sensors

MDPI

Article

Multipath Lightweight Deep Network Using Randomly Selected Dilated Convolution

Sangun Park and Dong Eui Chang *

School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, Korea; undol26@kaist.ac.kr

* Correspondence: dechang@kaist.ac.kr; Tel.: +82-42-350-7440

Abstract: Robot vision is an essential research field that enables machines to perform various tasks by classifying/detecting/segmenting objects as humans do. The classification accuracy of machine learning algorithms already exceeds that of a well-trained human, and the results are rather saturated. Hence, in recent years, many studies have been conducted in the direction of reducing the weight of the model and applying it to mobile devices. For this purpose, we propose a multipath lightweight deep network using randomly selected dilated convolutions. The proposed network consists of two sets of multipath networks (minimum 2, maximum 8), where the output feature maps of one path are concatenated with the input feature maps of the other path so that the features are reusable and abundant. We also replace the 3×3 standard convolution of each path with a randomly selected dilated convolution, which has the effect of increasing the receptive field. The proposed network lowers the number of floating point operations (FLOPs) and parameters by more than 50% and the classification error by 0.8% as compared to the state-of-the-art. We show that the proposed network is efficient.

Keywords: lightweight deep network; object classification; network design

Citation: Park, S.; Chang, D.E. Multipath Lightweight Deep Network Using Randomly Selected Dilated Convolution. *Sensors* **2021**, *21*, 7862. https://doi.org/10.3390/s21237862

Academic Editor: Stefanos Kollias

Received: 3 November 2021
Accepted: 24 November 2021
Published: 26 November 2021

1. Introduction

Object detection is one of the essential techniques that robots need to perform a variety of tasks. While humans can easily find and identify objects, robots are unable to do so. However, it is technically challenging to detect objects quickly and accurately in robot vision. Owing to its high importance, this field has received increased attention in recent years.

Deep convolutional neural networks (DCNNs) have attracted extensive attention in various computer vision applications such as object detection [1–7], object classification [8–15], and image segmentation [16–20]. DCNNs are composed of a series of convolutional layers, resulting in abundant features, more parameters, and complicated structures. These properties lead to a significant improvement in performance. Some of the prominent research involving DCNNs is as follows: combining networks in networks (NIN [21]), reducing the number of parameters by proposing a bottleneck layer (GoogLeNet [22]), placing several simple networks (VGGNet [11]), connecting an extra path between different layers (ResNet [12]), concatenating from previous layers to the next layers (DenseNet [13]), and increasing the number of channels as the layers get deeper (PyramidNet [23]).

However, as the applications of deep learning networks become more complex, the size of the model has increased rapidly. Nevertheless, deep learning networks are being deployed to lightweight devices such as mobile devices and automobiles. Large models have the following risks: memory limits, training/inference speed, performance degradation, and dead channels. The gradient required for training is proportional to the size of the model, so, even if the learning speed is increased through distributed learning, the training takes more time as the model grows. Many existing studies have attempted to solve this problem through training with multiple graphics processing units (GPUs), such as data

parallelization [24–27] and model parallelism [28–30]. Moreover, there are unnecessary channels that have little effect on the output result during the learning process. They can be a significant waste that continuously increases the computational complexity of the model. The pruning method [31] can resolve this problem.

Lightweight model design is designing a model with fewer parameters and computations while maintaining a similar level of performance. If the amount of computations is reduced, it enables deployment of the DCNN on low-power devices and secures real-time performance, and if the number of parameters is reduced, resources required for model storage and transmission are reduced. Therefore, it is very valuable to conduct research on lightweight model design.

In order to design lightweight models, we consider the following two fundamental questions:

1. Can the network be designed in a different way to make the model lighter?
2. How can we obtain richer feature maps than state-of-the-art (SOTA) traditional DCNNs?

We answer these questions with our proposed network, called the "multipath lightweight deep network using randomly selected dilated convolution". It consists of at least two multipath networks and uses a randomly selected dilated convolution to expand the receptive field.

Our main contributions are as follows.

- First, we design an extensible and modular network architecture. This model can be plugged into any existing network.
- Second, we reduce the the number of floating point operations (FLOPs) and parameters by more than 50%. Our model is composed of multipath network structures, so it is optimized for parallelization, and the model is light because the computation loss is small.
- Third, the proposed model has a wide receptive field and fewer parameters than the existing ones. This model can be placed in front of any network using the 3×3 standard convolution.

The remainder of this paper is organized as follows. Section 2 presents the related works about object classification methods and lightweight DCNN architectures. Section 3 explains and analyzes our proposed model. Section 4 shows our experimental results. Section 5 discusses the effect of our proposed network. Section 6 concludes the paper and presents possible future work.

2. Related Work

2.1. Object Classification

AlexNet [9] was the first network to popularize convolutional neural networks (CNNs). Unlike LeNet [8], AlexNet placed convolutional layers one after the other and improved performance by learning deep networks (8 layers) while utilizing GPU and rectified linear unit (ReLU) functions. The full-scale deep-layer era started with GoogLeNet [22]. GoogLeNet implemented an inception design to obtain features of different scales by applying different-scale convolution filters to the same layer. Especially, the bottleneck layer has a great effect on dimensionality reduction and computational cost reduction, so that a deeper network (22 layers) can be learned. However, as the network gets deeper, the gradient value saturates, which makes learning extremely slow. In addition, the error increases as the number of parameters increases. VGGNet [11] improved the performance by learning a deep network (19 layers) with only the simplest 3×3 convolution without changing the size of the receptive field.

The residual network (ResNet) [12] solved the vanishing gradient problem by adding shortcuts between adjacent layers, optimizing very deep networks (152 layers), and obtaining better performance with increasing depth based on the uncomplicated network, VGGNet. Later, DenseNet [13] showed improved performance with fewer parameters as compared to ResNet. Unlike ResNet, DenseNet connected a layer to all previous layers via

shortcut paths. By stacking feature maps, DenseNet can obtain very abundant feature maps and reduce the vanishing-gradient problems. Dual path networks (DPNs) [32] combined the advantages of ResNet for feature reuse and DenseNet for exploring new features. As the name suggests, DPNs consist of dual paths. One path is a ResNet network, while the other path is a DenseNet network.

2.2. Lightweight CNN Architectures

Existing object classification and detection models require a lot of computation power for training and testing, so expensive equipment such as GPUs is necessary. The model size is also relatively large, and it takes a lot of time to train and test, so improvement in model size and computational efficiency is essential for real-time application. To solve this problem, various attempts have been made to compress the deep network or reduce the amount of computation. In particular, recent studies have explored reducing the weight of the model while maintaining the performance of the existing model.

Based on the effect of Inception, Xception [33] proposed a depthwise-separable convolution network. SqueezeNet [34] reduced computation cost and the number of input channels by replacing some 3×3 convolution layers with point-wise convolution. MobileNet [35] proposed a lightweight architecture structure that can run on mobile devices through depthwise-separable convolutions. ShuffleNet [36] proposed a more efficient structure than MobileNet by applying group convolution to bottleneck layer operation and shuffle channels.

CondenseNet [14] achieved similar accuracy with a lower computational cost than other lightweight models such as MobileNet and ShuffleNet. It is a model with similar accuracy to DenseNet requiring one tenth of the computation power by pruning connections with less feature reuse by using learned group convolution, and increasing the growth rate as the network gets deeper. MobileNetV2 [37] proposed linear bottlenecks and an inverted residual to upgrade the architecture while improving performance in all indicators such as accuracy, the number of parameters, and amount of computation. ShuffleNetv2 [38] added a channel-splitting module to input and used concatenation instead of addition, resulting in faster processing speed with similar accuracy to ShuffleNetv1 as well as MobileNetV2. MobileNetv3 [39] used the NASNet [40] architecture to explore the structure and improved the performance by modifying the searched structure. CondenseNetV2 [15] used reactivating obsolete features not considered in CondenseNet and ShuffleNetV2. In addition, by adding a sparse feature reactivation (SFR) module after the existing learned group convolution, features were concatenated after processing.

3. Methods

In this section, we introduce the details of our proposed network. The model we refer to as the basic structure is DenseNet-based (including CondenseNet and CondenseNetV2) because the information of the previous layer is concatenated and the features are reused. That is, it is characterized by having much richer features compared to other networks that do not concatenate. In particular, CondenseNet can be used for lightweight models because it reduces the number of parameters ten times as compared to DenseNet and provides similar performance. Hence, we chose this as the basic model. Since CondenseNet has recently been improved to CondenseNetV2, we also applied the proposed method to CondenseNetV2.

The major differences between the proposed network and other network architectures are the presence of multipath networks and the expansion of the receptive field. Dividing the path into a pair of cross-shaped paths is the key to reducing the number of FLOPs and parameters. A detailed description is provided in Sections 3.2 and 3.3.

Figure 1 shows the overall architecture of the proposed network. MLDN described in Section 3.2 represents our proposed multipath lightweight deep network. In the figure, P in the dark purple box is the preprocessing module, R in the light purple box is ResNet, SP in the dark blue box splits the input into the number of paths, cat in the yellow box is

concatenation, H in the orange box is the composite function of MLDN, PL in the light blue box is the average pooling layer, FC in the green box denotes fully connected layers, and S in the red box is a softmax function. In and out in the white box are the input images and the predicted class, respectively. This network predicts the classes of objects through preprocessing, dense blocks, transition layers (average pooling), fully connected layers, and a softmax function. Preprocessing helps generate diverse feature maps. A denseblock has L MLDN and MLDN has p multiple paths. The various feature maps are produced after passing through each MLDN and denseblock. The pooling layer reduces the size of the output channels of each denseblock.

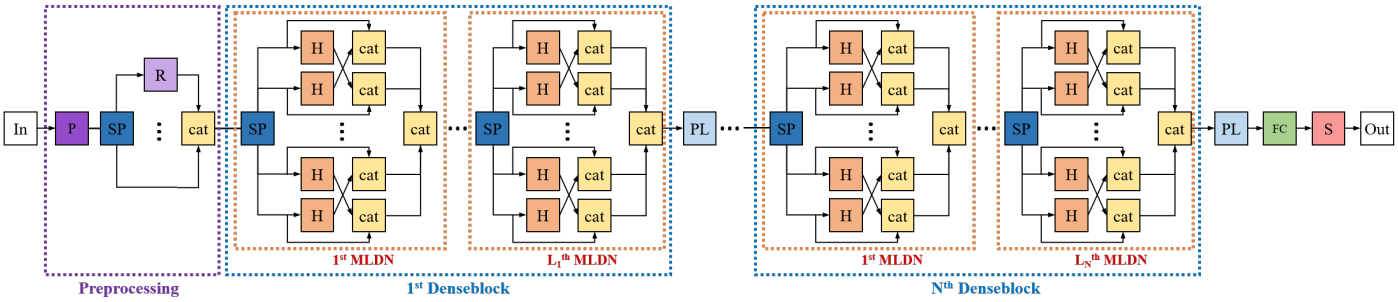


Figure 1. Overall procedure. The input goes through preprocessing and several dense blocks and pooling layers. During this procedure, the feature maps reduce the the number of FLOPs and parameters.

3.1. Preprocessing

Figure 2 describes preprocessing. Before using the input image itself, we first increment the feature maps by passing an initial convolutional layer (dark purple box). We mentioned that the network is divided into multiple paths. ResNet-18, which has the least number of FLOPs among various ResNets, is applied once to one of these paths. Henceforth, we use ResNet-18 wherever we mention ResNet.

This is similar to applying network-in-network (NIN) on GoogLeNet. Since it is one of the many processes, it does not increase the number of FLOPs or parameters significantly. Since ResNet performs an element-wise sum by shortcut connection of input features to output features, it transforms the feature itself without using it. Therefore, the path passing through ResNet and the path not passing through ResNet can be configured with different feature maps to improve performance.

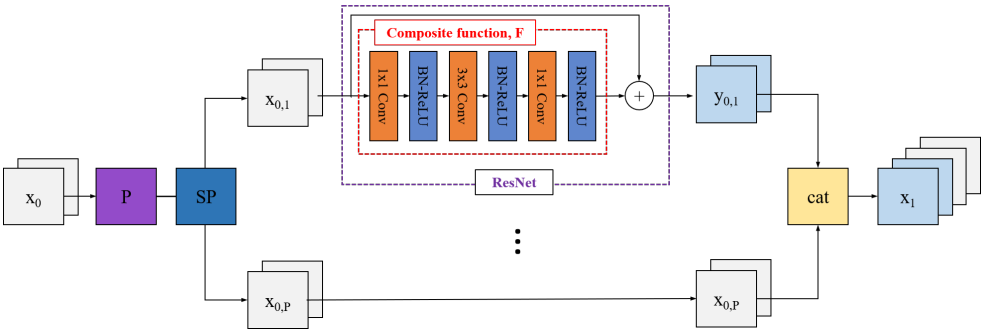


Figure 2. Preprocessing. Before splitting input feature maps, we apply initial convolution. Only one path has the ResNet module, and the other paths just pass the input feature maps.

The composite function, $F()$, consists of combination of convolution, batch normalization (BN) [41], and ReLU [42] layers including the bottleneck layer. We adjust the feature maps so that the number of output feature maps that pass through ResNet equals the number of output feature maps on the paths that do not. We can express the ResNet path as $y_1 = x_1 \oplus F(x_1)$, where x_1 are the input feature maps of the ResNet path, y_1 are the output feature maps of ResNet, and $\oplus$ is the element-wise sum operation.

3.2. Multipath Lightweight Deep Network

Most deep networks except GoogLeNet and DPN have only one path. GoogLeNet separates the network by placing the network in the network, but the networks do not exchange feature information with each other. DPN combines two networks to take advantage of both networks.

We propose a lightweight deep network with multiple paths to make a model suitable for weight reduction. The key to making a lightweight model as compared to existing networks is by reducing the number of parameters through multipath networks. Depending on the network design, the number of paths can be defined as a divisor of the growth rate. The growth rate of [13] controls the amount of information added to the network at each layer. For example, if the growth rate is 8, the possible paths are 2, 4, and 8. Algorithm 1 describes the proposed network, multipath lightweight deep network (MLDN), in nth denseblock in CondenseNet.

Algorithm 1 MLDN in nth denseblock in CondenseNet

Input: x_n
for $l = 1$ to L_n **do**
  $x_{n,l,2p-1}, x_{n,l,2p} = \text{split}(x_{n,l})$ where $1 \leq p \leq P/2$
  for $p = 1$ to $P/2$ **do**
    $y_{n,l,2p-1} = \text{cat}(x_{n,l,2p-1}, H_{n,l,2p}(x_{n,l,2p}))$
    $y_{n,l,2p} = \text{cat}(x_{n,l,2p}, H_{n,l,2p-1}(x_{n,l,2p-1}))$
  end for
  $x_{n,l+1} = \text{cat}(y_{n,l,1}, y_{n,l,2}, \ldots, y_{n,l,2p-1}, y_{n,l,2p})$
end for
Output: $x_{n+1} = x_{n,L_n+1}$

Here, n is the index of the denseblock, l is the index of layers, L_n is the number of layers in the nth denseblock, p is the index of the path, P is the number of paths, $H(\cdot)$ is the composite function, cat is the concatenation operation, and split is the split operation.

Figure 3 is an example of dividing the path into two ($P = 2$) in the lth layer. In Figure 3a, when $l = 1$, MLDN is explained as follows. First, input feature maps ($x_{n,1}$) split by the number of 2 ($x_{n,1,1}, x_{n,1,2}$). These split input feature maps are passed through the composite function ($H_{n,1,1}(x_{n,1,1}), H_{n,1,2}(x_{n,1,2})$). Next, they concatenate with the input feature maps of the opposite path ($y_{n,1,1} = \text{cat}(x_{n,1,1}, H_{n,1,2}(x_{n,1,2})), y_{n,1,2} = \text{cat}(x_{n,1,2}, H_{n,1,1}(x_{n,1,1}))$). Finally, all output feature maps are re-concatenated ($x_{n,2} = \text{cat}(y_{n,1,1}, y_{n,1,2})$. These output feature maps become the next input feature maps for the $(l+1)$th layer. This is repeated on all layers. In this way, information is exchanged between paths. Each composite function has two iterations of BN, ReLU, and convolution maps in series. For the first convolutional layer, learned group convolution (L-Conv) removes unimportant connections. For the second convolutional layer, group convolution (G-Conv) reduces the computational cost by partitioning the input features. In the composite function process, the number of feature maps does not increase and a constant k, the number of feature maps, is generated because of a bottleneck layer. More information about the composite function is in [14].

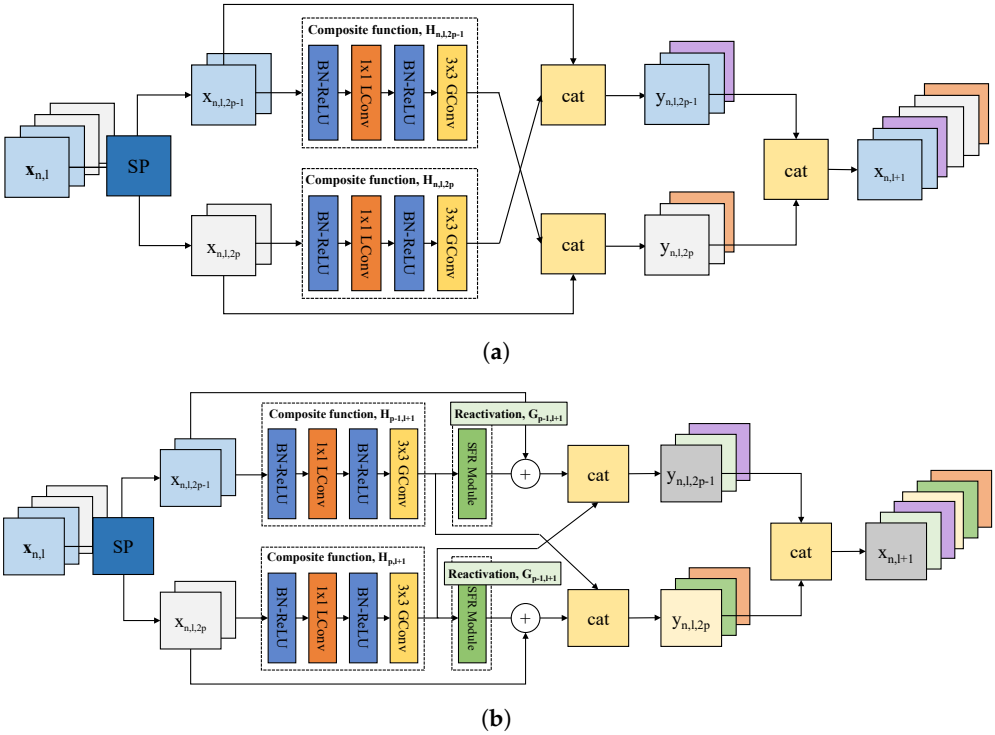


Figure 3. Multipath lightweight deep network (MLDN) in lth layers, nth denseblock. (**a**) MLDN in CondenseNet; (**b**) MLDN in CondenseNetV2. They pass their output feature maps to the other path.

CondenseNetV2 adds a sparse feature reactivation (SFR) module after the composite function, and the output feature maps of this SFR module are added by an element-wise sum with the input feature maps. Finally, the input feature maps from the $(2p-1)$th path and output feature maps from the $2p$th path are concatenated as in CondenseNet. Algorithm 2 explains how to plug in MLDN to CondenseNetV2. The composite function G in the SFR increases the output channels equal to the input feature maps. The rest is the same notation as in Algorithm 1. Since it is similar to MLDN+CondenseNet, the description has been omitted.

Algorithm 2 MLDN in nth denseblock in CondenseNetV2

Input: x_n
for $l = 1$ to L_n **do**
  $x_{n,l,2p-1}, x_{n,l,2p} = \text{split}(x_{n,l})$ where $1 \leq p \leq P/2$
  for $p = 1$ to $P/2$ **do**
    $y_{n,l,2p-1} = \text{cat}(x_{n,l,2p-1} \oplus G_{n,l,2p-1}(H_{n,l,2p-1}(x_{n,l,2p-1})), H_{n,l,2p}(x_{n,l,2p}))$
    $y_{n,l,2p} = \text{cat}(x_{n,l,2p} \oplus G_{n,l,2p}(H_{n,l,2p}(x_{n,l,2p})), H_{n,l,2p-1}(x_{n,l,2p-1}))$
  end for
  $x_{n,l+1} = \text{cat}(y_{n,l,1}, y_{n,l,2}, \ldots, y_{n,l,2p-1}, y_{n,l,2p})$
end for
Output: $x_{n+1} = x_{n,L_n+1}$

Our network architectures are shown in Table 1. A denseblock is composed of L-Conv (learned convolution) and G-Conv (grouped convolution). We choose $L = 14$ layers and P = [2,4,8] for each denseblock. The total growth rate k is [8,16,32], so the growth rate per path k_p is [4,4,4] ($k_{n,p} = k_n / p_n$). In the last layer, the output of the FC layer is the same as the number of labels in the dataset. As an example, the outputs of the FC layer in the cases of the CIFAR-10 and CIFAR-100 datasets are 10 and 100, respectively.

Table 1. Architecture of MLDN in CondenseNet/CondenseNetV2 on the CIFAR-10/100 dataset.

MLDN in CondenseNet	MLDN in CondenseNetV2	Feature Map Size
3×3 Conv (stride 1)	3×3 Conv (stride 1)	$32 \times 32 \times 16$
ResNet	ResNet	$32 \times 32 \times 16$
$\begin{bmatrix} 1 \times 1 \text{ L-Conv} \\ 3 \times 3 \text{ G-Conv} \end{bmatrix} \times 14 \times$ $2, k_p = 4, (k = 8)$	$\begin{bmatrix} 1 \times 1 \text{ L-Conv} \\ 3 \times 3 \text{ G-Conv} \\ \text{SFR Module} \end{bmatrix} \times 14 \times$ $2, k_p = 4, (k = 8)$	$32 \times 32 \times 128$
2×2 average pool, stride 2	2×2 average pool, stride 2	$16 \times 16 \times 128$
$\begin{bmatrix} 1 \times 1 \text{ L-Conv} \\ 3 \times 3 \text{ G-Conv} \end{bmatrix} \times 14 \times$ $4, k_p = 4, (k = 16)$	$\begin{bmatrix} 1 \times 1 \text{ L-Conv} \\ 3 \times 3 \text{ G-Conv} \\ \text{SFR Module} \end{bmatrix} \times 14 \times$ $4, k_p = 4, (k = 16)$	$16 \times 16 \times 352$
2×2 average pool, stride 2	2×2 average pool, stride 2	$8 \times 8 \times 352$
$\begin{bmatrix} 1 \times 1 \text{ L-Conv} \\ 3 \times 3 \text{ G-Conv} \end{bmatrix} \times 14 \times$ $8, k_p = 4, (k = 32)$	$\begin{bmatrix} 1 \times 1 \text{ L-Conv} \\ 3 \times 3 \text{ G-Conv} \\ \text{SFR Module} \end{bmatrix} \times 14 \times$ $8, k_p = 4, (k = 32)$	$8 \times 8 \times 800$
8×8 global average pool	8×8 global average pool	$1 \times 1 \times 800$
FC softmax	FC softmax	

3.3. Randomly Selected Dilated Convolution

A large receptive field can be used to improve network performance. However, this considerably increases the number of parameters and risks overfitting. Therefore, in general DCNNs, these problems are solved by combining convolution and pooling to lower the cost. Factorized convolution reduces the number of parameters and deepens the layers by replacing the feature maps of a large receptive field with a few other small feature maps, but it increases the depth of the network. To widen the receptive field and not deepen the network at the same time, we draw inspiration from dilated convolutions [43].

Dilated convolution expands small feature maps (3×3) into large feature maps (5×5, $7 \times 7 \ldots$) but, conversely, we reduce large feature maps to small feature maps while using all the information in the large feature maps. We propose a randomly selected dilated convolution (RSDC) with an extended receptive field but a relatively shallow layer.

Figure 4 explains our RSDC when the kernel size of the RSDC is 5. The left side of the figure shows the input feature maps, $*$ is the convolution symbol, and the right side of the figure shows the RSDC, where RSDC consists of M' feature maps. In this case, there are 25 weights. If we use this value as it is, however, it is the same as increasing the receptive field. We randomly select 9 of these weights and perform convolution with them. The reason for choosing 9 weights is that the size of the existing standard convolution is 3×3.

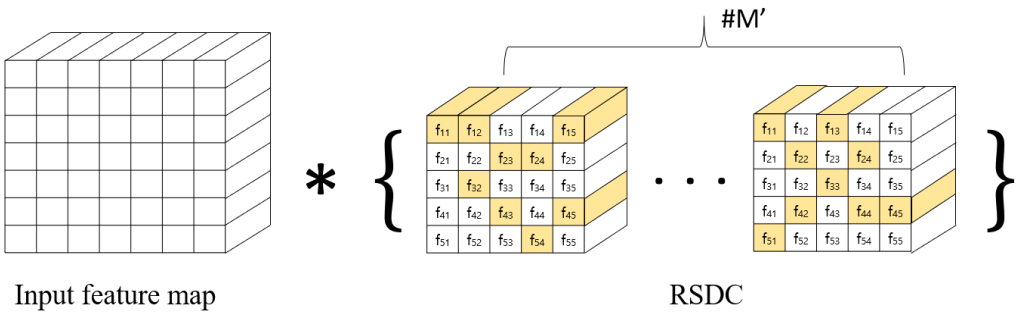


Figure 4. Randomly selected dilated convolution (RSDC) when kernel size is 5. The yellow on the right of the figure denotes randomly selected weights. There are M' output feature maps.

Algorithm 3 explains our network with RSDC in a general case. The $\mathbf{f}^{\text{input}}$ denotes the RSDC feature maps depicted on the right side of Figure 4 and $\mathbf{f}^{\text{output}}$ denotes randomly selected weights shown in yellow on the right side of Figure 4. The height and width of the input feature maps of RSDC are denoted by h, w, respectively; k_{rsdc} is the predefined kernel size for RSDC, e.g., 5 or 7; and i, j are height and width indices of the output feature maps of RSDC, respectively. The rand(x) function picks a random value of x, and the append(x) function appends x to the output.

Algorithm 3 RSDC before applying 3×3 standard convolution feature maps

Input: $\mathbf{f}^{\text{input}} = \{f_{h,w}\}$, where $1 \leq h, w \leq k_{rsdc}$
for $i = 1$ to 3 **do**
  for $j = 1$ to 3 **do**
    $f = \text{rand}(f_{h,w})$
    while f in $f_{i,j}$ **do**
      $f = \text{rand}(f_{h,w})$
    end while
    $\text{f}_{i,j} = \text{append}(f)$
  end for
end for
Output: $\mathbf{f}^{\text{output}} = \{\text{f}_{i,j}\}$, where $1 \leq i, j \leq 3$

In general, according to the factorized convolution method, the number of M feature maps of size k_{rsdc} that can be factorized into several 3×3 feature maps is given by:

$$k_{rsdc}^2 \times M \rightarrow 3^2 \times M \times \frac{(k_{rsdc} - 1)}{2}.$$

Since our RSDC uses only 9 weights, we can think of it as using 3×3 convolution feature maps (i.e., $k_{rsdc} = 3$ in the preceding equation). Therefore, to use fewer feature maps as compared to existing feature maps while obtaining a large receptive field, the following should be satisfied:

$$M' < M \times \frac{(k_{rsdc} - 1)}{2},$$

where M' is the number of feature maps of RSDC in Figure 4.

The MLDN exceeds SOTA (shown in Section 4.3.2), so we apply RSDC to MLDN. Among the composite functions of MLDN, a 3×3 convolution exists only once at the end. The proposed RSDC is located instead of the 3×3 standard convolution of MLDN. In one path of the first denseblock, we have L_1 layers and growth rate k_p, so the increasing number of output feature maps is $L_1 \times k_p$. Since the growth rate is too small in the first denseblock, we apply RSDC from the second denseblock.

4. Experimental Results

4.1. Datasets

We evaluated our proposed network on the CIFAR-10 and CIFAR-100 [44] datasets, and the ImageNet (ILSVRC) [45] datasets. The CIFAR-10 and CIFAR-100 datasets are composed of 32×32 pixel-sized RGB images corresponding to 10 and 100 classes, respectively. They have 50,000 training images and 10,000 testing images. We used a standard data-augmentation method [21,46–50] where the images were zero-padded to 4 pixels on all sides with a probability of 0.5, randomly cropped, and mirrored horizontally to keep the size of 32×32 pixels. We separated the 10,000 images from the training dataset into the validation dataset. The ImageNet dataset consists of 1000 classes and contains a total of 1.2 million training images and 50,000 validation images. We adopted the data-augmentation method of [12] at training time, rescaled the input image to 256×256 at test time, and then performed a 224×224 center crop.

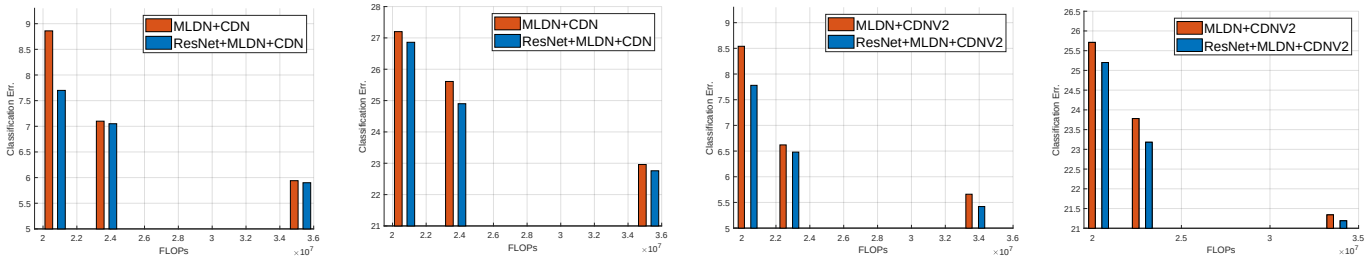
4.2. Training Settings

All models were trained by stochastic gradient descent (SGD) using similar optimization hyperparameters as in [14,15]. We adopted the Nesterov momentum weight of 0.9 without dampening and used a weight decay of 1×10^{-4}. All models were trained with a mini-batch size of 32 for 300 epochs. The cosine-shaped learning rate [51] was used, and it started at 0.1 and gradually decreased to 0. Dropout [52,53] with a drop rate of 0.1 was applied to train.

4.3. Performance Evaluation

4.3.1. The Effect of ResNet

We now show experimental validation for the fact that using ResNet is more effective than not using it as explained in Section 3.1. Figure 5 shows the classification error as a function of the number of FLOPs. In this paper, the classification error means top-1 error. Detailed values are given in Table 2.



(**a**) (**b**) (**c**) (**d**)

Figure 5. Comparison of the classification error rates with respect to the number of FLOPs on the effect of ResNet and MLDN. (**a**) MLDN in CondenseNet on CIFAR-10. (**b**) MLDN in CondenseNet on CIFAR-100. (**c**) MLDN in CondenseNetV2 on CIFAR-10. (**d**) MLDN in CondenseNetV2 on CIFAR-100.

Although the number of FLOPs in the case of the network with ResNet is slightly larger than those without ResNet (about 1M), we see a decrease in the classification error of the network. These values can be seen in the 1st (CDN) to the 3rd (MLDN+CDN) rows of the first column (Model) in Table 2. In the case of CondensetNetV2, a similar effect can be seen as shown in the 4th row (CDNV2) to the 6th row (MLDN+CDNV2) of the first column (Model) of Table 2.

Table 2. Comparison of the classification error rates (%) of CondenseNet/CondenseNetV2 (original) and MLDN+CDN/MLDN+CDNV2 (proposed) on the CIFAR-10 and CIFAR-100 datasets.

Model	Paths (*P*)	Growth Rate (*k*)	FLOPs [M]	Params [M]	C-10	C-100
CDN		[8,16,32]	65.8	0.52	6.03	23.71
MLDN+CDN	[2,2,2]	[8,16,32]	34.9	0.28	5.94	22.96
	[2,4,4]	[8,16,32]	23.4	0.16	7.1	25.61
	[2,4,8]	[8,16,32]	20.3	0.12	8.86	27.2
MLDN+CDN with ResNet	[2,2,2]	[8,16,32]	35.6	0.28	5.9	22.76
	[2,4,4]	[8,16,32]	24.1	0.17	7.05	24.9
	[2,4,8]	[8,16,32]	21.1	0.12	7.7	26.86
	[2,2,2]	[16,16,16]	61.0	0.22	5.41	22.25
	[2,2,2]	[32,32,32]	231.4	0.78	4.75	21.58
	[2,2,2]	[16,32,64]	134.6	1.04	4.69	18.05
	[2,4,4]	[16,32,64]	88.8	0.72	4.93	19.71
	[2,4,8]	[16,32,64]	76.5	0.39	5.28	21.35
CDNV2		[8,16,32]	64.3	0.51	5.80	22.31
MLDN+CDNV2	[2,2,2]	[8,16,32]	33.4	0.27	5.66	21.34
	[2,4,4]	[8,16,32]	22.4	0.15	6.62	23.78
	[2,4,8]	[8,16,32]	20.0	0.10	8.54	25.71
MLDN+CDNV2 with ResNet	[2,2,2]	[8,16,32]	34.1	0.26	5.42	21.19
	[2,4,4]	[8,16,32]	23.2	0.16	6.48	23.18
	[2,4,8]	[8,16,32]	20.7	0.11	7.78	25.20
	[2,2,2]	[16,16,16]	59.9	0.20	5.15	20.50
	[2,2,2]	[32,32,32]	230.0	0.77	4.49	18.43
	[2,2,2]	[16,32,64]	133.1	1.04	3.66	16.37
	[2,4,4]	[16,32,64]	86.9	0.70	4.55	18.25
	[2,4,8]	[16,32,64]	74.5	0.38	4.83	19.70

4.3.2. The Effect of Multiple Paths

We studied the effect of changing the number of paths and the growth rate on the proposed network. We designed the network such that each denseblock has 14 layers. We chose three sets of paths: P = [2,2,2], P = [2,4,4], and P = [2,4,8], such that P = [2,4,8] implies that the first denseblock has 2 paths, the second denseblock has 4 paths, and the third denseblock has 8 paths. The growth rate of the paths also increases such that the first denseblock has growth rate 8, the second denseblock has growth rate 16, and the third denseblock has growth rate 32. Figure 5 depicts the effect of multiple paths. In the case of MLDN+CDN/CDNV2 or ResNet+MLDN+CDN/CDNV2, p decreases from left (smaller FLOPs) to right (larger FLOPs) such as P = [2,2,2], P = [2,4,4], and P = [2,4,8]. Larger paths reduce the number of FLOPs because more operations are processed at one time. However, the number of output feature maps (actually, $\frac{\#\text{output feature maps}}{\#\text{paths}}$) used for training becomes smaller and the classification error increases.

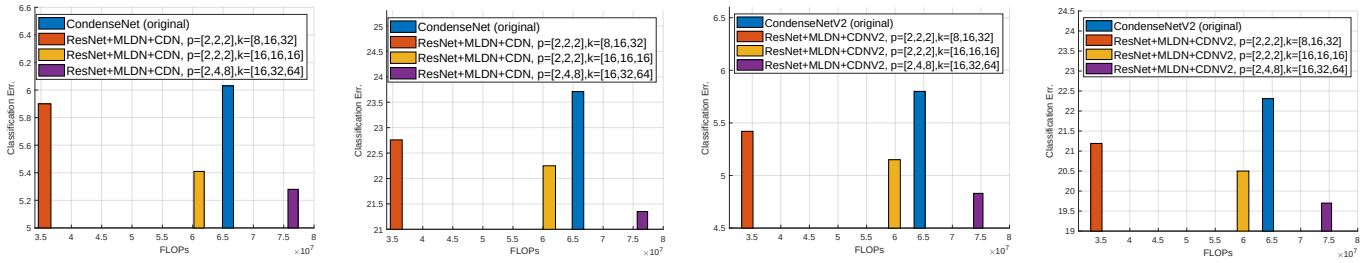
Detailed values are given in Table 2. It can be seen that the best result is shown when p = [2,2,2] with a constant size. Our MLDN+CDN model shows a 0.13% and 0.95% improvement in the classification error on CIFAR-10 and CIFAR-100, respectively. It reduces the number of FLOPs and parameters by 54.1% and 53.8% compared to the CDN, respectively. MLDN+CDNV2 improves by 0.37% and 1.12% in the classification error on CIFAR-10 and CIFAR-100 respectively. It reduces the number of FLOPs and parameters by 53% and 51% compared to the CDNV2, respectively.

4.3.3. The Effect of Changing MLDN Hyperparameters

We also performed the experiment with various MLDN hyperparameters. We compare the original network, our best models, increased and constant growth rates, and doubling growth rates in Figure 6.

First, we experimented with constant growth rates such as k = [16,16,16], and k = [32,32,32] when p = [2,2,2], so the growth rates per path were k_p = [8,8,8] and k_p = [16,16,16], respectively, which are sufficient to train well. However, in the case of k = [32,32,32], the number of FLOPs is too large to be meaningful. Hence, we did not plot for k = [32,32,32] in Figure 6 due to the scale problem. We see that as the growth rates increase with constant values, the classification error decreases but the number of FLOPs increases. There are trade-offs between these two.

Second, we experimented with doubling the growth rates. Unlike the best results where we chose p = [2,2,2] and k = [8,16,32], the number of FLOPs is slightly larger than the original (11M), but the classification error is reduced by 0.7% when p = [2,4,8] and k = [16,32,64]. This is because doubling k is enough to satisfy the training well. When p = [2,2,2] and k = [16,32,64], the number of FLOPs becomes too large to be worthwhile. Except for p = [2,4,8] and k = [16,32,64], the rest are not depicted in Figure 6 because of scale issues.

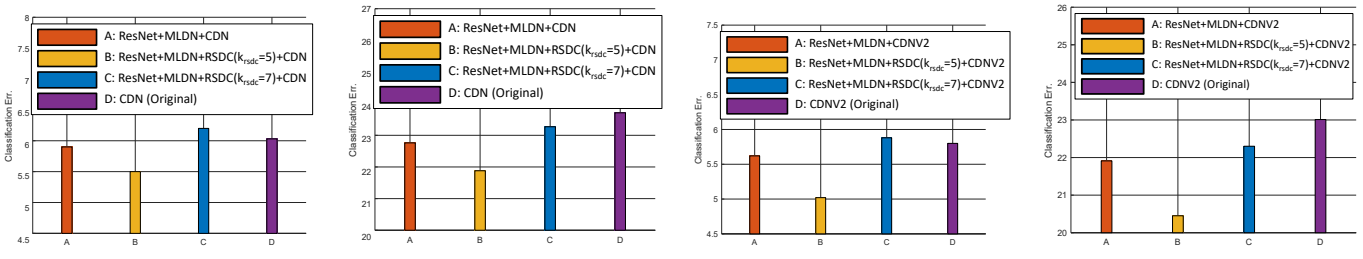


(**a**) (**b**) (**c**) (**d**)

Figure 6. Comparison of the classification error rates with respect to the number of FLOPs using MLDN on the effect of changing hyperparameters. (**a**) MLDN in CondenseNet on CIFAR-10. (**b**) MLDN in CondenseNet on CIFAR-100. (**c**) MLDN in CondenseNetV2 on CIFAR-10. (**d**) MLDN in CondenseNetV2 on CIFAR-100.

4.3.4. The Effect of RSDC

From the above results (in Sections 4.3.1 and 4.3.2), we confirm that having multiple paths and ResNet is more effective than the existing networks. Therefore, we applied RSDC to MLDN with ResNet to experiment with its effectiveness. The results were best when p = [2,2,2] and k = [8,16,32], so we set the same for this experiment. Figure 7 compares original, MLDN, and RSDC.



(**a**) (**b**) (**c**) (**d**)

Figure 7. Comparison of the classification error rates with respect to the number of FLOPs using RSDC. (**a**) ResNet+MLDN+RSDC+CDN on CIFAR-10. (**b**) ResNet+MLDN+RSDC+CDN on CIFAR-100. (**c**) ResNet+MLDN+RSDC+CDNV2 on CIFAR-10. (**d**) ResNet+MLDN+RSDC+CDNV2 on CIFAR-100.

First, we experimented by changing the kernel size of RSDC, k_{rsdc}. The classification error was lowest when k_{rsdc} was 5. This is because the input image size of the CIFAR

dataset is so small (32×32) and it is inefficient to use a large receptive field such as $k_{rsdc} = 7$. In addition, $k_{rsdc} = 5$ has larger selected weights than k_{rsdc} (9/25 vs. 9/49) for the same number of output feature maps. Detailed values are given in Table 3.

Table 3. Comparison of the classification error rates (%) of CondenseNet/CondenseNetV2 (original) and MLDN+CDN/MLDN+CDNV2 and MLDN+RSDC+CDN/MLDN+RSDC+CDNV2 (proposed) on the CIFAR-10 and CIFAR-100 datasets.

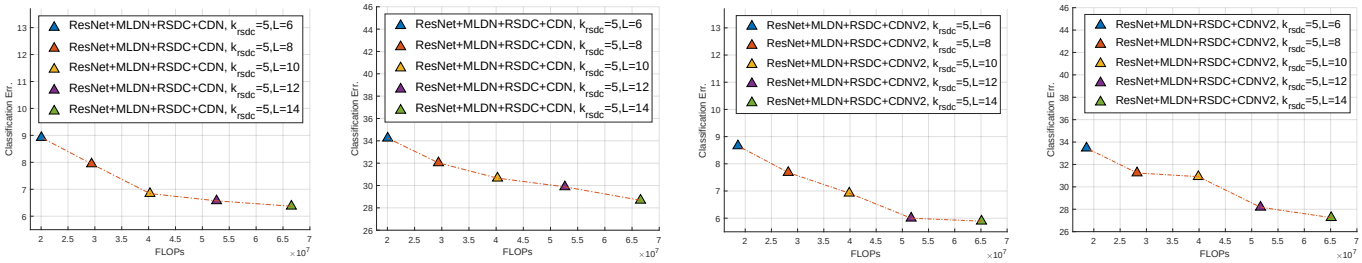
Model	k_{rsdc}	#L	C, G	FLOPs	Params	C-10	C-100
CDN		14	4	65.8M	0.52M	6.03	23.71
MLDN+CDN		14	4	35.6M	0.28M	5.9	22.76
MLDN+RSDC+CDN	5	14	4	36.1M	0.28M	5.5	21.88
	7	14	4	36.4M	0.28M	6.2	23.27
	5	14	2	66.6M	0.52M	6.37	28.68
	5	12	2	52.6M	0.41M	6.57	29.88
	5	10	2	40.3M	0.31M	6.84	30.66
	5	8	2	29.4M	0.22M	7.94	32.03
	5	6	2	20.1M	0.14M	8.92	34.24
CDNV2		14	4	64.3M	0.51M	5.80	23.01
MLDN+CDNV2		14	4	34.1M	0.27M	5.62	21.91
MLDN+RSDC+CDNV2	5	14	4	34.6M	0.27M	5.02	20.45
	7	14	4	35.2M	0.27M	5.88	22.30
	5	14	2	65.1M	0.5M	5.89	27.25
	5	12	2	51.7M	0.4M	6.00	28.18
	5	10	2	39.9M	0.29M	6.92	30.91
	5	8	2	28.2M	0.21M	7.68	31.24
	5	6	2	18.6M	0.13M	8.66	33.46

Second, we changed the number of layers. In [14,15], the condensation factor (C) and the number of groups (G) is 4. The condensation factor is the removal rate of the filter weight. With this factor, it is impossible to experiment with varying reductions in L for computational reasons such as size mismatch. Therefore, we changed this factor to 2. We changed L by 2 from 6 to 14. The smallest number of FLOPs and parameters is when $L = 6$, and the largest number of FLOPs and parameters is when $L = 14$. The classification error at $L = 10$ in Figure 8a or $L = 12$ in Figure 8c,d is similar to the previous classification error, but since the number of FLOPs is reduced by about 15 million, it is the best choice to obtain the lowest classification error with respect to the the number of FLOPs. When the number of layers is smaller than the L mentioned above, the classification error tends to increase remarkably because M', the number of output feature maps of RSDC, is not sufficient.

4.3.5. Comparison with SOTA

Table 4 compares the classification error rates with SOTA for various networks on the CIFAR datasets. It can be seen that our proposed network (bottom row of Table 4) significantly reduces the number of FLOPs and parameters compared to SOTA. In particular, compared to CondenseNet, the number of FLOPs is reduced by about 55%, and the classification error is reduced slightly by about 0.1%. Compared to CondenseNetV2, the number of FLOPs is reduced by about 53%, the number of parameters is reduced by about 53%, and the classification error is reduced by about 0.78%. This proves the effectiveness of the multipath method and RSDC. We also experimented with the ImageNet dataset. Compared with CondenseNet, our proposed model (the bottom row of Table 5) reduces

the number of FLOPs by more than 35% and improves top-1 classification error by more than 1.6%.



(**a**) (**b**) (**c**) (**d**)

Figure 8. Comparison of the classification error rates with respect to the number of FLOPs using RSDC with changes in the number of layers when $k_{rsdc} = 5$ and $G = C = 2$. (**a**) ResNet+MLDN+RSDC+CDN on CIFAR-10. (**b**) ResNet+MLDN+RSDC+CDN on CIFAR-100. (**c**) ResNet+MLDN+RSDC+CDNV2 on CIFAR-10. (**d**) ResNet+MLDN+RSDC+CDNV2 on CIFAR-100.

Table 4. Comparison of the classification error rate (%) with other convolutional networks on the CIFAR-10 and CIFAR-100 datasets.

Model	FLOPs	Params	CIFAR-10	CIFAR-100
ResNet-1001 [54]	2357M	16.1M	4.62	22.71
Stochastic-Depth-1202 [46]	2840M	19.4M	4.91	-
Wide-ResNet-28 [55]	5248M	36.5M	4.00	19.25
ResNeXt-29 [56]	10,704M	68.1M	3.58	17.31
DenseNet-190 [13]	9388M	25.6M	3.46	17.18
NASNet-A [40]	-	3.3M	3.41	-
CondenseNet(light)-160	1084M	3.1M	3.46	17.55
CondenseNet-182	513M	4.2M	3.76	18.47
ResNet-based				
CP [57]	62M	-	8.2	-
PFEC [58]	90M	0.73M	6.94	-
LECN [59]	124M	1.21M	5.27	23.91
NISP [60]	142M	0.96M	6.88	-
FPGM [61]	121M	-	6.24	-
DenseNet-based				
LECN [59]	190M	0.66M	5.19	25.28
CondenseNet [14]	66M	0.52M	6.03	23.71
CondenseNetV2-146 [15]	64M	0.51M	5.8	23.01
MLDN+RSDC-based				
MLDN+RSDC+CDN	36M	0.28M	5.5	21.8
MLDN+RSDC+CDNV2	35M	0.27M	5.02	20.45

Table 5. Comparison of classification error rate (%) with other SOTA networks on the ImageNet datasets.

Model	FLOPs	Params	Top-1	Top-5
Inception V1 [22]	1448M	6.6M	30.2	10.1
1.0 MobileNet-224 [35]	569M	4.2M	29.4	10.5
ShuffleNet 2x [36]	524M	5.3M	29.1	10.2
NASNet-A (N = 4) [40]	564M	5.3M	26.0	8.4
ShuffleNetV2 1.5x [38]	299M	-	27.4	9.4
1.0 MobileNetV2 [62]	300M	3.4M	28.0	9.0
MobileNetV3 L. 1.0x [39]	219M	5.4M	24.8	-
CondenseNet (G = C = 8) [14]	274M	2.9M	29.0	10.0
CondenseNetV2-C [15]	309M	6.1M	24.1	7.3
MLDN+RSDC+CDN	177M	2.3M	27.4	9.2

5. Discussion

We summarize the effect of our network as follows. First, the reuse of parameters is excellent. Unlike the existing convolutional neural network models that use only the last high-level feature maps and drop the previously produced feature maps, the DenseNet-based model uses both high-complexity feature maps as well as low-level feature maps to be more effective. Because the channel of DenseNet is narrow, it shows good performance with small parameters compared to other networks. Therefore, regularization is not required.

Second, our model satisfies the model complexity by crossing connections and passes one feature map to the other with a cross-shaped structure, not by increasing the number of channels. In addition, this mixing of information between paths has the same effect as shuffling for each group in ShuffleNet. The vanishing gradient problem is smoothed out by transferring the error directly to the beginning of the network during the backpropagation process.

Third, the numbers of FLOPs and parameters are reduced. In group convolution, the number of parameters decreases in proportion to the number of groups according to the relation:

$$HWk^2 \frac{C}{G} \frac{M}{G} G = \frac{HWk^2CM}{G},$$

where H is the height of input feature maps, W is the width of input feature maps, k is the convolution filter size, C is the number of input channels, M is the number of output channels, and G is the number of groups.

Since our network is divided into P paths, the following should be satisfied:

$$HWk^2 \frac{C}{PG} \frac{M}{PG} PG = \frac{HWk^2CM}{PG},$$

where P is the number of paths and the rest of the notation is the same as in the above equation. That is, our network has the effect of reducing parameters by $1/P$ compared to other models.

Finally, our network is divided by the number of paths defined in advance for each denseblock, which is more beneficial for parallelization than existing networks. Besides, our network can easily be plugged into any CNN that adopts the concatenation-based feature reuse mechanism.

However, our network takes about 4 times longer to train as compared to the original models (CondenseNet or CondenseNetV2). This is due to the following two reasons. First, we concatenate the output feature maps through the composite function from one path to the other. This process requires intensive computational resources while the original

network does not due to the absence of the process. Second, the computational cost increases in the process of splitting the input feature maps and concatenating the output feature maps.

6. Conclusions

This paper has dealt with the effects of multiple paths and randomly selected dilated convolutions on lightweight deep networks. Our proposed network has multiple paths, and the diversity is enhanced by adding a ResNet in front of one path. We concatenate the output feature maps through the composite function from one path to the other. This helps to produce rich feature maps and is more suitable for parallelization than other models. The architecture of the proposed network is modularized and can be expanded by increasing or decreasing the number of paths. By adding RSDC instead of the 3×3 standard convolution, we obtain the effect of a large receptive field and improve the result. We compared our network with various SOTA networks and demonstrated better results (more than half the number of FLOPs and parameters, but similar classification error) on the CIFAR10/100 dataset.

In the future, we plan to apply multiple paths and RSDC to models other than DenseNet-based networks. Moreover, we need to train on datasets with larger input images, such as ImageNet [45]. It is expected that this will allow us to achieve meaningful results applying the large kernel size of RSDC.

Author Contributions: Conceptualization, S.P.; methodology, S.P.; software, S.P.; validation, S.P.; formal analysis, S.P.; investigation, S.P.; resources, S.P.; data curation, S.P.; writing—original draft preparation, S.P.; writing—review and editing, S.P. and D.E.C.; visualization, S.P.; supervision, D.E.C.; project administration, S.P.; funding acquisition, D.E.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by the NRF grant funded by the Korean government (MSIT) (2021R1A2C2010585), by IITP (2021-0-00590, Decentralized High-Performance Consensus for Large-Scale Blockchains), and by the BK21 FOUR program.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: CIFAR-10/100 dataset (accessed on 25 November 2021): https://www.cs.toronto.edu/~kriz/cifar.html, ILSVRC dataset (accessed on 25 November 2021): https://www.image-net.org/challenges/LSVRC/.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolutional neural network
DCNN	Deep convolutional neural network
GPU	Graphics processing units
SOTA	State-of-the-art
MLDN	Multipath lightweight deep network
RSDC	Randomly selected dilated convolution
CDN	CondenseNet
CDNV2	CondenseNetV2
FLOPs	Floating point operations
Params	the number of parameters
FC	Fully connected layers

References

1. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
2. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99. [CrossRef]
3. Girshick, R. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1440–1448.
4. Dai, J.; Li, Y.; He, K.; Sun, J. R-fcn: Object detection via region-based fully convolutional networks. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 379–387.
5. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
6. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
7. Shen, Z.; Liu, Z.; Li, J.; Jiang, Y.G.; Chen, Y.; Xue, X. Dsod: Learning deeply supervised object detectors from scratch. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1919–1927.
8. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
9. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
10. Zeiler, M.D.; Fergus, R. Visualizing and understanding convolutional networks. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 818–833.
11. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
12. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
13. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
14. Huang, G.; Liu, S.; Van der Maaten, L.; Weinberger, K.Q. Condensenet: An efficient densenet using learned group convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 2752–2761.
15. Yang, L.; Jiang, H.; Cai, R.; Wang, Y.; Song, S.; Huang, G.; Tian, Q. CondenseNet V2: Sparse Feature Reactivation for Deep Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 19–25 June 2021; pp. 3569–3578.
16. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3431–3440.
17. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, 5–9 October 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 234–241.
18. Badrinarayanan, V.; Kendall, A.; Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 2481–2495. [CrossRef] [PubMed]
19. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 834–848. [CrossRef] [PubMed]
20. Zhuge, M.; Fan, D.P.; Liu, N.; Zhang, D.; Xu, D.; Shao, L. Salient object detection via integrity learning. *arXiv* **2021**, arXiv:2101.07663.
21. Lin, M.; Chen, Q.; Yan, S. Network in network. *arXiv* **2013**, arXiv:1312.4400.
22. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
23. Han, D.; Kim, J.; Kim, J. Deep pyramidal residual networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 5927–5935.
24. Kim, Y.; Choi, H.; Lee, J.; Kim, J.S.; Jei, H.; Roh, H. Towards an optimized distributed deep learning framework for a heterogeneous multi-GPU cluster. *Clust. Comput.* **2020**, *23*, 2287–2300. [CrossRef]
25. Sergeev, A.; Del Balso, M.H. fast and easy distributed deep learning in TensorFlow. *arXiv* **2018**, arXiv:1802.05799.
26. Kim, S.; Yu, G.I.; Park, H.; Cho, S.; Jeong, E.; Ha, H.; Lee, S.; Jeong, J.S.; Chun, B.G. Parallax: Sparsity-aware data parallel training of deep neural networks. In Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25–28 March 2019; pp. 1–15.
27. Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv* **2014**, arXiv:1404.5997.
28. Jia, Z.; Zaharia, M.; Aiken, A. Beyond data and model parallelism for deep neural networks. *arXiv* **2018**, arXiv:1807.05358.

29. Park, J.H.; Yun, G.; Chang, M.Y.; Nguyen, N.T.; Lee, S.; Choi, J.; Noh, S.H.; Choi, Y.r. Hetpipe: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism. In Proceedings of the 2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20), Boston, MA, USA, 15–17 July 2020; pp. 307–321.
30. Koliousis, A.; Watcharapichat, P.; Weidlich, M.; Mai, L.; Costa, P.; Pietzuch, P. CROSSBOW: Scaling deep learning with small batch sizes on multi-gpu servers. *arXiv* **2019**, arXiv:1901.02244.
31. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both weights and connections for efficient neural networks. *arXiv* **2015**, arXiv:1506.02626.
32. Chen, Y.; Li, J.; Xiao, H.; Jin, X.; Yan, S.; Feng, J. Dual path networks. *arXiv* **2017**, arXiv:1707.01629.
33. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
34. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
35. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
36. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 6848–6856.
37. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
38. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 116–131.
39. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 27–28 October 2019; pp. 1314–1324.
40. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 8697–8710.
41. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 448–456.
42. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the ICML, Haifa, Israel, 21–24 June 2010.
43. Yu, F.; Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv* **2015**, arXiv:1511.07122.
44. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Tech Report; 2009. Available online: https://www.cs.toronto.edu/kriz/cifar.html (accessed on 25 November 2021).
45. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition,Miami, Florida, USA, 20–25 June 2009; pp. 248–255.
46. Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; Weinberger, K.Q. Deep networks with stochastic depth. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 646–661.
47. Lee, C.Y.; Xie, S.; Gallagher, P.; Zhang, Z.; Tu, Z. Deeply-supervised nets. In Proceedings of the Artificial Intelligence and Statistics, PMLR, San Diego, CA, USA, 9–12 May 2015; pp. 562–570.
48. Romero, A.; Ballas, N.; Kahou, S.E.; Chassang, A.; Gatta, C.; Bengio, Y. Fitnets: Hints for thin deep nets. *arXiv* **2014**, arXiv:1412.6550.
49. Springenberg, J.T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv* **2014**, arXiv:1412.6806.
50. Srivastava, R.K.; Greff, K.; Schmidhuber, J. Training very deep networks. *arXiv* **2015**, arXiv:1507.06228.
51. Loshchilov, I.; Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv* **2016**, arXiv:1608.03983.
52. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
53. Caterini, A.L.; Chang, D.E. *Deep Neural Networks in a Mathematical Framework*; Springer: Berlin/Heidelberg, Germany, 2018.
54. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity mappings in deep residual networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 630–645.
55. Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv* **2016**, arXiv:1605.07146.
56. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1492–1500.
57. He, Y.; Zhang, X.; Sun, J. Channel pruning for accelerating very deep neural networks. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1389–1397.
58. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient convnets. *arXiv* **2016**, arXiv:1608.08710.
59. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2736–2744.

60. Yu, R.; Li, A.; Chen, C.F.; Lai, J.H.; Morariu, V.I.; Han, X.; Gao, M.; Lin, C.Y.; Davis, L.S. Nisp: Pruning networks using neuron importance score propagation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 9194–9203.
61. He, Y.; Liu, P.; Wang, Z.; Hu, Z.; Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–20 June 2019; pp. 4340–4349.
62. Howard, A.; Zhmoginov, A.; Chen, L.C.; Sandler, M.; Zhu, M. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv* **2018**, arXiv:1801.04381.