Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet.

1. Introduction

Convolutional neural networks (CNNs) have become the dominant machine learning approach for visual object recognition. Although they were originally introduced over 20 years ago [18], improvements in computer hardware and network structure have enabled the training of truly deep CNNs only recently. The original LeNet5 [19] consisted of 5 layers, VGG featured 19 [28], and only last year Highway Networks [33] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

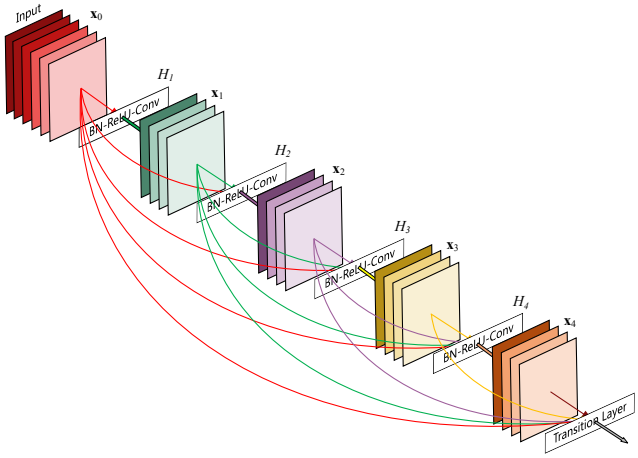

Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the network. Many recent publications address this or related problems. ResNets [11] and Highway Networks [33] bypass signal from one layer to the next via identity connections. Stochastic depth [13] shortens ResNets by randomly dropping layers during training to allow better information and gradient flow. FractalNets [17] repeatedly combine several parallel layer sequences with different number of convolutional blocks to obtain a large nominal depth, while maintaining many short paths in the network. Although these different approaches vary in network topology and training procedure, they all share a key characteristic: they create short paths from early layers to later layers.

*Authors contributed equally

1063-6919/17 $31.00 © 2017 IEEE
DOI 10.1109/CVPR.2017.243

In this paper, we propose an architecture that distills this insight into a simple connectivity pattern: to ensure maximum information flow between layers in the network, we connect *all layers* (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Figure 1 illustrates this layout schematically. Crucially, in contrast to ResNets, we never combine features through summation before they are passed into a layer; instead, we combine features by concatenating them. Hence, the ℓ^{th} layer has ℓ inputs, consisting of the feature-maps of all preceding convolutional blocks. Its own feature-maps are passed on to all $L-\ell$ subsequent layers. This introduces $\frac{L(L+1)}{2}$ connections in an L-layer network, instead of just L, as in traditional architectures. Because of its dense connectivity pattern, we refer to our approach as *Dense Convolutional Network (DenseNet)*.

A possibly counter-intuitive effect of this dense connectivity pattern is that it requires *fewer* parameters than traditional convolutional networks, as there is no need to relearn redundant feature-maps. Traditional feed-forward architectures can be viewed as algorithms with a state, which is passed on from layer to layer. Each layer reads the state from its preceding layer and writes to the subsequent layer. It changes the state but also passes on information that needs to be preserved. ResNets [11] make this information preservation explicit through additive identity transformations. Recent variations of ResNets [13] show that many layers contribute very little and can in fact be randomly dropped during training. This makes the state of ResNets similar to (unrolled) recurrent neural networks [21], but the number of parameters of ResNets is substantially larger because each layer has its own weights. Our proposed DenseNet architecture explicitly differentiates between information that is added to the network and information that is preserved. DenseNet layers are very narrow (*e.g.*, 12 filters per layer), adding only a small set of feature-maps to the "collective knowledge" of the network and keep the remaining feature-maps unchanged—and the final classifier makes a decision based on all feature-maps in the network.

Besides better parameter efficiency, one big advantage of DenseNets is their improved flow of information and gradients throughout the network, which makes them easy to train. Each layer has direct access to the gradients from the loss function and the original input signal, leading to an implicit deep supervision [20]. This helps training of deeper network architectures. Further, we also observe that dense connections have a regularizing effect, which reduces overfitting on tasks with smaller training set sizes.

We evaluate DenseNets on four highly competitive benchmark datasets (CIFAR-10, CIFAR-100, SVHN, and ImageNet). Our models tend to require much fewer parameters than existing algorithms with comparable accuracy. Further, we significantly outperform the current state-of-the-art results on most of the benchmark tasks.

2. Related Work

The exploration of network architectures has been a part of neural network research since their initial discovery. The recent resurgence in popularity of neural networks has also revived this research domain. The increasing number of layers in modern networks amplifies the differences between architectures and motivates the exploration of different connectivity patterns and the revisiting of old research ideas.

A cascade structure similar to our proposed dense network layout has already been studied in the neural networks literature in the 1980s [3]. Their pioneering work focuses on fully connected multi-layer perceptrons trained in a layer-by-layer fashion. More recently, fully connected cascade networks to be trained with batch gradient descent were proposed [39]. Although effective on small datasets, this approach only scales to networks with a few hundred parameters. In [9, 23, 30, 40], utilizing multi-level features in CNNs through skip-connnections has been found to be effective for various vision tasks. Parallel to our work, [1] derived a purely theoretical framework for networks with cross-layer connections similar to ours.

Highway Networks [33] were amongst the first architectures that provided a means to effectively train end-to-end networks with more than 100 layers. Using bypassing paths along with gating units, Highway Networks with hundreds of layers can be optimized without difficulty. The bypassing paths are presumed to be the key factor that eases the training of these very deep networks. This point is further supported by ResNets [11], in which pure identity mappings are used as bypassing paths. ResNets have achieved impressive, record-breaking performance on many challenging image recognition, localization, and detection tasks, such as ImageNet and COCO object detection [11]. Recently, *stochastic depth* was proposed as a way to successfully train a 1202-layer ResNet [13]. Stochastic depth improves the training of deep residual networks by dropping layers randomly during training. This shows that not all layers may be needed and highlights that there is a great amount of redundancy in deep (residual) networks. Our paper was partly inspired by that observation. ResNets with *pre-activation* also facilitate the training of state-of-the-art networks with $>$ 1000 layers [12].

An orthogonal approach to making networks deeper (*e.g.*, with the help of skip connections) is to increase the network *width*. The GoogLeNet [35, 36] uses an "Inception module" which concatenates feature-maps produced by filters of different sizes. In [37], a variant of ResNets with wide generalized residual blocks was proposed. In fact, simply increasing the number of filters in each layer of

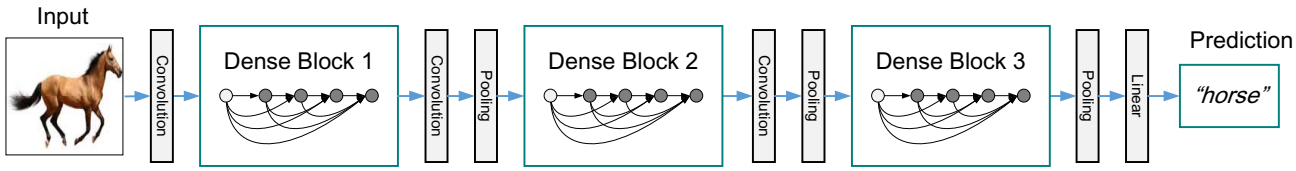

Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

ResNets can improve its performance provided the depth is sufficient [41]. FractalNets also achieve competitive results on several datasets using a wide network structure [17].

Instead of drawing representational power from extremely deep or wide architectures, DenseNets exploit the potential of the network through *feature reuse*, yielding condensed models that are easy to train and highly parameter-efficient. Concatenating feature-maps learned by *different layers* increases variation in the input of subsequent layers and improves efficiency. This constitutes a major difference between DenseNets and ResNets. Compared to Inception networks [35, 36], which also concatenate features from different layers, DenseNets are simpler and more efficient.

There are other notable network architecture innovations which have yielded competitive results. The Network in Network (NIN) [22] structure includes micro multi-layer perceptrons into the filters of convolutional layers to extract more complicated features. In Deeply Supervised Network (DSN) [20], internal layers are directly supervised by auxiliary classifiers, which can strengthen the gradients received by earlier layers. Ladder Networks [26, 25] introduce lateral connections into autoencoders, producing impressive accuracies on semi-supervised learning tasks. In [38], Deeply-Fused Nets (DFNs) were proposed to improve information flow by combining intermediate layers of different base networks. The augmentation of networks with pathways that minimize reconstruction losses was also shown to improve image classification models [42].

3. DenseNets

Consider a single image $\mathbf{x}_0$ that is passed through a convolutional network. The network comprises L layers, each of which implements a non-linear transformation $H_\ell(\cdot)$, where ℓ indexes the layer. $H_\ell(\cdot)$ can be a composite function of operations such as Batch Normalization (BN) [14], rectified linear units (ReLU) [6], Pooling [19], or Convolution (Conv). We denote the output of the ℓ^{th} layer as $\mathbf{x}_\ell$.

ResNets. Traditional convolutional feed-forward networks connect the output of the ℓ^{th} layer as input to the $(\ell+1)^{th}$ layer [16], which gives rise to the following layer transition: $\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1})$. ResNets [11] add a skip-connection that bypasses the non-linear transformations with an identity function:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}. \tag{1}$$

An advantage of ResNets is that the gradient can flow directly through the identity function from later layers to the earlier layers. However, the identity function and the output of H_ℓ are combined by summation, which may impede the information flow in the network.

Dense connectivity. To further improve the information flow between layers we propose a different connectivity pattern: we introduce direct connections from any layer to all subsequent layers. Figure 1 illustrates the layout of the resulting DenseNet schematically. Consequently, the ℓ^{th} layer receives the feature-maps of all preceding layers, $\mathbf{x}_0, \ldots, \mathbf{x}_{\ell-1}$, as input:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]), \tag{2}$$

where $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]$ refers to the concatenation of the feature-maps produced in layers $0, \ldots, \ell-1$. Because of its dense connectivity we refer to this network architecture as *Dense Convolutional Network (DenseNet)*. For ease of implementation, we concatenate the multiple inputs of $H_\ell(\cdot)$ in eq. (2) into a single tensor.

Composite function. Motivated by [12], we define $H_\ell(\cdot)$ as a composite function of three consecutive operations: batch normalization (BN) [14], followed by a rectified linear unit (ReLU) [6] and a 3×3 convolution (Conv).

Pooling layers. The concatenation operation used in Eq. (2) is not viable when the size of feature-maps changes. However, an essential part of convolutional networks is down-sampling layers that change the size of feature-maps. To facilitate down-sampling in our architecture we divide the network into multiple densely connected *dense blocks*; see Figure 2. We refer to layers between blocks as *transition layers*, which do convolution and pooling. The transition layers used in our experiments consist of a batch normalization layer and an 1×1 convolutional layer followed by a 2×2 average pooling layer.

Growth rate. If each function H_ℓ produces k feature-maps, it follows that the ℓ^{th} layer has $k_0 + k \times (\ell-1)$ input feature-maps, where k_0 is the number of channels in the input layer. An important difference between DenseNet and existing network architectures is that DenseNet can have very narrow layers, *e.g.*, $k = 12$. We refer to the hyper-parameter k as the *growth rate* of the network. We show in Section 4 that a relatively small growth rate is sufficient to

Layers	Output Size	DenseNet-121($k=32$)	DenseNet-169($k=32$)	DenseNet-201($k=32$)	DenseNet-161($k=48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

obtain state-of-the-art results on the datasets that we tested on. One explanation for this is that each layer has access to all the preceding feature-maps in its block and, therefore, to the network's "collective knowledge". One can view the feature-maps as the global state of the network. Each layer adds k feature-maps of its own to this state. The growth rate regulates how much new information each layer contributes to the global state. The global state, once written, can be accessed from everywhere within the network and, unlike in traditional network architectures, there is no need to replicate it from layer to layer.

Bottleneck layers. Although each layer only produces k output feature-maps, it typically has many more inputs. It has been noted in [36, 11] that a 1×1 convolution can be introduced as *bottleneck* layer before each 3×3 convolution to reduce the number of input feature-maps, and thus to improve computational efficiency. We find this design especially effective for DenseNet and we refer to our network with such a bottleneck layer, *i.e.*, to the BN-ReLU-Conv(1×1)-BN-ReLU-Conv(3×3) version of H_ℓ, as DenseNet-B. In our experiments, we let each 1×1 convolution produce $4k$ feature-maps.

Compression. To further improve model compactness, we can reduce the number of feature-maps at transition layers. If a dense block contains m feature-maps, we let the following transition layer generate $\lfloor \theta m \rfloor$ output feature-maps, where $0 < \theta \leq 1$ is referred to as the compression factor. When $\theta = 1$, the number of feature-maps across transition layers remains unchanged. We refer the DenseNet with $\theta < 1$ as DenseNet-C, and we set $\theta = 0.5$ in our experiment. When both the bottleneck and transition layers with $\theta < 1$ are used, we refer to our model as DenseNet-BC.

Implementation Details. On all datasets except ImageNet, the DenseNet used in our experiments has three dense blocks that each has an equal number of layers. Before entering the first dense block, a convolution with 16 (or twice the growth rate for DenseNet-BC) output channels is performed on the input images. For convolutional layers with kernel size 3×3, each side of the inputs is zero-padded by one pixel to keep the feature-map size fixed. We use 1×1 convolution followed by 2×2 average pooling as transition layers between two contiguous dense blocks. At the end of the last dense block, a global average pooling is performed and then a softmax classifier is attached. The feature-map sizes in the three dense blocks are $32\times$ 32, 16×16, and 8×8, respectively. We experiment with the basic DenseNet structure with configurations $\{L = 40, k = 12\}$, $\{L = 100, k = 12\}$ and $\{L = 100, k = 24\}$. For DenseNet-BC, the networks with configurations $\{L = 100, k = 12\}$, $\{L=250, k=24\}$ and $\{L=190, k=40\}$ are evaluated.

In our experiments on ImageNet, we use a DenseNet-BC structure with 4 dense blocks on 224×224 input images. The initial convolution layer comprises $2k$ convolutions of size 7×7 with stride 2; the number of feature-maps in all other layers also follow from setting k. The exact network configurations we used on ImageNet are shown in Table 1.

4. Experiments

We empirically demonstrate DenseNet's effectiveness on several benchmark datasets and compare with state-of-the-art architectures, especially with ResNet and its variants.

4.1. Datasets

CIFAR. The two CIFAR datasets [15] consist of colored natural images with 32×32 pixels. CIFAR-10 (C10) con-

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [31]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [33]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [41]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

sists of images drawn from 10 and CIFAR-100 (C100) from 100 classes. The training and test sets contain 50,000 and 10,000 images respectively, and we hold out 5,000 training images as a validation set. We adopt a standard data augmentation scheme (mirroring/shifting) that is widely used for these two datasets [11, 13, 17, 22, 27, 20, 31, 33]. We denote this data augmentation scheme by a "+" mark at the end of the dataset name (*e.g.*, C10+). For preprocessing, we normalize the data using the channel means and standard deviations. For the final run we use all 50,000 training images and report the final test error at the end of training.

SVHN. The Street View House Numbers (SVHN) dataset [24] contains 32×32 colored digit images. There are 73,257 images in the training set, 26,032 images in the test set, and 531,131 images for additional training. Following common practice [7, 13, 20, 22, 29] we use all the training data without any data augmentation, and a validation set with 6,000 images is split from the training set. We select the model with the lowest validation error during training and report the test error. We follow [41] and divide the pixel values by 255 so they are in the $[0, 1]$ range.

ImageNet. The ILSVRC 2012 classification dataset [2] consists 1.2 million images for training, and 50,000 for validation, from $1,000$ classes. We adopt the same data augmentation scheme for training images as in [8, 11, 12], and apply a single-crop or 10-crop with size 224×224 at test time. Following [11, 12, 13], we report classification errors on the validation set.

4.2. Training

All the networks are trained using stochastic gradient descent (SGD). On CIFAR and SVHN we train using batch size 64 for 300 and 40 epochs, respectively. The initial learning rate is set to 0.1, and is divided by 10 at 50% and 75% of the total number of training epochs. On ImageNet, we train models for 90 epochs with a batch size of 256. The learning rate is set to 0.1 initially, and is lowered by 10 times at epoch 30 and 60. Due to GPU memory constraints, our largest model (DenseNet-161) is trained with a mini-batch size 128. To compensate for the smaller batch size, we train this model for 100 epochs, and divide the learning rate by 10 at epoch 90.

Following [8], we use a weight decay of 10^{-4} and a Nesterov momentum [34] of 0.9 without dampening. We adopt the weight initialization introduced by [10]. For the three datasets without data augmentation, *i.e.*, C10, C100 and SVHN, we add a dropout layer [32] after each convolutional layer (except the first one) and set the dropout rate to 0.2. The test errors were only evaluated once for each task and model setting.

Model	top-1	top-5
DenseNet-121 ($k=32$)	25.02 (23.61)	7.71 (6.66)
DenseNet-169 ($k=32$)	23.80 (22.08)	6.85 (5.92)
DenseNet-201 ($k=32$)	22.58 (21.46)	6.34 (5.54)
DenseNet-161 ($k=48$)	22.33 (20.85)	6.15 (5.30)

Table 3: The top-1 and top-5 error rates on the ImageNet validation set, with single-crop (10-crop) testing.

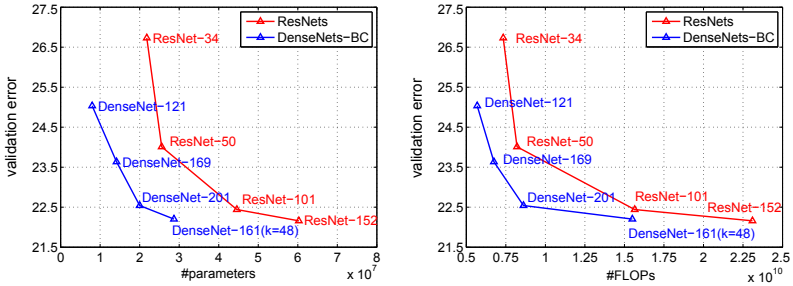

Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

4.3. Classification Results on CIFAR and SVHN

We train DenseNets with different depths, L, and growth rates, k. The main results on CIFAR and SVHN are shown in Table 2. To highlight general trends, we mark all results that outperform the existing state-of-the-art in **boldface** and the overall best result in **blue**.

Accuracy. Possibly the most noticeable trend may originate from the bottom row of Table 2, which shows that DenseNet-BC with $L = 190$ and $k = 40$ outperforms the existing state-of-the-art consistently on *all* the CIFAR datasets. Its error rates of 3.46% on C10+ and 17.18% on C100+ are significantly lower than the error rates achieved by wide ResNet architecture [41]. Our best results on C10 and C100 (without data augmentation) are even more encouraging: both are close to 30% lower than FractalNet with drop-path regularization [17]. On SVHN, with dropout, the DenseNet with $L = 100$ and $k = 24$ also surpasses the current best result achieved by wide ResNet. However, the 250-layer DenseNet-BC doesn't further improve the performance over its shorter counterpart. This may be explained by that SVHN is a relatively easy task, and extremely deep models may overfit to the training set.

Capacity. Without compression or bottleneck layers, there is a general trend that DenseNets perform better as L and k increase. We attribute this primarily to the corresponding growth in model capacity. This is best demonstrated by the column of C10+ and C100+. On C10+, the error drops from 5.24% to 4.10% and finally to 3.74% as the number of parameters increases from 1.0M, over 7.0M to 27.2M. On C100+, we observe a similar trend. This suggests that DenseNets can utilize the increased representational power of bigger and deeper models. It also indicates that they do not suffer from overfitting or the optimization difficulties of residual networks [11].

Parameter Efficiency. The results in Table 2 indicate that DenseNets utilize parameters more efficiently than alternative architectures (in particular, ResNets). The DenseNet-BC with bottleneck structure and dimension reduction at transition layers is particularly parameter-efficient. For example, our 250-layer model only has 15.3M parameters, but it consistently outperforms other models such as FractalNet and Wide ResNets that have more than 30M parameters. We also highlight that DenseNet-BC with $L = 100$ and $k = 12$ achieves comparable performance (*e.g.*, 4.51% vs 4.62% error on C10+, 22.27% vs 22.71% error on C100+) as the 1001-layer pre-activation ResNet using 90% fewer parameters. Figure 4 (right panel) shows the training loss and test errors of these two networks on C10+. The 1001-layer deep ResNet converges to a lower training loss value but a similar test error. We analyze this effect in more detail below.

Overfitting. One positive side-effect of the more efficient use of parameters is a tendency of DenseNets to be less prone to overfitting. We observe that on the datasets without data augmentation, the improvements of DenseNet architectures over prior work are particularly pronounced. On C10, the improvement denotes a 29% relative reduction in error from 7.33% to 5.19%. On C100, the reduction is about 30% from 28.20% to 19.64%. In our experiments, we observed potential overfitting in a single setting: on C10, a $4\times$ growth of parameters produced by increasing $k = 12$ to $k = 24$ lead to a modest increase in error from 5.77% to 5.83%. The DenseNet-BC bottleneck and compression layers appear to be an effective way to counter this trend.

4.4. Classification Results on ImageNet

We evaluate DenseNet-BC with different depths and growth rates on the ImageNet classification task, and compare it with state-of-the-art ResNet architectures. To ensure a fair comparison between the two architectures, we eliminate all other factors such as differences in data preprocessing and optimization settings by adopting the publicly available Torch implementation for ResNet by [8][1]. We simply replace the ResNet model with the DenseNet-BC network, and keep all the experiment settings *exactly* the same as those used for ResNet. The only exception is our largest DenseNet model is trained with a mini-batch size of 128

[1] https://github.com/facebook/fb.resnet.torch

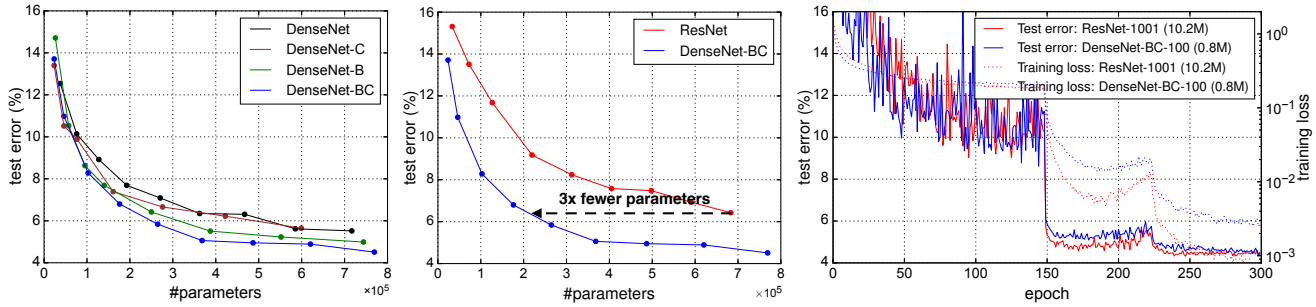

Figure 4: *Left:* Comparison of the parameter efficiency on C10+ between DenseNet variations. *Middle:* Comparison of the parameter efficiency between DenseNet-BC and (pre-activation) ResNets. DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy. *Right:* Training and testing curves of the 1001-layer pre-activation ResNet [12] with more than 10M parameters and a 100-layer DenseNet with only 0.8M parameters.

because of GPU memory limitations; we train this model for 100 epochs with a third learning rate drop after epoch 90 to compensate for the smaller batch size.

We report the single-crop and 10-crop validation errors of DenseNets on ImageNet in Table 3. Figure 3 shows the single-crop top-1 validation errors of DenseNets and ResNets as a function of the number of parameters (left) and FLOPs (right). The results presented in the figure reveal that DenseNets perform on par with the state-of-the-art ResNets, whilst requiring significantly fewer parameters and computation to achieve comparable performance. For example, a DenseNet-201 with 20M parameters model yields similar validation error as a 101-layer ResNet with more than 40M parameters. Similar trends can be observed from the right panel, which plots the validation error as a function of the number of FLOPs: a DenseNet that requires as much computation as a ResNet-50 performs on par with a ResNet-101, which requires twice as much computation.

It is worth noting that our experimental setup implies that we use hyperparameter settings that are optimized for ResNets but not for DenseNets. It is conceivable that more extensive hyper-parameter searches may further improve the performance of DenseNet on ImageNet.[2]

5. Discussion

Superficially, DenseNets are quite similar to ResNets: Eq. (2) differs from Eq. (1) only in that the inputs to $H_\ell(\cdot)$ are concatenated instead of summed. However, the implications of this seemingly small modification lead to substantially different behaviors of the two network architectures.

Model compactness. As a direct consequence of the input concatenation, the feature-maps learned by any of the DenseNet layers can be accessed by all subsequent layers. This encourages feature reuse throughout the network, and leads to more compact models.

[2]Our DenseNet implementation contains some memory inefficiencies which temporarily precludes experiments with over 30M parameters.

The left two plots in Figure 4 show the result of an experiment that aims to compare the parameter efficiency of all variants of DenseNets (left) and also a comparable ResNet architecture (middle). We train multiple small networks with varying depths on C10+ and plot their test accuracies as a function of network parameters. In comparison with other popular network architectures, such as AlexNet [16] or VGG-net [28], ResNets with pre-activation use fewer parameters while typically achieving better results [12]. Hence, we compare DenseNet ($k = 12$) against this architecture. The training setting for DenseNet is kept the same as in the previous section.

The graph shows that DenseNet-BC is consistently the most parameter efficient variant of DenseNet. Further, to achieve the same level of accuracy, DenseNet-BC only requires around 1/3 of the parameters of ResNets (middle plot). This result is in line with the results on ImageNet we presented in Figure 3. The right plot in Figure 4 shows that a DenseNet-BC with only 0.8M trainable parameters is able to achieve comparable accuracy as the 1001-layer (pre-activation) ResNet [12] with 10.2M parameters.

Implicit Deep Supervision. One explanation for the improved accuracy of dense convolutional networks may be that individual layers receive additional supervision from the loss function through the shorter connections. One can interpret DenseNets to perform a kind of "deep supervision". The benefits of deep supervision have previously been shown in deeply-supervised nets (DSN; [20]), which have classifiers attached to every hidden layer, enforcing the intermediate layers to learn discriminative features.

DenseNets perform a similar deep supervision in an implicit fashion: a single classifier on top of the network provides direct supervision to all layers through at most two or three transition layers. However, the loss function and gradient of DenseNets are substantially less complicated, as the same loss function is shared between all layers.

Stochastic vs. deterministic connection. There is an interesting connection between dense convolutional net-

works and stochastic depth regularization of residual networks [13]. In stochastic depth, layers in residual networks are randomly dropped, which creates direct connections between the surrounding layers. As the pooling layers are never dropped, the network results in a similar connectivity pattern as DenseNet: there is a small probability for any two layers, between the same pooling layers, to be directly connected—if all intermediate layers are randomly dropped. Although the methods are ultimately quite different, the DenseNet interpretation of stochastic depth may provide insights into the success of this regularizer.

Feature Reuse. By design, DenseNets allow layers access to feature-maps from all of its preceding layers (although sometimes through transition layers). We conduct an experiment to investigate if a trained network takes advantage of this opportunity. We first train a DenseNet on C10+ with $L = 40$ and $k = 12$. For each convolutional layer ℓ within a block, we compute the average (absolute) weight assigned to connections with layer s. Figure 5 shows a heat-map for all three dense blocks. The average absolute weight serves as a surrogate for the dependency of a convolutional layer on its preceding layers. A red dot in position (ℓ, s) indicates that the layer ℓ makes, on average, strong use of feature-maps produced s-layers before. Several observations can be made from the plot:

1. All layers spread their weights over many inputs within the same block. This indicates that features extracted by very early layers are, indeed, directly used by deep layers throughout the same dense block.
2. The weights of the transition layers also spread their weight across all layers within the preceding dense block, indicating information flow from the first to the last layers of the DenseNet through few indirections.
3. The layers within the second and third dense block consistently assign the least weight to the outputs of the transition layer (the top row of the triangles), indicating that the transition layer outputs many redundant features (with low weight on average). This is in keeping with the strong results of DenseNet-BC where exactly these outputs are compressed.
4. Although the final classification layer, shown on the very right, also uses weights across the entire dense block, there seems to be a concentration towards final feature-maps, suggesting that there may be some more high-level features produced late in the network.

6. Conclusion

We proposed a new convolutional network architecture, which we refer to as Dense Convolutional Network (DenseNet). It introduces direct connections between any

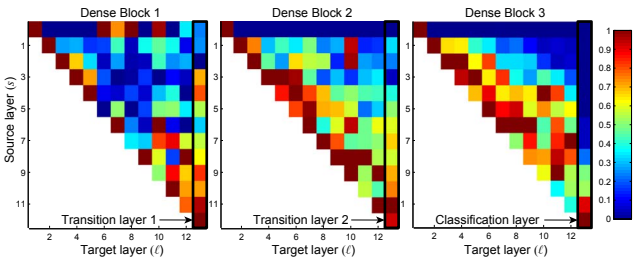

Figure 5: The average absolute filter weights of convolutional layers in a trained DenseNet. The color of pixel (s, ℓ) encodes the average $L1$ norm (normalized by number of input feature-maps) of the weights connecting convolutional layer s to ℓ within a dense block. Three columns highlighted by black rectangles correspond to two transition layers and the classification layer. The first row encodes weights connected to the input layer of the dense block.

two layers with the same feature-map size. We showed that DenseNets scale naturally to hundreds of layers, while exhibiting no optimization difficulties. In our experiments, DenseNets tend to yield consistent improvement in accuracy with growing number of parameters, without any signs of performance degradation or overfitting. Under multiple settings, it achieved state-of-the-art results across several highly competitive datasets. Moreover, DenseNets require substantially fewer parameters and less computation to achieve state-of-the-art performances. Because we adopted hyperparameter settings optimized for residual networks in our study, we believe that further gains in accuracy of DenseNets may be obtained by more detailed tuning of hyperparameters and learning rate schedules.

Whilst following a simple connectivity rule, DenseNets naturally integrate the properties of identity mappings, deep supervision, and diversified depth. They allow feature reuse throughout the networks and can consequently learn more compact and, according to our experiments, more accurate models. Because of their compact internal representations and reduced feature redundancy, DenseNets may be good feature extractors for various computer vision tasks that build on convolutional features, *e.g.*, [4, 5]. We plan to study such feature transfer with DenseNets in future work.

Acknowledgements. The authors are supported in part by the III-1618134, III-1526012, IIS-1149882 grants from the National Science Foundation, and the Bill and Melinda Gates foundation. Gao Huang is supported by the International Postdoctoral Exchange Fellowship Program of China Postdoctoral Council (No.20150015). Zhuang Liu is supported by the National Basic Research Program of China Grants 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61361136003. We also thank Daniel Sedra, Geoff Pleiss and Yu Sun for many insightful discussions.

References

[1] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. Adanet: Adaptive structural learning of artificial neural networks. *arXiv preprint arXiv:1607.01097*, 2016. 2

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[3] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *NIPS*, 1989. 2

[4] J. R. Gardner, M. J. Kusner, Y. Li, P. Upchurch, K. Q. Weinberger, and J. E. Hopcroft. Deep manifold traversal: Changing labels with convolutional features. *arXiv preprint arXiv:1511.06421*, 2015. 8

[5] L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. *Nature Communications*, 2015. 8

[6] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011. 3

[7] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013. 5

[8] S. Gross and M. Wilber. Training and investigating residual nets, 2016. 5, 6

[9] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015. 2

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

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

[12] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 2, 3, 5, 7

[13] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 1, 2, 5, 8

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Tech Report*, 2009. 4

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 3, 7

[17] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016. 1, 3, 5, 6

[18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 3

[20] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015. 2, 3, 5, 7

[21] Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016. 2

[22] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014. 3, 5

[23] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 2

[24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning, 2011. In *NIPS Workshop*, 2011. 5

[25] M. Pezeshki, L. Fan, P. Brakel, A. Courville, and Y. Bengio. Deconstructing the ladder network architecture. In *ICML*, 2016. 3

[26] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *NIPS*, 2015. 3

[27] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015. 5

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*. 1, 7

[29] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, pages 3288–3291. IEEE, 2012. 5

[30] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, 2013. 2

[31] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 5

[32] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014. 5

[33] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, 2015. 1, 2, 5

[34] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013. 5

[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2, 3

[36] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2, 3, 4

[37] S. Targ, D. Almeida, and K. Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016. 2

[38] J. Wang, Z. Wei, T. Zhang, and W. Zeng. Deeply-fused nets. *arXiv preprint arXiv:1605.07716*, 2016. 3

[39] B. M. Wilamowski and H. Yu. Neural network learning without backpropagation. *IEEE Transactions on Neural Networks*, 21(11):1793–1803, 2010. 2

[40] S. Yang and D. Ramanan. Multi-scale recognition with dag-cnns. In *ICCV*, 2015. 2

[41] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 3, 5, 6

[42] Y. Zhang, K. Lee, and H. Lee. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *ICML*, 2016. 3