Densely Connected Convolutional Networks

a summary by Elmar Stellnberger

Department of Information Technology, University of Klagenfurt, Austria
estellnb@elstel.org

1. Abstract & Introduction

Densely connected convolutional networks (CNNs) implement a special architecture for convolutional neuronal networks in which every layer in a so called dense block connects to all subsequent layers within the same block. Each layer consequently draws its input from all preceding layers and not just from the directly preceding layer as in traditionally designed CNNs. DenseNets have many compelling advantages: They encourage feature propagation, feature reuse, alleviate the vanishing gradient problem, they make use of existing parameters more efficiently than comparable architectures and they are less prone to overfitting even without data augmentation. The following article first shortly discusses approaches with some similarity to the DenseNet architecture and it then explains the DenseNet architecture in greater detail. We summarize with the evaluation of the experiments of the reference article along with a subsequent discussion of the results [1].

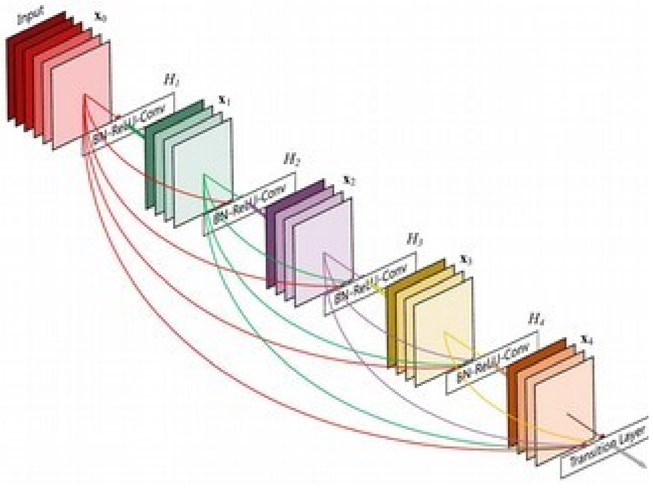


Figure 1: A 5-layer dense block with a growth rate of k=4. Each layer takes all preceding activation maps as input.

2. Deep Networks, Related Work

Improvements in computer hardware and network structure have made more and more deeper networks recently possible [1]. Truly deep networks seemed to be the key for better performance. However with increasing network depth the accuracy got saturated and even degraded rapidly thereafter. This was not caused by overfitting because overly deep networks expose a higher training error than their relatively shallower counterparts [2].

The problem exposed by overly deep networks is mainly the problem of vanishing/exploding gradients: More and more weights approach zero through learning. Input data gets “washed out” and information is lost when the network is used to classify images.

When learning takes place the final classification error is assessed by comparing the actual output of the network by the desired result. First of all the current weights are adjusted basing on the activation of the connected neurons in the previous layer (*Hebbian Theory, neurons that fire together wire together,* $\partial y/\partial w_i = a_i$*:* w_i *~ weight for input from neuron i ,* a_i *~ activation of neuron i in the previous layer, y convolution* *function* = $\sum w_i \cdot a_i$). Only weights for neurons active with the current example image need to adjusted; the others don´t matter. If our neuron has n inputs or dendrites then we need to calculate the respective partial differential ($\partial C/\partial w_i$, *C ~Cost*) to obtain an n-dimensional gradient where each weight w_i makes up one dimension. The result of the entire integral $\partial y/\partial w_i \cdot \partial \sigma/\partial y \cdot \partial C/\partial \sigma = \partial C/\partial w_i$ is minimized to obtain the minimal training error where σ is the activation function and C the cost function. Then it comes to back-propagation, i.e. it is calculated how the neurons in the previous layer would need to have been activated in order to achieve the desired result by use of the current weight configuration ($\partial y/\partial a_i = w_i$). The problem is now that this kind of back-propagation only works well for a limited amount of layers.

Previous attempts to get the vanishing gradient problem under control were mainly focused on intermediate normalization layers as well as by a normalized initialization of the network [2]. However recent results have shown that this problem can be tackled most efficiently by the right architectural choice.

A possibility that almost suggests itself is to introduce classifiers at every hidden or intermediate layer than just to have one at the end. The result of the additional classifiers is not used for user output but merely for learning to shorten the back-propagation paths. A discriminative classifier at a certain level is therefore a proxy for the goodness of the quality of the respective intermediate layer [3]. With DenseNets there is no need for such additional classifiers as each layer directly contributes to the output.

The most compelling idea to alleviate the vanishing gradient problem is however to introduce short paths or residual mappings in a deep network. One of the most successful architectural improvements which realise this idea was introduced by the so called ResNets. A limited number of layers are bypassed by an identity connection which is then simply added to the result of the last layer [2]. ResNets are used as a benchmark for the success of DenseNets throughout the paper. The paper can show that the DenseNet architecture is even more powerful than the idea of ResNets. There have been many similar ideas: Highway Networks were one of the first networks that could increase depth by identity connections along with gating units. FractalNets combine several parallel layers [1]. GoogleNets are just another approach to increase the depth and width of CNNs. GoogleNets combine a 5x5, a 3x3, a 1x1 convolution and a 3x3 pooling in an inception module in parallel. The idea behind this is that visual information should be processed at various scales [4]. Another interesting approach is stochastic depth where layers are dropped randomly [1]. This is somehow similar to randomly deactivating neurons within a given layer because the network has to learn and maintain alternative paths to achieve the same result. What all of these architectures have in common are the existence of shorter paths from the beginning to the end which do not necessarily pass through each layer.

3. The DenseNet Architecture

The traditional archtitecture for CNNs was to apply a function $x_l = H_l(x_{l-1})$ at each layer l while the network as a whole has L layers. $H_l(x)$ is usually a composite function comprising of individual functions such as batch normalization (BN), rectified linear units (ReLU), pooling and convolution. Batch normalization puts a value in relation to the values of neighbouring activation maps. What counts is not so much the absolute value but the difference towards the surrounding. Rectified linear units is an activation function which is more performant than f.i. the sigmoid activation function. It is furthermore very simple to implement max(0,x) and models biological neurons better than the sigmoid or tanh function [5]. The activation function implements a means of non-linearity in addition to convolution and thus enables the network to make decisions. Pooling can take the maximum or average over a neighbouring area for each pixel and is usually applied on neighbouring areas of one activation map but can also be applied across planes to make f.i. either or decisions possible (max-pooling). Pooling over spatial regions results in a certain degree of translation invariance and takes only the most relevant result in case of max-pooling [6]. When used with a stride or step size greater than one it reduces the amount of data. The convolution operation in computer graphics is well known apart from convolutional neuronal networks in order to detect edges or to blur images where the Gaussian blur is most well known.

The ResNets used for comparison throughout the paper implement a residual connection of the form $x_l = H_l(x_{l-1}) + x_{l-1}$. The DenseNet approach has proven that a simple concatenation $x_l = H_l([x_0,x_1,..,\ldots,x_{l-2},x_{l-1}])$ can even be more efficient. The paper states that concatenation is only feasible with activation maps of the same size. Concatenation could thus be envisioned as stitching two images together to a larger image which has then twice the width of the input images. However it can be seen from the article that concatenated activation maps which stem from different layers are multiplied with different kernel weights. Consequently the structure of the l-tuple $[x_0,x_1,..,\ldots,x_{l-2},x_{l-1}]$ needs to be maintained as is. Besides the depth L of the network the growth rate k is one of the most important parameters. In each step k different kernels are applied to the input activation maps in order to produce k new output activation maps. That results in $k_0 + k\cdot(l-1)$ input activation maps for layer l. As each layer has access to all preceding activation maps a relatively small growth rate k suffices for DenseNets.

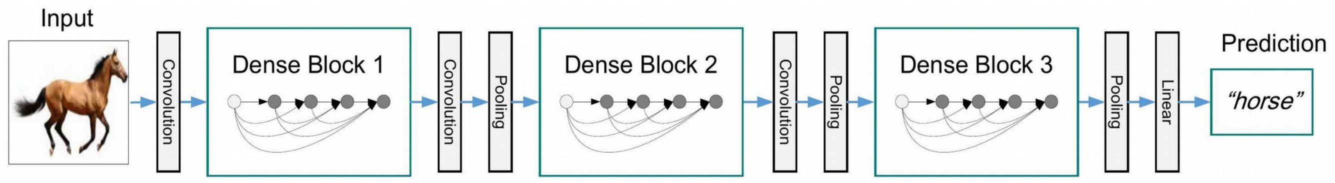
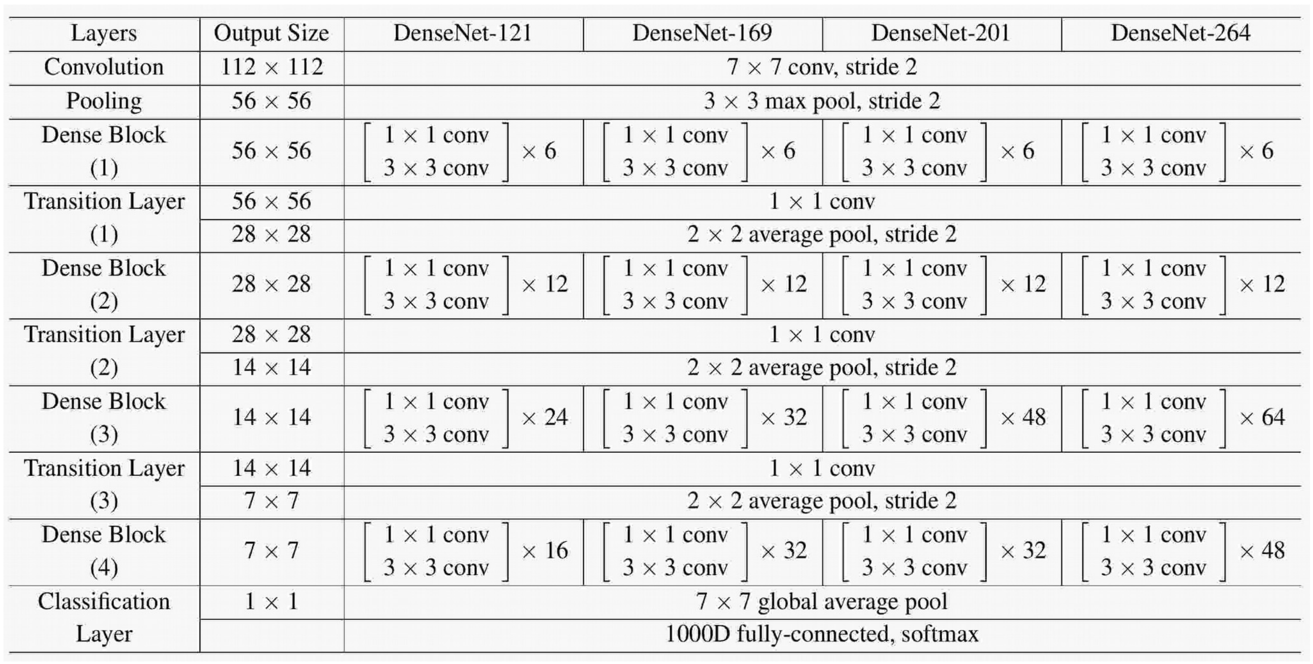


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layer and are responsible to reduce the sizes of the activation maps.

However one of the most important features of convolutional neuronal networks is the data reduction. You get a large image as input, extract only relevant features from that image and finally condense all values to a simple prediction value of the object class as humans know it which is done by a fully connected layer like a softmax layer at the end in contrast to the preceding convolution layers. As already stated no



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
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	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: The layers of a DenseNet with k=32. Each convolution layer in deed refers to three layers namely BN-ReLU-Conv. This architecture has been applied on the ImageNet data set. There were three dense blocks for all other data sets.

data reduction can take place with the fully interconnected approach where each layer is connected to every other layer in a feed forward fashion (figure 1) because for concatenation the input activation maps need to have the same size. The solution to this problem is that the network is divided into three dense blocks each dense block being fully interconnected in its interior, $x_l = H_l([x_0,x_1,.., …,x_{l-2},x_{l-1}])$. The dense blocks themselves are connected on-to-the-other in a linear fashion. Between the dense blocks an additional convolution and most important pooling for data reduction can take place (figure 2). The individual $H_l(\vec{x})$ inside the dense blocks comprise of batch normalization (BN), followed by a rectified linear unit (ReLU) and a 3x3 convolution. Table 1 shows how each step is combined as part of the full network and what output sizes are generated at each step in order to achieve the necessary data reduction.

As you can see from table 1 an additional 1x1 convolution can be introduced before each 3x3 convolution. The 1x1 convolution itself would not make sense if it would not be combined with a ReLU activation function and a preceding batch normalization. The activation function reduces the the number of input activation maps and thus improves computational efficiency. This variant is called DenseNet-B.

Besides the 1x1 convolution compression is an additional step that can be performed at the transition layers between two dense blocks. The DenseNet-C variant has a compression factor of θ = 0.5 which means that only the half of the activation maps are passed on between dense blocks.

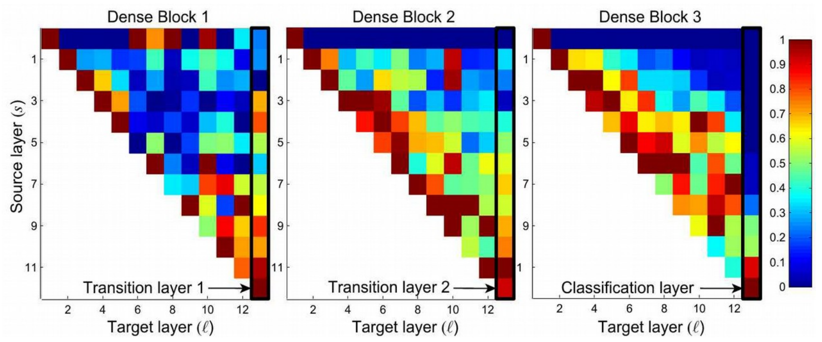
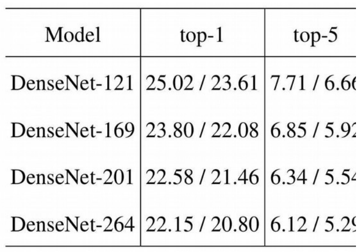


Figure 3: Average absolute filter weights of convolution layers in a trained DenseNet. Blue refers to a low weight while red refers to a high absolute weight value.

Figure 3 shows us how each layer in each of the three dense blocks makes use of the inputs of the preceding layers within each dense block (see the respective column). The first row in each of the three triangles states how much the inputs from the transition layer are used throughout each subsequent layer. As these rows are mostly blue indicating low usage they are a good candidate for compression. We can also see that not only the values from the diagonal are redish but also some preceding rows above the diagonal which means that inputs from more than the immediately preceding layer are used in deed. Especially the transition layer (right column) makes use of many preceding layers.

4. Experiments

The performance of DenseNets has been evaluated against four different datasets: CIFAR (C10, C100), SVHN and ImageNet. CIFAR-10 consists of images of 10 object classes while CIFAR-100 differs between 100 object classes. There have been 50,000, 10,000 and 5,000 images in the training, test and validation set. C10+ and C100+ denote the same image sets with data augmentation such as mirroring and shifting. The Street View House Numbers (SVHN) dataset has a training, test and validation set of 73,000, 26,000 and 6,000 images. The SVHN is a relatively easy task so that extremely deep models may overfit it. The ImageNet contains 1.2 million



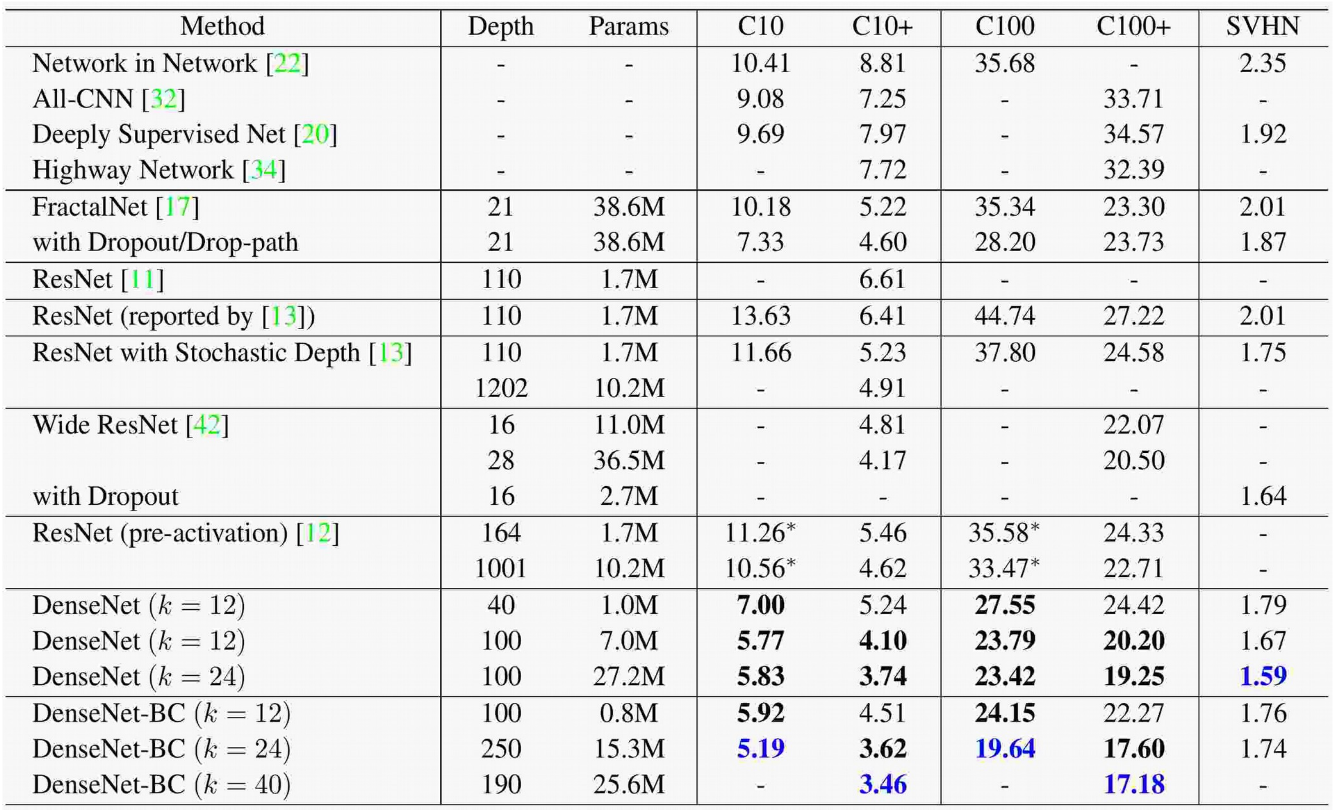
Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.6
DenseNet-169	23.80 / 22.08	6.85 / 5.9
DenseNet-201	22.58 / 21.46	6.34 / 5.5
DenseNet-264	22.15 / 20.80	6.12 / 5.2

Table 2: Top-1 and top-5 error rates on the ImageNet validation set

images for training and 50,000 for validation. The original article does not contain a comparison of ImageNet results for DenseNet and other architectures but it contains a table with top-1 and top-5 error rates (table 2). We have found some other results for the ImageNet data set in the article about deeply-supervised nets (DSN) [3]. However a direct comparison with ResNets would have been more interesting.

All the networks have been trained using stochastic gradient descent (SGD). CIFAR (C10, C100) and SVHN were trained using 40 epochs and ImageNet using 90 epochs. For CIFAR (C10, C100) and SVHN the learning rate was initially set to 0.1 and divided by 10 at 50% and 75% of the total training epochs. On ImageNet the initial learning rate was also set to 0.1 and lowered by ten times at epoch 30 and 60. They used a weight decay of 10^{-4} and a Nesterov momentum of 0.9 without dampening. For the three data sets without data augmentation (C10, C100, SVHN) a dropout layer with dropout rate 0.2 has been added after each convolution layer in order to prevent overfitting.

Table 3 shows us the results of the experiments. For both the CIFAR and SVHN data sets the best results (blue) could be achieved by DenseNets. For the CIFAR data set the DenseNet-BC variant has performed best while plain DenseNets were better on the SVHN data set. We can see that DenseNets perform better as L and k increase.



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	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 [42]	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 3: Error rates in % for the CIFAR and SVHN datasets. "+" indicates data augmentation and "" experiments conducted directly by the authors of the reference article [1]*

This means that DenseNets can utilize the increased representational power of bigger and deeper models. DenseNets and especially the DenseNet-BC variant can utilize parameters more efficiently than alternative architectures. DenseNets are less prone to overfitting as they use parameters more efficiently. It can be seen that improvements of DenseNets on data sets without data augmentation are particularly pronounced.

Figure 4 shows that DenseNets can perform on par with state of the art ResNets but require significantly fewer parameters and computational power (~½ of flops). Figure 5 shows the parameter efficiency for different types of DenseNets on the C10+ data set. With this dataset they require three times fewer parameters than ResNets.

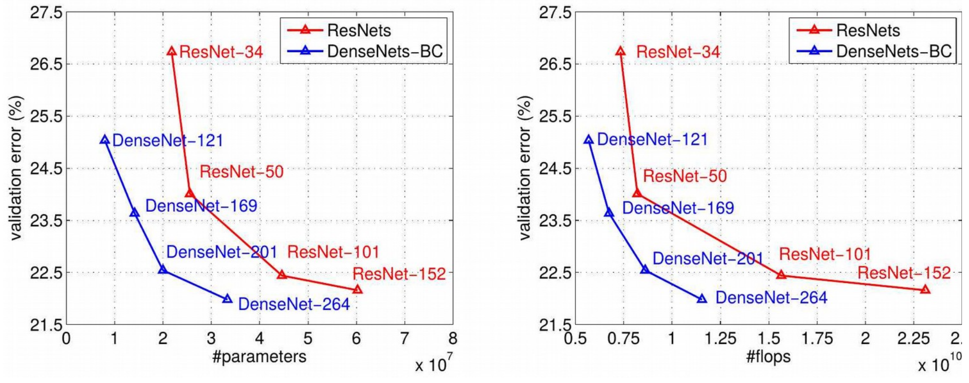


Figure 4: Comparison of DenseNets versus ResNets on the ImageNet validation dataset in terms of flops and number of parameters

Figure 6 shows that a Densenet-BC with 0.8M parameters achieves comparable accuracy to a 10.2M parameter ResNet. The test error rate is the same while the improved training error rate for ResNet points to a kind of overfitting which can not yield a respective effect for the test error rate.

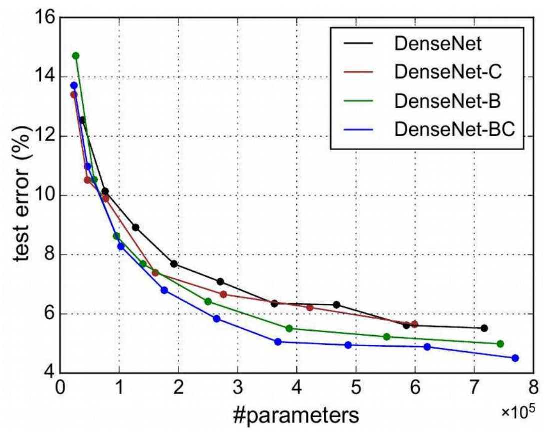


Figure 5: Parameter efficiency of different DenseNet variants on the C10+ data set

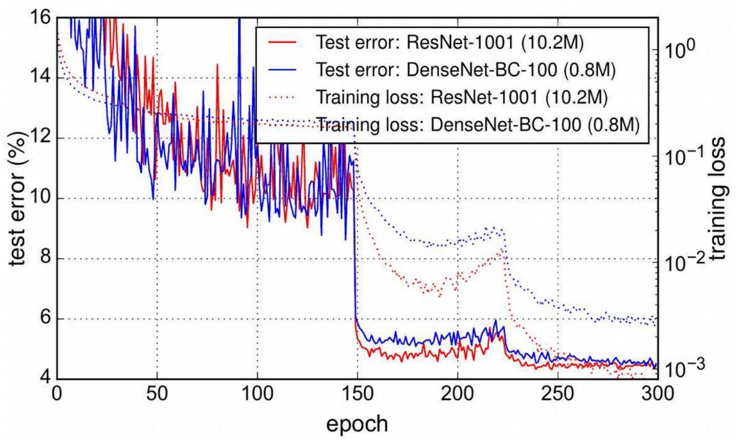


Figure 6: Training and test error curves of the 1001-layer pre-activation ResNet with more than 10M parameters and a 100-layer DenseNet-BC with 0.8M parameters

5. Conclusions

The DenseNet architecture scales naturally to hundreds of layers and yields a consistent improvement in accuracy with a growing number of parameters. DenseNets require substantially less parameters and computational power than comparable architectures. The tests in the article at hand have been conducted with parameters optimized for ResNets. The authors suppose that a parameter setting particularly fine tuned for DenseNets could even have performed better. The idea of dense blocks being fully interconnected in a feed-forward fashion at least within their interior naturally integrates many architectural features of existing CNNs like identity mappings, deep supervision and diversified depth.

References

[1] G. Huang, Z. Liu, L. van der Maaten, K. Q. Weinberger, “Densely Connected Convolutional Networks”, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4700-4708.

[2] K. He, X. Zhang, S. Rem and J. Sun, “Deep residual learning for image recognition”, in CVPR 2016.

[3] C-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, “Deeply-Supervised Nets”, in AISTATS 2015.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, “Going deeper with convolutions”, in CVPR, 2015.

[5] X. Glorot, A. Bordes, Y. Bengio, “Deep Sparse Rectifier Neural Networks”, in AISTATS, 2011

[6] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning”, MIT Press, 2016, ISBN 9780262035613