Convolutional Neural Networks II

Milan Straka

March 14, 2022

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Designing and Training Neural Networks

Designing and training a neural network is not a one-shot action, but instead an iterative procedure.

- When choosing hyperparameters, it is important to verify that the model does not underfit and does not overfit.
- Underfitting can be checked by trying increasing model capacity or training longer, and observing whether the training performance increases.
- Overfitting can be tested by observing train/dev difference, or by trying stronger regularization and observing whether the development performance improves.

Regarding hyperparameters:

- We need to set the number of training epochs so that development performance stops increasing during training (usually later than when the training performance plateaus).
- Generally, we want to use large enough batch size, but such a one which does not slow us down too much (GPUs sometimes allow larger batches without slowing down training). However, because larger batch size implies less noise in the gradient, small batch size sometimes work as regularization (especially for vanilla SGD algorithm).

Main Takeaways From Previous Lecture

- Convolutions can provide
  - local interactions in spacial/temporal dimensions
  - shift invariance
  - *much* less parameters than a fully connected layer
- Usually repeated 3×3 convolutions are enough, no need for larger filter sizes.
- When pooling is performed, double the number of channels (i.e., the first convolution following the pooling layer will have twice as many output channels).
- If your network is deep enough (the last hidden neurons have a large receptive fields), final fully connected layers are not needed, and global average pooling is enough.
- Batch normalization is a great regularization method for CNNs, allowing removal/decrease of dropout and L^2 regularization.
- Small weight decay (i.e., L^2 regularization) of usually 1e-4 is still useful for regularizing convolutional kernels.

ResNet – 2015 (3.6% error)

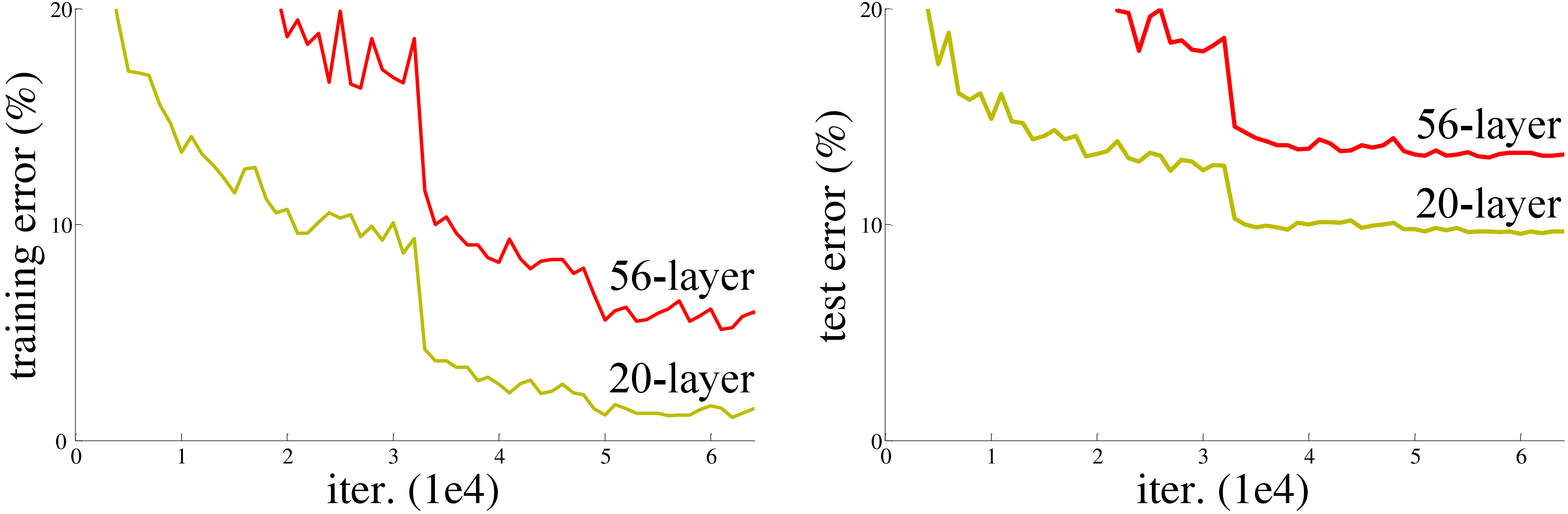


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Figure 1 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

ResNet – 2015 (3.6% error)

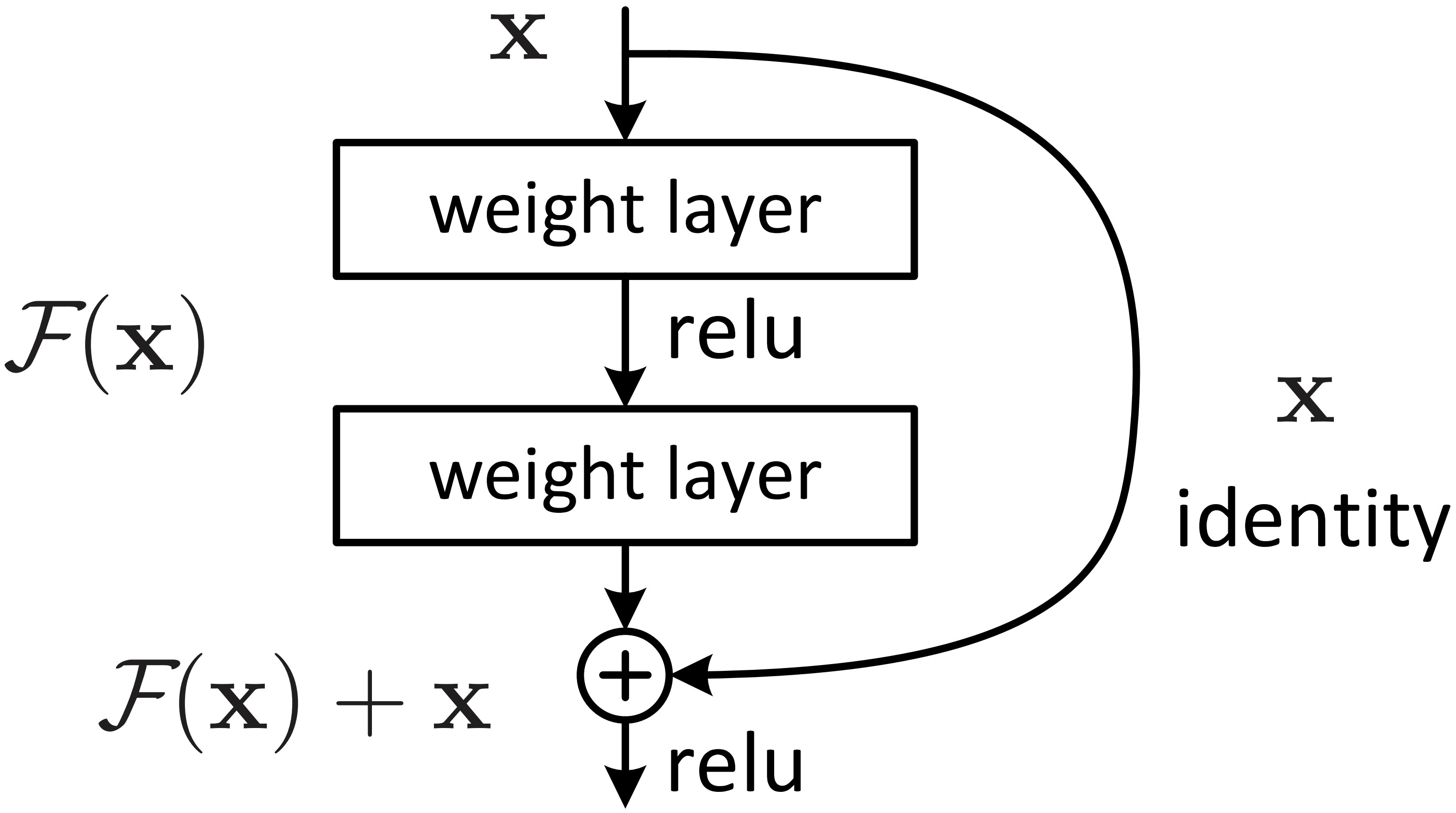


Figure 2. Residual learning: a building block.

Figure 2 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

ResNet – 2015 (3.6% error)

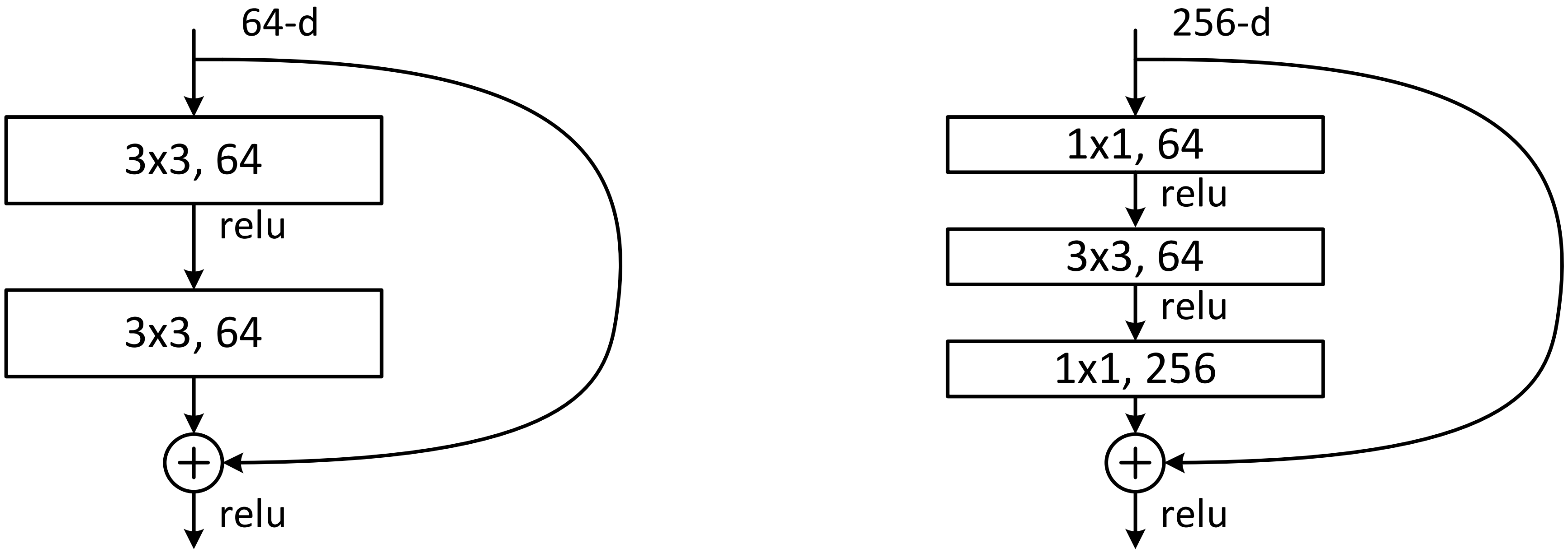
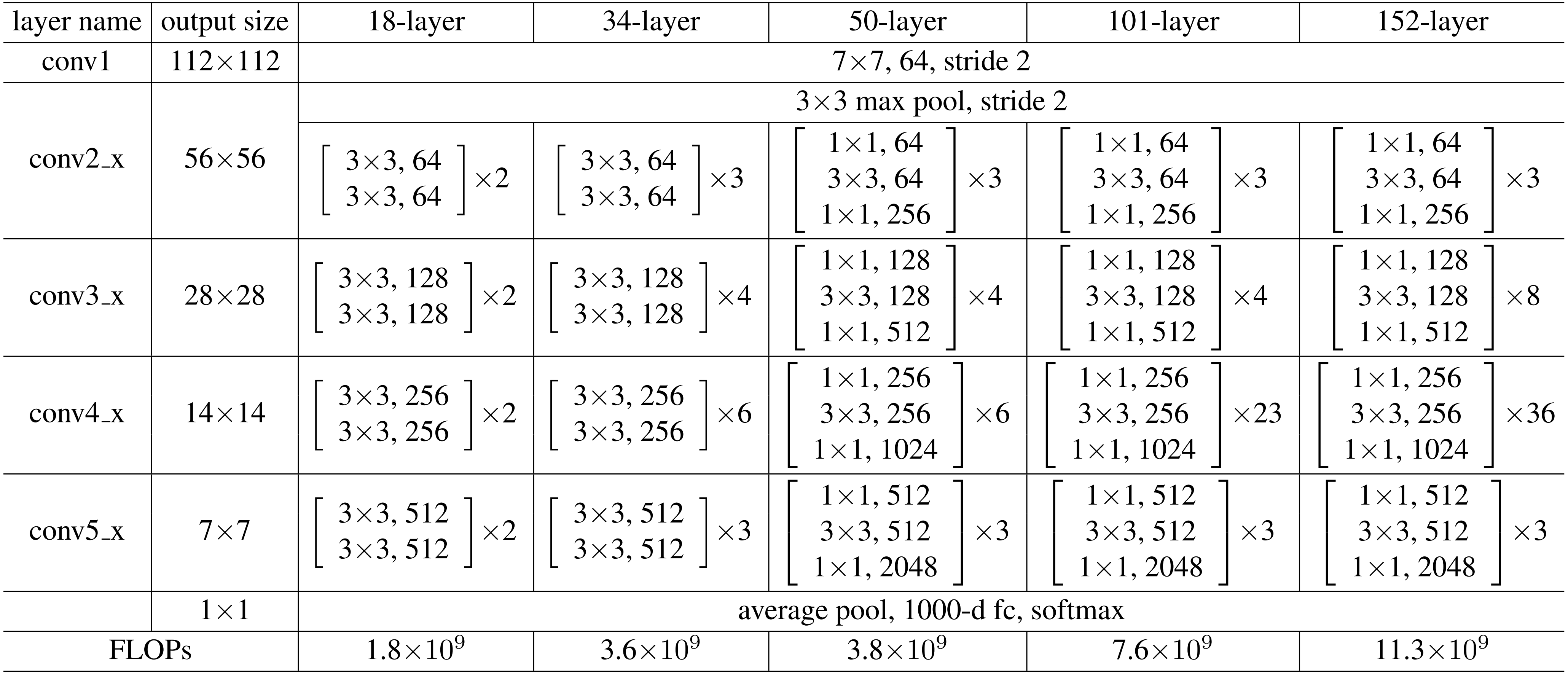


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

Figure 5 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

ResNet – 2015 (3.6% error)



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

ResNet – 2015 (3.6% error)

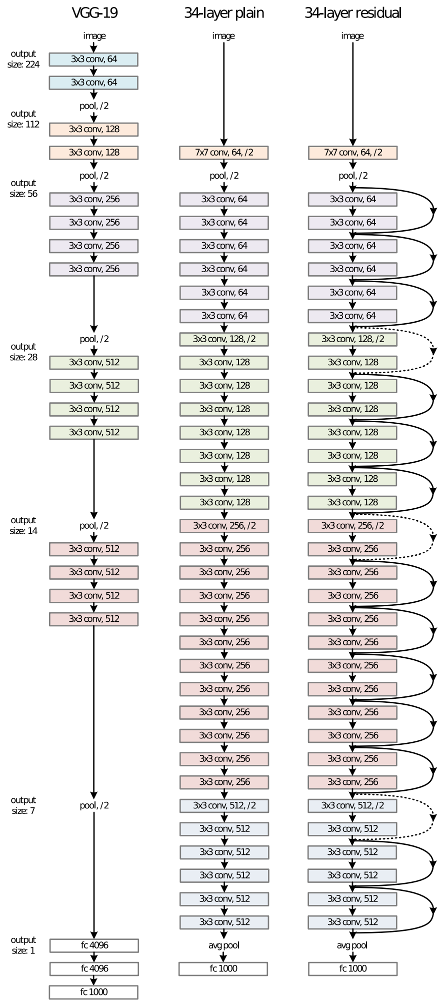


Figure 3 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

The residual connections cannot be applied directly when number of channels increases.

The authors considered several alternatives, and chose the one where in case of channels increase a 1×1 convolution + BN is used on the projections to match the required number of channels. The required spacial resolution is achieved by using stride 2.

ResNet – 2015 (3.6% error)

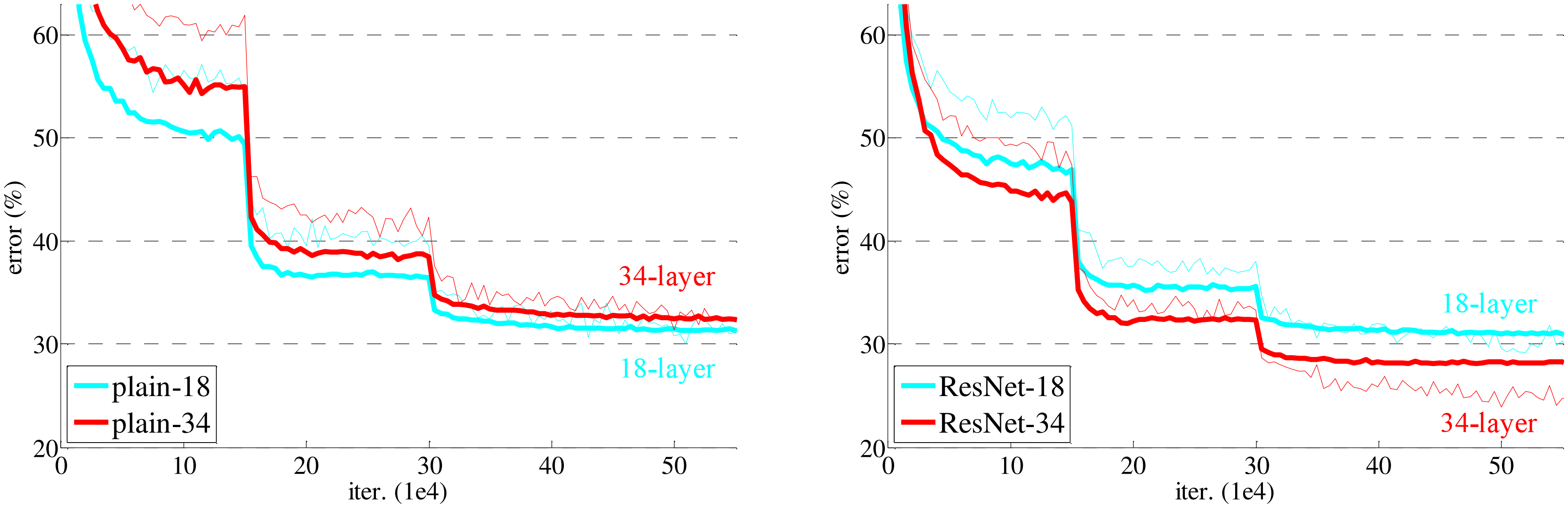


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Figure 4 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

ResNet – 2015 (3.6% error)

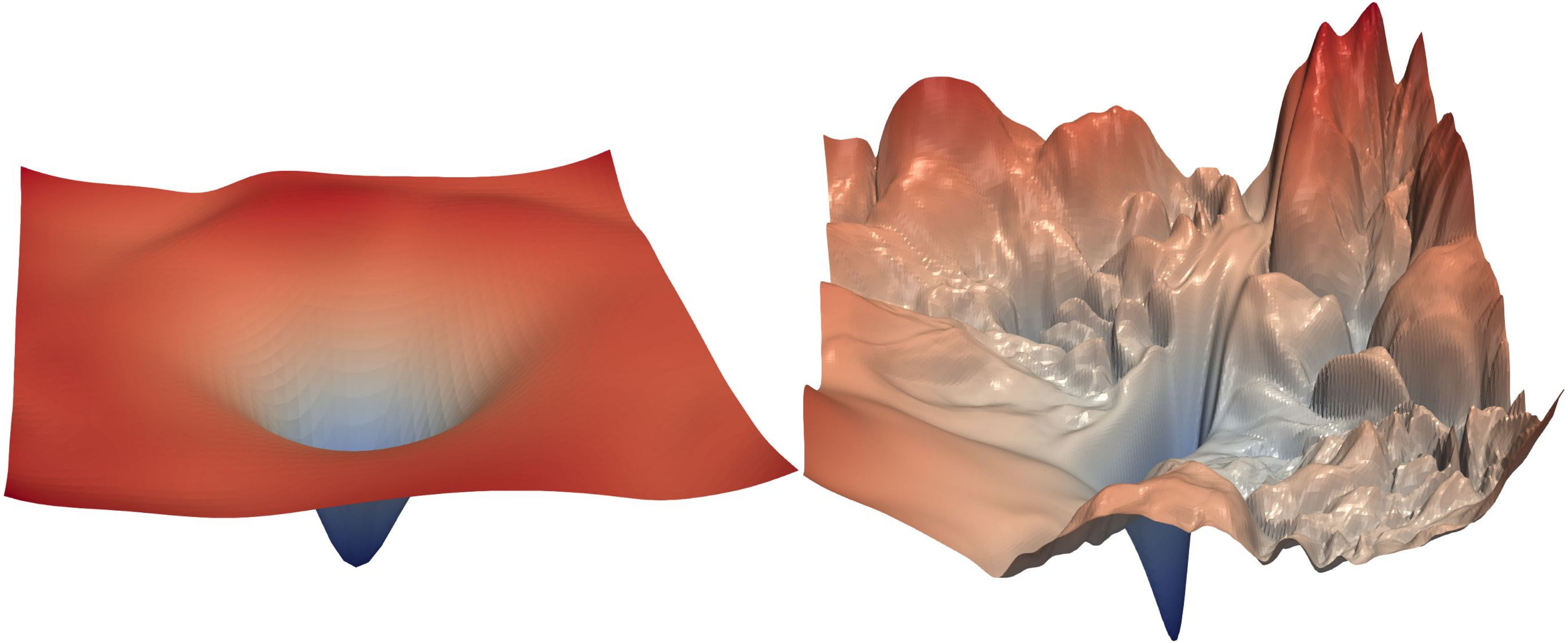


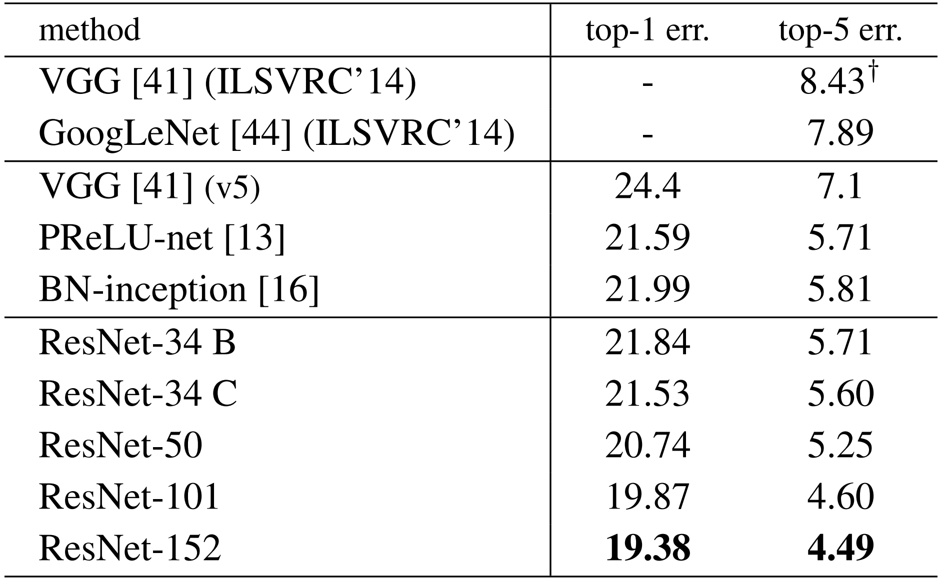
Figure 1 of "Visualizing the Loss Landscape of Neural Nets", https://arxiv.org/abs/1712.09913

ResNet – 2015 (3.6% error)

Training details:

- batch normalizations after each convolution and before activation
- SGD with batch size 256 and momentum of 0.9
- learning rate starts with 0.1 and is divided by 10 when error plateaus
- no dropout, weight decay 0.0001
- during training, an image is resized with its shorter side randomly sampled in the range $[256, 480]$, and a random 224×224 crop is used
- during testing, 10-crop evaluation strategy is used
  - for the best results, the scores across multiple scales are averaged – the images are resized so that their smaller size is in $\{224, 256, 384, 480, 640\}$

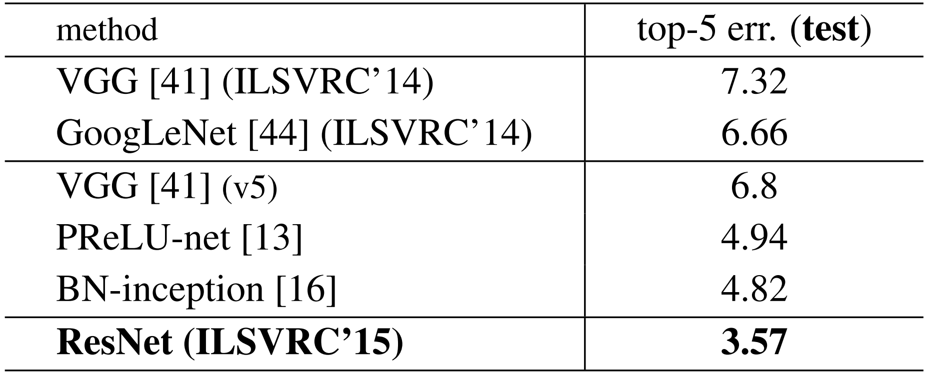
ResNet – 2015 (3.6% error)



method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	$8.43^\dagger$
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Table 4 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385



method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

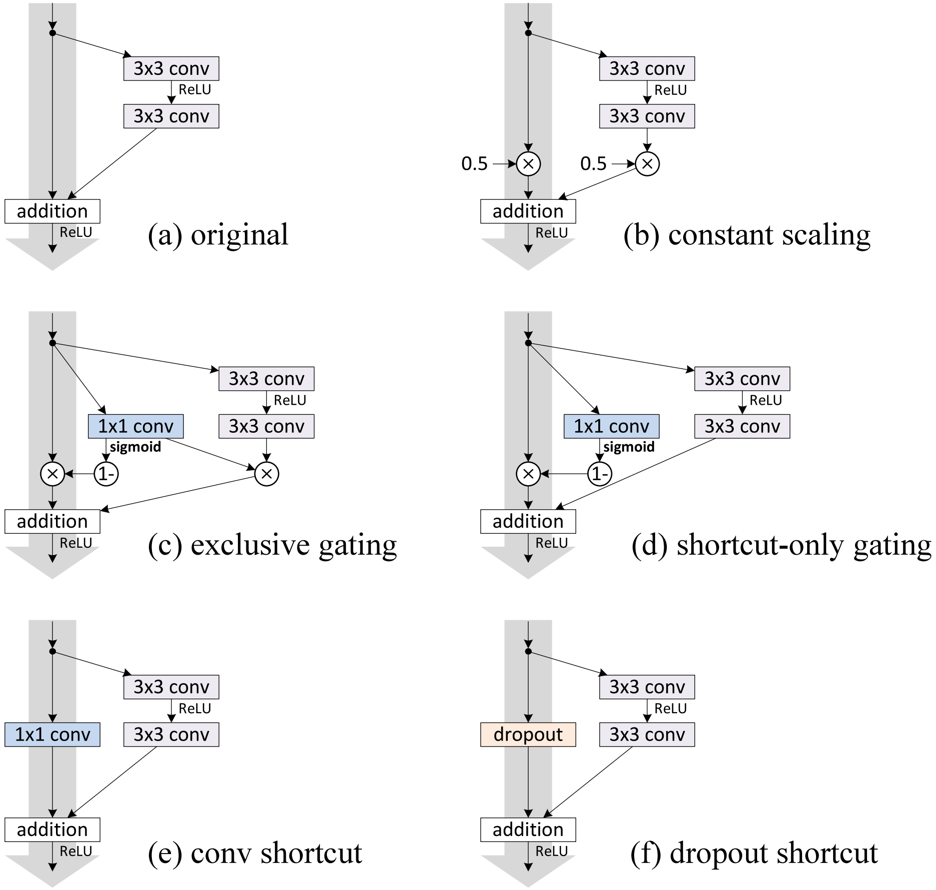
Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

Table 5 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

The ResNet-34 B uses the 1×1 convolution on residual connections with different number of input and output channels; ResNet-34 C uses this convolution on all residual connections. Variant B is used for ResNet-50/101/152.

ResNet Ablations – Shortcuts

The authors of ResNet published an ablation study several months after the original paper.



(a) original

(b) constant scaling

(c) exclusive gating

(d) shortcut-only gating

(e) conv shortcut

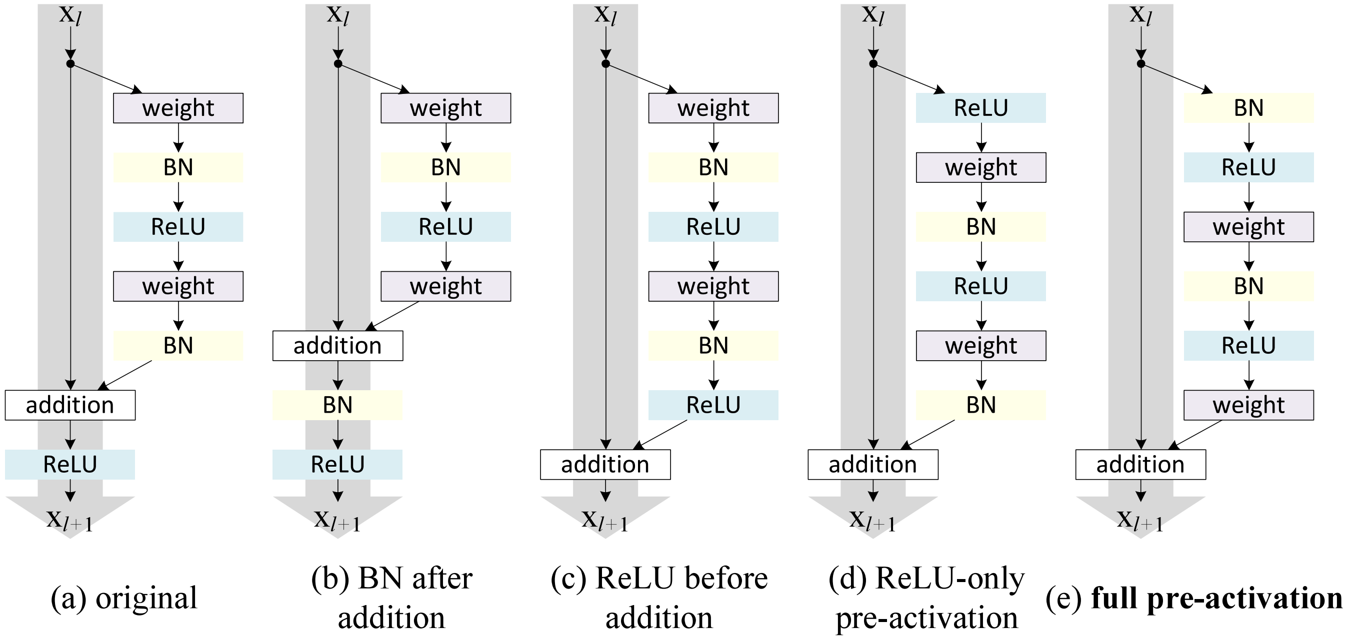
(f) dropout shortcut

Figure 2 of "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027

case	Fig.	on shortcut	on $\mathcal{F}$	error (%)	remark
original [1]	Fig. 2(a)	1	1	**6.61**	
constant scaling	Fig. 2(b)	0	1	fail	This is a plain net
		0.5	1	fail	
		0.5	0.5	12.35	frozen gating
exclusive gating	Fig. 2(c)	$1 - g(\mathbf{x})$	$g(\mathbf{x})$	fail	init b_g=0 to −5
		$1 - g(\mathbf{x})$	$g(\mathbf{x})$	8.70	init b_g=-6
		$1 - g(\mathbf{x})$	$g(\mathbf{x})$	9.81	init b_g=-7
shortcut-only gating	Fig. 2(d)	$1 - g(\mathbf{x})$	1	12.86	init b_g=0
		$1 - g(\mathbf{x})$	1	6.91	init b_g=-6
1×1 conv shortcut	Fig. 2(e)	1×1 conv	1	12.22	
dropout shortcut	Fig. 2(f)	dropout 0.5	1	fail	

Table 1 of "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027

ResNet Ablations – Activations



(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

Figure 4 of "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

Table 2 of "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027

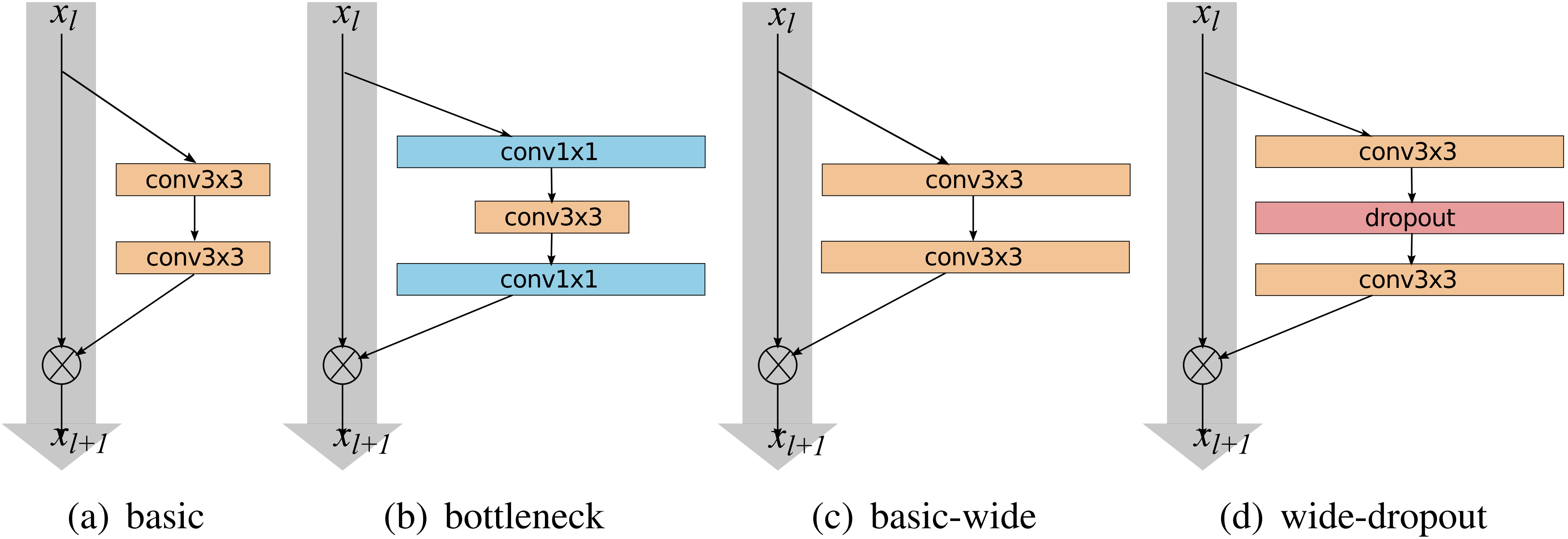
ResNet Ablations – Pre-Activation Results

The *pre-activation* architecture was evaluated also on ImageNet, in a single-crop regime.

method	augmentation	train crop	test crop	top-1	top-5
ResNet-152, original Residual Unit [1]	scale	224×224	224×224	23.0	6.7
ResNet-152, original Residual Unit [1]	scale	224×224	320×320	21.3	5.5
ResNet-152, **pre-act** Residual Unit	scale	224×224	320×320	21.1	5.5
ResNet-200, original Residual Unit [1]	scale	224×224	320×320	21.8	6.0
ResNet-200, **pre-act** Residual Unit	scale	224×224	320×320	**20.7**	**5.3**
ResNet-200, **pre-act** Residual Unit	scale+asp ratio	224×224	320×320	**20.1**†	**4.8**†
Inception v3 [19]	scale+asp ratio	299×299	299×299	21.2	5.6

Table 5 of "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027

WideNet



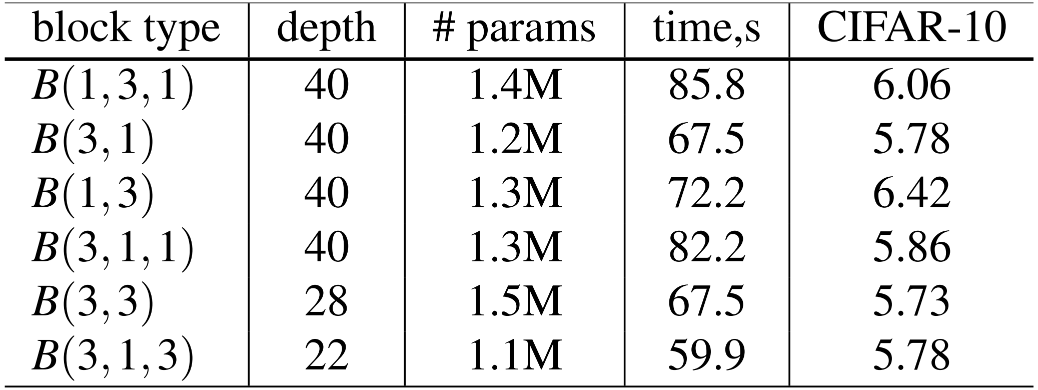
(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

Figure 1 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146

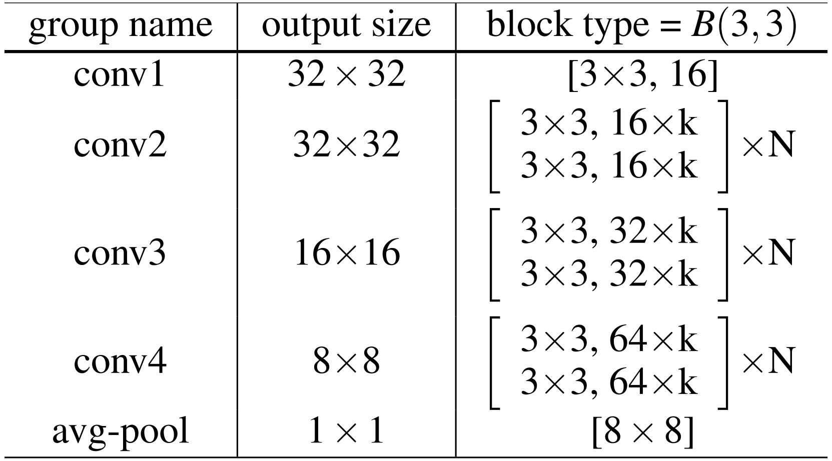
WideNet

- Authors do not consider bottleneck blocks. Instead, they experiment with different *block types*, e.g., $B(1,3,1)$ or $B(3,3)$.



block type	depth	# params	time,s	CIFAR-10
$B(1,3,1)$	40	1.4M	85.8	6.06
$B(3,1)$	40	1.2M	67.5	5.78
$B(1,3)$	40	1.3M	72.2	6.42
$B(3,1,1)$	40	1.3M	82.2	5.86
$B(3,3)$	28	1.5M	67.5	5.73
$B(3,1,3)$	22	1.1M	59.9	5.78

Table 2 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146



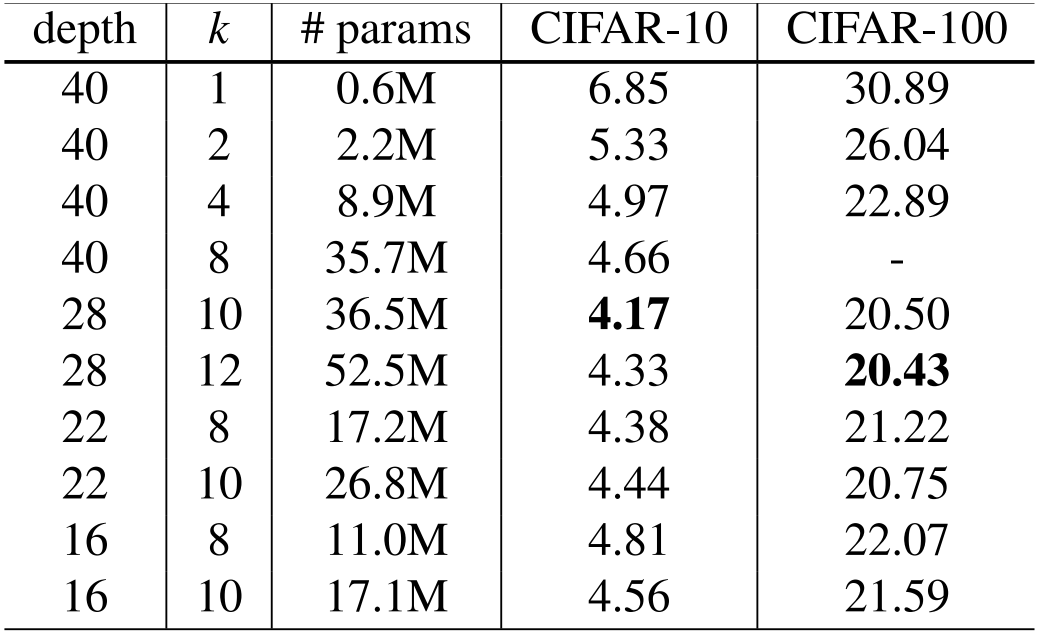
group name	output size	block type = $B(3,3)$
conv1	32×32	$[3\times3, 16]$
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Table 1 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146

The $B(3,3)$ is used in further experiments, unless specified otherwise.

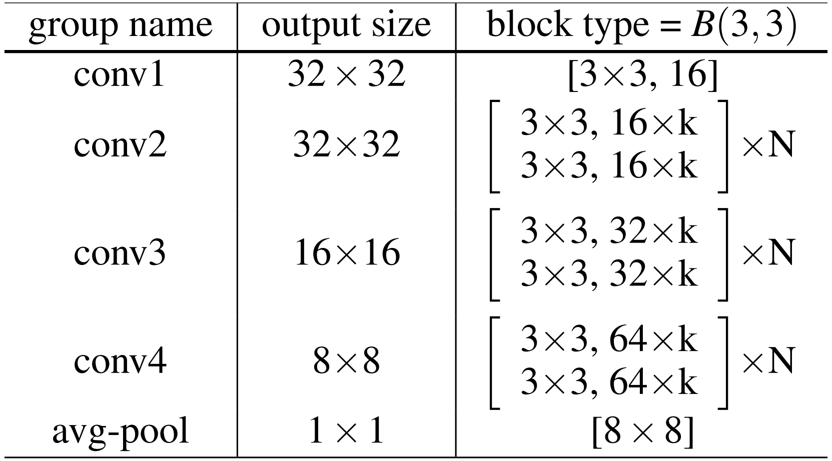
WideNet

- Authors evaluate various *widening factors* k



depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Table 4 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146



group name	output size	block type = $B(3,3)$
conv1	32×32	$[3\times3, 16]$
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Table 1 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146

WideNet

- Authors measure the effect of *dropping out* inside the residual block (but not the residual connection itself)

depth	k	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.00	19.25	-
28	10	✓	**3.89**	**18.85**	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

Table 6 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Table 1 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146

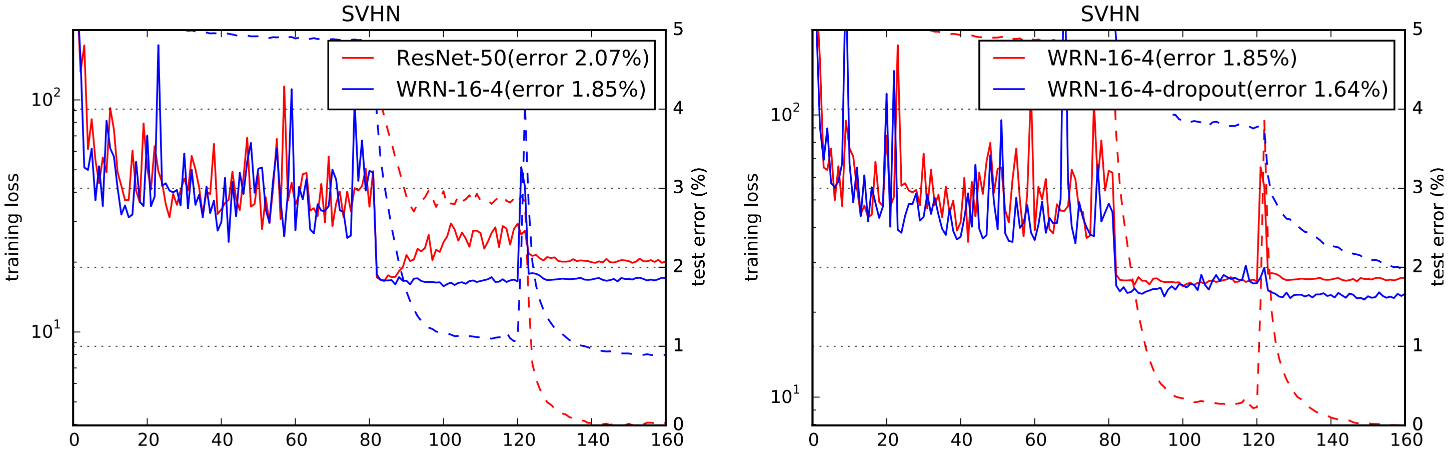
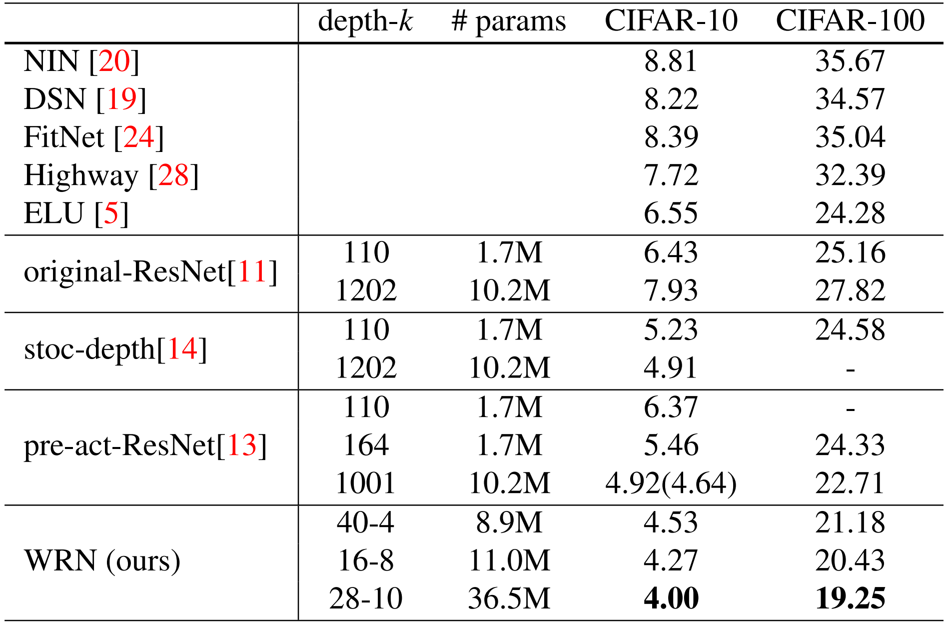


Figure 3 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146

WideNet – Results

<table>
<tr><th>Dataset</th><th>Results</th></tr>
<tr><td>CIFAR</td><td>

<table>
<tr><th></th><th>depth-<i>k</i></th><th># params</th><th>CIFAR-10</th><th>CIFAR-100</th></tr>
<tr><td>NIN [20]</td><td></td><td></td><td>8.81</td><td>35.67</td></tr>
<tr><td>DSN [19]</td><td></td><td></td><td>8.22</td><td>34.57</td></tr>
<tr><td>FitNet [24]</td><td></td><td></td><td>8.39</td><td>35.04</td></tr>
<tr><td>Highway [28]</td><td></td><td></td><td>7.72</td><td>32.39</td></tr>
<tr><td>ELU [5]</td><td></td><td></td><td>6.55</td><td>24.28</td></tr>
<tr><td>original-ResNet[11]</td><td>110</td><td>1.7M</td><td>6.43</td><td>25.16</td></tr>
<tr><td></td><td>1202</td><td>10.2M</td><td>7.93</td><td>27.82</td></tr>
<tr><td>stoc-depth[14]</td><td>110</td><td>1.7M</td><td>5.23</td><td>24.58</td></tr>
<tr><td></td><td>1202</td><td>10.2M</td><td>4.91</td><td>-</td></tr>
<tr><td>pre-act-ResNet[13]</td><td>110</td><td>1.7M</td><td>6.37</td><td>-</td></tr>
<tr><td></td><td>164</td><td>1.7M</td><td>5.46</td><td>24.33</td></tr>
<tr><td></td><td>1001</td><td>10.2M</td><td>4.92(4.64)</td><td>22.71</td></tr>
<tr><td>WRN (ours)</td><td>40-4</td><td>8.9M</td><td>4.53</td><td>21.18</td></tr>
<tr><td></td><td>16-8</td><td>11.0M</td><td>4.27</td><td>20.43</td></tr>
<tr><td></td><td>28-10</td><td>36.5M</td><td>4.00</td><td>19.25</td></tr>
</table>
<i>Table 5 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146</i>
</td></tr>
<tr><td>ImageNet</td><td>

<table>
<tr><th>Model</th><th>top-1 err, %</th><th>top-5 err, %</th><th>#params</th><th>time/batch 16</th></tr>
<tr><td>ResNet-50</td><td>24.01</td><td>7.02</td><td>25.6M</td><td>49</td></tr>
<tr><td>ResNet-101</td><td>22.44</td><td>6.21</td><td>44.5M</td><td>82</td></tr>
<tr><td>ResNet-152</td><td>22.16</td><td>6.16</td><td>60.2M</td><td>115</td></tr>
<tr><td>WRN-50-2-bottleneck</td><td>21.9</td><td>6.03</td><td>68.9M</td><td>93</td></tr>
<tr><td>pre-ResNet-200</td><td>21.66</td><td>5.79</td><td>64.7M</td><td>154</td></tr>
</table>
<i>Table 8 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146</i>
</td></tr>
</table>

DenseNet

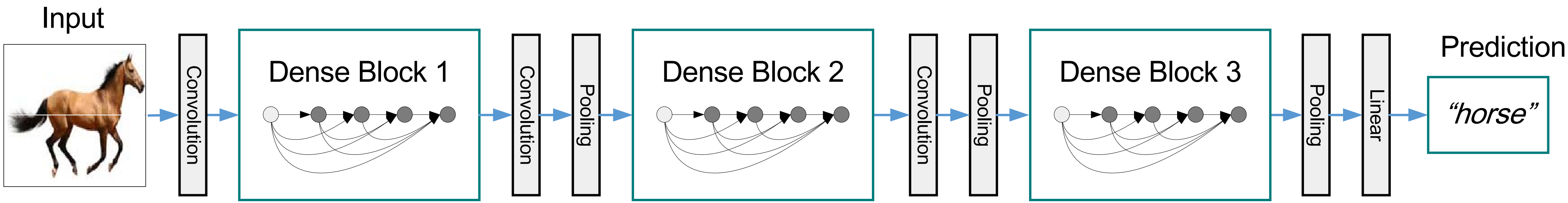


Figure 2 of "Densely Connected Convolutional Networks", https://arxiv.org/abs/1608.06993

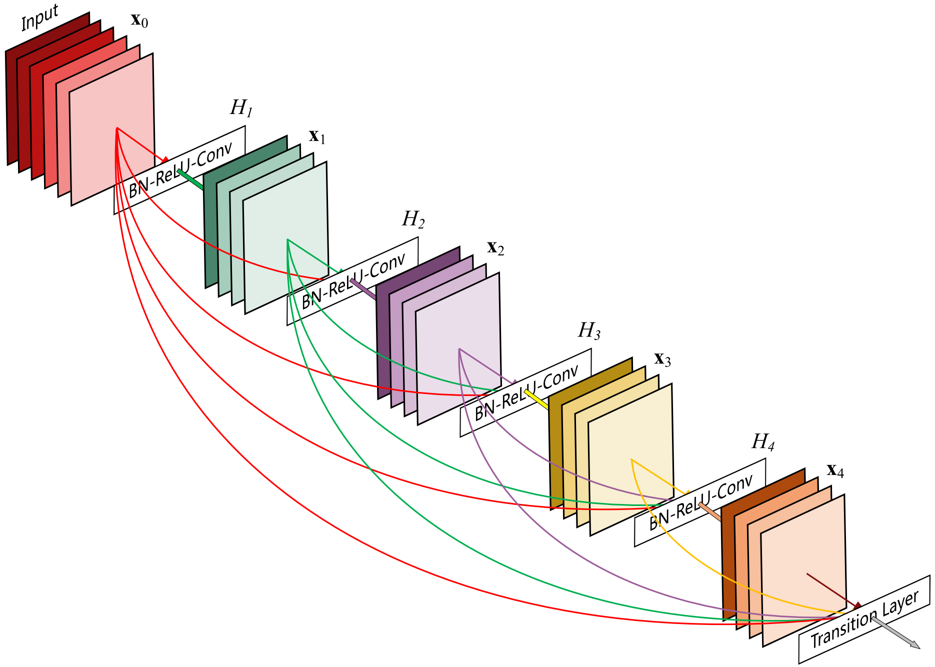
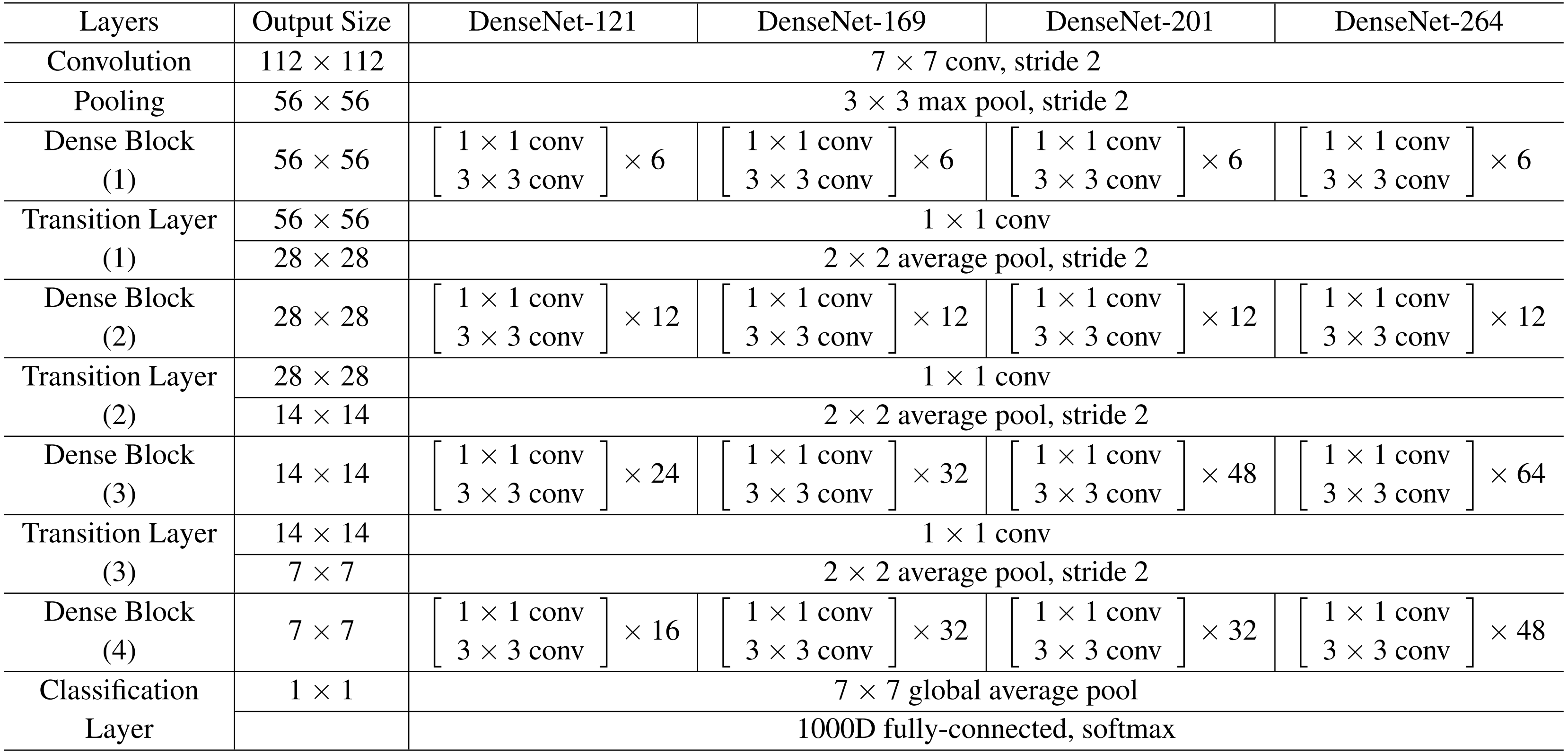


Figure 1 of "Densely Connected Convolutional Networks", https://arxiv.org/abs/1608.06993

DenseNet – Architecture

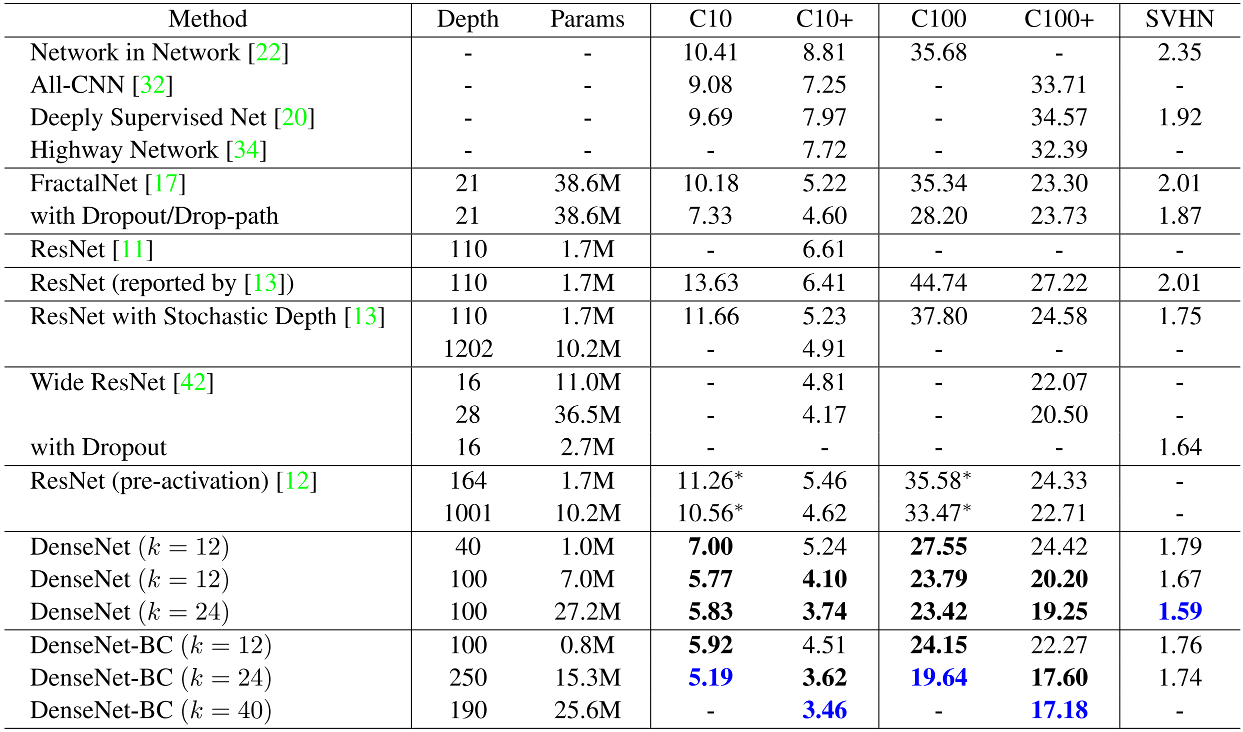
The initial convolution generates 64 channels, each 1×1 convolution in dense block 256, each 3×3 convolution in dense block 32, and the transition layer reduces the number of channels in the initial convolution by half.



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 of "Densely Connected Convolutional Networks", https://arxiv.org/abs/1608.06993

DenseNet – Results



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 2 of "Densely Connected Convolutional Networks", https://arxiv.org/abs/1608.06993

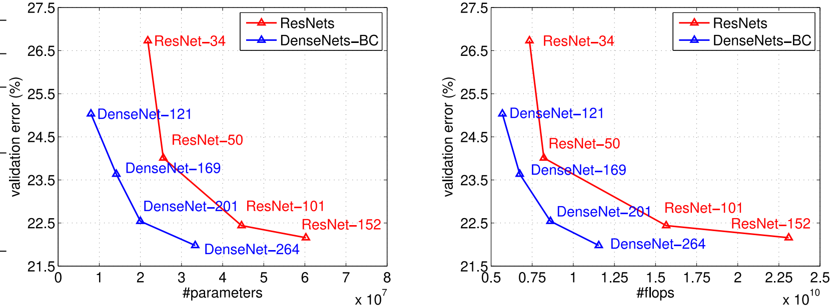


Figure 3 of "Densely Connected Convolutional Networks", https://arxiv.org/abs/1608.06993

PyramidNet

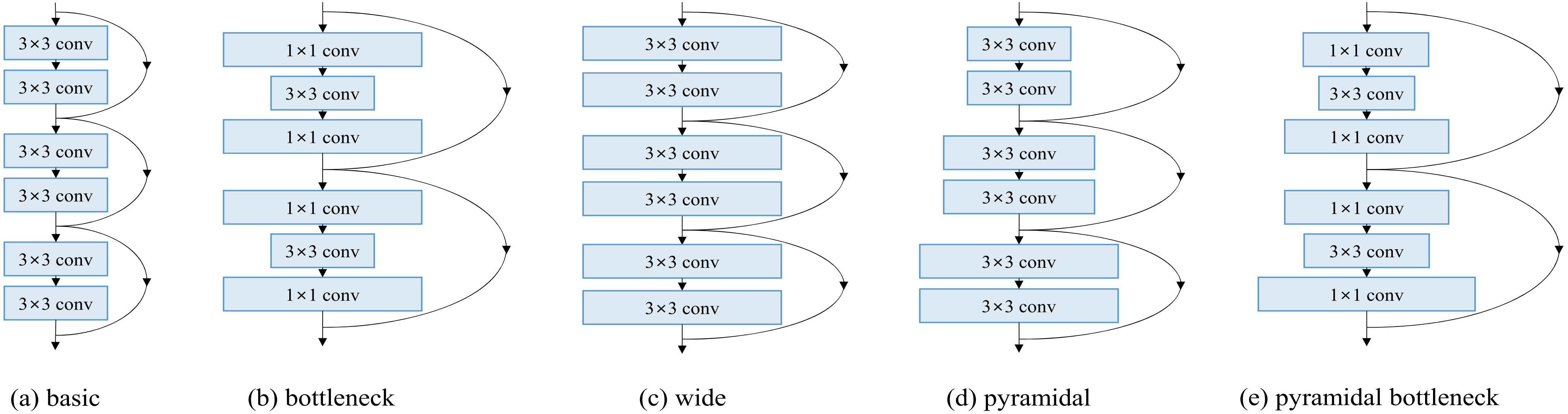


Figure 1 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915

PyramidNet – Growth Rate

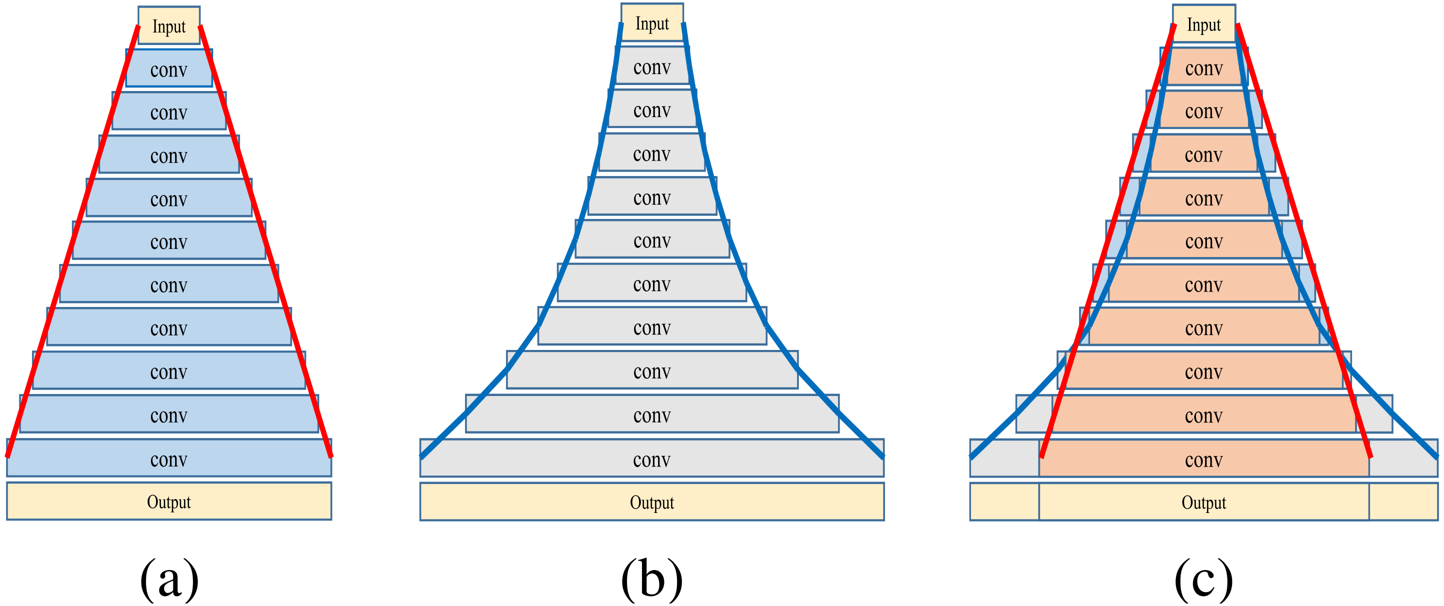


Figure 2 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915

In architectures up until now, number of filters doubled when spacial resolution was halved.

Such exponential growth would suggest gradual widening rule $D_k = \lfloor D_{k-1} \cdot \alpha^{1/N} \rfloor$.

However, the authors employ a linear widening rule $D_k = \lfloor D_{k-1} + \alpha/N \rfloor$, where D_k is number of filters in the k-th out of N convolutional block and α is number of filters to add in total.

PyramidNet – Residual Connections

No residual connection can be a real identity – the authors propose to zero-pad missing channels, where the zero-pad channels correspond to newly computed features.

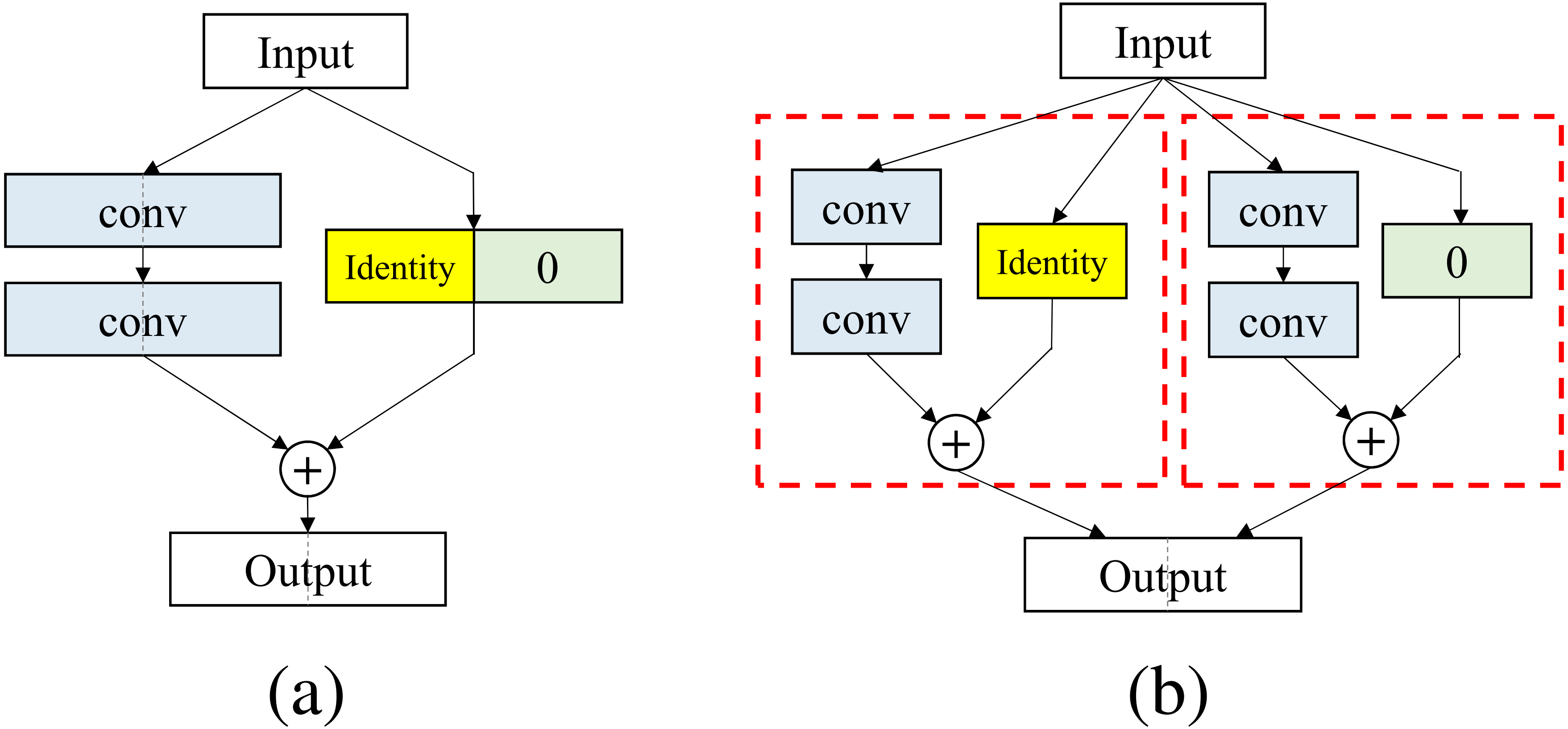
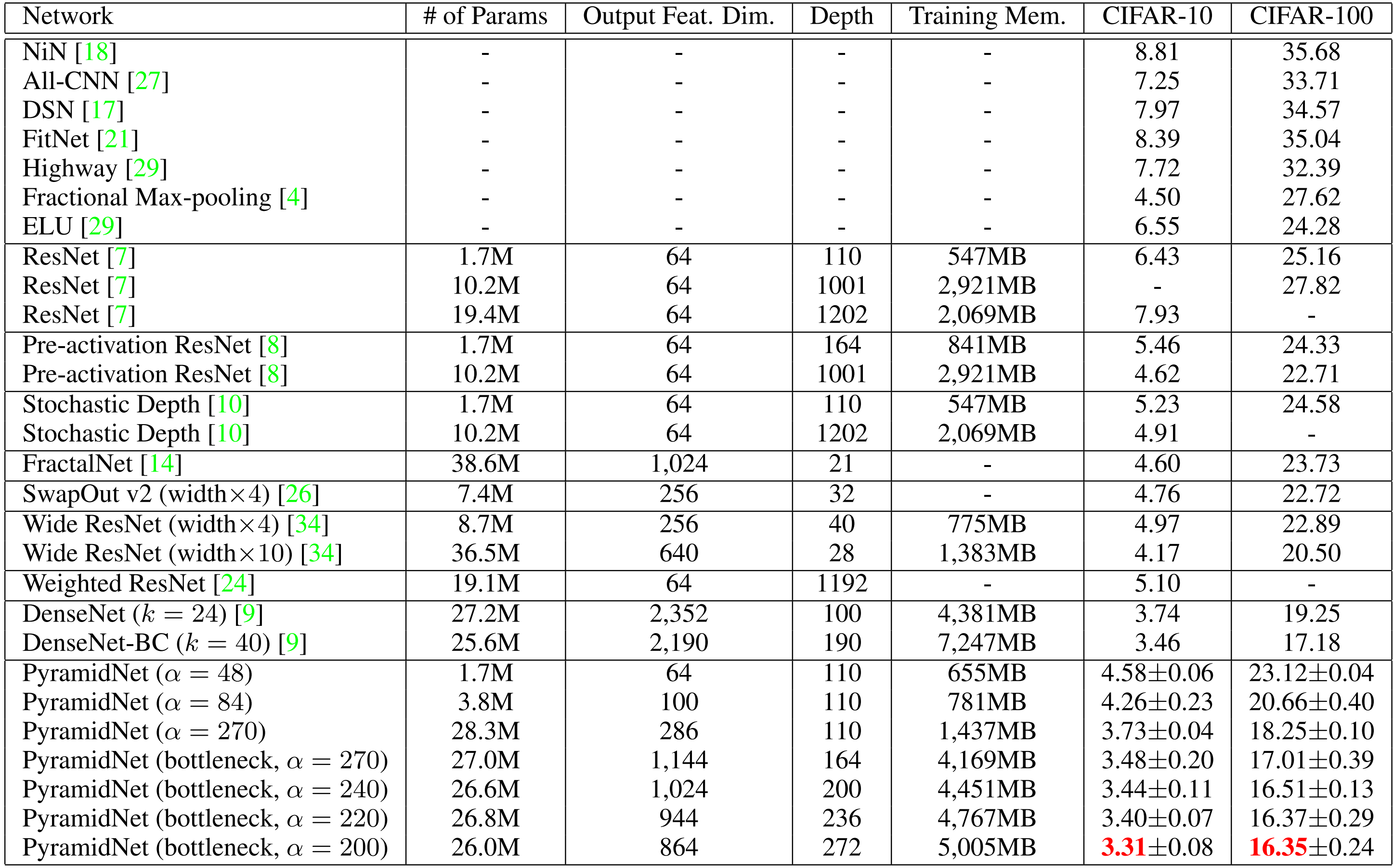


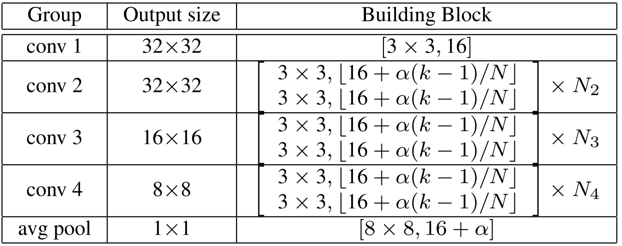
Figure 5 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915

PyramidNet – CIFAR Results



Network	# of Params	Output Feat. Dim.	Depth	Training Mem.	CIFAR-10	CIFAR-100
NiN [18]	-	-	-	-	8.81	35.68
All-CNN [27]	-	-	-	-	7.25	33.71
DSN [17]	-	-	-	-	7.97	34.57
FitNet [21]	-	-	-	-	8.39	35.04
Highway [29]	-	-	-	-	7.72	32.39
Fractional Max-pooling [4]	-	-	-	-	4.50	27.62
ELU [29]	-	-	-	-	6.55	24.28
ResNet [7]	1.7M	64	110	547MB	6.43	25.16
ResNet [7]	10.2M	64	1001	2,921MB	-	27.82
ResNet [7]	19.4M	64	1202	2,069MB	7.93	-
Pre-activation ResNet [8]	1.7M	64	164	841MB	5.46	24.33
Pre-activation ResNet [8]	10.2M	64	1001	2,921MB	4.62	22.71
Stochastic Depth [10]	1.7M	64	110	547MB	5.23	24.58
Stochastic Depth [10]	10.2M	64	1202	2,069MB	4.91	-
FractalNet [14]	38.6M	1,024	21	-	4.60	23.73
SwapOut v2 (width×4) [26]	7.4M	256	32	-	4.76	22.72
Wide ResNet (width×4) [34]	8.7M	256	40	775MB	4.97	22.89
Wide ResNet (width×10) [34]	36.5M	640	28	1,383MB	4.17	20.50
Weighted ResNet [24]	19.1M	64	1192	-	5.10	-
DenseNet ($k = 24$) [9]	27.2M	2,352	100	4,381MB	3.74	19.25
DenseNet-BC ($k = 40$) [9]	25.6M	2,190	190	7,247MB	3.46	17.18
PyramidNet ($\alpha = 48$)	1.7M	64	110	655MB	4.58±0.06	23.12±0.04
PyramidNet ($\alpha = 84$)	3.8M	100	110	781MB	4.26±0.23	20.66±0.40
PyramidNet ($\alpha = 270$)	28.3M	286	110	1,437MB	3.73±0.04	18.25±0.10
PyramidNet (bottleneck, $\alpha = 270$)	27.0M	1,144	164	4,169MB	3.48±0.20	17.01±0.39
PyramidNet (bottleneck, $\alpha = 240$)	26.6M	1,024	200	4,451MB	3.44±0.11	16.51±0.13
PyramidNet (bottleneck, $\alpha = 220$)	26.8M	944	236	4,767MB	3.40±0.07	16.37±0.29
PyramidNet (bottleneck, $\alpha = 200$)	26.0M	864	272	5,005MB	**3.31**±0.08	**16.35**±0.24

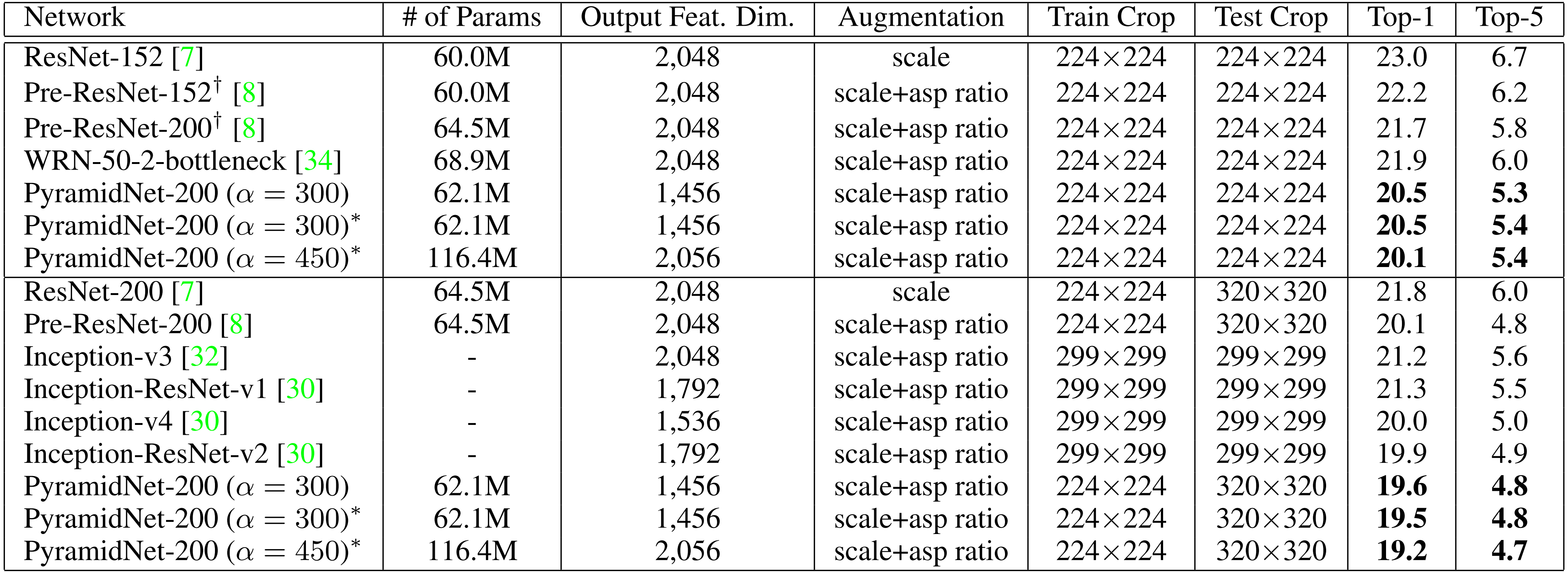
Table 4 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915



Group	Output size	Building Block
conv 1	32×32	$[3\times3, 16]$
conv 2	32×32	$\begin{bmatrix} 3\times3, \lfloor 16+\alpha(k-1)/N \rfloor \\ 3\times3, \lfloor 16+\alpha(k-1)/N \rfloor \end{bmatrix} \times N_2$
conv 3	16×16	$\begin{bmatrix} 3\times3, \lfloor 16+\alpha(k-1)/N \rfloor \\ 3\times3, \lfloor 16+\alpha(k-1)/N \rfloor \end{bmatrix} \times N_3$
conv 4	8×8	$\begin{bmatrix} 3\times3, \lfloor 16+\alpha(k-1)/N \rfloor \\ 3\times3, \lfloor 16+\alpha(k-1)/N \rfloor \end{bmatrix} \times N_4$
avg pool	1×1	$[8\times8, 16+\alpha]$

Table 1 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915

PyramidNet – ImageNet Results



Network	# of Params	Output Feat. Dim.	Augmentation	Train Crop	Test Crop	Top-1	Top-5
ResNet-152 [7]	60.0M	2,048	scale	224×224	224×224	23.0	6.7
Pre-ResNet-152† [8]	60.0M	2,048	scale+asp ratio	224×224	224×224	22.2	6.2
Pre-ResNet-200† [8]	64.5M	2,048	scale+asp ratio	224×224	224×224	21.7	5.8
WRN-50-2-bottleneck [34]	68.9M	2,048	scale+asp ratio	224×224	224×224	21.9	6.0
PyramidNet-200 ($\alpha = 300$)	62.1M	1,456	scale+asp ratio	224×224	224×224	**20.5**	**5.3**
PyramidNet-200 ($\alpha = 300$)*	62.1M	1,456	scale+asp ratio	224×224	224×224	**20.5**	**5.4**
PyramidNet-200 ($\alpha = 450$)*	116.4M	2,056	scale+asp ratio	224×224	224×224	**20.1**	**5.4**
ResNet-200 [7]	64.5M	2,048	scale	224×224	320×320	21.8	6.0
Pre-ResNet-200 [8]	64.5M	2,048	scale+asp ratio	224×224	320×320	20.1	4.8
Inception-v3 [32]	-	2,048	scale+asp ratio	299×299	299×299	21.2	5.6
Inception-ResNet-v1 [30]	-	1,792	scale+asp ratio	299×299	299×299	21.3	5.5
Inception-v4 [30]	-	1,536	scale+asp ratio	299×299	299×299	20.0	5.0
Inception-ResNet-v2 [30]	-	1,792	scale+asp ratio	299×299	299×299	19.9	4.9
PyramidNet-200 ($\alpha = 300$)	62.1M	1,456	scale+asp ratio	224×224	320×320	**19.6**	**4.8**
PyramidNet-200 ($\alpha = 300$)*	62.1M	1,456	scale+asp ratio	224×224	320×320	**19.5**	**4.8**
PyramidNet-200 ($\alpha = 450$)*	116.4M	2,056	scale+asp ratio	224×224	320×320	**19.2**	**4.7**

Table 5 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915

ResNeXt

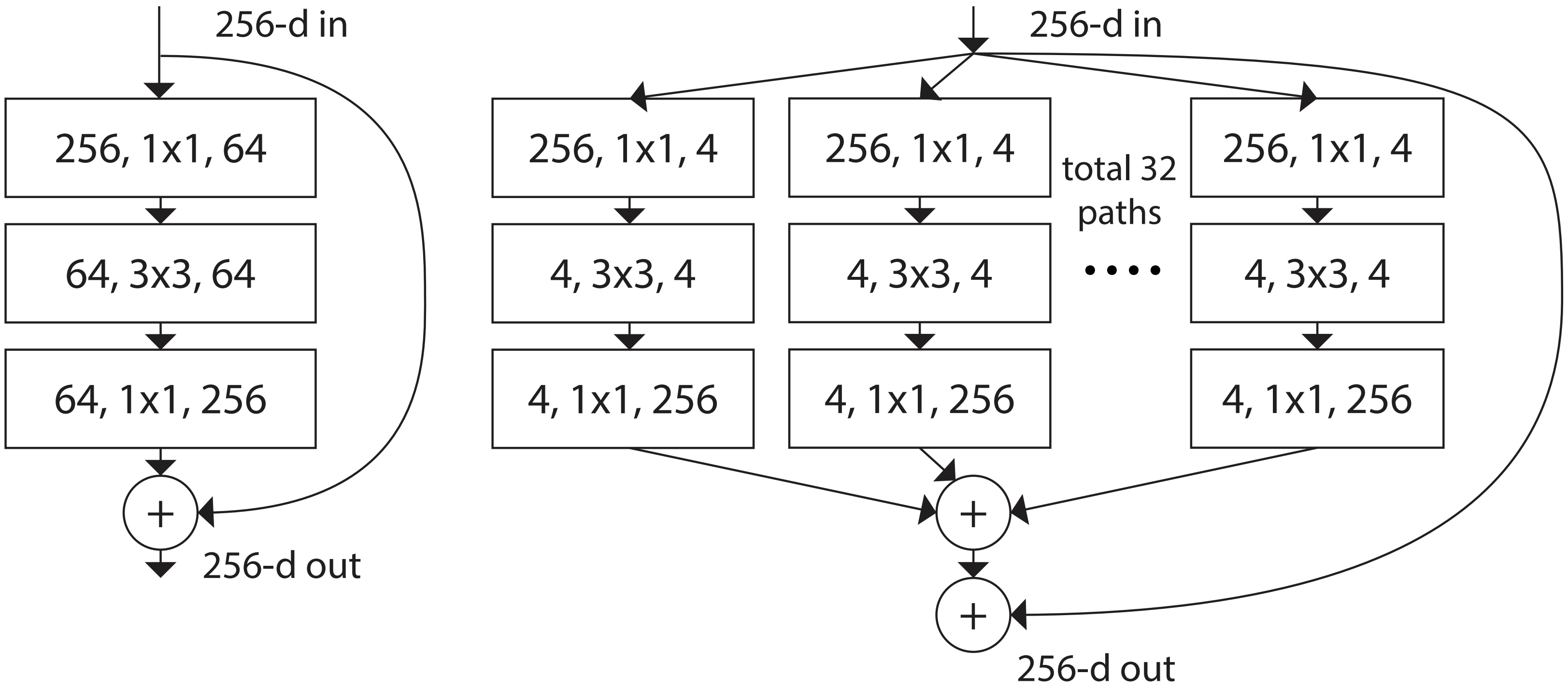


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Figure 1 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431

ResNeXt

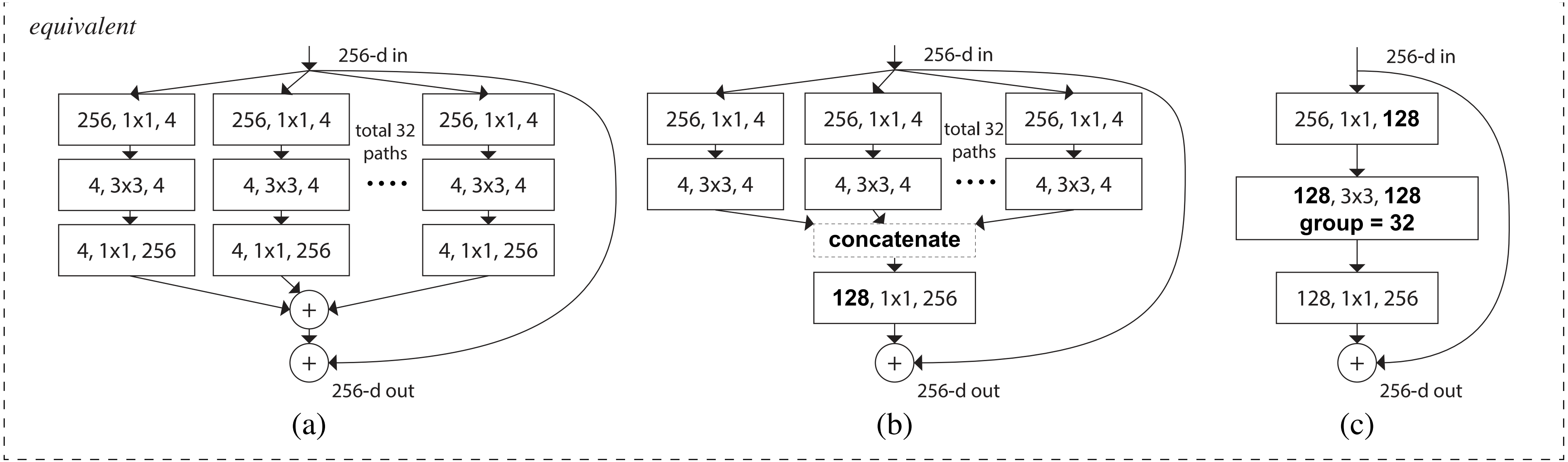
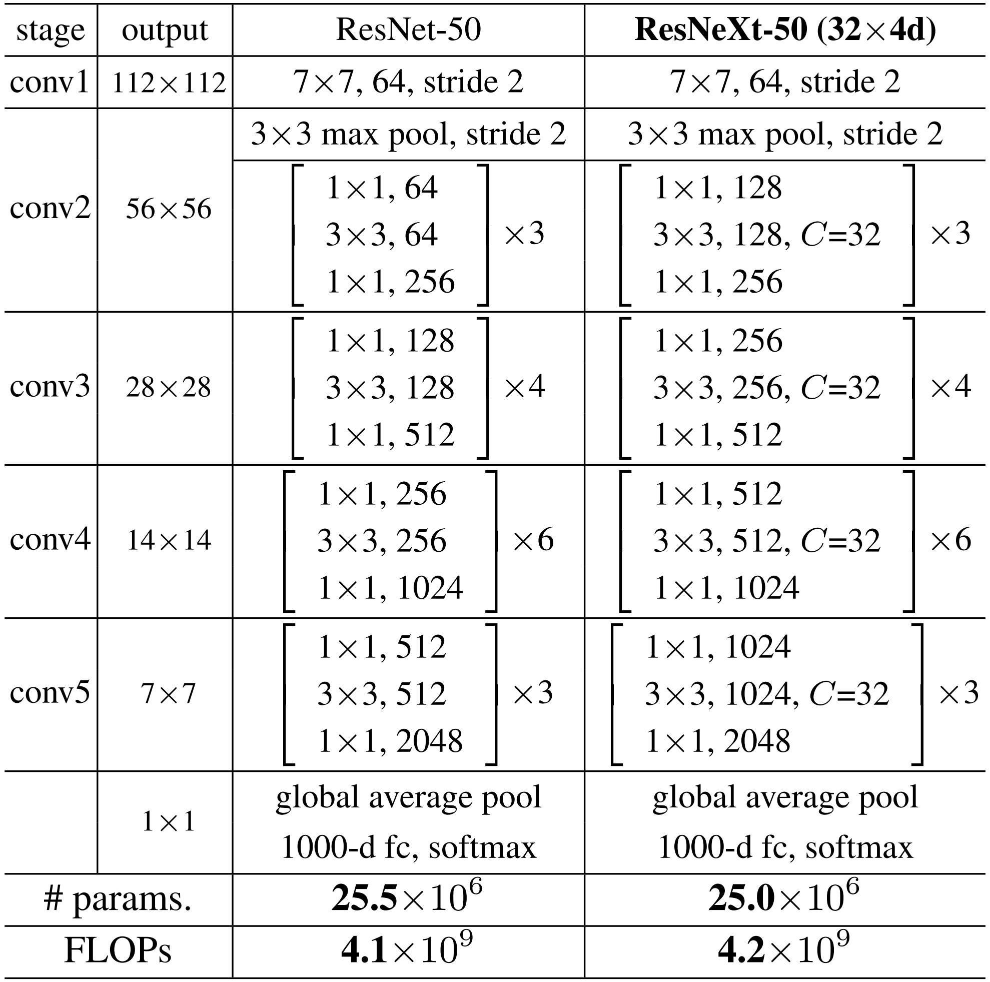


Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivalent to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [24]. Notations in **bold** text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

Figure 3 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431

ResNeXt



stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\left[\begin{array}{l}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times 3$	$\left[\begin{array}{l}1\times1, 128\\3\times3, 128, C{=}32\\1\times1, 256\end{array}\right]\times 3$
conv3	28×28	$\left[\begin{array}{l}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times 4$	$\left[\begin{array}{l}1\times1, 256\\3\times3, 256, C{=}32\\1\times1, 512\end{array}\right]\times 4$
conv4	14×14	$\left[\begin{array}{l}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times 6$	$\left[\begin{array}{l}1\times1, 512\\3\times3, 512, C{=}32\\1\times1, 1024\end{array}\right]\times 6$
conv5	7×7	$\left[\begin{array}{l}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times 3$	$\left[\begin{array}{l}1\times1, 1024\\3\times3, 1024, C{=}32\\1\times1, 2048\end{array}\right]\times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^{6}$	$\mathbf{25.0}\times10^{6}$
FLOPs		$\mathbf{4.1}\times10^{9}$	$\mathbf{4.2}\times10^{9}$

Table 1 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431

ResNeXt

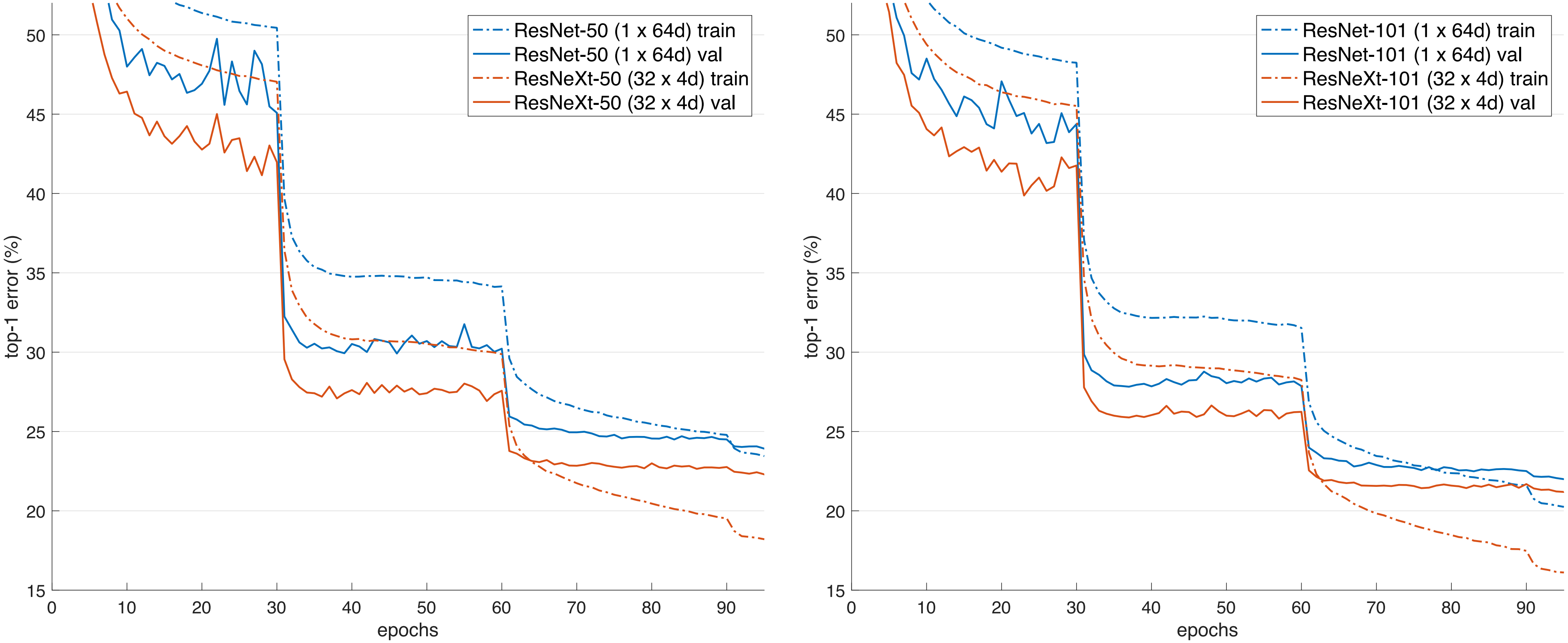
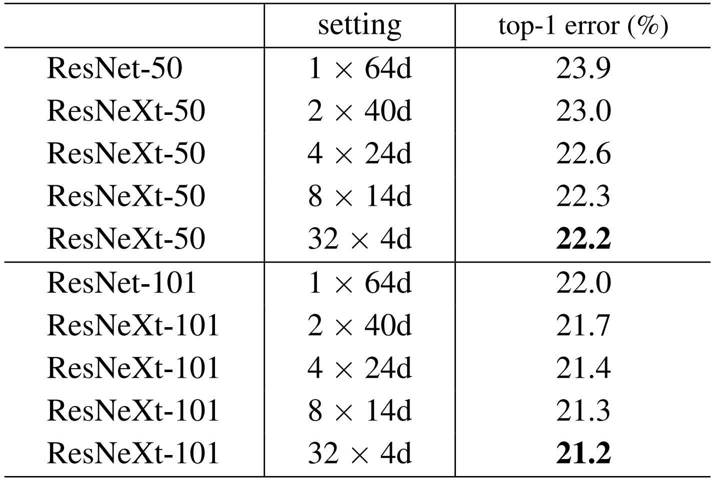


Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (∼4.1 billion FLOPs, ∼25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (∼7.8 billion FLOPs, ∼44 million parameters).

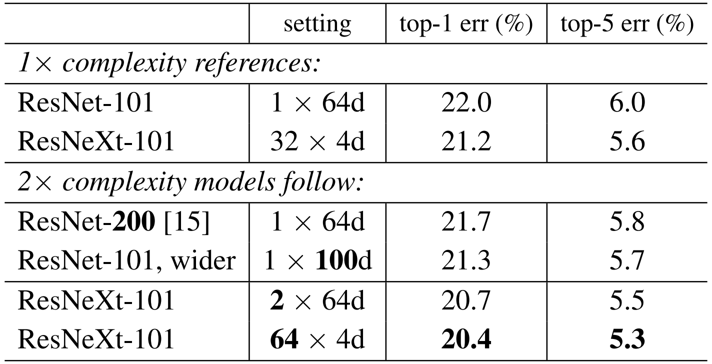
Figure 5 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431

ResNeXt



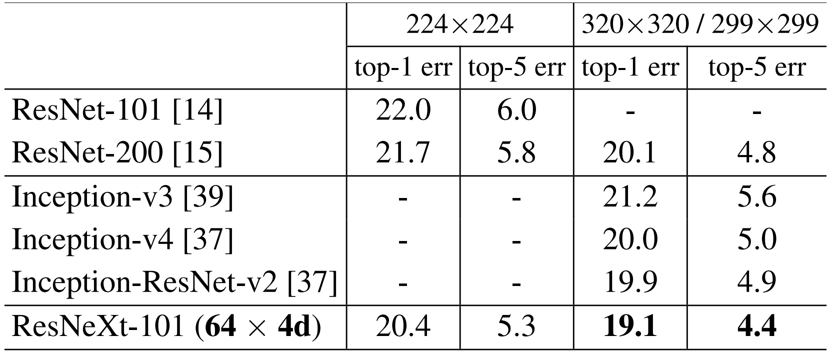
	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431



	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431



	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [14]	22.0	6.0	-	-
ResNet-200 [15]	21.7	5.8	20.1	4.8
Inception-v3 [39]	-	-	21.2	5.6
Inception-v4 [37]	-	-	20.0	5.0
Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (**64 × 4d**)	20.4	5.3	**19.1**	**4.4**

Table 5 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431

Deep Networks with Stochastic Depth

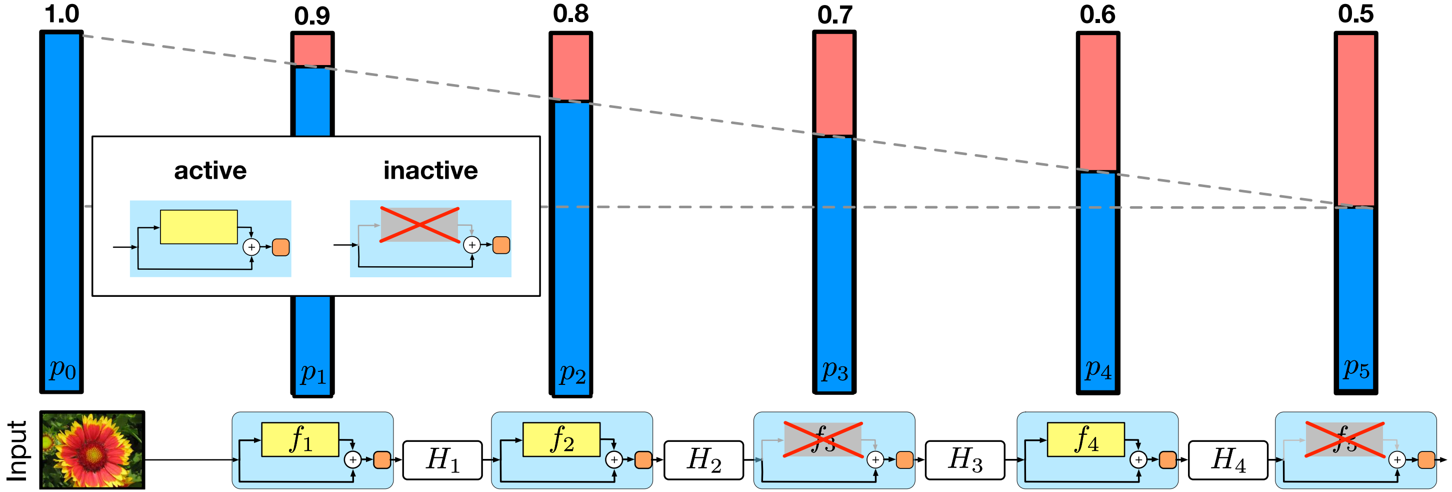
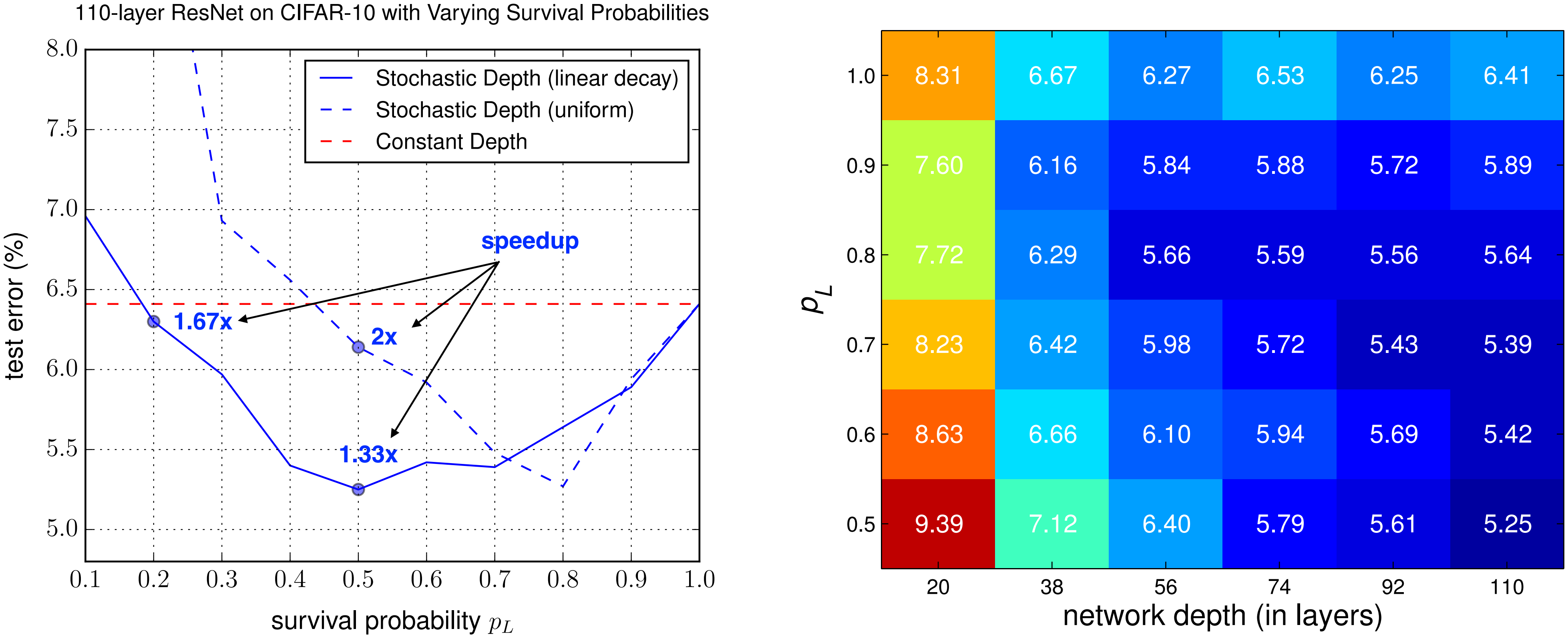


Figure 2 of "Deep Networks with Stochastic Depth", https://arxiv.org/abs/1603.09382

We drop a whole block (but not the residual connection) with probability $1 - p_l$. During inference, we multiply the block output by p_l to compensate; or we can use the alternative approach like in regular dropout, where we divide the activation by p_l during training only.

All p_l can be set to a constant, but more effective approach is to utilize a simple linear decay $p_l = 1 - \frac{l}{L}(1 - p_L)$, where p_L is the final probability of the last layer, motivated by the intuition that the initial blocks extract low-level features utilized by the later layers, and should therefore be present.

Deep Networks with Stochastic Depth



p_L	20	38	56	74	92	110
1.0	8.31	6.67	6.27	6.53	6.25	6.41
0.9	7.60	6.16	5.84	5.88	5.72	5.89
0.8	7.72	6.29	5.66	5.59	5.56	5.64
0.7	8.23	6.42	5.98	5.72	5.43	5.39
0.6	8.63	6.66	6.10	5.94	5.69	5.42
0.5	9.39	7.12	6.40	5.79	5.61	5.25

Figure 8 of "Deep Networks with Stochastic Depth", https://arxiv.org/abs/1603.09382

According to the ablation experiments, linear decay with $p_L = 0.5$ was selected.

Deep Networks with Stochastic Depth

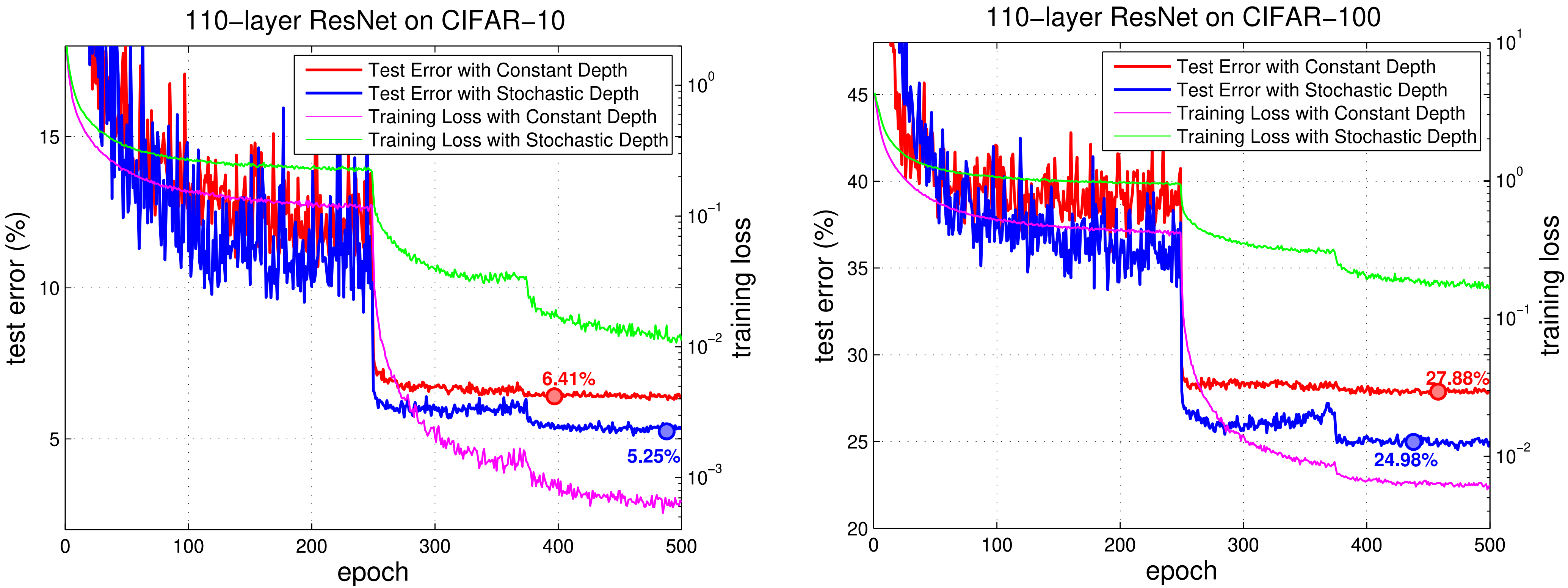


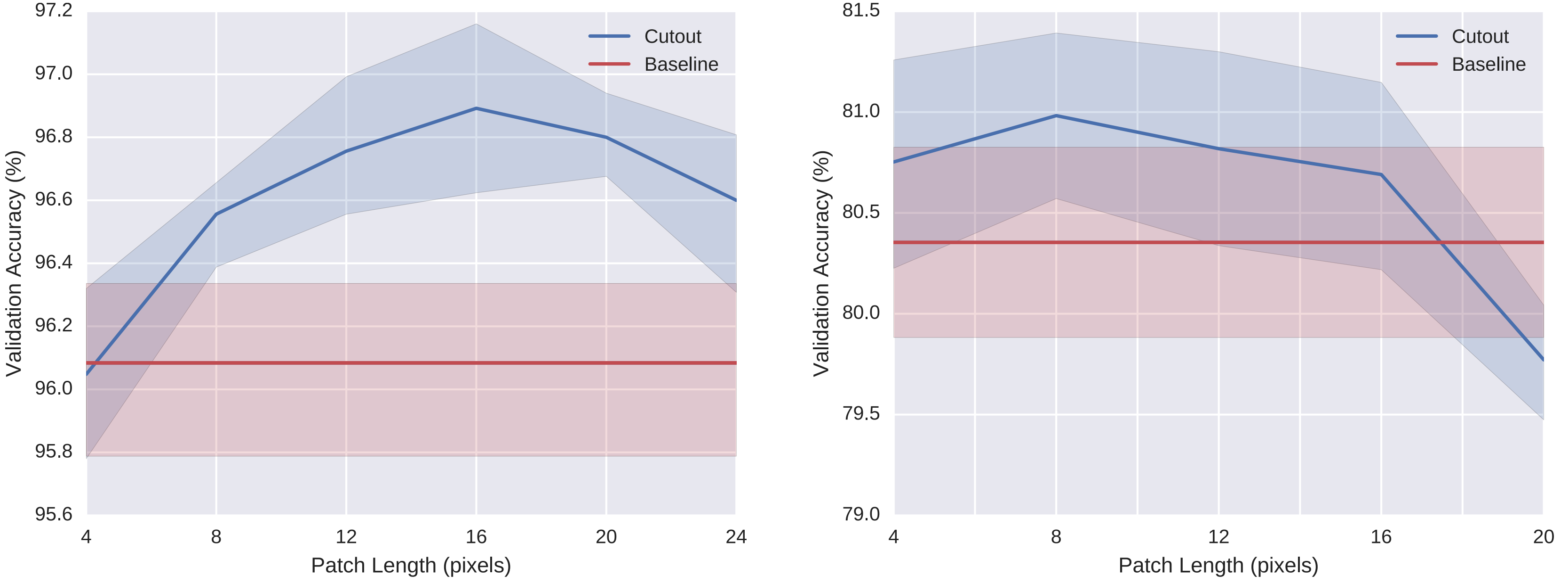
Figure 3 of "Deep Networks with Stochastic Depth", https://arxiv.org/abs/1603.09382

Cutout

Figure 1 of "Improved Regularization of Convolutional Neural Networks with Cutout", https://arxiv.org/abs/1708.04552

Drop 16×16 square in the input image, with randomly chosen center. The pixels are replaced by their mean value from the dataset.

Cutout



(a) CIFAR-10 (b) CIFAR-100

Figure 3 of "Improved Regularization of Convolutional Neural Networks with Cutout", https://arxiv.org/abs/1708.04552

Method	C10	C10+	C100	C100+	SVHN
ResNet18 [5]	10.63 ± 0.26	4.72 ± 0.21	36.68 ± 0.57	22.46 ± 0.31	-
ResNet18 + cutout	9.31 ± 0.18	3.99 ± 0.13	34.98 ± 0.29	21.96 ± 0.24	-
WideResNet [22]	6.97 ± 0.22	3.87 ± 0.08	26.06 ± 0.22	18.8 ± 0.08	1.60 ± 0.05
WideResNet + cutout	$\mathbf{5.54 \pm 0.08}$	3.08 ± 0.16	$\mathbf{23.94 \pm 0.15}$	18.41 ± 0.27	$\mathbf{1.30 \pm 0.03}$
Shake-shake regularization [4]	-	2.86	-	15.85	-
Shake-shake regularization + cutout	-	$\mathbf{2.56 \pm 0.07}$	-	$\mathbf{15.20 \pm 0.21}$	-

Table 1 of "Improved Regularization of Convolutional Neural Networks with Cutout", https://arxiv.org/abs/1708.04552

DropBlock

Dropout drops individual values, SpatialDropout drops whole channels, DropBlock drops rectangular areas in all channels at the same time.

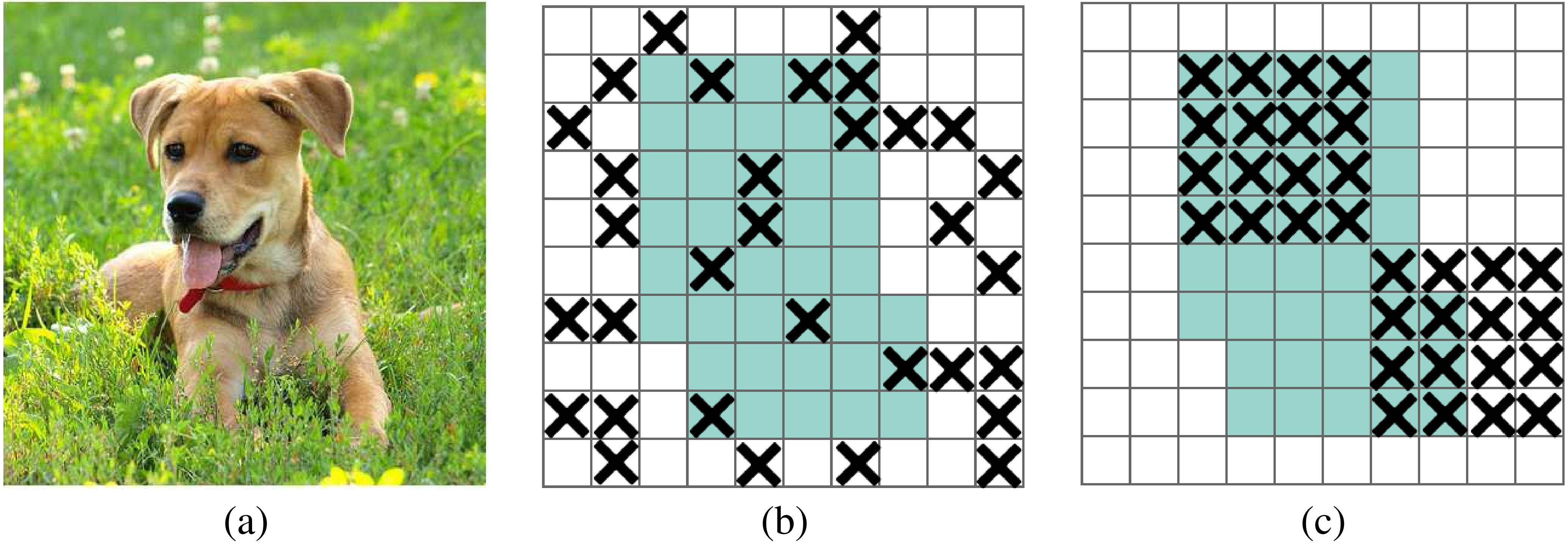


Figure 1 of "DropBlock: A regularization method for convolutional networks", https://arxiv.org/abs/1810.12890

The authors mention that they also tried applying DropBlock in every channel separately, but that masking all channels equally “tends to work better in our experiments”.

DropBlock

Algorithm 1 DropBlock

```
Input: output activations of a layer ($A$), $block\_size$, $\gamma$, $mode$
if $mode$ == $Inference$ then
    return $A$
end if
Randomly sample mask $M$: $M_{i,j} \sim Bernoulli(\gamma)$
For each zero position $M_{i,j}$, create a spatial square mask with the center being $M_{i,j}$, the width, height being $block\_size$ and set all the values of $M$ in the square to be zero (see Figure 2).
Apply the mask: $A = A \times M$
Normalize the features: $A = A \times \mathbf{count}(M)/\mathbf{count\_ones}(M)$
```

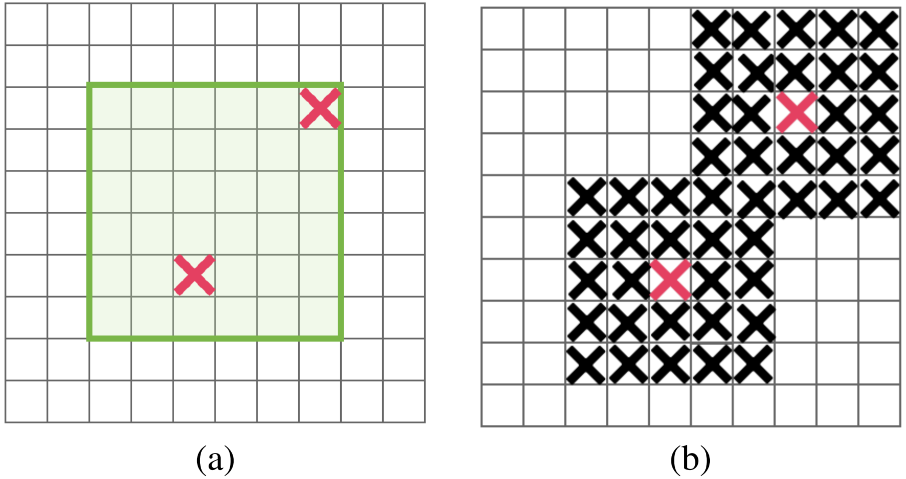


Figure 2 of "DropBlock: A regularization method for convolutional networks", https://arxiv.org/abs/1810.12890

DropBlock

The authors have chosen *block size=7* and also employ linear schedule of the *keep probability*, which starts at 1 and linearly decays until the target value is reached at the end of training.

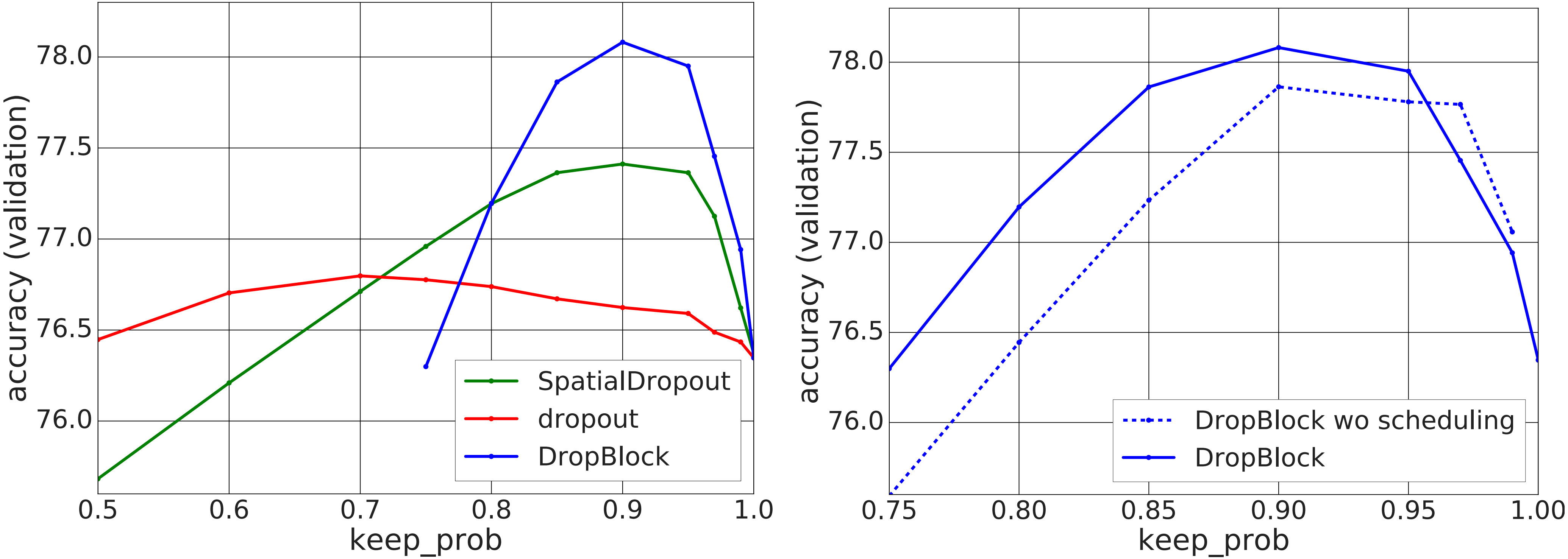


Figure 3 of "DropBlock: A regularization method for convolutional networks", https://arxiv.org/abs/1810.12890

DropBlock

Model	top-1(%)	top-5(%)
ResNet-50	76.51 ± 0.07	93.20 ± 0.05
ResNet-50 + dropout (kp=0.7) [1]	76.80 ± 0.04	93.41 ± 0.04
ResNet-50 + DropPath (kp=0.9) [17]	77.10 ± 0.08	93.50 ± 0.05
ResNet-50 + SpatialDropout (kp=0.9) [20]	77.41 ± 0.04	93.74 ± 0.02
ResNet-50 + Cutout [23]	76.52 ± 0.07	93.21 ± 0.04
ResNet-50 + AutoAugment [27]	77.63	93.82
ResNet-50 + label smoothing (0.1) [28]	77.17 ±0.05	93.45 ±0.03
ResNet-50 + DropBlock, (kp=0.9)	78.13 ± 0.05	94.02 ± 0.02
ResNet-50 + DropBlock (kp=0.9) + label smoothing (0.1)	78.35 ± 0.05	94.15 ± 0.03

Table 1 of "DropBlock: A regularization method for convolutional networks", https://arxiv.org/abs/1810.12890

The results are averages of three runs.

Squeeze and Excitation

The ILSVRC 2017 winner was SENet, *Squeeze and Excitation Network*, augmenting existing architectures by a **squeeze and excitation** block, which learns to emphasise informative channels and suppress less useful ones according to global information.

- **squeeze (global information embedding)** computes the average value of every channel;
- **excitation (adaptive recalibration)** computes a weight for every channel using a sigmoid activation function and multiplies the corresponding channel with it. To not increase the number of parameters too much (by C^2), an additional small hidden layer with $C/16$ neurons is employed (to reduce the additional parameters to $C^2/8$ only).

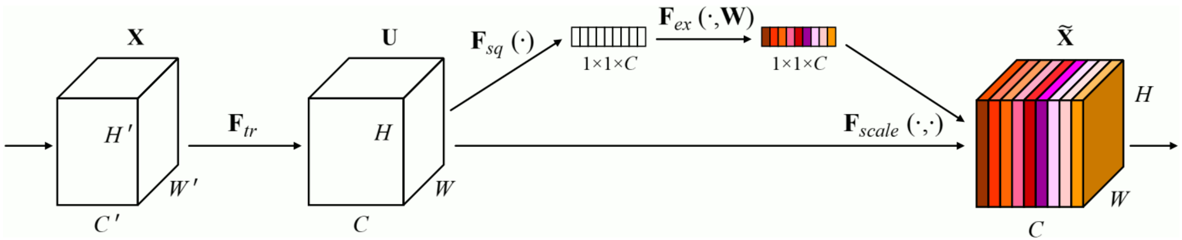


Figure 1 of "Squeeze-and-Excitation Networks", https://arxiv.org/abs/1709.01507

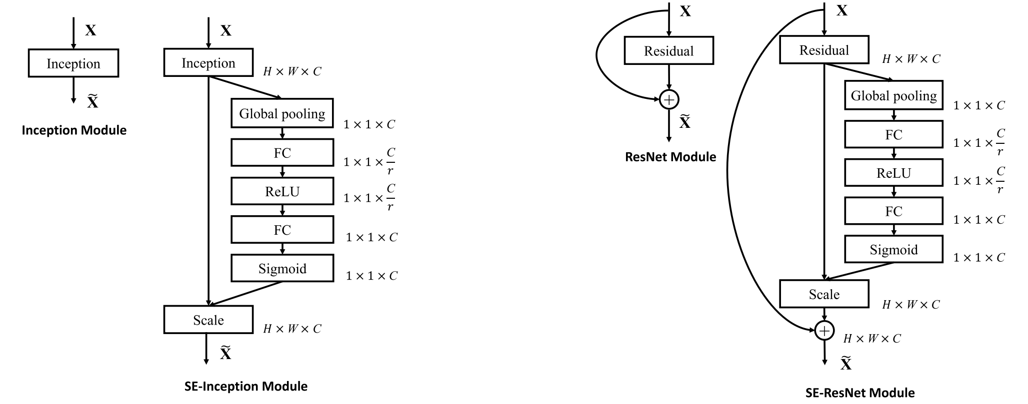


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

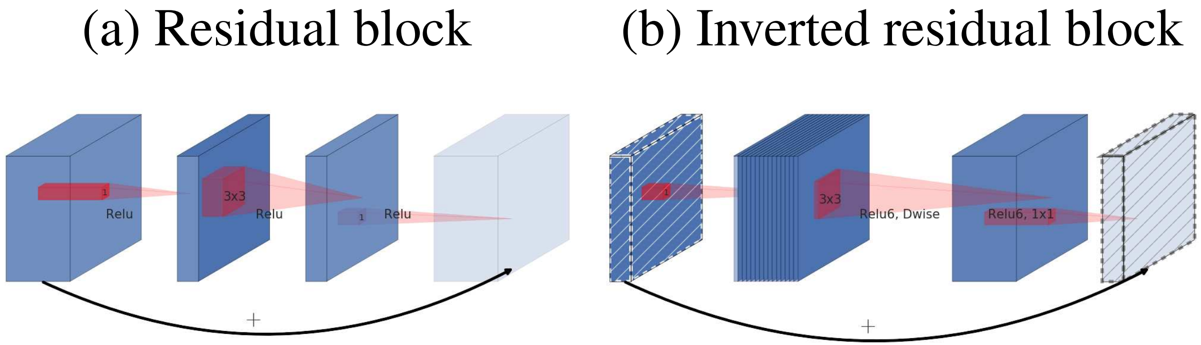
Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

Figure 2 of "Squeeze-and-Excitation Networks", https://arxiv.org/abs/1709.01507

Mobile Inverted Bottleneck Convolution

When designing convolutional neural networks for mobile phones, the following **mobile inverted bottleneck** block was proposed.

- Regular convolution is replaced by **separable convolution**, which consists of
  - a **depthwise separable** convolution (for example 3×3) acting on each channel separately (which reduces time and space complexity of a regular convolution by a factor equal to the number of channels);
  - a **pointwise** 1×1 convolution acting on each position independently (which reduces time and space complexity of a regular convolution by a factor of $3 \cdot 3$).


(a) Residual block

(b) Inverted residual block

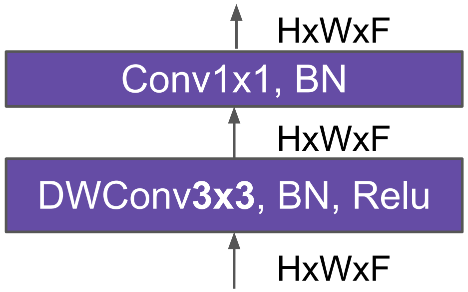
Figure 3 of "MobileNetV2: Inverted Residuals and Linear Bottlenecks", https://arxiv.org/abs/1801.04381

- The residual connections connect bottlenecks (layers with least channels).
- There is no nonlinear activation on the bottlenecks (it would lead to loss of information given small capacity of bottlenecks).

Mobile Inverted Bottleneck Convolution

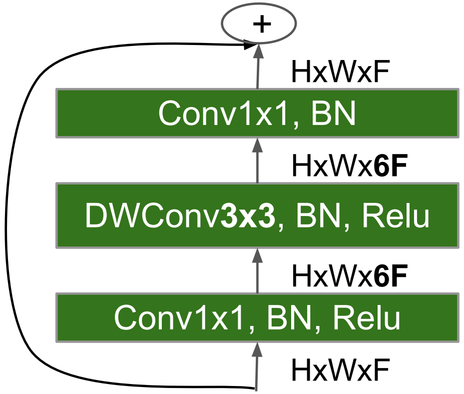
The mobile inverted bottleneck convolution is denoted for example as *MBConv6 k3x3*, where the 6 denotes expansion factor after the bottleneck and 3×3 is the kernel size of the separable convolution.

Furthermore, the mobile inverted bottleneck convolution can be augmented with squeeze and excitation blocks.


HxWxF
Conv1x1, BN
HxWxF
DWConv**3x3**, BN, Relu
HxWxF

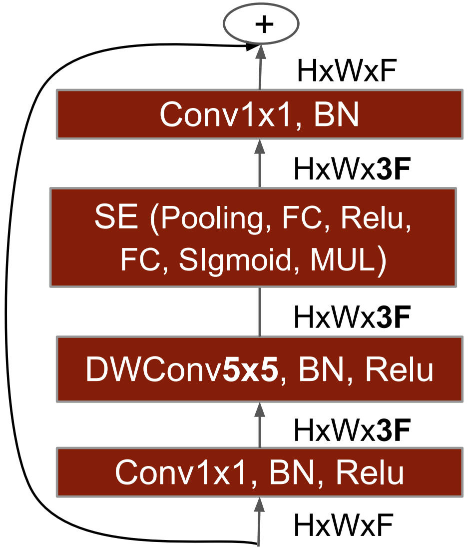
(d) SepConv (k3x3)

Figure 7 of "MnasNet: Platform-Aware Neural Architecture Search for Mobile", https://arxiv.org/abs/1807.11626


+
HxWxF
Conv1x1, BN
HxWx**6F**
DWConv**3x3**, BN, Relu
HxWx**6F**
Conv1x1, BN, Relu
HxWxF

(c) MBConv6 (k3x3)

Figure 7 of "MnasNet: Platform-Aware Neural Architecture Search for Mobile", https://arxiv.org/abs/1807.11626


+
HxWxF
Conv1x1, BN
HxWx**3F**
SE (Pooling, FC, Relu, FC, SIgmoid, MUL)
HxWx**3F**
DWConv**5x5**, BN, Relu
HxWx**3F**
Conv1x1, BN, Relu
HxWxF

(b) MBConv3 (k5x5)

Figure 7 of "MnasNet: Platform-Aware Neural Architecture Search for Mobile", https://arxiv.org/abs/1807.11626

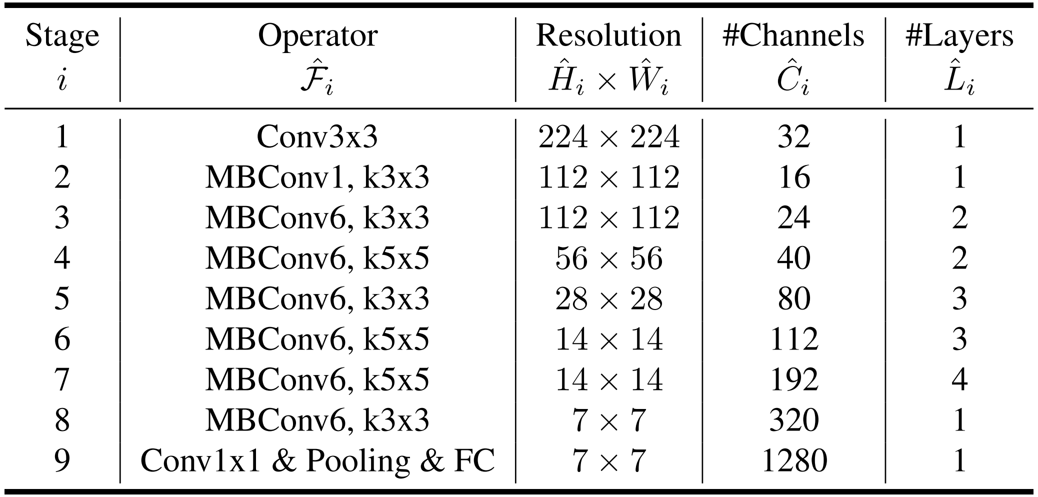
EfficientNet

In 2019, very performant and efficient convolutional architecture **EfficientNet** was proposed.

The EfficientNet architecture was created using a multi-objective neural architecture search that optimized both accuracy and computation complexity.

The resulting network is denoted as **EfficientNet-B0** baseline network.

It was trained using RMSProp with $\beta{=}0.9$ and momentum 0.9, weight decay 1e-5, and initial learning rate 0.256 decayed by 0.97 every 2.4 epochs. Dropout with dropout rate 0.2 is used on the last layer, stochastic depth with survival probability 0.8 is employed, and $\operatorname{swish}(\boldsymbol x) \stackrel{\text{def}}{=} \boldsymbol x \cdot \sigma(\boldsymbol x)$ activation function is utilized.



Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 1 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946

EfficientNet – Compound Scaling

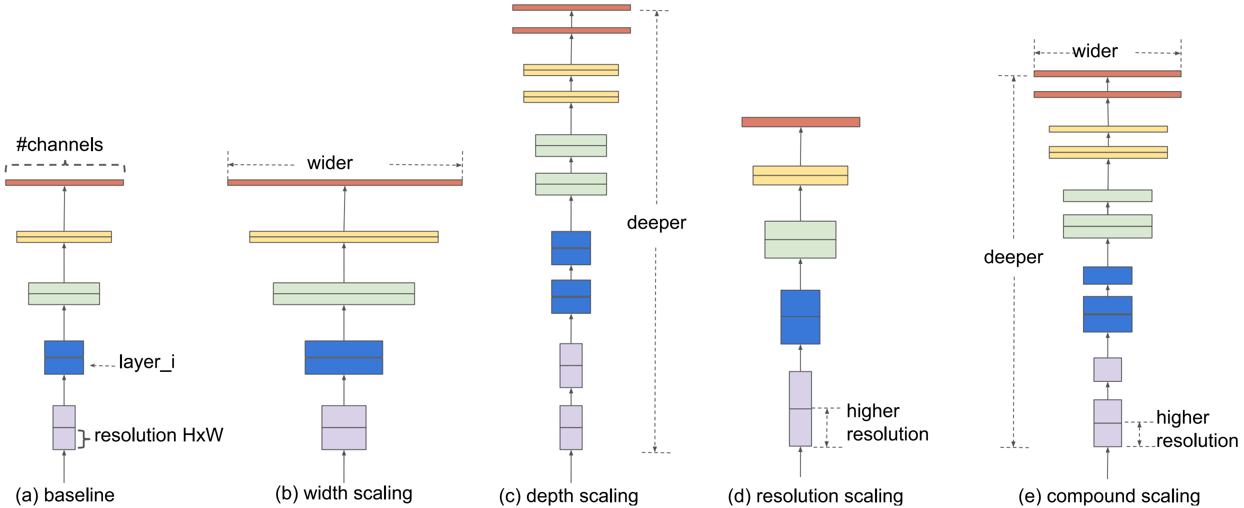


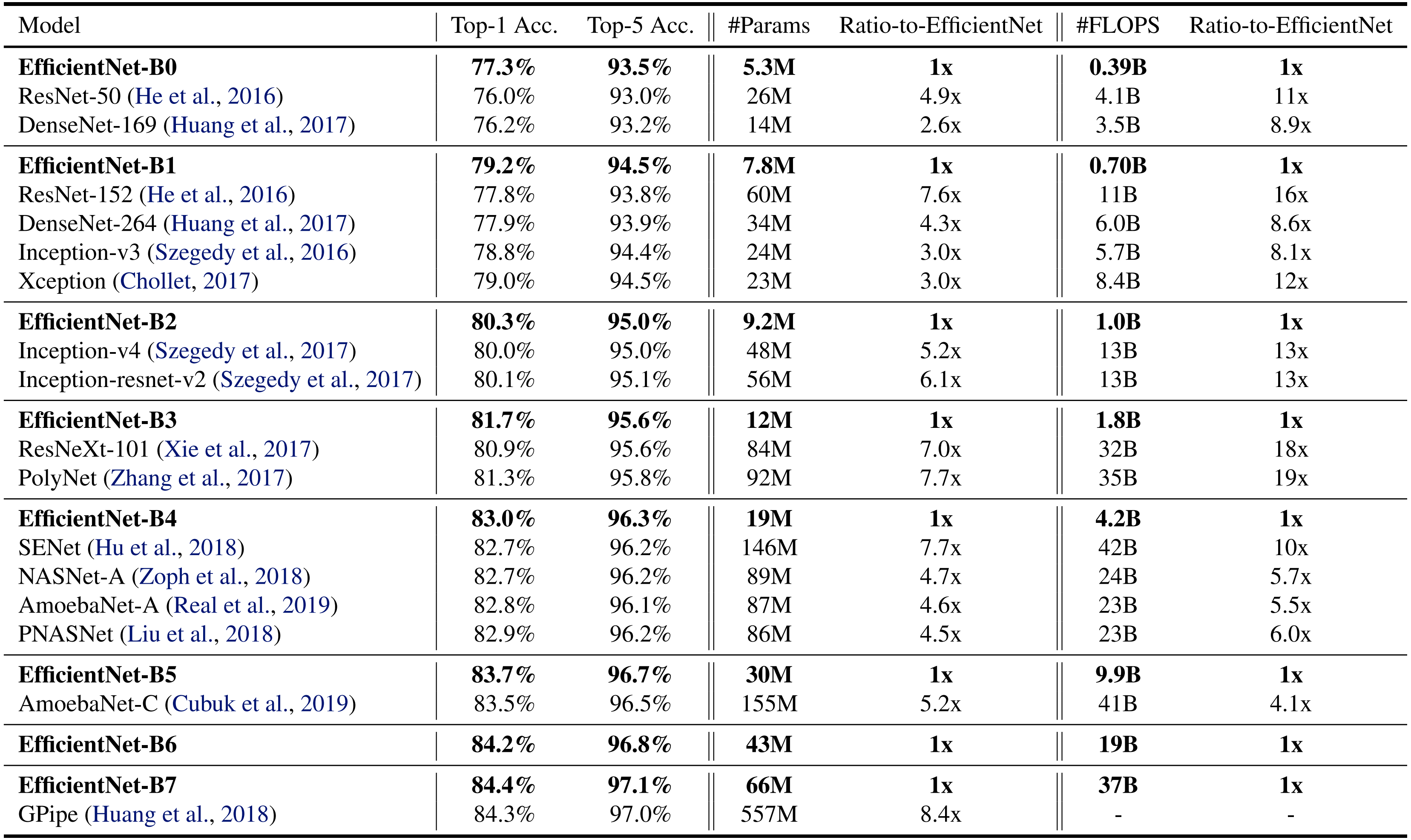
Figure 2 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946

To effectively scale the network, the authors propose a simultaneous increase of three qualities:

- **width**, which is the number of channels;
- **depth**, which is the number of layers;
- **resolution**, which is the input image resolution.

By a grid search on a network with double computation complexity, the best trade-off of scaling width by 1.1, depth by 1.2 and resolution by 1.15 was found ($1.1^2 \cdot 1.2 \cdot 1.15^2 \approx 2$).

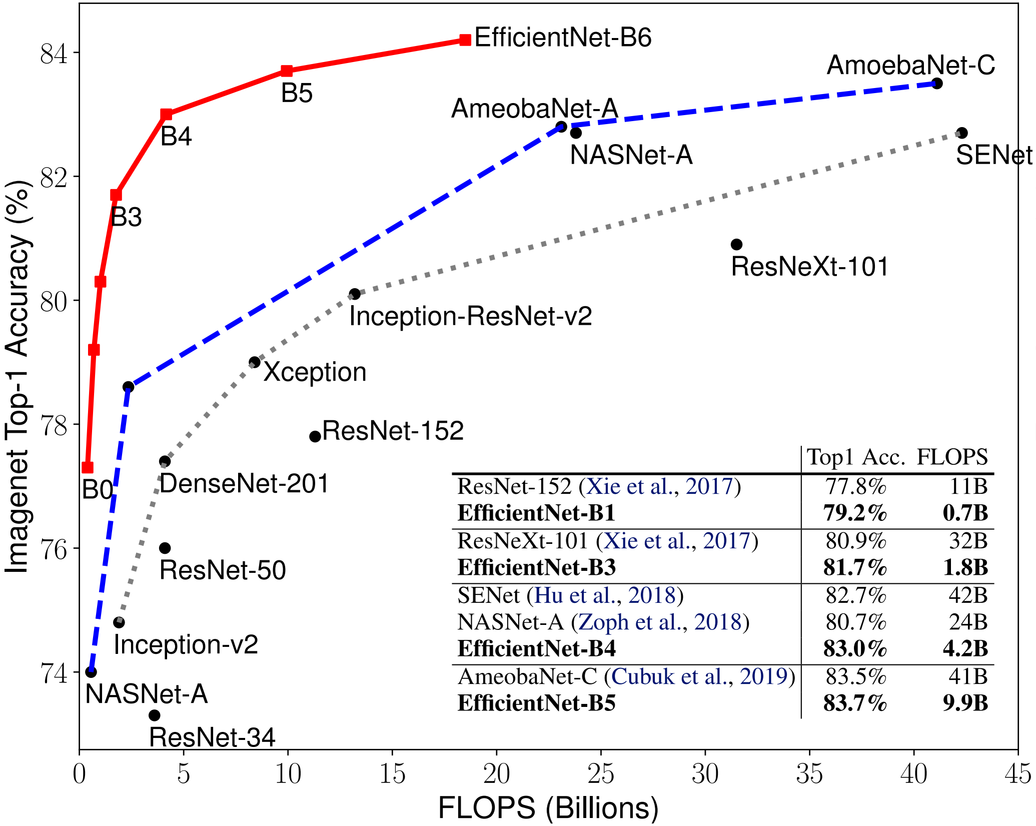
EfficientNet – Results



Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B0	**77.3%**	**93.5%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.2%**	**94.5%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.3%**	**95.0%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.7%**	**95.6%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**83.0%**	**96.3%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.7%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.2%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

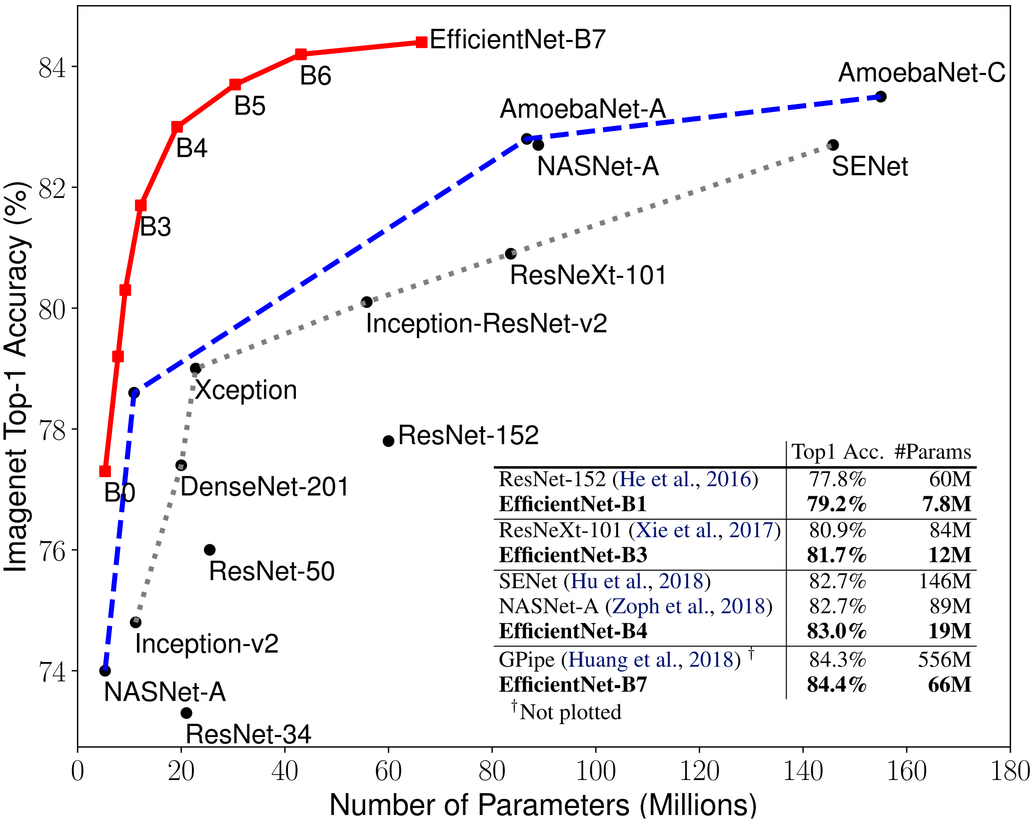
Table 2 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946

EfficientNet – Results



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.2%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.7%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**83.0%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.7%**	**9.9B**

Figure 5 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

† Not plotted

Figure 1 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946

EfficientNetV2

In April 2021, an improved version of EfficientNet, **EfficientNetV2**, was published. It is currently one of the best available CNNs for image recognition.

The improvements between EfficientNet and EfficientNetV2 are not large:

- The separable convolutions have fewer parameters, but are slow to execute on modern hardware. The authors therefore “fuse” the 1×1 convolution and a 3×3 depthwise convolution into a regular convolution, which has more parameters and require more computation, but is in fact executed faster.
- Very large images make training very slow. EfficientNetV2 avoids aggressively scaling the image sizes, limiting maximum image size to 480.
- The authors utilize progressive training – the image size is gradually increased during training, as is the regularization strength (dropout, mixup, RandAugment magnitude).

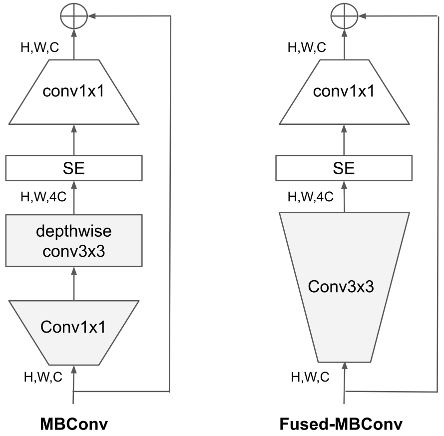


Figure 2. Structure of MBConv and Fused-MBConv.

Figure 2 of "EfficientNetV2: Smaller Models and Faster Training", https://arxiv.org/abs/2104.00298

EfficientNetV2 – Architecture

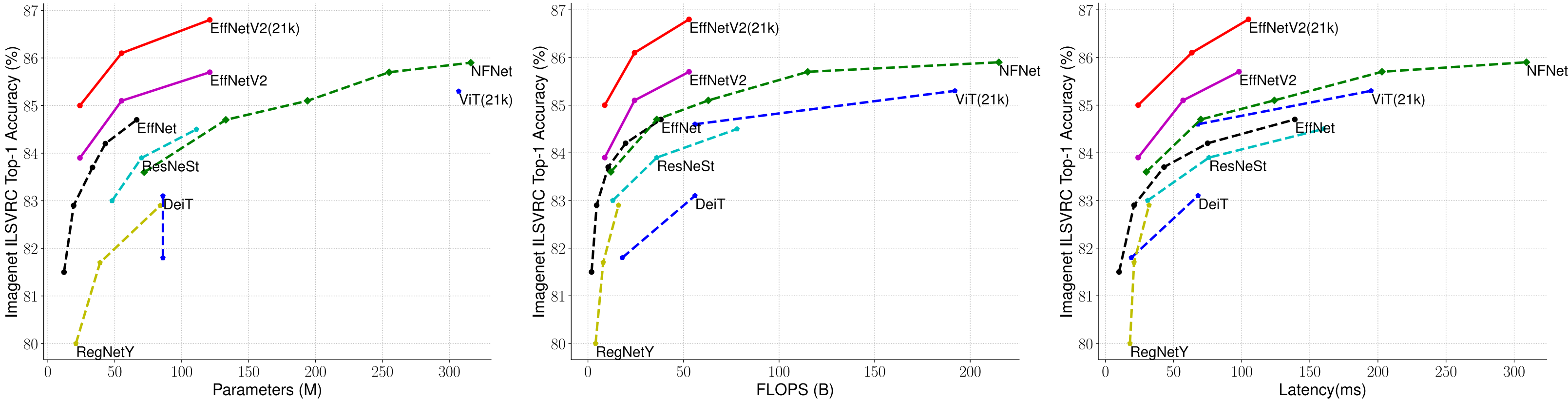
Table 4. EfficientNetV2-S architecture – MBConv and Fused-MBConv blocks are described in Figure 2.

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Table 4 of "EfficientNetV2: Smaller Models and Faster Training", https://arxiv.org/abs/2104.00298

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Table 1 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946



(a) Parameters (b) FLOPs (c) GPU V100 Latency (batch 16)

Figure 5. **Model Size, FLOPs, and Inference Latency** – Latency is measured with batch size 16 on V100 GPU. `21k` denotes pretrained on ImageNet21k images, others are just trained on ImageNet ILSVRC2012. Our EfficientNetV2 has slightly better parameter efficiency with EfficientNet, but runs 3x faster for inference.

Figure 5 of "EfficientNetV2: Smaller Models and Faster Training", https://arxiv.org/abs/2104.00298

EfficientNetV2 – Results

	Model	Top-1 Acc.	Params	FLOPs	Infer-time(ms)	Train-time (hours)
ConvNets & Hybrid	EfficientNet-B3 (Tan & Le, 2019a)	81.5%	12M	1.9B	19	10
	EfficientNet-B4 (Tan & Le, 2019a)	82.9%	19M	4.2B	30	21
	EfficientNet-B5 (Tan & Le, 2019a)	83.7%	30M	10B	60	43
	EfficientNet-B6 (Tan & Le, 2019a)	84.3%	43M	19B	97	75
	EfficientNet-B7 (Tan & Le, 2019a)	84.7%	66M	38B	170	139
	RegNetY-8GF (Radosavovic et al., 2020)	81.7%	39M	8B	21	-
	RegNetY-16GF (Radosavovic et al., 2020)	82.9%	84M	16B	32	-
	ResNeSt-101 (Zhang et al., 2020)	83.0%	48M	13B	31	-
	ResNeSt-200 (Zhang et al., 2020)	83.9%	70M	36B	76	-
	ResNeSt-269 (Zhang et al., 2020)	84.5%	111M	78B	160	-
	TResNet-L (Ridnik et al., 2020)	83.8%	56M	-	45	-
	TResNet-XL (Ridnik et al., 2020)	84.3%	78M	-	66	-
	EfficientNet-X (Li et al., 2021)	84.7%	73M	91B	-	-
	NFNet-F0 (Brock et al., 2021)	83.6%	72M	12B	30	8.9
	NFNet-F1 (Brock et al., 2021)	84.7%	133M	36B	70	20
	NFNet-F2 (Brock et al., 2021)	85.1%	194M	63B	124	36
	NFNet-F3 (Brock et al., 2021)	85.7%	255M	115B	203	65
	NFNet-F4 (Brock et al., 2021)	85.9%	316M	215B	309	126
	LambdaResNet-420-hybrid (Bello, 2021)	84.9%	125M	-	-	67
	BotNet-T7-hybrid (Srinivas et al., 2021)	84.7%	75M	46B	-	95
	BiT-M-R152x2 (21k) (Kolesnikov et al., 2020)	85.2%	236M	135B	500	-
Vision Transformers	ViT-B/32 (Dosovitskiy et al., 2021)	73.4%	88M	13B	13	-
	ViT-B/16 (Dosovitskiy et al., 2021)	74.9%	87M	56B	68	-
	DeiT-B (ViT+reg) (Touvron et al., 2021)	81.8%	86M	18B	19	-
	DeiT-B-384 (ViT+reg) (Touvron et al., 2021)	83.1%	86M	56B	68	-
	T2T-ViT-19 (Yuan et al., 2021)	81.4%	39M	8.4B	-	-
	T2T-ViT-24 (Yuan et al., 2021)	82.2%	64M	13B	-	-
	ViT-B/16 (21k) (Dosovitskiy et al., 2021)	84.6%	87M	56B	68	-
	ViT-L/16 (21k) (Dosovitskiy et al., 2021)	85.3%	304M	192B	195	172
ConvNets (ours)	**EfficientNetV2-S**	83.9%	22M	8.8B	24	7.1
	EfficientNetV2-M	85.1%	54M	24B	57	13
	EfficientNetV2-L	85.7%	120M	53B	98	24
	EfficientNetV2-S (21k)	84.9%	22M	8.8B	24	9.0
	EfficientNetV2-M (21k)	86.2%	54M	24B	57	15
	EfficientNetV2-L (21k)	86.8%	120M	53B	98	26
	EfficientNetV2-XL (21k)	87.3%	208M	94B	-	45

Table 7 of "EfficientNetV2: Smaller Models and Faster Training", https://arxiv.org/abs/2104.00298

Transfer Learning

In many situations, we would like to utilize a model trained on a different dataset – generally, this cross-dataset usage is called **transfer learning**.

In image processing, models trained on ImageNet are frequently used as general **feature extraction models**.

The easiest scenario is to take a ImageNet model, drop the last classification layer, and use the result of the global average pooling as image features. The ImageNet model is not modified during training.

For efficiency, we may precompute the image features **once** and reuse it later many times.

Transfer Learning – Finetuning

After we have successfully trained a network employing an ImageNet model, we may improve performance further by **finetuning** – training the full network including the ImageNet model, allowing the feature extraction to adapt to the current dataset.

- The layers after the ImageNet models **should** be already trained to convergence.
- Usually a smaller learning rate is necessary, because the original model probably finished training with a very small learning rate. A good starting point is one tenth of the original starting learning rate (therefore, 0.0001 for Adam).
- We have to think about batch normalization, data augmentation or other regularization techniques.

Transposed Convolution

So far, the convolution operation produces either an output of the same size, or it produced a smaller one if stride was larger than one.

In order to come up with **upscaling convolution**, we start by considering how a gradient is backpropagated through a fully connected layer and a regular convolution.

In a fully connected layer without activation:

- during the forward pass, input $\boldsymbol{x}$ is multiplied by the weight matrix $\boldsymbol{W}$ as $\boldsymbol{x}\boldsymbol{W}$;
- during the backward pass, the gradient $\boldsymbol{g}$ is multiplied by the *transposed* weight matrix as $\boldsymbol{g}\boldsymbol{W}^T$.

Transposed Convolution

Analogously, in a convolutional layer without activation:

- during the forward pass, the cross-correlation operation between input I and kernel K is performed as

$$(\mathsf{K} \star \mathsf{I})_{i,j,o} = \sum_{m,n,c} \mathsf{I}_{i\cdot S+m,j\cdot S+n,c}\mathsf{K}_{m,n,c,o};$$

- during the backward pass, we obtain $\mathsf{G}_{i,j,o} = \frac{\partial L}{\partial (\mathsf{K}\star\mathsf{I})_{i,j,o}}$ and we need to backpropagate it to obtain $\frac{\partial L}{\partial \mathsf{I}_{i,j,c}}$. It is not difficult to show that

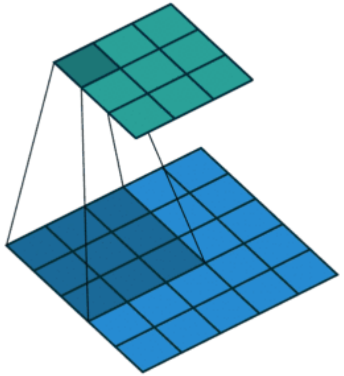
$$\frac{\partial L}{\partial \mathsf{I}_{i,j,c}} = \sum_{\substack{i',m\\ i'\cdot S+m=i}} \sum_{\substack{j',n\\ j'\cdot S+n=j}} \sum_{o} \mathsf{G}_{i',j',o}\mathsf{K}_{m,n,c,o}.$$

This operation is called **transposed** or **upscaling** convolution and stride greater than one makes the output larger, not smaller.

Transposed Convolution Animation

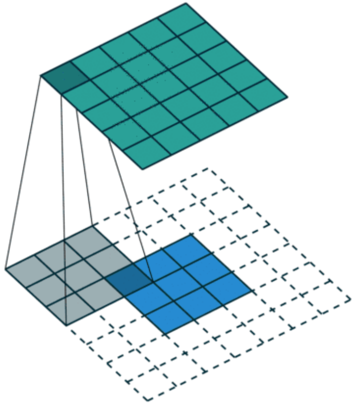
Illustration of the padding schemes and different strides for a 3×3 kernel.

- **valid**, stride=1, regular:



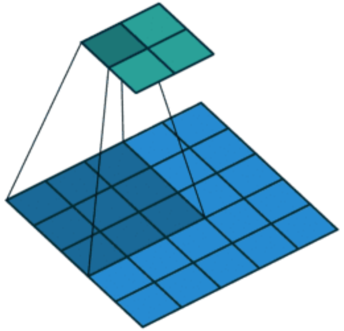
https://github.com/vdumoulin/conv_arithmetic

transposed:



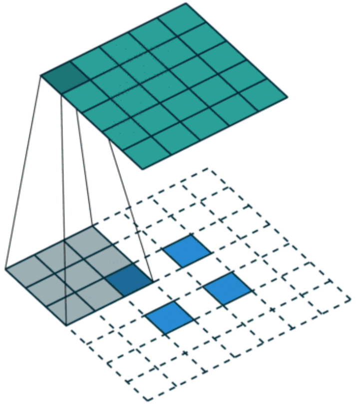
https://github.com/vdumoulin/conv_arithmetic

- **valid**, stride=2, regular:



https://github.com/vdumoulin/conv_arithmetic

transposed:

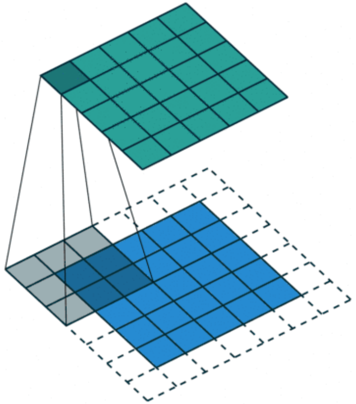


https://github.com/vdumoulin/conv_arithmetic

Transposed Convolution Animation

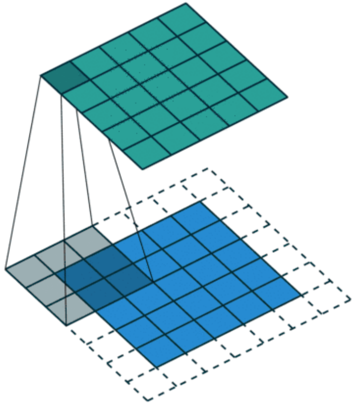
Illustration of the padding schemes and different strides for a 3×3 kernel.

- **same**, stride=1, regular:



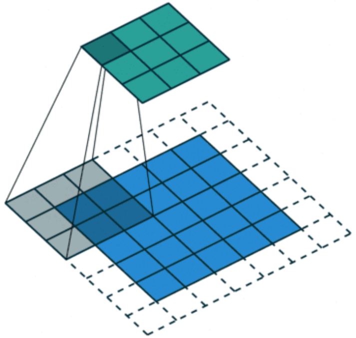
https://github.com/vdumoulin/conv_arithmetic

transposed:



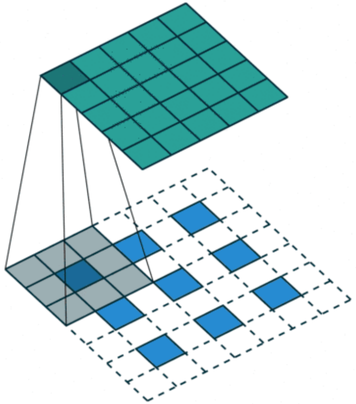
https://github.com/vdumoulin/conv_arithmetic

- **same**, stride=2, regular:



https://github.com/vdumoulin/conv_arithmetic

transposed:

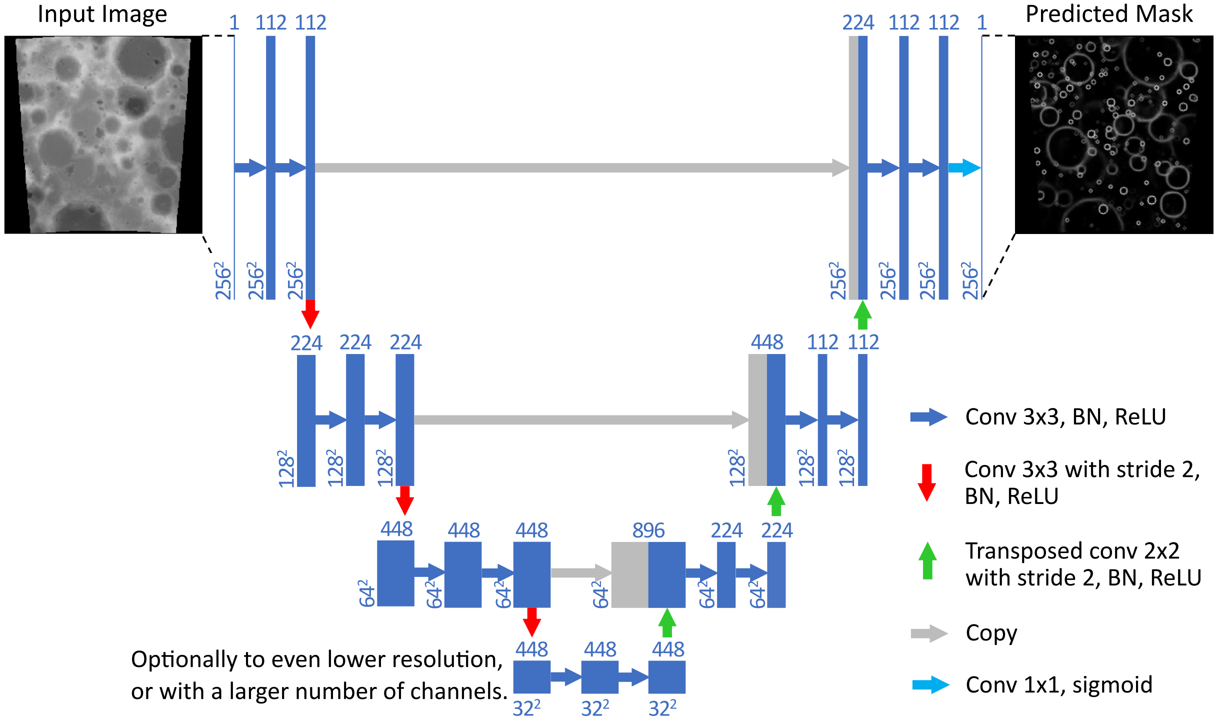


https://github.com/vdumoulin/conv_arithmetic

Transposed Convolution

Given that the transposed convolution must be implemented for efficient backpropagation of a regular convolution, it is usually available for direct usage in neural network frameworks.

It is frequently used to perform upscaling of an image, as an “inverse” operation to pooling (or convolution with stride > 1), which is useful for example in *image segmentation*:



Modification of Figure 2 of "Lunar Crater Identification via Deep Learning", https://arxiv.org/abs/1803.02192