Convolutional Neural Networks II

Milan Straka

March 13, 2023
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 $3 \times 3$ 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.
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
Figure 2. Residual learning: a building block.

Figure 2 of "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385
Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on $56 \times 56$ feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.
<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td></td>
<td></td>
<td>7×7, 64, stride 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2_x</td>
<td>56×56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td>[1×1, 64]</td>
<td>[1×1, 64]</td>
<td>[1×1, 64]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3×3 max pool, stride 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3_x</td>
<td>28×28</td>
<td>1×1, 128</td>
<td>3×3, 128</td>
<td>1×1, 128</td>
<td>3×3, 128</td>
<td>1×1, 128</td>
</tr>
<tr>
<td></td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
<td></td>
</tr>
<tr>
<td>conv4_x</td>
<td>14×14</td>
<td>1×1, 256</td>
<td>3×3, 256</td>
<td>1×1, 256</td>
<td>3×3, 256</td>
<td>1×1, 256</td>
</tr>
<tr>
<td></td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[1×1, 1024]</td>
<td>[1×1, 1024]</td>
<td>[1×1, 1024]</td>
<td></td>
</tr>
<tr>
<td>conv5_x</td>
<td>7×7</td>
<td>1×1, 512</td>
<td>3×3, 512</td>
<td>1×1, 512</td>
<td>3×3, 512</td>
<td>1×1, 512</td>
</tr>
<tr>
<td></td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[1×1, 2048]</td>
<td>[1×1, 2048]</td>
<td>[1×1, 2048]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>average pool, 1000-d fc, softmax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.8×10⁹</td>
<td>3.6×10⁹</td>
<td>3.8×10⁹</td>
<td>7.6×10⁹</td>
<td>11.3×10⁹</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 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 $1 \times 1$ 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% ILSVRC top-5 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% ILSVRC top-5 error)

Figure 1 of "Visualizing the Loss Landscape of Neural Nets", https://arxiv.org/abs/1712.09913
ResNet – 2015 (3.6% ILSVRC top-5 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 the error plateaus”
  - 600k training iterations are used (120 epochs, each containing 1.281M images)
  - according to one graph (and to their later paper) they decay at 25% and 50% of the training, so after epochs 30 and 60
    - other concurrent papers also use exponential decay or 25%-50%-75%
- 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 $224 \times 224$ 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 $\{224, 256, 384, 480, 640\}$
The ResNet-34 B uses the $1 \times 1$ 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.
The authors of ResNet published an ablation study several months after the original paper.

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

Table 1. Classification error on the CIFAR-10 test set using ResNet-110 [1], with different types of shortcut connections applied to all Residual Units. We report “fail” when the test error is higher than 20%.

<table>
<thead>
<tr>
<th>case</th>
<th>Fig.</th>
<th>on shortcut</th>
<th>on F</th>
<th>error (%)</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>original [1]</td>
<td>Fig. 2(a)</td>
<td>1</td>
<td>1</td>
<td>6.61</td>
<td>This is a plain net</td>
</tr>
<tr>
<td>constant scaling</td>
<td>Fig. 2(b)</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>frozen gating</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>1</td>
<td>fail</td>
<td>init $b_2=0$ to $-5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>0.5</td>
<td>12.35</td>
<td>init $b_2=-6$</td>
</tr>
<tr>
<td>exclusive gating</td>
<td>Fig. 2(c)</td>
<td>1 - $g(x)$</td>
<td>$g(x)$</td>
<td>fail</td>
<td>init $b_2=0$ to $-5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - $g(x)$</td>
<td>$g(x)$</td>
<td>8.70</td>
<td>init $b_2=-6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - $g(x)$</td>
<td>$g(x)$</td>
<td>9.81</td>
<td>init $b_2=-7$</td>
</tr>
<tr>
<td>shortcut-only gating</td>
<td>Fig. 2(d)</td>
<td>1 - $g(x)$</td>
<td>1</td>
<td>12.86</td>
<td>init $b_2=0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - $g(x)$</td>
<td>1</td>
<td>6.91</td>
<td>init $b_2=-6$</td>
</tr>
<tr>
<td>conv shortcut</td>
<td>Fig. 2(e)</td>
<td>$1 \times 1$ conv</td>
<td>1</td>
<td>12.22</td>
<td></td>
</tr>
<tr>
<td>dropout shortcut</td>
<td>Fig. 2(f)</td>
<td>dropout 0.5</td>
<td>1</td>
<td>fail</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 2. Classification error (%) on the CIFAR-10 test set using different activation functions.

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

Table 2 of "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027
The *pre-activation* architecture was evaluated also on ImageNet, in a single-crop regime.

**Table 5.** Comparisons of single-crop error on the ILSVRC 2012 validation set. All ResNets are trained using the same hyper-parameters and implementations as [1]). Our Residual Units are the full pre-activation version (Fig. 4(e)). †: code/model available at [https://github.com/facebook/fb.resnet.torch/tree/master/pretrained](https://github.com/facebook/fb.resnet.torch/tree/master/pretrained), using scale and aspect ratio augmentation in [20].

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

Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)
Authors do not consider bottleneck blocks. Instead, they experiment with different block types, e.g., $B(1, 3, 1)$ or $B(3, 3)$.

<table>
<thead>
<tr>
<th>block type</th>
<th>depth</th>
<th># params</th>
<th>time, s</th>
<th>CIFAR-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B(1, 3, 1)$</td>
<td>40</td>
<td>1.4M</td>
<td>85.8</td>
<td>6.06</td>
</tr>
<tr>
<td>$B(3, 1, 1)$</td>
<td>40</td>
<td>1.3M</td>
<td>82.2</td>
<td>5.86</td>
</tr>
<tr>
<td>$B(3, 3)$</td>
<td>28</td>
<td>1.5M</td>
<td>67.5</td>
<td>5.73</td>
</tr>
<tr>
<td>$B(3, 1, 3)$</td>
<td>22</td>
<td>1.1M</td>
<td>59.9</td>
<td>5.78</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>group name</th>
<th>output size</th>
<th>block type = $B(3, 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>$32 \times 32$</td>
<td>$[3 \times 3, 16]$</td>
</tr>
<tr>
<td>conv2</td>
<td>$32 \times 32$</td>
<td>$\begin{bmatrix} 3 \times 3, 16 \times k \ 3 \times 3, 16 \times k \end{bmatrix} \times N$</td>
</tr>
<tr>
<td>conv3</td>
<td>$16 \times 16$</td>
<td>$\begin{bmatrix} 3 \times 3, 32 \times k \ 3 \times 3, 32 \times k \end{bmatrix} \times N$</td>
</tr>
<tr>
<td>conv4</td>
<td>$8 \times 8$</td>
<td>$\begin{bmatrix} 3 \times 3, 64 \times k \ 3 \times 3, 64 \times k \end{bmatrix} \times N$</td>
</tr>
<tr>
<td>avg-pool</td>
<td>$1 \times 1$</td>
<td>$[8 \times 8]$</td>
</tr>
</tbody>
</table>

Table 2 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146
Authors evaluate various *widening factors* \( k \):

<table>
<thead>
<tr>
<th>depth</th>
<th>( k )</th>
<th># params</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1</td>
<td>0.6M</td>
<td>6.85</td>
<td>30.89</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>2.2M</td>
<td>5.33</td>
<td>26.04</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>8.9M</td>
<td>4.97</td>
<td>22.89</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>35.7M</td>
<td>4.66</td>
<td>-</td>
</tr>
<tr>
<td>28</td>
<td>10</td>
<td>36.5M</td>
<td>4.17</td>
<td>20.50</td>
</tr>
<tr>
<td>28</td>
<td>12</td>
<td>52.5M</td>
<td>4.33</td>
<td>20.43</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>17.2M</td>
<td>4.38</td>
<td>21.22</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>26.8M</td>
<td>4.44</td>
<td>20.75</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>11.0M</td>
<td>4.81</td>
<td>22.07</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>17.1M</td>
<td>4.56</td>
<td>21.59</td>
</tr>
</tbody>
</table>

**Table 1 of “Wide Residual Networks”, https://arxiv.org/abs/1605.07146**

<table>
<thead>
<tr>
<th>group name</th>
<th>output size</th>
<th>block type = ( B(3, 3) )</th>
</tr>
</thead>
</table>
| conv1      | 32 \times 32 | \[
\begin{bmatrix}
3 \times 3, 16 \\
3 \times 3, 16 
\end{bmatrix}
\times N
\] |
| conv2      | 32 \times 32 | \[
\begin{bmatrix}
3 \times 3, 16 \times k \\
3 \times 3, 16 \times k 
\end{bmatrix}
\times N
\] |
| conv3      | 16 \times 16 | \[
\begin{bmatrix}
3 \times 3, 32 \times k \\
3 \times 3, 32 \times k 
\end{bmatrix}
\times N
\] |
| conv4      | 8 \times 8  | \[
\begin{bmatrix}
3 \times 3, 64 \times k \\
3 \times 3, 64 \times k 
\end{bmatrix}
\times N
\] |
| avg-pool   | 1 \times 1  | [8 \times 8]                |

**Table 4 of “Wide Residual Networks”, https://arxiv.org/abs/1605.07146**
Authors measure the effect of **dropping out** inside the residual block (but not the residual connection itself).

<table>
<thead>
<tr>
<th>depth</th>
<th>$k$</th>
<th>dropout</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td></td>
<td>5.02</td>
<td>24.03</td>
<td>1.85</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>✓</td>
<td>5.24</td>
<td>23.91</td>
<td>1.64</td>
</tr>
<tr>
<td>28</td>
<td>10</td>
<td></td>
<td>4.00</td>
<td>19.25</td>
<td>-</td>
</tr>
<tr>
<td>28</td>
<td>10</td>
<td>✓</td>
<td>3.89</td>
<td><strong>18.85</strong></td>
<td>-</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td></td>
<td>6.43</td>
<td>29.89</td>
<td>2.08</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>✓</td>
<td>6.28</td>
<td>29.78</td>
<td>1.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>group name</th>
<th>output size</th>
<th>block type = $B(3, 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>$32 \times 32$</td>
<td>$[3 \times 3, 16] \times N$</td>
</tr>
<tr>
<td>conv2</td>
<td>$32 \times 32$</td>
<td>$[3 \times 3, 16 \times k] \times N$</td>
</tr>
<tr>
<td>conv3</td>
<td>$16 \times 16$</td>
<td>$[3 \times 3, 32 \times k] \times N$</td>
</tr>
<tr>
<td>conv4</td>
<td>$8 \times 8$</td>
<td>$[3 \times 3, 64 \times k] \times N$</td>
</tr>
<tr>
<td>avg-pool</td>
<td>$1 \times 1$</td>
<td>$[8 \times 8]$</td>
</tr>
</tbody>
</table>

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

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

#### Dataset

<table>
<thead>
<tr>
<th></th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>depth-k</td>
</tr>
<tr>
<td>NIN [20]</td>
<td></td>
</tr>
<tr>
<td>DSN [19]</td>
<td></td>
</tr>
<tr>
<td>FitNet [24]</td>
<td></td>
</tr>
<tr>
<td>Highway [28]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1202</td>
</tr>
<tr>
<td>stoc-depth[14]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>1202</td>
</tr>
<tr>
<td>pre-act-ResNet[13]</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>1001</td>
</tr>
<tr>
<td>WRN (ours)</td>
<td>40-4</td>
</tr>
<tr>
<td></td>
<td>16-8</td>
</tr>
<tr>
<td></td>
<td>28-10</td>
</tr>
</tbody>
</table>

*Table 5 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146*

#### ImageNet

<table>
<thead>
<tr>
<th>Model</th>
<th>top-1 err, %</th>
<th>top-5 err, %</th>
<th>#params</th>
<th>time/batch 16</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

*Table 8 of "Wide Residual Networks", https://arxiv.org/abs/1605.07146*
The initial convolution generates 64 channels, each $1 \times 1$ convolution in dense block 128, each $3 \times 3$ convolution in dense block 32, and the transition layer reduces the number of channels in the initial convolution by half.

<table>
<thead>
<tr>
<th>Layers</th>
<th>Output Size</th>
<th>DenseNet-121</th>
<th>DenseNet-169</th>
<th>DenseNet-201</th>
<th>DenseNet-264</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>$112 \times 112$</td>
<td>7 × 7 conv, stride 2</td>
<td>7 × 7 conv, stride 2</td>
<td>7 × 7 conv, stride 2</td>
<td>7 × 7 conv, stride 2</td>
</tr>
<tr>
<td>Pooling</td>
<td>$56 \times 56$</td>
<td>3 × 3 max pool, stride 2</td>
<td>3 × 3 max pool, stride 2</td>
<td>3 × 3 max pool, stride 2</td>
<td>3 × 3 max pool, stride 2</td>
</tr>
<tr>
<td>Dense Block (1)</td>
<td>$56 \times 56$</td>
<td>$1 \times 1$ conv × 6</td>
<td>$1 \times 1$ conv × 6</td>
<td>$1 \times 1$ conv × 6</td>
<td>$1 \times 1$ conv × 6</td>
</tr>
<tr>
<td>Transition Layer (1)</td>
<td>$56 \times 56$</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
</tr>
<tr>
<td>Dense Block (2)</td>
<td>$28 \times 28$</td>
<td>$1 \times 1$ conv × 12</td>
<td>$1 \times 1$ conv × 12</td>
<td>$1 \times 1$ conv × 12</td>
<td>$1 \times 1$ conv × 12</td>
</tr>
<tr>
<td>Transition Layer (2)</td>
<td>$28 \times 28$</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
</tr>
<tr>
<td>Dense Block (3)</td>
<td>$14 \times 14$</td>
<td>$1 \times 1$ conv × 24</td>
<td>$1 \times 1$ conv × 32</td>
<td>$1 \times 1$ conv × 48</td>
<td>$1 \times 1$ conv × 64</td>
</tr>
<tr>
<td>Transition Layer (3)</td>
<td>$14 \times 14$</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
<td>$2 \times 2$ average pool, stride 2</td>
</tr>
<tr>
<td>Dense Block (4)</td>
<td>$7 \times 7$</td>
<td>$1 \times 1$ conv × 16</td>
<td>$1 \times 1$ conv × 32</td>
<td>$1 \times 1$ conv × 32</td>
<td>$1 \times 1$ conv × 48</td>
</tr>
<tr>
<td>Classification Layer</td>
<td>$1 \times 1$</td>
<td>7 × 7 global average pool</td>
<td>1000D fully-connected, softmax</td>
<td>1000D fully-connected, softmax</td>
<td>1000D fully-connected, softmax</td>
</tr>
</tbody>
</table>

Table 1 of “Densely Connected Convolutional Networks”, https://arxiv.org/abs/1608.06993
## DenseNet – Results

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Depth</th>
<th>Params</th>
<th>C10</th>
<th>C10+</th>
<th>C100</th>
<th>C100+</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network in Network [22]</td>
<td>-</td>
<td>-</td>
<td>10.41</td>
<td>8.81</td>
<td>35.68</td>
<td>-</td>
<td>2.35</td>
</tr>
<tr>
<td>All-CNN [32]</td>
<td>-</td>
<td>-</td>
<td>9.08</td>
<td>7.25</td>
<td>-</td>
<td>33.71</td>
<td>-</td>
</tr>
<tr>
<td>Deeply Supervised Net [20]</td>
<td>-</td>
<td>-</td>
<td>9.69</td>
<td>7.97</td>
<td>-</td>
<td>34.57</td>
<td>1.92</td>
</tr>
<tr>
<td>Highway Network [34]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.72</td>
<td>-</td>
<td>32.39</td>
<td>-</td>
</tr>
<tr>
<td>FractalNet [17]</td>
<td>21</td>
<td>38.6M</td>
<td>10.18</td>
<td>5.22</td>
<td>35.34</td>
<td>23.30</td>
<td>2.01</td>
</tr>
<tr>
<td>with Dropout/Drop-path</td>
<td>21</td>
<td>38.6M</td>
<td>7.33</td>
<td>4.60</td>
<td>28.20</td>
<td>23.73</td>
<td>1.87</td>
</tr>
<tr>
<td>ResNet [11]</td>
<td>110</td>
<td>1.7M</td>
<td>-</td>
<td>6.61</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ResNet (reported by [13])</td>
<td>110</td>
<td>1.7M</td>
<td>13.63</td>
<td>6.41</td>
<td>44.74</td>
<td>27.22</td>
<td>2.01</td>
</tr>
<tr>
<td>ResNet with Stochastic Depth [13]</td>
<td>110</td>
<td>1.7M</td>
<td>11.66</td>
<td>5.23</td>
<td>37.80</td>
<td>24.58</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>1202</td>
<td>10.2M</td>
<td>-</td>
<td>4.91</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Wide ResNet [42]</td>
<td>16</td>
<td>11.0M</td>
<td>-</td>
<td>4.81</td>
<td>-</td>
<td>22.07</td>
<td>-</td>
</tr>
<tr>
<td>with Dropout</td>
<td>28</td>
<td>36.5M</td>
<td>-</td>
<td>4.17</td>
<td>-</td>
<td>20.50</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>2.7M</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.64</td>
</tr>
<tr>
<td>ResNet (pre-activation) [12]</td>
<td>164</td>
<td>1.7M</td>
<td>11.26</td>
<td>5.46</td>
<td>35.58</td>
<td>24.33</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1001</td>
<td>10.2M</td>
<td>10.56</td>
<td>4.62</td>
<td>33.47</td>
<td>22.71</td>
<td>-</td>
</tr>
<tr>
<td>DenseNet (k = 12)</td>
<td>40</td>
<td>1.0M</td>
<td>7.00</td>
<td>5.24</td>
<td>27.55</td>
<td>24.42</td>
<td>1.79</td>
</tr>
<tr>
<td>DenseNet (k = 12)</td>
<td>100</td>
<td>7.0M</td>
<td>5.77</td>
<td>4.10</td>
<td>23.79</td>
<td>20.20</td>
<td>1.67</td>
</tr>
<tr>
<td>DenseNet (k = 24)</td>
<td>100</td>
<td>27.2M</td>
<td>5.83</td>
<td>3.74</td>
<td>23.42</td>
<td>19.25</td>
<td>1.59</td>
</tr>
<tr>
<td>DenseNet-BC (k = 12)</td>
<td>100</td>
<td>0.8M</td>
<td>5.92</td>
<td>4.51</td>
<td>24.15</td>
<td>22.77</td>
<td>1.76</td>
</tr>
<tr>
<td>DenseNet-BC (k = 24)</td>
<td>250</td>
<td>15.3M</td>
<td>5.19</td>
<td>3.62</td>
<td>19.64</td>
<td>17.60</td>
<td>1.74</td>
</tr>
<tr>
<td>DenseNet-BC (k = 40)</td>
<td>190</td>
<td>25.6M</td>
<td>-</td>
<td>3.46</td>
<td>-</td>
<td>17.18</td>
<td>-</td>
</tr>
</tbody>
</table>

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
In architectures up until now, number of filters doubled when spatial 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 $\alpha$ is number of filters to add in total.
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

<table>
<thead>
<tr>
<th>Network</th>
<th># of Params</th>
<th>Output Feat. Dim.</th>
<th>Depth</th>
<th>Training Mem.</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>NiN [18]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.81</td>
<td>35.68</td>
</tr>
<tr>
<td>All-CNN [27]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.25</td>
<td>33.71</td>
</tr>
<tr>
<td>DSN [17]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.97</td>
<td>34.57</td>
</tr>
<tr>
<td>FitNet [21]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.39</td>
<td>35.04</td>
</tr>
<tr>
<td>Highway [29]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.72</td>
<td>32.39</td>
</tr>
<tr>
<td>Fractional Max-pooling [4]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4.50</td>
<td>27.62</td>
</tr>
<tr>
<td>ELU [29]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6.55</td>
<td>24.28</td>
</tr>
<tr>
<td>ResNet [7]</td>
<td>1.7M</td>
<td>64</td>
<td>110</td>
<td>547MB</td>
<td>6.43</td>
<td>25.16</td>
</tr>
<tr>
<td>ResNet [7]</td>
<td>10.2M</td>
<td>64</td>
<td>1001</td>
<td>2,921MB</td>
<td>-</td>
<td>27.82</td>
</tr>
<tr>
<td>ResNet [7]</td>
<td>19.4M</td>
<td>64</td>
<td>1202</td>
<td>2,069MB</td>
<td>7.93</td>
<td>-</td>
</tr>
<tr>
<td>Pre-activation ResNet [8]</td>
<td>1.7M</td>
<td>64</td>
<td>164</td>
<td>841MB</td>
<td>5.46</td>
<td>24.33</td>
</tr>
<tr>
<td>Pre-activation ResNet [8]</td>
<td>10.2M</td>
<td>64</td>
<td>1001</td>
<td>2,921MB</td>
<td>4.62</td>
<td>22.71</td>
</tr>
<tr>
<td>Stochastic Depth [10]</td>
<td>1.7M</td>
<td>64</td>
<td>110</td>
<td>547MB</td>
<td>5.23</td>
<td>24.58</td>
</tr>
<tr>
<td>Stochastic Depth [10]</td>
<td>10.2M</td>
<td>64</td>
<td>1202</td>
<td>2,069MB</td>
<td>4.91</td>
<td>-</td>
</tr>
<tr>
<td>FractalNet [14]</td>
<td>38.6M</td>
<td>1,024</td>
<td>21</td>
<td>-</td>
<td>4.60</td>
<td>23.73</td>
</tr>
<tr>
<td>SwapOut v2 (width×4) [26]</td>
<td>7.4M</td>
<td>256</td>
<td>32</td>
<td>-</td>
<td>4.76</td>
<td>22.72</td>
</tr>
<tr>
<td>Wide ResNet (width×4) [34]</td>
<td>8.7M</td>
<td>256</td>
<td>40</td>
<td>775MB</td>
<td>4.97</td>
<td>22.89</td>
</tr>
<tr>
<td>Wide ResNet (width×10) [34]</td>
<td>36.5M</td>
<td>640</td>
<td>28</td>
<td>1,383MB</td>
<td>4.17</td>
<td>20.50</td>
</tr>
<tr>
<td>Weighted ResNet [23]</td>
<td>19.1M</td>
<td>64</td>
<td>1192</td>
<td>-</td>
<td>5.10</td>
<td>-</td>
</tr>
<tr>
<td>DenseNet (k = 24) [9]</td>
<td>27.2M</td>
<td>2,352</td>
<td>100</td>
<td>4,381MB</td>
<td>3.74</td>
<td>19.25</td>
</tr>
<tr>
<td>DenseNet-BC (k = 40) [9]</td>
<td>25.6M</td>
<td>2,190</td>
<td>190</td>
<td>7,247MB</td>
<td>3.46</td>
<td>17.18</td>
</tr>
<tr>
<td>PyramidNet (α = 48)</td>
<td>1.7M</td>
<td>64</td>
<td>110</td>
<td>655MB</td>
<td>4.58±0.06</td>
<td>23.12±0.04</td>
</tr>
<tr>
<td>PyramidNet (α = 84)</td>
<td>3.8M</td>
<td>100</td>
<td>110</td>
<td>781MB</td>
<td>4.26±0.23</td>
<td>20.66±0.40</td>
</tr>
<tr>
<td>PyramidNet (α = 270)</td>
<td>28.3M</td>
<td>286</td>
<td>110</td>
<td>1,437MB</td>
<td>3.73±0.04</td>
<td>18.25±0.10</td>
</tr>
<tr>
<td>PyramidNet (bottleneck, α = 270)</td>
<td>27.0M</td>
<td>1,144</td>
<td>164</td>
<td>4,169MB</td>
<td>3.48±0.20</td>
<td>17.01±0.39</td>
</tr>
<tr>
<td>PyramidNet (bottleneck, α = 240)</td>
<td>26.6M</td>
<td>1,024</td>
<td>200</td>
<td>4,451MB</td>
<td>3.44±0.11</td>
<td>16.51±0.13</td>
</tr>
<tr>
<td>PyramidNet (bottleneck, α = 220)</td>
<td>26.8M</td>
<td>944</td>
<td>236</td>
<td>4,767MB</td>
<td>3.40±0.07</td>
<td>16.37±0.29</td>
</tr>
<tr>
<td>PyramidNet (bottleneck, α = 200)</td>
<td>26.0M</td>
<td>864</td>
<td>272</td>
<td>5,005MB</td>
<td>3.31±0.08</td>
<td>16.35±0.24</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Group</th>
<th>Output size</th>
<th>Building Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv 1</td>
<td>32×32</td>
<td>3 × 3, 16 ÷ (k−1)/N</td>
</tr>
<tr>
<td>conv 2</td>
<td>32×32</td>
<td>3 × 3, 16 ÷ (k−1)/N</td>
</tr>
<tr>
<td>conv 3</td>
<td>16×16</td>
<td>3 × 3, 16 ÷ (k−1)/N</td>
</tr>
<tr>
<td>conv 4</td>
<td>8×8</td>
<td>3 × 3, 16 ÷ (k−1)/N</td>
</tr>
<tr>
<td>avg pool</td>
<td>1×1</td>
<td>[8 × 8, 16 ÷ α]</td>
</tr>
</tbody>
</table>

Table 4 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915
### PyramidNet – ImageNet Results

<table>
<thead>
<tr>
<th>Network</th>
<th># of Params</th>
<th>OutputFeat. Dim.</th>
<th>Augmentation</th>
<th>Train Crop</th>
<th>Test Crop</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-152 [7]</td>
<td>60.0M</td>
<td>2,048</td>
<td>scale</td>
<td>224×224</td>
<td>224×224</td>
<td>23.0</td>
<td>6.7</td>
</tr>
<tr>
<td>Pre-ResNet-152† [8]</td>
<td>60.0M</td>
<td>2,048</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>224×224</td>
<td>22.2</td>
<td>6.2</td>
</tr>
<tr>
<td>Pre-ResNet-200† [8]</td>
<td>64.5M</td>
<td>2,048</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>224×224</td>
<td>21.7</td>
<td>5.8</td>
</tr>
<tr>
<td>WRN-50-2-bottleneck [34]</td>
<td>68.9M</td>
<td>2,048</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>224×224</td>
<td>21.9</td>
<td>6.0</td>
</tr>
<tr>
<td>PyramidNet-200 (α = 300)</td>
<td>62.1M</td>
<td>1,456</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>224×224</td>
<td>20.5</td>
<td>5.3</td>
</tr>
<tr>
<td>PyramidNet-200 (α = 300)*</td>
<td>62.1M</td>
<td>1,456</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>224×224</td>
<td>20.5</td>
<td>5.4</td>
</tr>
<tr>
<td>PyramidNet-200 (α = 450)*</td>
<td>116.4M</td>
<td>2,056</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>224×224</td>
<td>20.1</td>
<td>5.4</td>
</tr>
<tr>
<td>ResNet-200 [7]</td>
<td>64.5M</td>
<td>2,048</td>
<td>scale</td>
<td>224×224</td>
<td>320×320</td>
<td>21.8</td>
<td>6.0</td>
</tr>
<tr>
<td>Pre-ResNet-200 [8]</td>
<td>64.5M</td>
<td>2,048</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>320×320</td>
<td>20.1</td>
<td>4.8</td>
</tr>
<tr>
<td>Inception-v3 [32]</td>
<td>-</td>
<td>2,048</td>
<td>scale+asp ratio</td>
<td>299×299</td>
<td>299×299</td>
<td>21.2</td>
<td>5.6</td>
</tr>
<tr>
<td>Inception-v4 [30]</td>
<td>-</td>
<td>1,536</td>
<td>scale+asp ratio</td>
<td>299×299</td>
<td>299×299</td>
<td>20.0</td>
<td>5.0</td>
</tr>
<tr>
<td>PyramidNet-200 (α = 300)</td>
<td>62.1M</td>
<td>1,456</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>320×320</td>
<td>19.6</td>
<td>4.8</td>
</tr>
<tr>
<td>PyramidNet-200 (α = 300)*</td>
<td>62.1M</td>
<td>1,456</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>320×320</td>
<td>19.5</td>
<td>4.8</td>
</tr>
<tr>
<td>PyramidNet-200 (α = 450)*</td>
<td>116.4M</td>
<td>2,056</td>
<td>scale+asp ratio</td>
<td>224×224</td>
<td>320×320</td>
<td>19.2</td>
<td>4.7</td>
</tr>
</tbody>
</table>

Table 5 of "Deep Pyramidal Residual Networks", https://arxiv.org/abs/1610.02915
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 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*
<table>
<thead>
<tr>
<th>stage</th>
<th>output</th>
<th>ResNet-50</th>
<th>ResNeXt-50 (32x4d)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>7x7, 64, stride 2</td>
<td>7x7, 64, stride 2</td>
</tr>
<tr>
<td>conv1</td>
<td>112x112</td>
<td>3x3 max pool, stride 2</td>
<td>3x3 max pool, stride 2</td>
</tr>
<tr>
<td>conv2</td>
<td>56x56</td>
<td>1x1, 64</td>
<td>1x1, 128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x3, 64</td>
<td>3x3, 128, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x1, 256</td>
<td>1x1, 256</td>
</tr>
<tr>
<td>conv3</td>
<td>28x28</td>
<td>1x1, 128</td>
<td>1x1, 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x3, 128</td>
<td>3x3, 256, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x1, 512</td>
<td>1x1, 512</td>
</tr>
<tr>
<td>conv4</td>
<td>14x14</td>
<td>1x1, 256</td>
<td>1x1, 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x3, 256</td>
<td>3x3, 512, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x1, 1024</td>
<td>1x1, 1024</td>
</tr>
<tr>
<td>conv5</td>
<td>7x7</td>
<td>1x1, 512</td>
<td>1x1, 1024</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x3, 512</td>
<td>3x3, 1024, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x1, 2048</td>
<td>1x1, 2048</td>
</tr>
<tr>
<td></td>
<td>global average pool</td>
<td>1000-d fc, softmax</td>
<td>global average pool</td>
</tr>
<tr>
<td></td>
<td>1x1</td>
<td>1000-d fc, softmax</td>
<td>1000-d fc, softmax</td>
</tr>
<tr>
<td># params.</td>
<td>25.5x10^6</td>
<td>25.0x10^6</td>
<td></td>
</tr>
<tr>
<td>FLOPs</td>
<td>4.1x10^9</td>
<td>4.2x10^9</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 of "Aggregated Residual Transformations for Deep Neural Networks", https://arxiv.org/abs/1611.05431
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).
<table>
<thead>
<tr>
<th>setting</th>
<th>top-1 error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50 1 × 64d</td>
<td>23.9</td>
</tr>
<tr>
<td>ResNeXt-50 2 × 40d</td>
<td>23.0</td>
</tr>
<tr>
<td>ResNeXt-50 4 × 24d</td>
<td>22.6</td>
</tr>
<tr>
<td>ResNeXt-50 8 × 14d</td>
<td>22.3</td>
</tr>
<tr>
<td>ResNeXt-50 32 × 4d</td>
<td>22.2</td>
</tr>
<tr>
<td>ResNet-101 1 × 64d</td>
<td>22.0</td>
</tr>
<tr>
<td>ResNeXt-101 2 × 40d</td>
<td>21.7</td>
</tr>
<tr>
<td>ResNeXt-101 4 × 24d</td>
<td>21.4</td>
</tr>
<tr>
<td>ResNeXt-101 8 × 14d</td>
<td>21.3</td>
</tr>
<tr>
<td>ResNeXt-101 32 × 4d</td>
<td>21.2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>setting</th>
<th>top-1 err (%)</th>
<th>top-5 err (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-101 1 × 64d</td>
<td>22.0</td>
<td>6.0</td>
</tr>
<tr>
<td>ResNeXt-101 32 × 4d</td>
<td>21.2</td>
<td>5.6</td>
</tr>
</tbody>
</table>

1 × complexity references:

2 × complexity models follow:

<table>
<thead>
<tr>
<th>setting</th>
<th>top-1 err (%)</th>
<th>top-5 err (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-200 [15] 1 × 64d</td>
<td>21.7</td>
<td>5.8</td>
</tr>
<tr>
<td>ResNet-101, wider 1 × 100d</td>
<td>21.3</td>
<td>5.7</td>
</tr>
<tr>
<td>ResNeXt-101 2 × 64d</td>
<td>20.7</td>
<td>5.5</td>
</tr>
<tr>
<td>ResNeXt-101 64 × 4d</td>
<td>20.4</td>
<td>5.3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>224×224</th>
<th>320×320 / 299×299</th>
</tr>
</thead>
<tbody>
<tr>
<td>top-1 err</td>
<td>top-5 err</td>
</tr>
<tr>
<td>ResNet-101 [14]</td>
<td>22.0</td>
</tr>
<tr>
<td>ResNet-200 [15]</td>
<td>21.7</td>
</tr>
<tr>
<td>Inception-v3 [39]</td>
<td>-</td>
</tr>
<tr>
<td>Inception-v4 [37]</td>
<td>-</td>
</tr>
<tr>
<td>Inception-ResNet-v2 [37]</td>
<td>-</td>
</tr>
<tr>
<td>ResNeXt-101 (64 × 4d)</td>
<td>20.4</td>
</tr>
</tbody>
</table>

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

110-layer ResNet on CIFAR-10

110-layer ResNet on CIFAR-100
Drop $16 \times 16$ square in the input image, with randomly chosen center. The pixels are replaced by their mean value from the dataset.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>C10</th>
<th>C10+</th>
<th>C100</th>
<th>C100+</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet18 [5]</td>
<td>10.63 ± 0.26</td>
<td>4.72 ± 0.21</td>
<td>36.68 ± 0.57</td>
<td>22.46 ± 0.31</td>
<td>-</td>
</tr>
<tr>
<td>ResNet18 + cutout</td>
<td>9.31 ± 0.18</td>
<td>3.99 ± 0.13</td>
<td>34.98 ± 0.29</td>
<td>21.96 ± 0.24</td>
<td>-</td>
</tr>
<tr>
<td>WideResNet [22]</td>
<td>6.97 ± 0.22</td>
<td>3.87 ± 0.08</td>
<td>26.06 ± 0.22</td>
<td>18.8 ± 0.08</td>
<td>1.60 ± 0.05</td>
</tr>
<tr>
<td>WideResNet + cutout</td>
<td>5.54 ± 0.08</td>
<td>3.08 ± 0.16</td>
<td>23.94 ± 0.15</td>
<td>18.41 ± 0.27</td>
<td>1.30 ± 0.03</td>
</tr>
<tr>
<td>Shake-shake regularization + cutout</td>
<td>-</td>
<td>2.56 ± 0.07</td>
<td>-</td>
<td>15.20 ± 0.21</td>
<td>-</td>
</tr>
</tbody>
</table>

*Figure 3* 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”.

(a)  
(b)  
(c)
Algorithm 1 DropBlock

1: **Input**: output activations of a layer \( A \), \( block\_size \), \( \gamma \), \( mode \)
2: if \( mode == \) Inference then
3:     return \( A \)
4: end if
5: Randomly sample mask \( M: M_{i,j} \sim \text{Bernoulli}(\gamma) \)
6: 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).
7: Apply the mask: \( A = A \times M \)
8: Normalize the features: \( A = A \times \frac{\text{count}(M)}{\text{count\_ones}(M)} \)

*Figure 2 of "DropBlock: A regularization method for convolutional networks", https://arxiv.org/abs/1810.12890*
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
## Table 1 of "DropBlock: A regularization method for convolutional networks", https://arxiv.org/abs/1810.12890

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

The results are averages of three runs.
To perform CutMix:

- First we sample $\lambda$ uniformly from $(0, 1)$.
- We sample bounding box center uniformly.
- Width and height are set to $W \sqrt{\lambda}$, $H \sqrt{\lambda}$.
- Labels are combined as $\lambda y_A + (1 - \lambda) y_B$. 

Table 3 of "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features", https://arxiv.org/abs/1905.04899

Table 4 of "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features", https://arxiv.org/abs/1905.04899
Table 5: Comparison of state-of-the-art regularization methods on CIFAR-100.

<table>
<thead>
<tr>
<th>Model Configuration</th>
<th>Top-1 Error (%)</th>
<th>Top-5 Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>16.45</td>
<td>3.69</td>
</tr>
<tr>
<td>+ StochDepth [17]</td>
<td>15.86</td>
<td>3.33</td>
</tr>
<tr>
<td>+ Label smoothing (ε=0.1) [38]</td>
<td>16.73</td>
<td>3.37</td>
</tr>
<tr>
<td>+ Cutout + Label smoothing (ε=0.1)</td>
<td>15.61</td>
<td>3.88</td>
</tr>
<tr>
<td>+ DropBlock [8]</td>
<td>15.73</td>
<td>3.26</td>
</tr>
<tr>
<td>+ DropBlock + Label smoothing (ε=0.1)</td>
<td>15.16</td>
<td>3.86</td>
</tr>
<tr>
<td>+ Mixup (α=0.5) [48]</td>
<td>15.78</td>
<td>4.04</td>
</tr>
<tr>
<td>+ Mixup (α=1.0) [48]</td>
<td>15.63</td>
<td>3.99</td>
</tr>
<tr>
<td>+ Manifold Mixup (α=1.0) [42]</td>
<td>16.14</td>
<td>4.07</td>
</tr>
<tr>
<td>+ Cutout + Mixup (α=1.0)</td>
<td>15.46</td>
<td>3.42</td>
</tr>
<tr>
<td>+ Cutout + Manifold Mixup (α=1.0)</td>
<td>15.09</td>
<td>3.35</td>
</tr>
<tr>
<td>+ ShakeDrop [46]</td>
<td>15.08</td>
<td>2.72</td>
</tr>
<tr>
<td>+ CutMix</td>
<td>14.47</td>
<td>2.97</td>
</tr>
<tr>
<td>+ CutMix + ShakeDrop [46]</td>
<td><strong>13.81</strong></td>
<td><strong>2.29</strong></td>
</tr>
</tbody>
</table>

In the following, $\lambda \sim \text{Beta}(\alpha, \alpha)$.

Table 7: Impact of CutMix on CIFAR-100.

<table>
<thead>
<tr>
<th>Model Configuration</th>
<th>Top-1 Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>3.85</td>
</tr>
<tr>
<td>+ Cutout</td>
<td>3.10</td>
</tr>
<tr>
<td>+ Mixup (α=1.0)</td>
<td>3.09</td>
</tr>
<tr>
<td>+ Manifold Mixup (α=1.0)</td>
<td>3.15</td>
</tr>
<tr>
<td>+ CutMix</td>
<td><strong>2.88</strong></td>
</tr>
</tbody>
</table>

Figure 3: Impact of $\alpha$ and CutMix layer depth on CIFAR-100 top-1 error.
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).
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 $3 \times 3$) 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 $1 \times 1$ convolution acting on each position independently (which reduces time and space complexity of a regular convolution by a factor of $3 \cdot 3$).
- 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).
The mobile inverted bottleneck convolution is denoted for example as $\text{MBConv6 } k3x3$, where the 6 denotes expansion factor after the bottleneck and $3 \times 3$ is the kernel size of the separable convolution.

Furthermore, the mobile inverted bottleneck convolution can be augmented with squeeze and excitation blocks.
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 \( \text{swish}(x) \overset{\text{def}}{=} x \odot \sigma(x) \) activation function is utilized.

<table>
<thead>
<tr>
<th>Stage ( i )</th>
<th>Operator</th>
<th>Resolution ( H_i \times W_i )</th>
<th>#Channels ( C_i )</th>
<th>#Layers ( L_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Conv3x3</td>
<td>224 x 224</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>MBConv1, k3x3</td>
<td>112 x 112</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>MBConv6, k3x3</td>
<td>112 x 112</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>MBConv6, k5x5</td>
<td>56 x 56</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>MBConv6, k3x3</td>
<td>28 x 28</td>
<td>80</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>MBConv6, k5x5</td>
<td>14 x 14</td>
<td>112</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>MBConv6, k5x5</td>
<td>14 x 14</td>
<td>192</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>MBConv6, k3x3</td>
<td>7 x 7</td>
<td>320</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Conv1x1 &amp; Pooling &amp; FC</td>
<td>7 x 7</td>
<td>1280</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Acc.</th>
<th>Top-5 Acc.</th>
<th>#Params</th>
<th>Ratio-to-EfficientNet</th>
<th>#FLOPS</th>
<th>Ratio-to-EfficientNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>EfficientNet-B0</td>
<td>77.3%</td>
<td>93.5%</td>
<td>5.3M</td>
<td>1x</td>
<td>0.39B</td>
<td>1x</td>
</tr>
<tr>
<td>ResNet-50 (He et al., 2016)</td>
<td>76.0%</td>
<td>93.0%</td>
<td>26M</td>
<td>4.9x</td>
<td>4.1B</td>
<td>11x</td>
</tr>
<tr>
<td>DenseNet-169 (Huang et al., 2017)</td>
<td>76.2%</td>
<td>93.2%</td>
<td>14M</td>
<td>2.6x</td>
<td>3.5B</td>
<td>8.9x</td>
</tr>
<tr>
<td>EfficientNet-B1</td>
<td>79.2%</td>
<td>94.5%</td>
<td>7.8M</td>
<td>1x</td>
<td>0.70B</td>
<td>1x</td>
</tr>
<tr>
<td>ResNet-152 (He et al., 2016)</td>
<td>77.8%</td>
<td>93.8%</td>
<td>60M</td>
<td>7.6x</td>
<td>11B</td>
<td>16x</td>
</tr>
<tr>
<td>DenseNet-264 (Huang et al., 2017)</td>
<td>77.9%</td>
<td>93.9%</td>
<td>34M</td>
<td>4.3x</td>
<td>6.0B</td>
<td>6.6x</td>
</tr>
<tr>
<td>Inception-v3 (Szegedy et al., 2016)</td>
<td>78.8%</td>
<td>94.4%</td>
<td>24M</td>
<td>3.0x</td>
<td>5.7B</td>
<td>8.1x</td>
</tr>
<tr>
<td>Xception (Chollet, 2017)</td>
<td>79.0%</td>
<td>94.5%</td>
<td>23M</td>
<td>3.0x</td>
<td>8.4B</td>
<td>12x</td>
</tr>
<tr>
<td>EfficientNet-B2</td>
<td>80.3%</td>
<td>95.0%</td>
<td>9.2M</td>
<td>1x</td>
<td>1.0B</td>
<td>1x</td>
</tr>
<tr>
<td>Inception-v4 (Szegedy et al., 2017)</td>
<td>80.0%</td>
<td>95.0%</td>
<td>48M</td>
<td>5.2x</td>
<td>13B</td>
<td>13x</td>
</tr>
<tr>
<td>Inception-resnet-v2 (Szegedy et al., 2017)</td>
<td>80.1%</td>
<td>95.1%</td>
<td>56M</td>
<td>6.1x</td>
<td>13B</td>
<td>13x</td>
</tr>
<tr>
<td>EfficientNet-B3</td>
<td>81.7%</td>
<td>95.6%</td>
<td>12M</td>
<td>1x</td>
<td>1.8B</td>
<td>1x</td>
</tr>
<tr>
<td>ResNet3-101 (Xie et al., 2017)</td>
<td>80.9%</td>
<td>95.6%</td>
<td>84M</td>
<td>7.0x</td>
<td>32B</td>
<td>18x</td>
</tr>
<tr>
<td>PolyNet (Zhang et al., 2017)</td>
<td>81.3%</td>
<td>95.8%</td>
<td>92M</td>
<td>7.7x</td>
<td>35B</td>
<td>19x</td>
</tr>
<tr>
<td>EfficientNet-B4</td>
<td>83.0%</td>
<td>96.3%</td>
<td>19M</td>
<td>1x</td>
<td>4.2B</td>
<td>1x</td>
</tr>
<tr>
<td>SENet (Hu et al., 2018)</td>
<td>82.7%</td>
<td>96.2%</td>
<td>146M</td>
<td>7.7x</td>
<td>42B</td>
<td>10x</td>
</tr>
<tr>
<td>NASNet-A (Zoph et al., 2018)</td>
<td>82.7%</td>
<td>96.2%</td>
<td>89M</td>
<td>4.7x</td>
<td>24B</td>
<td>5.7x</td>
</tr>
<tr>
<td>AmoebaNet-A (Real et al., 2019)</td>
<td>82.8%</td>
<td>96.1%</td>
<td>87M</td>
<td>4.6x</td>
<td>23B</td>
<td>5.5x</td>
</tr>
<tr>
<td>PNASNet (Liu et al., 2018)</td>
<td>82.9%</td>
<td>96.2%</td>
<td>86M</td>
<td>4.5x</td>
<td>23B</td>
<td>6.0x</td>
</tr>
<tr>
<td>EfficientNet-B5</td>
<td>83.7%</td>
<td>96.7%</td>
<td>30M</td>
<td>1x</td>
<td>9.9B</td>
<td>1x</td>
</tr>
<tr>
<td>AmoebaNet-C (Cubuk et al., 2019)</td>
<td>83.5%</td>
<td>96.5%</td>
<td>155M</td>
<td>5.2x</td>
<td>41B</td>
<td>4.1x</td>
</tr>
<tr>
<td>EfficientNet-B6</td>
<td>84.2%</td>
<td>96.8%</td>
<td>43M</td>
<td>1x</td>
<td>19B</td>
<td>1x</td>
</tr>
<tr>
<td>EfficientNet-B7</td>
<td>84.4%</td>
<td>97.1%</td>
<td>66M</td>
<td>1x</td>
<td>37B</td>
<td>1x</td>
</tr>
<tr>
<td>GPipe (Huang et al., 2018)</td>
<td>84.3%</td>
<td>97.0%</td>
<td>557M</td>
<td>8.4x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946
Figure 5 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946

Figure 1 of "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", https://arxiv.org/abs/1905.11946
In April 2021, an improved version of EfficientNet, **EfficientNetV2**, was published. It is currently one of very good CNNs available 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 $1 \times 1$ convolution and a $3 \times 3$ 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.](https://arxiv.org/abs/2104.00298)
Table 4. EfficientNetV2-S architecture – MBConv and Fused-MBConv blocks are described in Figure 2.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Operator</th>
<th>Stride</th>
<th>#Channels</th>
<th>#Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Conv3x3</td>
<td>2</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Fused-MBConv1, k3x3</td>
<td>1</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Fused-MBConv4, k3x3</td>
<td>2</td>
<td>48</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Fused-MBConv4, k3x3</td>
<td>2</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>MBConv4, k3x3, SE0.25</td>
<td>2</td>
<td>128</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>MBConv6, k3x3, SE0.25</td>
<td>1</td>
<td>160</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>MBConv6, k3x3, SE0.25</td>
<td>2</td>
<td>256</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Conv1x1 &amp; Pooling &amp; FC</td>
<td>-</td>
<td>1280</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

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**

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Acc</th>
<th>Params</th>
<th>PLOPs</th>
<th>Infer-time (ms)</th>
<th>Train-time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EfficientNet-B3</td>
<td>81.5%</td>
<td>12M</td>
<td>1.9B</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>EfficientNet-B4</td>
<td>82.9%</td>
<td>19M</td>
<td>4.2B</td>
<td>30</td>
<td>21</td>
</tr>
<tr>
<td>EfficientNet-B5</td>
<td>83.7%</td>
<td>30M</td>
<td>10B</td>
<td>60</td>
<td>43</td>
</tr>
<tr>
<td>EfficientNet-B6</td>
<td>84.3%</td>
<td>43M</td>
<td>19B</td>
<td>97</td>
<td>75</td>
</tr>
<tr>
<td>EfficientNet-B7</td>
<td>84.7%</td>
<td>66M</td>
<td>38B</td>
<td>170</td>
<td>139</td>
</tr>
<tr>
<td>RegNetY-SGF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RegNetY-16GF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ResNetSt-101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ResNetSt-200</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ResNetSt-269</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TResNet-L</td>
<td>83.8%</td>
<td>56M</td>
<td>-</td>
<td>45</td>
<td>-</td>
</tr>
<tr>
<td>TResNet-XL</td>
<td>84.3%</td>
<td>78M</td>
<td>-</td>
<td>66</td>
<td>-</td>
</tr>
<tr>
<td>EfficientNet-X-L</td>
<td>84.7%</td>
<td>73M</td>
<td>91B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NFNNet-F0</td>
<td>83.6%</td>
<td>72M</td>
<td>12B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NFNNet-F1</td>
<td>84.7%</td>
<td>133M</td>
<td>36B</td>
<td>70</td>
<td>20</td>
</tr>
<tr>
<td>NFNNet-F2</td>
<td>85.1%</td>
<td>194M</td>
<td>63B</td>
<td>124</td>
<td>36</td>
</tr>
<tr>
<td>NFNNet-F3</td>
<td>85.7%</td>
<td>255M</td>
<td>115B</td>
<td>203</td>
<td>65</td>
</tr>
<tr>
<td>NFNNet-F4</td>
<td>85.9%</td>
<td>316M</td>
<td>215B</td>
<td>309</td>
<td>126</td>
</tr>
<tr>
<td>LambdaResNet-420-hybrid</td>
<td>84.9%</td>
<td>125M</td>
<td>-</td>
<td>-</td>
<td>67</td>
</tr>
<tr>
<td>BotNet-T7-hybrid</td>
<td>84.7%</td>
<td>75M</td>
<td>46B</td>
<td>-</td>
<td>95</td>
</tr>
<tr>
<td>BiT-M-R152x2 (21k)</td>
<td>85.2%</td>
<td>236M</td>
<td>135B</td>
<td>500</td>
<td>-</td>
</tr>
</tbody>
</table>

**ConvNets & Hybrid**

**Vision Transformers**

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Acc</th>
<th>Params</th>
<th>PLOPs</th>
<th>Infer-time (ms)</th>
<th>Train-time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT-B/32</td>
<td>73.4%</td>
<td>88M</td>
<td>13B</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>VT-B/16</td>
<td>74.9%</td>
<td>87M</td>
<td>56B</td>
<td>68</td>
<td>-</td>
</tr>
<tr>
<td>DeiT-B (ViT+reg)</td>
<td>81.8%</td>
<td>86M</td>
<td>18B</td>
<td>19</td>
<td>-</td>
</tr>
<tr>
<td>DeiT-B-384 (ViT+reg)</td>
<td>83.1%</td>
<td>86M</td>
<td>56B</td>
<td>68</td>
<td>-</td>
</tr>
<tr>
<td>T2T-ViT-19 (Yuan et al., 2021)</td>
<td>81.4%</td>
<td>39M</td>
<td>8.4B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T2T-ViT-24 (Yuan et al., 2021)</td>
<td>82.2%</td>
<td>64M</td>
<td>13B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VT-B/16 (21k) (Dosovitskiy et al., 2021)</td>
<td>84.6%</td>
<td>87M</td>
<td>56B</td>
<td>68</td>
<td>-</td>
</tr>
<tr>
<td>VT-L/16 (21k) (Dosovitskiy et al., 2021)</td>
<td>85.3%</td>
<td>304M</td>
<td>192B</td>
<td>195</td>
<td>172</td>
</tr>
</tbody>
</table>

**ConvNets (ours)**

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Acc</th>
<th>Params</th>
<th>PLOPs</th>
<th>Infer-time (ms)</th>
<th>Train-time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EfficientNetV2-S</td>
<td>83.9%</td>
<td>22M</td>
<td>8.8B</td>
<td>24</td>
<td>7.1</td>
</tr>
<tr>
<td>EfficientNetV2-M</td>
<td>85.1%</td>
<td>54M</td>
<td>24B</td>
<td>57</td>
<td>13</td>
</tr>
<tr>
<td>EfficientNetV2-L</td>
<td>85.7%</td>
<td>120M</td>
<td>53B</td>
<td>98</td>
<td>24</td>
</tr>
<tr>
<td>EfficientNetV2-S (21k)</td>
<td>84.9%</td>
<td>22M</td>
<td>8.8B</td>
<td>24</td>
<td>9.0</td>
</tr>
<tr>
<td>EfficientNetV2-M (21k)</td>
<td>86.2%</td>
<td>54M</td>
<td>24B</td>
<td>57</td>
<td>15</td>
</tr>
<tr>
<td>EfficientNetV2-L (21k)</td>
<td>86.8%</td>
<td>120M</td>
<td>53B</td>
<td>98</td>
<td>26</td>
</tr>
<tr>
<td>EfficientNetV2-XY (21k)</td>
<td>87.3%</td>
<td>208M</td>
<td>94B</td>
<td>-</td>
<td>45</td>
</tr>
</tbody>
</table>

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, the input $X$ is multiplied by the weight matrix $W$ as $XW$;
- during the backward pass, the gradient $G$ is multiplied by the *transposed* weight matrix as $GW^T$. 
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

$$\left(K \ast I\right)_{i,j,o} = \sum_{m,n,c} I_{i \cdot S+m, j \cdot S+n, c} K_{m,n,c,o};$$

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

$$\frac{\partial L}{\partial I_{i,j,c}} = \sum_{i',m} \sum_{j',n} \sum_{o} G_{i',j',o} K_{m,n,c,o} \cdot I_{i \cdot S+m, j \cdot S+n, c}.$$

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 $3 \times 3$ kernel.

- **valid**, stride=1, regular: 
  
- **valid**, stride=2, regular:
  
  [Images of padding schemes and different strides]

  https://github.com/vdumoulin/conv_arithmetic

  [Images of padding schemes and different strides]

  https://github.com/vdumoulin/conv_arithmetic
Transposed Convolution Animation

Illustration of the padding schemes and different strides for a $3 \times 3$ kernel.

- **same**, stride=1, regular:
  - regular:
    - https://github.com/vdumoulin/conv_arithmetic
  - transposed:
    - https://github.com/vdumoulin/conv_arithmetic

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