Convolutional Neural Networks

Milan Straka

March 25, 2019
Convergence

The training process might or might not converge. Even if it does, it might converge slowly or quickly.

We have already discussed two factors influencing it on the previous lecture:

- saturating non-linearities,
- parameter initialization strategies.

Another prominent method for dealing with slow or diverging training is *gradient clipping*.
Convergence – Gradient Clipping

$J(w, b)$

$w$

$b$

Figure 8.3, page 289 of Deep Learning Book, http://deeplearningbook.org
Using a given maximum norm, we may *clip* the gradient.

\[
g \left\{ \begin{array}{ll}
g & \text{if } ||g|| \leq c \\
c \frac{g}{||g||} & \text{if } ||g|| > c
\end{array} \right.
\]

The clipping can be per weight (*clipvalue* of `tf.keras.optimizers.Optimizer`), per variable or for the gradient as a whole (*clipnorm* of `tf.keras.optimizers.Optimizer`).
Going Deeper
Convolutional Networks

Consider data with some structure (temporal data, speech, images, ...).

Unlike densely connected layers, we might want:

- local interactions;
- parameter sharing (equal response everywhere);
- shift invariance.
Convolution Operation

For a functions $x$ and $w$, convolution $x \ast w$ is defined as

$$(x \ast w)(t) = \int x(a)w(t - a) \, da.$$ 

For vectors, we have

$$(x \ast w)_t = \sum_i x_i w_{t-i}.$$ 

Convolution operation can be generalized to two dimensions by

$$(I \ast K)_{i,j} = \sum_{m,n} I_{m,n} K_{i-m,j-n}.$$ 

Closely related is cross-corellation, where $K$ is flipped:

$$S_{i,j} = \sum_{m,n} I_{i+m,j+n} K_{m,n}.$$
Convolution

Figure 9.1, page 334 of Deep Learning Book, http://deeplearningbook.org
Convolutional Networks

Gradient Clipping
Convolution
CNNs
AlexNet
Deep Prior
VGG
Inception
BatchNorm
ResNet
The $K$ is usually called a *kernel* or a *filter*, and we generally apply several of them at the same time.

Consider an input image with $C$ channels. The convolution layer with $F$ filters of width $W$, height $H$ and stride $S$ produces an output with $F$ channels, is parametrized by a kernel $K$ of total size $W \times H \times C \times F$ and is computed as

$$(I * K)_{i,j,k} = \sum_{m,n,o} I_{i \cdot S + m, j \cdot S + n, o} K_{m,n,o,k}.$$  

We can consider the kernel to be composed of $F$ independent kernels, one for every output channel.

Note that while only local interactions are performed in the image spacial dimensions (width and height), we combine input channels in a fully connected manner.
Convolution Layer

There are multiple padding schemes, most common are:

- **valid**: Only use valid pixels, which causes the result to be smaller than the input.
- **same**: Pad original image with zero pixels so that the result is exactly the size of the input.

There are two prevalent image formats (called *data_format* in TensorFlow):

- **channels_last**: The dimensions of the 4-dimensional image tensor are batch, height, width, and channels.
  
  The original TensorFlow format, faster on CPU.

- **channels_first**: The dimensions of the 4-dimensional image tensor are batch, channel, height, and width.
  
  Usual GPU format (used by CUDA and nearly all frameworks); on TensorFlow, not all CPU kernels are available with this layout.

TensorFlow has been implementing an approach that will convert data format to **channels_first** automatically depending on the backend.
Pooling is an operation similar to convolution, but we perform a fixed operation instead of multiplying by a kernel.

- Max pooling (minor translation invariance)
- Average pooling
High-level CNN Architecture

We repeatedly use the following block:

1. Convolution operation
2. Non-linear activation (usually ReLU)
3. Pooling
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253, 40–186,624–64,896–64,896–43,264–4096–4096–1000.
AlexNet – 2012 (16.4% error)

Training details:

- 2 GPUs for 5-6 days
- SGD with batch size 128, momentum 0.9, weight decay 0.0005
- initial learning rate 0.01, manually divided by 10 when validation error rate stopped improving
- ReLU non-linearities
- dropout with rate 0.5 on fully-connected layers
- data augmentation using translations and horizontal reflections (choosing random $224 \times 224$ patches from $256 \times 256$ images)
  - during inference, 10 patches are used (four corner patches and a center patch, as well as their reflections)
AlexNet – ReLU vs tanh

Figure 1 of paper "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky et al.
AlexNet built on already existing CNN architectures, mostly on LeNet, which achieved 0.8% test error on MNIST.

Figure 2 of paper "Gradient-Based Learning Applied to Document Recognition", http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.
The primary visual cortex recognizes Gabor functions.

Figure 9.18, page 370 of Deep Learning Book, http://deeplearningbook.org
Similar functions are recognized in the first layer of a CNN.
CNNs as Regularizers – Deep Prior

Figure 1 of paper “Deep Prior”, https://arxiv.org/abs/1712.05016

(a) Ground truth

(b) SRResNet [18], Trained

(c) Bicubic, Not trained

(d) Deep prior, Not trained
CNNs as Regularizers – Deep Prior

- Gradient Clipping
- Convolution
- CNNs
- AlexNet
- Deep Prior
- VGG
- Inception
- BatchNorm
- ResNet

Figure 7 of paper “Deep Prior”, https://arxiv.org/abs/1712.05016
Figure 5: **Inpainting diversity.** Left: original image (black pixels indicate holes). The remaining four images show results obtained using deep prior corresponding to different input vector $z$.

*Figure 5 of supplementary materials of paper "Deep Prior", https://arxiv.org/abs/1712.05016*
CNNs as Regularizers – Deep Prior

Deep Prior paper website with supplementary material
VGG – 2014 (6.8% error)

**ConvNet Configuration**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
<td>19 weight layers</td>
</tr>
</tbody>
</table>

- **input (224 x 224 RGB image)**
  - conv3-64
  - conv3-64
  - LRN
  - conv3-64
  - maxpool
- conv3-128
  - conv3-128
  - maxpool
- conv3-256
  - conv3-256
  - conv3-256
  - maxpool
- conv3-512
  - conv3-512
  - conv3-512
  - maxpool
- **maxpool**
  - FC-4096
  - FC-4096
  - FC-1000
  - soft-max

**Table 2: Number of parameters** (in millions).

<table>
<thead>
<tr>
<th>Network</th>
<th>A, A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of parameters</td>
<td></td>
<td>133</td>
<td>133</td>
<td>134</td>
<td>138</td>
</tr>
</tbody>
</table>

Figure 1 of paper "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556.

Figure 1 of paper "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567.

Figure 2 of paper "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556.
### Gradient Clipping

**VGG – 2014 (6.8% error)**

<table>
<thead>
<tr>
<th>Method</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
<th>top-5 test error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG (2 nets, multi-crop &amp; dense eval.)</td>
<td>23.7</td>
<td>6.8</td>
<td>6.8</td>
</tr>
<tr>
<td>VGG (1 net, multi-crop &amp; dense eval.)</td>
<td>24.4</td>
<td>7.1</td>
<td>7.0</td>
</tr>
<tr>
<td>VGG (ILSVRC submission, 7 nets, dense eval.)</td>
<td>24.7</td>
<td>7.5</td>
<td>7.3</td>
</tr>
<tr>
<td>GoogLeNet (Szegedy et al., 2014) (1 net)</td>
<td>-</td>
<td></td>
<td>7.9</td>
</tr>
<tr>
<td>GoogLeNet (Szegedy et al., 2014) (7 nets)</td>
<td>-</td>
<td></td>
<td>6.7</td>
</tr>
<tr>
<td>MSRA (He et al., 2014) (11 nets)</td>
<td>-</td>
<td>-</td>
<td>8.1</td>
</tr>
<tr>
<td>MSRA (He et al., 2014) (1 net)</td>
<td>27.9</td>
<td>9.1</td>
<td>9.1</td>
</tr>
<tr>
<td>Clarifai (Russakovsky et al., 2014) (multiple nets)</td>
<td>-</td>
<td>-</td>
<td>11.7</td>
</tr>
<tr>
<td>Clarifai (Russakovsky et al., 2014) (1 net)</td>
<td>-</td>
<td>-</td>
<td>12.5</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus (Zeiler &amp; Fergus, 2013) (6 nets)</td>
<td>36.0</td>
<td>14.7</td>
<td>14.8</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus (Zeiler &amp; Fergus, 2013) (1 net)</td>
<td>37.5</td>
<td>16.0</td>
<td>16.1</td>
</tr>
<tr>
<td>OverFeat (Sermanet et al., 2014) (7 nets)</td>
<td>34.0</td>
<td>13.2</td>
<td>13.6</td>
</tr>
<tr>
<td>OverFeat (Sermanet et al., 2014) (1 net)</td>
<td>35.7</td>
<td>14.2</td>
<td>-</td>
</tr>
<tr>
<td>Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)</td>
<td>38.1</td>
<td>16.4</td>
<td>16.4</td>
</tr>
<tr>
<td>Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)</td>
<td>40.7</td>
<td>18.2</td>
<td>-</td>
</tr>
</tbody>
</table>

*Figure 2 of paper “Very Deep Convolutional Networks For Large-Scale Image Recognition”, https://arxiv.org/abs/1409.1556.*
Inception (GoogLeNet) – 2014 (6.7% error)

Figure 2 of paper "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842.
Inception (GoogLeNet) – 2014 (6.7% error)

Figure 2 of paper "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842.
**Inception (GoogLeNet) – 2014 (6.7% error)**

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/ stride</th>
<th>output size</th>
<th>depth</th>
<th>#1×1</th>
<th>#3×3 reduce</th>
<th>#3×3</th>
<th>#5×5 reduce</th>
<th>#5×5 proj</th>
<th>pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7×7/2</td>
<td>112×112×64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>56×56×64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td>3×3/1</td>
<td>56×56×192</td>
<td>2</td>
<td>64</td>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>28×28×192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28×28×256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td></td>
<td>159</td>
<td>128M</td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28×28×480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>64</td>
<td></td>
<td>380K</td>
<td>304M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>14×14×480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14×14×512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>64</td>
<td></td>
<td>364K</td>
<td>73M</td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14×14×512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td></td>
<td>437K</td>
<td>88M</td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14×14×512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td></td>
<td>463K</td>
<td>100M</td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14×14×528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td></td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4e)</td>
<td>14×14×832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
<td></td>
<td>840K</td>
<td>170M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>7×7×832</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (5a)</td>
<td>7×7×832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
<td></td>
<td>1072K</td>
<td>54M</td>
</tr>
<tr>
<td>inception (5b)</td>
<td>7×7×1024</td>
<td>2</td>
<td>384</td>
<td>192</td>
<td>384</td>
<td>48</td>
<td>128</td>
<td>128</td>
<td></td>
<td>1388K</td>
<td>71M</td>
</tr>
<tr>
<td>avg pool</td>
<td>7×7/1</td>
<td>1×1×1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout (40%)</td>
<td>1×1×1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1×1×1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>softmax</td>
<td>1×1×1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 of paper "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842.
Inception (GoogLeNet) – 2014 (6.7% error)

Also note the two auxiliary classifiers (they have weight 0.3).
Batch Normalization

*Internal covariate shift* refers to the change in the distributions of hidden node activations due to the updates of network parameters during training.

Let \( \mathbf{x} = (x_1, \ldots, x_d) \) be \( d \)-dimensional input. We would like to normalize each dimension as

\[
\hat{x}_i = \frac{x_i - \mathbb{E}[x_i]}{\sqrt{\text{Var}[x_i]}}.
\]

Furthermore, it may be advantageous to learn suitable scale \( \gamma_i \) and shift \( \beta_i \) to produce normalized value

\[
y_i = \gamma_i \hat{x}_i + \beta_i.
\]
Consider a mini-batch of \( m \) examples \((\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})\).

**Batch normalizing transform** of the mini-batch is the following transformation.

**Inputs:** Mini-batch \((\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}), \varepsilon \in \mathbb{R}\)

**Outputs:** Normalized batch \((\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(m)})\)

- \( \mu \leftarrow \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^{(i)} \)
- \( \sigma^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}^{(i)} - \mu)^2 \)
- \( \hat{x}^{(i)} \leftarrow (\mathbf{x}^{(i)} - \mu)/\sqrt{\sigma^2 + \varepsilon} \)
- \( y^{(i)} \leftarrow \gamma \hat{x}^{(i)} + \beta \)

Batch normalization is commonly added just before a nonlinearity. Therefore, we replace \( f(W\mathbf{x} + \mathbf{b}) \) by \( f(BN(W\mathbf{x})) \).

During inference, \( \mu \) and \( \sigma^2 \) are fixed. They are either precomputed after training on the whole training data, or an exponential moving average is updated during training.
Batch Normalization

When a batch normalization is used on a fully connected layer, each neuron is normalized individually across the minibatch.

However, for convolutional networks we would like the normalization to honour their properties, most notably the shift invariance. We therefore normalize each channel across not only the minibatch, but also across all corresponding spatial/temporal locations.

Adapted from Figure 2 of paper "Group Normalization", https://arxiv.org/abs/1803.08494.
Inception with BatchNorm (4.8% error)

Figure 2: Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

<table>
<thead>
<tr>
<th>Model</th>
<th>Steps to 72.2%</th>
<th>Max accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>$31.0 \cdot 10^6$</td>
<td>72.2%</td>
</tr>
<tr>
<td>BN-Baseline</td>
<td>$13.3 \cdot 10^6$</td>
<td>72.7%</td>
</tr>
<tr>
<td>BN-x5</td>
<td>$2.1 \cdot 10^6$</td>
<td>73.0%</td>
</tr>
<tr>
<td>BN-x30</td>
<td>$2.7 \cdot 10^6$</td>
<td>74.8%</td>
</tr>
<tr>
<td>BN-x5-Sigmoid</td>
<td></td>
<td>69.8%</td>
</tr>
</tbody>
</table>

Figure 3: For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.

Inception v2 and v3 – 2015 (3.6% error)

Figure 1 of paper "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567.

Figure 3 of paper "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567.
Inception v2 and v3 – 2015 (3.6% error)

Figure 5. Inception modules where each $5 \times 5$ convolution is replaced by two $3 \times 3$ convolutions, as suggested by principle 3 of Section 2.

Figure 5 of paper “Rethinking the Inception Architecture for Computer Vision”, https://arxiv.org/abs/1512.00567.

Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the $17 \times 17$ grid. (The filter sizes are picked using principle 3)

Figure 6 of paper “Rethinking the Inception Architecture for Computer Vision”, https://arxiv.org/abs/1512.00567.

Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest ($8 \times 8$) grid to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by $1 \times 1$ convolutions) is increased compared to the spatial aggregation.

Figure 7 of paper “Rethinking the Inception Architecture for Computer Vision”, https://arxiv.org/abs/1512.00567.
## Inception v2 and v3 – 2015 (3.6% error)

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/stride or remarks</th>
<th>input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td>$3 \times 3 / 2$</td>
<td>$299 \times 299 \times 3$</td>
</tr>
<tr>
<td>conv</td>
<td>$3 \times 3 / 1$</td>
<td>$149 \times 149 \times 32$</td>
</tr>
<tr>
<td>conv padded</td>
<td>$3 \times 3 / 1$</td>
<td>$147 \times 147 \times 32$</td>
</tr>
<tr>
<td>pool</td>
<td>$3 \times 3 / 2$</td>
<td>$147 \times 147 \times 64$</td>
</tr>
<tr>
<td>conv</td>
<td>$3 \times 3 / 1$</td>
<td>$73 \times 73 \times 64$</td>
</tr>
<tr>
<td>conv</td>
<td>$3 \times 3 / 2$</td>
<td>$71 \times 71 \times 80$</td>
</tr>
<tr>
<td>conv</td>
<td>$3 \times 3 / 1$</td>
<td>$35 \times 35 \times 192$</td>
</tr>
<tr>
<td>3×Inception</td>
<td>As in figure 5</td>
<td>$35 \times 35 \times 288$</td>
</tr>
<tr>
<td>5×Inception</td>
<td>As in figure 6</td>
<td>$17 \times 17 \times 768$</td>
</tr>
<tr>
<td>2×Inception</td>
<td>As in figure 7</td>
<td>$8 \times 8 \times 1280$</td>
</tr>
<tr>
<td>pool</td>
<td>$8 \times 8$</td>
<td>$8 \times 8 \times 2048$</td>
</tr>
<tr>
<td>linear</td>
<td>logits</td>
<td>$1 \times 1 \times 2048$</td>
</tr>
<tr>
<td>softmax</td>
<td>classifier</td>
<td>$1 \times 1 \times 1000$</td>
</tr>
</tbody>
</table>

Table 1 of paper “Rethinking the Inception Architecture for Computer Vision”, https://arxiv.org/abs/1512.00567.
Inception v2 and v3 – 2015 (3.6% error)

<table>
<thead>
<tr>
<th>Network</th>
<th>Top-1 Error</th>
<th>Top-5 Error</th>
<th>Cost Bn Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogLeNet [20]</td>
<td>29%</td>
<td>9.2%</td>
<td>1.5</td>
</tr>
<tr>
<td>BN-GoogLeNet</td>
<td>26.8%</td>
<td>-</td>
<td>1.5</td>
</tr>
<tr>
<td>BN-Inception [7]</td>
<td>25.2%</td>
<td>7.8</td>
<td>2.0</td>
</tr>
<tr>
<td>Inception-v2</td>
<td>23.4%</td>
<td>-</td>
<td>3.8</td>
</tr>
<tr>
<td>Inception-v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSProp</td>
<td>23.1%</td>
<td>6.3</td>
<td>3.8</td>
</tr>
<tr>
<td>Inception-v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label Smoothing</td>
<td>22.8%</td>
<td>6.1</td>
<td>3.8</td>
</tr>
<tr>
<td>Inception-v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factorized 7 × 7</td>
<td>21.6%</td>
<td>5.8</td>
<td>4.8</td>
</tr>
<tr>
<td>Inception-v2 BN-auxiliary</td>
<td>21.2%</td>
<td>5.6%</td>
<td>4.8</td>
</tr>
</tbody>
</table>

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 2. Residual learning: a building block.

Figure 2 of paper "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></td>
<td>7×7, 64, stride 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3×3 max pool, stride 2</td>
<td></td>
</tr>
<tr>
<td>conv2-x</td>
<td>56×56</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>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
</tr>
<tr>
<td>conv3-x</td>
<td>28×28</td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[1×1, 128]</td>
<td>[1×1, 128]</td>
<td>[1×1, 128]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×4</td>
<td>×4</td>
<td>×4</td>
<td>×8</td>
</tr>
<tr>
<td>conv4-x</td>
<td>14×14</td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×6</td>
<td>×6</td>
<td>×23</td>
<td>×36</td>
</tr>
<tr>
<td>conv5-x</td>
<td>7×7</td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
</tr>
<tr>
<td>1×1</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>1.8×10^9</td>
<td>3.6×10^9</td>
<td>3.8×10^9</td>
<td>7.6×10^9</td>
<td>11.3×10^9</td>
<td></td>
</tr>
</tbody>
</table>

The residual connections cannot be applied directly when number of channels increase. The authors considered several alternatives, and chose the one where in case of channels increase a $1 \times 1$ convolution is used on the projections to match the required number of channels.
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 paper "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385.
ResNet – 2015 (3.6% error)

Figure 1 of paper “Visualizing the Loss Landscape of Neural Nets”, https://arxiv.org/abs/1712.09913.
ResNet – 2015 (3.6% error)

<table>
<thead>
<tr>
<th>method</th>
<th>top-1 err.</th>
<th>top-5 err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>-</td>
<td>8.43†</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>-</td>
<td>7.89</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>24.4</td>
<td>7.1</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>21.99</td>
<td>5.81</td>
</tr>
<tr>
<td>ResNet-34 B</td>
<td>21.84</td>
<td>5.71</td>
</tr>
<tr>
<td>ResNet-34 C</td>
<td>21.53</td>
<td>5.60</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>20.74</td>
<td>5.25</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>19.87</td>
<td>4.60</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>19.38</td>
<td>4.49</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>method</th>
<th>top-5 err. (test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>7.32</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>6.66</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>6.8</td>
</tr>
<tr>
<td>PReLU-net [13]</td>
<td>4.94</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>4.82</td>
</tr>
<tr>
<td><strong>ResNet (ILSVRC’15)</strong></td>
<td><strong>3.57</strong></td>
</tr>
</tbody>
</table>

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

---


Main Takeaways

- 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 number of channels.

- Final fully connected layers are not needed, global average pooling is usually enough.

- Batch normalization is a great regularization method for CNNs.
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 paper "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 (32×4d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td>7×7, 64, stride 2</td>
<td>7×7, 64, stride 2</td>
</tr>
<tr>
<td>conv2</td>
<td>56×56</td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1×1, 64</td>
<td>1×1, 128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3×3, 64</td>
<td>3×3, 128, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1×1, 256</td>
<td>1×1, 256</td>
</tr>
<tr>
<td>conv3</td>
<td>28×28</td>
<td>1×1, 128</td>
<td>1×1, 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3×3, 128</td>
<td>3×3, 256, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1×1, 512</td>
<td>1×1, 512</td>
</tr>
<tr>
<td>conv4</td>
<td>14×14</td>
<td>1×1, 256</td>
<td>1×1, 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3×3, 256</td>
<td>3×3, 512, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1×1, 1024</td>
<td>1×1, 1024</td>
</tr>
<tr>
<td>conv5</td>
<td>7×7</td>
<td>1×1, 512</td>
<td>1×1, 1024</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3×3, 5, 2</td>
<td>3×3, 1024, C=32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
</tr>
<tr>
<td></td>
<td>1×1</td>
<td>global average pool</td>
<td>global average pool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000-d fc, softmax</td>
<td>1000-d fc, softmax</td>
</tr>
<tr>
<td># params.</td>
<td></td>
<td><strong>25.5×10^6</strong></td>
<td><strong>25.0×10^6</strong></td>
</tr>
<tr>
<td>FLOPs</td>
<td></td>
<td><strong>4.1×10^9</strong></td>
<td><strong>4.2×10^9</strong></td>
</tr>
</tbody>
</table>

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

*Figure 5 of paper “Aggregated Residual Transformations for Deep Neural Networks”, https://arxiv.org/abs/1611.05431*
Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)