Convolutional Neural Networks

Charles Ollion - Olivier Grisel
used everywhere for vision

[Krizhevsky 2012]

[Ciresan et al. 2013]

[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]
Many other applications

Speech recognition & speech synthesis
Many other applications

Speech recognition & speech synthesis

Natural Language Processing
Many other applications

Speech recognition & speech synthesis

Natural Language Processing

Protein/DNA binding prediction
Many other applications

Speech recognition & speech synthesis

Natural Language Processing

Protein/DNA binding prediction

Any problem with a spatial (or sequential) structure
ConvNets for image classification

CNN = Convolutional Neural Networks = ConvNet
ConvNets for image classification

CNN = Convolutional Neural Networks = ConvNet


https://m2dsupsdclclass.github.io/lectures-labs/slides/04_conv_nets/index.html#1
Outline

Convolutions
Outline

Convolutions

CNNs for Image Classification
Outline

Convolutions

CNNs for Image Classification

CNN Architectures
Convolutions
Motivations

Standard Dense Layer for an image input:

```python
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

How many parameters in the Dense layer?
Motivations

Standard Dense Layer for an image input:

```python
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

How many parameters in the Dense layer?

$$640 \times 480 \times 3 \times 1000 + 1000 = 922M!$$
Motivations

Standard Dense Layer for an image input:

```python
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

How many parameters in the Dense layer?

\[640 \times 480 \times 3 \times 1000 + 1000 = 922M!\]

Spatial organization of the input is destroyed by Flatten
Motivations

Standard Dense Layer for an image input:

```python
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

How many parameters in the Dense layer?

\[ 640 \times 480 \times 3 \times 1000 + 1000 = 922M! \]

Spatial organization of the input is destroyed by `Flatten`

We never use Dense layers directly on large images. Most standard solution is **convolution** layers
Fully Connected Network: MLP

```python
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```
Fully Connected Network: MLP

```python
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

Convolutional Network

```python
input_image = Input(shape=(28, 28, 1))
x = Conv2D(32, 5, activation='relu')(input_image)
x = MaxPool2D(2, strides=2)(x)
x = Conv2D(64, 3, activation='relu')(x)
x = MaxPool2D(2, strides=2)(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
convnet = Model(inputs=input_image, outputs=x)
```
Fully Connected Network: MLP

```python
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

Convolutional Network

```python
input_image = Input(shape=(28, 28, 1))
x = Conv2D(32, 5, activation='relu')(input_image)
x = MaxPool2D(2, strides=2)(x)
x = Conv2D(64, 3, activation='relu')(x)
x = MaxPool2D(2, strides=2)(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
convnet = Model(inputs=input_image, outputs=x)
```

2D spatial organization of features preserved until Flatten.
Convolution in a neural network

- $x$ is a $3 \times 3$ chunk (dark area) of the image (*blue array*)
- Each output neuron is parametrized with the $3 \times 3$ weight matrix $\mathbf{w}$ (*small numbers*)

These slides extensively use convolution visualisation by V. Dumoulin available at
https://m2dsupsdiclass.github.io/lectures-labs/slides/04_conv_nets/index.html#1
Convolution in a neural network

- $x$ is a $3 \times 3$ chunk (dark area) of the image (*blue array*)
- Each output neuron is parametrized with the $3 \times 3$ weight matrix $w$ (*small numbers*)

The activation obtained by sliding the $3 \times 3$ window and computing:

$$z(x) = \text{relu}(w^T x + b)$$
Motivations

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance
Motivations

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: strong prior for vision!
Motivations

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: strong prior for vision!

Animal Vision Analogy

Hubel & Wiesel, RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX (1959)
Why Convolution

Discrete convolution (actually cross-correlation) between two functions \( f \) and \( g \):

\[
(f \ast g)(x) = \sum_{a+b=x} f(a) \cdot g(b) = \sum_{a} f(a) \cdot g(x + a)
\]
Why Convolution

Discrete convolution (actually cross-correlation) between two functions $f$ and $g$:

$$(f \star g)(x) = \sum_{a+b=x} f(a) \cdot g(b) = \sum_{a} f(a) \cdot g(x + a)$$

2D-convolutions (actually 2D cross-correlation):

$$(f \star g)(x, y) = \sum_{n} \sum_{m} f(n, m) \cdot g(x + n, y + m)$$
Why Convolution

Discrete convolution (actually cross-correlation) between two functions $f$ and $g$:

$$(f \ast g)(x) = \sum_{a+b=x} f(a). g(b) = \sum_a f(a). g(x + a)$$

2D-convolutions (actually 2D cross-correlation):

$$(f \ast g)(x, y) = \sum_n \sum_m f(n, m). g(x + n, y + m)$$

$f$ is a convolution kernel or filter applied to the 2-d map $g$ (our image)
Example: convolution image

- Image: $im$ of dimensions $5 \times 5$
- Kernel: $k$ of dimensions $3 \times 3$

$$(k \ast im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m). im(x + n - 1, y + m - 1)$$
Channels

Colored image = tensor of shape (height, width, channels)
Channels

Colored image = tensor of shape (height, width, channels)

Convolutions are usually computed for each channel and summed:
Channels

Colored image = tensor of shape (height, width, channels)

Convolutions are usually computed for each channel and summed:
Multiple convolutions
Multiple convolutions
Multiple convolutions
Multiple convolutions
Multiple convolutions

28x28x3

5x5x3x4

24x24x4
Multiple convolutions

- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension: $\text{length} - \text{kernel\_size} + 1$
Strides

- Strides: increment step size for the convolution operator
- Reduces the size of the output map

Example with kernel size $3 \times 3$ and a stride of 2 (image in blue)
Padding

- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s
Dealing with shapes

Kernel or Filter shape \( (F, F, C^i, C^o) \)

- \( F \times F \) kernel size,
- \( C^i \) input channels
- \( C^o \) output channels
Dealing with shapes

Kernel or Filter shape \((F, F, C^i, C^o)\)

- \(F \times F\) kernel size,
- \(C^i\) input channels
- \(C^o\) output channels

Number of parameters: \((F \times F \times C^i + 1) \times C^o\)
Dealing with shapes

Kernel or Filter shape \((F, F, C^i, C^o)\)

- \(F \times F\) kernel size,
- \(C^i\) input channels
- \(C^o\) output channels

Number of parameters: \((F \times F \times C^i + 1) \times C^o\)

Activations or Feature maps shape:

- Input \((W^i, H^i, C^i)\)
- Output \((W^o, H^o, C^o)\)
Dealing with shapes

**Kernel or Filter** shape \((F, F, C^i, C^o)\)

- \(F \times F\) kernel size,
- \(C^i\) input channels
- \(C^o\) output channels

Number of parameters: \((F \times F \times C^i + 1) \times C^o\)

**Activations or Feature maps** shape:

- Input \((W^i, H^i, C^i)\)
- Output \((W^o, H^o, C^o)\)

\[ W^o = (W^i - F + 2P)/S + 1 \]
Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units
Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units

Schematic from Stanford

max pool with 2x2 filters and stride 2

https://m2dsupsdiclass.github.io/lectures-labs/slides/04_conv_nets/index.html#1
Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units
Architectures
Classic ConvNet Architecture

Input
Classic ConvNet Architecture

Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2
Classic ConvNet Architecture

Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

Output

- Fully connected layers
- Softmax
AlexNet

AlexNet

First conv layer: kernel 11x11x3x96 stride 4

AlexNet

First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: (11, 11, 3, 96)
- Output shape: (55, 55, 96)
- Number of parameters: 34,944
- Equivalent MLP parameters: $43.7 \times 1e9$


https://m2dsupsdlclass.github.io/lectures-labs/slides/04_conv_nets/index.html#1
AlexNet

INPUT:      [227x227x3]
CONV1:      [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1:  [27x27x96]   3x3 filters at stride 2
CONV2:      [27x27x256]  256 5x5 filters at stride 1, pad 2
MAX POOL2:  [13x13x256]  3x3 filters at stride 2
CONV3:      [13x13x384]  384 3x3 filters at stride 1, pad 1
CONV4:      [13x13x384]  384 3x3 filters at stride 1, pad 1
CONV5:      [13x13x256]  256 3x3 filters at stride 1, pad 1
MAX POOL3:  [6x6x256]    3x3 filters at stride 2
FC6:        [4096]       4096 neurons
FC7:        [4096]       4096 neurons
FC8:        [1000]       1000 neurons (softmax logits)
Hierarchical representation

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
VGG-16

VGG in Keras

```python
model.add(Convolution2D(64, 3, 3, activation='relu', input_shape=(3, 224, 224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
```
## Memory and Parameters

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Shape</th>
<th>Activation Maps</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT:</td>
<td>[224x224x3]</td>
<td>150K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-64:</td>
<td>[224x224x64]</td>
<td>3.2M</td>
<td>(3x3x3)x64 = 1,728</td>
</tr>
<tr>
<td>CONV3-64:</td>
<td>[224x224x64]</td>
<td>3.2M</td>
<td>(3x3x64)x64 = 36,864</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[112x112x64]</td>
<td>800K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-128:</td>
<td>[112x112x128]</td>
<td>1.6M</td>
<td>(3x3x64)x128 = 73,728</td>
</tr>
<tr>
<td>CONV3-128:</td>
<td>[112x112x128]</td>
<td>1.6M</td>
<td>(3x3x128)x128 = 147,456</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[56x56x128]</td>
<td>400K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-256:</td>
<td>[56x56x256]</td>
<td>800K</td>
<td>(3x3x128)x256 = 294,912</td>
</tr>
<tr>
<td>CONV3-256:</td>
<td>[56x56x256]</td>
<td>800K</td>
<td>(3x3x256)x256 = 589,824</td>
</tr>
<tr>
<td>CONV3-256:</td>
<td>[56x56x256]</td>
<td>800K</td>
<td>(3x3x256)x256 = 589,824</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[28x28x256]</td>
<td>200K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[28x28x512]</td>
<td>400K</td>
<td>(3x3x256)x512 = 1,179,648</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[28x28x512]</td>
<td>400K</td>
<td>(3x3x512)x512 = 2,359,296</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[28x28x512]</td>
<td>400K</td>
<td>(3x3x512)x512 = 2,359,296</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[14x14x512]</td>
<td>100K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[14x14x512]</td>
<td>100K</td>
<td>(3x3x512)x512 = 2,359,296</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[14x14x512]</td>
<td>100K</td>
<td>(3x3x512)x512 = 2,359,296</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[14x14x512]</td>
<td>100K</td>
<td>(3x3x512)x512 = 2,359,296</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[7x7x512]</td>
<td>25K</td>
<td>0</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x4096]</td>
<td>4096</td>
<td>7x7x512x4096 = 102,760,448</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x4096]</td>
<td>4096</td>
<td>4096x4096 = 16,777,216</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x1000]</td>
<td>1000</td>
<td>4096x1000 = 4,096,000</td>
</tr>
</tbody>
</table>

**TOTAL activations**: 24M x 4 bytes $\approx$ 93MB / image (x2 for backward)

**TOTAL parameters**: 138M x 4 bytes $\approx$ 552MB (x2 for plain SGD, x4 for Adam)
## Memory and Parameters

<table>
<thead>
<tr>
<th>Layer</th>
<th>Activation maps</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT:</td>
<td>[224x224x3]</td>
<td>150K</td>
</tr>
<tr>
<td>CONV3-64:</td>
<td>[224x224x64]</td>
<td>3.2M</td>
</tr>
<tr>
<td>CONV3-64:</td>
<td>[224x224x64]</td>
<td>3.2M</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[112x112x64]</td>
<td>800K</td>
</tr>
<tr>
<td>CONV3-128</td>
<td>[112x112x128]</td>
<td>1.6M</td>
</tr>
<tr>
<td>CONV3-128</td>
<td>[112x112x128]</td>
<td>1.6M</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[56x56x128]</td>
<td>400K</td>
</tr>
<tr>
<td>CONV3-256</td>
<td>[56x56x256]</td>
<td>800K</td>
</tr>
<tr>
<td>CONV3-256</td>
<td>[56x56x256]</td>
<td>800K</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[28x28x256]</td>
<td>200K</td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[28x28x512]</td>
<td>400K</td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[28x28x512]</td>
<td>400K</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[14x14x512]</td>
<td>100K</td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[14x14x512]</td>
<td>100K</td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[14x14x512]</td>
<td>100K</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[7x7x512]</td>
<td>25K</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x4096]</td>
<td>4096</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x4096]</td>
<td>4096</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x1000]</td>
<td>1000</td>
</tr>
</tbody>
</table>

TOTAL activations: 24M x 4 bytes ~= 93MB / image (x2 for backward)
TOTAL parameters: 138M x 4 bytes ~= 552MB (x2 for plain SGD, x4 for Adam)
ResNet

Even deeper models:

34, 50, 101, 152 layers

ResNet

A block learns the residual w.r.t. identity

ResNet

A block learns the residual w.r.t. identity

Figure 2. Residual learning: a building block.

- Good optimization properties

ResNet

ResNet50 Compared to VGG:

Superior accuracy in all vision tasks
5.25% top-5 error vs 7.1%

ResNet

ResNet50 Compared to VGG:

Superior accuracy in all vision tasks
5.25% top-5 error vs 7.1%

Less parameters
25M vs 138M


https://m2dsupsdlclass.github.io/lectures-labs/slides/04_conv_nets/index.html#1
ResNet

ResNet50 Compared to VGG:

Superior accuracy in all vision tasks
5.25% top-5 error vs 7.1%

Less parameters
25M vs 138M

Computational complexity
3.8B Flops vs 15.3B Flops

ResNet

ResNet50 Compared to VGG:

Superior accuracy in all vision tasks 5.25% top-5 error vs 7.1%

Less parameters 25M vs 138M

Computational complexity 3.8B Flops vs 15.3B Flops

Fully Convolutional until the last layer

Deeper is better

ImageNet experiments

ImageNet Classification top-5 error (%)
State of the art

- Finding right architectures: Active area or research
State of the art

- Finding right architectures: Active area or research

<table>
<thead>
<tr>
<th>Model</th>
<th>Params</th>
<th>×+</th>
<th>1/5-Acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V3</td>
<td>23.8M</td>
<td>5.72B</td>
<td>78.0 / 93.9</td>
</tr>
<tr>
<td>Xception</td>
<td>22.8M</td>
<td>8.37B</td>
<td>79.0 / 94.5</td>
</tr>
<tr>
<td>Inception ResNet V2</td>
<td>55.8M</td>
<td>13.2B</td>
<td>80.4 / 95.3</td>
</tr>
<tr>
<td>ResNeXt-101 (64x4d)</td>
<td>83.6M</td>
<td>31.5B</td>
<td>80.9 / 95.6</td>
</tr>
<tr>
<td>PolyNet</td>
<td>92.0M</td>
<td>34.7B</td>
<td>81.3 / 95.8</td>
</tr>
<tr>
<td>Dual-Path-Net-131</td>
<td>79.5M</td>
<td>32.0B</td>
<td>81.5 / 95.8</td>
</tr>
<tr>
<td>Squeeze-Excite-Net</td>
<td>145.8M</td>
<td>42.3B</td>
<td>82.7 / <strong>96.2</strong></td>
</tr>
<tr>
<td>GeNet-2</td>
<td>156M</td>
<td>–</td>
<td>72.1 / 90.4</td>
</tr>
<tr>
<td>Block-QNN-B, N=3</td>
<td>–</td>
<td>–</td>
<td>75.7 / 92.6</td>
</tr>
<tr>
<td>Hierarchical (2, 64)</td>
<td>64M</td>
<td>–</td>
<td>79.7 / 94.8</td>
</tr>
<tr>
<td>PNASNet-5 (4, 216)</td>
<td>86.1M</td>
<td>25.0B</td>
<td><strong>82.9</strong> / 96.1</td>
</tr>
<tr>
<td>NASNet-A (6, 168)</td>
<td>88.9M</td>
<td>23.8B</td>
<td>82.7 / <strong>96.2</strong></td>
</tr>
<tr>
<td>AmoebaNet-B (6, 190)</td>
<td>84.0M</td>
<td>22.3B</td>
<td>82.3 / 96.1</td>
</tr>
<tr>
<td>AmoebaNet-C (6, 168)</td>
<td>85.5M</td>
<td>22.5B</td>
<td>82.7 / 96.1</td>
</tr>
<tr>
<td>AmoebaNet-A (6, 190)</td>
<td>86.7M</td>
<td>23.1B</td>
<td>82.8 / 96.1</td>
</tr>
<tr>
<td>AmoebaNet-A (6, 204)</td>
<td>99.6M</td>
<td>26.2B</td>
<td>82.8 / <strong>96.2</strong></td>
</tr>
</tbody>
</table>
State of the art

- Finding right architectures: Active area or research

Modular building blocks engineering
State of the art

- Finding right architectures: Active area or research

Modular building blocks engineering

see also DenseNets, Wide ResNets, Fractal ResNets, ResNeXts, Pyramidal ResNets
State of the art

- Finding right architectures: Active area or research

Automated Architecture search:

- reinforcement learning
- evolutionary algorithms
Comparison of models

Top 1-accuracy, performance and size on ImageNet


https://m2dsupsdlclass.github.io/lectures-labs/slides/04_conv_nets/index.html#1
Pre-trained models
Pre-trained models

Training a model on ImageNet from scratch takes \textit{days or weeks}. 
Pre-trained models

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!
Pre-trained models

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!

Transfer learning

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images
Pre-trained models

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!

Fine-tuning

Retraining the (some) parameters of the network (given enough data)
Pre-trained models

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!

**Fine-tuning**

Retraining the (some) parameters of the network (given enough data)

- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
Pre-trained models

Training a model on ImageNet from scratch takes \textbf{days or weeks}.

Many models trained on ImageNet and their weights are publicly available!

Fine-tuning

Retraining the (some) parameters of the network (given enough data)

- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning
Data Augmentation
Data Augmentation
Data Augmentation

With Keras:

```python
from keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    channel_shift_range=9,
    fill_mode='nearest'
)

train_flow = image_gen.flow_from_directory(train_folder)
model.fit_generator(train_flow, train_flow.n)
```
Lab 4: Back here in 15min!