Convolutional Neural Networks
Neural Nets: a particularly useful Black Box

Convolutional Neural Network

image X

label Y

“Penguin”
Classic Object Recognition

Feature extractors

- Edges
- Texture
- Colors

Classifier

- Segments
- Parts

“Penguin”

image X

label Y

Slide by Philip Isola
Classic Object Recognition

Feature extractors

- Edges
- Texture
- Colors

Classifier

- Segments
- Parts

image $X$

label $Y$

“Penguin”

Slide by Philip Isola
Learning Features

Feature extractors

- Edges
- Texture
- Colors

image X → Learned → label Y
Neural Network: algorithm + feature + data!
Vanilla (fully-connected) Neural Networks
Fully Connected Layer

Example: 200x200 image
40K hidden units

~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..
Locally Connected Layer

Example: 200x200 image  
40K hidden units  
Filter size: 10x10  
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels
Convolutional Layer
Convolutional Layer
Convolutional Layer

Ranzato
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
before:

input layer  hidden layer  output layer

now:

Fei-Fei Li & Andrej Karpathy
Neural Network vs ConvNet

- **Fully connected:**
  - Connects to everything

- **Convnet:**
  - Connects locally

Slide credit: Karpathy and Fei-Fei
Convolution Layer

32x32x3 image

width: 32
height: 32
depth: 3
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
**Convolution Layer**

32x32x3 image

5x5x3 filter

*Convolve* the filter with the image i.e. “slide over the image spatially, computing dot products”

Filters always extend the full depth of the input volume
Convolution Layer

32x32x3 image
5x5x3 filter \( w \)

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. \( 5 \times 5 \times 3 = 75 \)-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"
The brain/neuron view of CONV Layer

E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume.
convolving the first filter in the input gives the first slice of depth in the output volume.
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

- Conv, ReLU e.g. 6 5x5x3 filters
- Conv, ReLU e.g. 10 5x5x6 filters
- Conv, ReLU
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
7x7 input (spatially)
assume 3x3 filter
7x7 input (spatially)
assume 3x3 filter
7x7 input (spatially)
assume 3x3 filter
7x7 input (spatially) 
assume 3x3 filter
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
7x7 input (spatially) assume 3x3 filter applied \textit{with stride} 2
Stride

7x7 input (spatially) assume 3x3 filter applied with stride 2
7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):
- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33\)
In practice: Common to zero pad the border

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

\[(\text{N} - \text{F}) / \text{stride} + 1\]

(recall:)

\[(\text{N} - \text{F}) / \text{stride} + 1\]
In practice: Common to zero pad the border

- e.g. input 7x7
- 3x3 filter, applied with **stride 1**
- **pad with 1 pixel** border => what is the output?

**7x7 output!**

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```
### In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7

3x3 filter, applied with **stride 1**

**pad with 1 pixel** border => what is the output?

#### 7x7 output!

In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with 

\[
\frac{F-1}{2}. \text{ (will preserve size spatially)}
\]

e.g. F = 3 => zero pad with 1

F = 5 => zero pad with 2

F = 7 => zero pad with 3
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?
Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:
MAX POOLING

Single depth slice

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
ConvNets: Typical Stage

One stage (zoom)

\[ \text{Convol.} \rightarrow \text{Pooling} \]

courtesy of K. Kavukcuoglu
Case Study: LeNet-5

(Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]
Handwritten digit classification

[Courtesy of Yann LeCun]
<table>
<thead>
<tr>
<th></th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>227x22</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Each block is a HxWxC volume.
You transform one volume to another with convolution.
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: \( \frac{(227-11)}{4} + 1 = 55 \)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume [55x55x96]
Parameters: (11*11*3)*96 = 35K
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: $(55-3)/2+1 = 27$
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2
## VGG16

<table>
<thead>
<tr>
<th></th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>224x22</td>
<td>224x22</td>
<td>112x112</td>
<td>56x56</td>
<td>28x28</td>
<td>14x14</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td></td>
<td>43</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>512</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

All filters 3x3  
All filters followed by ReLU
### ConvNet Configuration

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input</th>
<th>Output</th>
<th>Memory</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>[224x224x3]</td>
<td></td>
<td>224x224x3=150K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-64</td>
<td>[224x224x64]</td>
<td>memory: 224x224x64=3.2M</td>
<td>params: (3<em>3</em>64) = 1,728</td>
<td></td>
</tr>
<tr>
<td>CONV3-64</td>
<td>[224x224x64]</td>
<td>memory: 224x224x64=3.2M</td>
<td>params: (3<em>3</em>64) = 36,864</td>
<td></td>
</tr>
<tr>
<td>POOL2</td>
<td>[112x112x64]</td>
<td>memory: 112x112x64=800K</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CONV3-128</td>
<td>[112x112x128]</td>
<td>memory: 112x112x128=1.6M</td>
<td>params: (3<em>3</em>128) = 73,728</td>
<td></td>
</tr>
<tr>
<td>CONV3-128</td>
<td>[112x112x128]</td>
<td>memory: 112x112x128=1.6M</td>
<td>params: (3<em>3</em>128) = 147,456</td>
<td></td>
</tr>
<tr>
<td>POOL2</td>
<td>[56x56x128]</td>
<td>memory: 56x56x128=400K</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CONV3-256</td>
<td>[56x56x256]</td>
<td>memory: 56x56x256=800K</td>
<td>params: (3<em>3</em>256) = 294,912</td>
<td></td>
</tr>
<tr>
<td>CONV3-256</td>
<td>[56x56x256]</td>
<td>memory: 56x56x256=800K</td>
<td>params: (3<em>3</em>256) = 589,824</td>
<td></td>
</tr>
<tr>
<td>CONV3-256</td>
<td>[56x56x256]</td>
<td>memory: 56x56x256=800K</td>
<td>params: (3<em>3</em>256) = 589,824</td>
<td></td>
</tr>
<tr>
<td>POOL2</td>
<td>[28x28x256]</td>
<td>memory: 28x28x256=200K</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[28x28x512]</td>
<td>memory: 28x28x512=400K</td>
<td>params: (3<em>3</em>512) = 1,179,648</td>
<td></td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[28x28x512]</td>
<td>memory: 28x28x512=400K</td>
<td>params: (3<em>3</em>512) = 2,359,296</td>
<td></td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[28x28x512]</td>
<td>memory: 28x28x512=400K</td>
<td>params: (3<em>3</em>512) = 2,359,296</td>
<td></td>
</tr>
<tr>
<td>POOL2</td>
<td>[14x14x512]</td>
<td>memory: 14x14x512=100K</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[14x14x512]</td>
<td>memory: 14x14x512=100K</td>
<td>params: (3<em>3</em>512) = 2,359,296</td>
<td></td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[14x14x512]</td>
<td>memory: 14x14x512=100K</td>
<td>params: (3<em>3</em>512) = 2,359,296</td>
<td></td>
</tr>
<tr>
<td>CONV3-512</td>
<td>[14x14x512]</td>
<td>memory: 14x14x512=100K</td>
<td>params: (3<em>3</em>512) = 2,359,296</td>
<td></td>
</tr>
<tr>
<td>POOL2</td>
<td>[7x7x512]</td>
<td>memory: 7x7x512=25K</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FC: [1x1x4096]</td>
<td></td>
<td>memory: 4096</td>
<td>params: 7x7x512*4096 = 102,760,448</td>
<td></td>
</tr>
<tr>
<td>FC: [1x1x4096]</td>
<td></td>
<td>memory: 4096</td>
<td>params: 4096*4096 = 16,777,216</td>
<td></td>
</tr>
<tr>
<td>FC: [1x1x1000]</td>
<td></td>
<td>memory: 1000</td>
<td>params: 4096*1000 = 4,096,000</td>
<td></td>
</tr>
</tbody>
</table>
Case Study: ResNet

[He et al., 2015]

spatial dimension only 56x56!
Case Study: ResNet  [He et al., 2015]

- Plain net
  \[ x \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow H(x) \]

- Residual net
  \[ F(x) \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow \text{weight layer} \]
  \[ H(x) = F(x) + x \rightarrow \text{relu} \]

\( x \)

Identity
Convert HxW image into a F-dimensional vector

- What’s the probability this image is a cloudy (F=1)
- Which of 1000 categories is this image? (F=1000)
- At what GPS coord was this image taken? (F=2)
- Identify the X,Y coordinates of 28 face keypoints of an image of a human (F=56)
e.g. Image Classification

Slide by David Fouhey
ImageNet Challenge (1000 object classes), Fei-Fei et al.
Revolution of Depth


(slide from Kaiming He’s recent presentation)
What is your data not big enough?

Horizontal Flip  Color Jitter  Image Cropping
Training a CNN – Augmentation

• Apply transformations that don’t affect the output
• Produces more data but you have to be careful that it doesn’t change the meaning of the output
Project 4

Slide by David Fouhey