Convolutional Neural Networks

Erkut Erdem
Hacettepe University
Computer Vision Lab (HUCVL)
Convolution Layer

32x32x3 image

32 \text{ height}

32 \text{ width}

3 \text{ depth}
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 \( \omega \)

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)

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

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

consider a second, green filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

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!
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

- **CONV, ReLU**
  - e.g. $6 \times 5 \times 3$ filters
  - $3 \rightarrow 6 \rightarrow 28 \rightarrow 24$

- **CONV, ReLU**
  - e.g. $10 \times 5 \times 6$ filters
  - $3 \rightarrow 6 \rightarrow 28 \rightarrow 24$
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x, y] \ast g[x, y] = \sum_{n_1 = -\infty}^{\infty} \sum_{n_2 = -\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2] \]

elementwise multiplication and sum of a filter and the signal (image)
Preview

RELU RELU RELU RELU RELU
CONV CONV CONV CONV CONV

POOL POOL POOL POOL

FC

car
truck
airplane
ship
horse
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

32
32
3

1
28
28
1
A closer look at spatial dimensions:

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

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

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

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

7x7 input (spatially) assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

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

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

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:

- stride 1 => \((7 - 3)/1 + 1 = 5\)
- stride 2 => \((7 - 3)/2 + 1 = 3\)
- stride 3 => \((7 - 3)/3 + 1 = 2.33\)

\[\]
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?

(recall:)
(N - F) / 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!
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!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with \((F-1)/2\). (will preserve size spatially)
e.g. \(F = 3\) => zero pad with 1
\(F = 5\) => zero pad with 2
\(F = 7\) => zero pad with 3

<table>
<thead>
<tr>
<th></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>
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Recap: Convolution Layer

Convolving a $3 \times 3$ kernel over a $4 \times 4$ input using unit strides (i.e., $i = 4, k = 3, s = 1$ and $p = 0$).
Computing the output values of a 2D discrete convolution $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, and $p_1 = p_2 = 1$.
Examples

time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples

time:

Input volume: \textbf{32x32x3}
10 5x5 filters with stride 1, pad 2

Output volume size:
\(\frac{32+2*2-5}{1}+1 = 32\) spatially, so
\textbf{32x32x10}
Examples

time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Examples
time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has \(5 \times 5 \times 3 + 1 = 76\) params (+1 for bias)

\(\Rightarrow 76 \times 10 = 760\)
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
Common settings:

- $K = \text{(powers of 2, e.g. } 32, 64, 128, 512)$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ? \text{ (whatever fits)}$
- $F = 1, S = 1, P = 0$
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
The brain/neuron view of CONV Layer

A 32x32x3 image

A 5x5x3 filter

1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number: the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)

It’s just a neuron with local connectivity...
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”
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
Activation Functions

Activation function diagram:

- $w_0 x_0$: synapse from axon to dendrite
- $w_1 x_1$, $w_2 x_2$: inputs from dendrites
- $\sum_i w_i x_i + b$: weighted sum
- $f$: activation function (e.g., sigmoid, ReLU)
- Output axon

Mathematical representation:

$$f \left( \sum_i w_i x_i + b \right)$$
Activation Functions

**Sigmoid**

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**

\[ \tanh(x) \]

**ReLU**

\[ \text{max}(0, x) \]
Activation Functions

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. \(\exp()\) is a bit compute expensive

Sigmoid
Activation Functions

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

\[ \text{tanh}(x) \]

[LeCun et al., 1991]
Activation Functions

- Computes $f(x) = \max(0, x)$
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

ReLU
(Rectified Linear Unit)

[Krizhevsky et al., 2012]
two more layers to go: POOL/FC
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:

![Diagram of pooling layer](image)
Max Pooling

Single depth

Max pool with 2x2 filters and stride 2
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
Common settings:

F = 2, S = 2
F = 3, S = 2

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks.
[ConvNetJS demo: training on CIFAR-10]

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Case studies
Case Study: LeNet-5 [LeCun et al., 1998]

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]
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: \((227-11)/4+1 = 55\)
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: $(11 \times 11 \times 3) \times 96 = 35K$
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: 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)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

best model

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 73,728
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

(not counting biases)
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
INPUT: [224x224x3]  memory: 224*224*3 = 150K  params: 0
CONV3-64: [224x224x64]  memory: 224*224*64 = 3.2M  params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64 = 3.2M  params: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory: 112*112*64 = 800K  params: 0
CONV3-128: [112x112x128]  memory: 112*112*128 = 1.6M  params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory: 112*112*128 = 1.6M  params: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory: 56*56*128 = 400K  params: 0
CONV3-256: [56x56x256]  memory: 56*56*256 = 800K  params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory: 56*56*256 = 800K  params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory: 56*56*256 = 800K  params: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory: 28*28*256 = 200K  params: 0
CONV3-512: [28x28x512]  memory: 28*28*512 = 400K  params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory: 28*28*512 = 400K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory: 28*28*512 = 400K  params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory: 14*14*512 = 100K  params: 0
CONV3-512: [14x14x512]  memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory: 7*7*512 = 25K  params: 0
FC: [1x1x4096]  memory: 4096  params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory: 4096  params: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory: 1000  params: 4096*1000 = 4,096,000

**TOTAL memory:** 24M * 4 bytes \(\sim=\) 93MB / image (only forward! \(\sim\)*2 for bwd)

**TOTAL params:** 138M parameters

*Note:*

- Most memory is in early CONV
- Most params are in late FC

*slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson*
Case Study: GoogLeNet [Szegedy et al., 2014]

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: ResNet \[\text{He et al., 2015}\]
ILSVRC 2015 winner
(3.6\% top 5 error)

MSRA @ ILSVRC & COCO 2015 Competitions

• **1st places in all five main tracks**
  • ImageNet Classification: “Ultra-deep” (quote Yann) 152-layer nets
  • ImageNet Detection: 16\% better than 2nd
  • ImageNet Localization: 27\% better than 2nd
  • COCO Detection: 11\% better than 2nd
  • COCO Segmentation: 12\% better than 2nd

*improvements are relative numbers


Slide from Kaiming He’s recent presentation [https://www.youtube.com/watch?v=1PGLj-uKT1w](https://www.youtube.com/watch?v=1PGLj-uKT1w)
Case Study: ResNet  

[He et al., 2015]  
ILSVRC 2015 winner  
(3.6% top 5 error)

2-3 weeks of training on 8 GPU machine  
at runtime: faster than a VGGNet!  
(even though it has 8x more layers)

(slide from Kaiming He’s recent presentation)
Case Study: ResNet [He et al., 2015]

34-layer plain

7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128

34-layer residual

7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128

224x224x3

spatial dimension only 56x56!
Visualizing CNN (Layer 1)

Visualizing CNN (Layer 2)

Visualizing CNN (Layer 3)

Visualizing CNN (Layer 4)

Part that Triggered Filter

Top Image Patches

Visualizing CNN (Layer 5)

Part that Triggered Filter

Top Image Patches

Tips and Tricks
• Shuffle the training samples

• Use Dropout and Batch Normalization for regularization
Input representation

- Centered (0-mean) RGB values.

“Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image”
Data Augmentation

• The neural net has 60M real-valued parameters and 650,000 neurons

• It overfits a lot. Therefore, they train on 224x224 patches extracted randomly from 256x256 images, and also their horizontal reflections.

“This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter-dependent.”

[Krizhevsky et al. 2012]
Data Augmentation

• Alter the intensities of the RGB channels in training images.

“Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1…This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.”

[Krizhevsky et al. 2012]
Data Augmentation

Horizontal flips
Data Augmentation

Get creative!

Random mix/combinations of:
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Transfer Learning with ConvNets

1. Train on Imagenet
Transfer Learning with ConvNets

1. Train on Imagenet

2. Small dataset: **feature extractor**

   - Freeze these
   - Train this
Transfer Learning with ConvNets

1. Train on Imagenet
2. Small dataset: feature extractor
   Freeze these
   Train this
3. Medium dataset: finetuning
   more data = retrain more of the network (or all of it)
   Freeze these
   Train this
Transfer Learning with ConvNets

1. Train on Imagenet

2. Small dataset: feature extractor
   - Freeze these
   - Train this

3. Medium dataset: finetuning
   - More data = retrain more of the network (or all of it)
   - Freeze these
   - Tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers
   - Train this
Today ConvNets are everywhere

Classification

Retrieval

[Krizhevsky 2012]
Today ConvNets are everywhere

Detection

Segmentation

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

[Farabet et al., 2012]
Today ConvNets are everywhere

self-driving cars

NVIDIA Tegra X1
Today ConvNets are everywhere

[Simonyan et al. 2014]

[Goodfellow 2014]
Today ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]
Today ConvNets are everywhere

[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]
Today ConvNets are everywhere
Today ConvNets are everywhere

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010
Today ConvNets are everywhere

<table>
<thead>
<tr>
<th>Describes without errors</th>
<th>Describes with minor errors</th>
<th>Somewhat related to the image</th>
<th>Unrelated to the image</th>
</tr>
</thead>
<tbody>
<tr>
<td>A person riding a motorcycle on a dirt road.</td>
<td>Two dogs play in the grass.</td>
<td>A skateboarder does a trick on a ramp.</td>
<td>A dog is jumping to catch a frisbee.</td>
</tr>
<tr>
<td>A group of young people playing a game of frisbee.</td>
<td>Two hockey players are fighting over the puck.</td>
<td>A little girl in a pink hat is blowing bubbles.</td>
<td>A refrigerator filled with lots of food and drinks.</td>
</tr>
<tr>
<td>A herd of elephants walking across a dry grass field.</td>
<td>A close up of a cat laying on a couch.</td>
<td>A red motorcycle parked on the side of the road.</td>
<td>A yellow school bus parked in a parking lot.</td>
</tr>
</tbody>
</table>

[Vinyals et al., 2015]
Today ConvNets are everywhere