Lecture 14:
– Deep Convolutional Networks
Announcement

• Midterm exam on Nov 30, 2018 at 09.00 in rooms D6 & D9

• More info in Piazza
Last time… Three key ideas

• (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations

• End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extract

• Distributed Representations
  - No single neuron “encodes” everything
  - Groups of neurons work together
Last time... Intro. to Deep Learning

VISION

SIFT/HOG → K-Means/pooling → classifier → “car”
  fixed          unsupervised       supervised

SPEECH

MFCC → Mixture of Gaussians → classifier → ‘d ē p’
  fixed          unsupervised       supervised

NLP

This burrito place is yummy and fun! → Parse Tree Syntactic → n-grams → classifier → “+”
  fixed          unsupervised       supervised
Last time... **Intro. to Deep Learning**

- **“Shallow” models**
  
  ![Car Image](image1)
  
<table>
<thead>
<tr>
<th>hand-crafted Feature Extractor</th>
<th>“Simple” Trainable Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fixed</strong></td>
<td><strong>learned</strong></td>
</tr>
</tbody>
</table>

- **Deep models**
  
  ![Car Image](image2)

<table>
<thead>
<tr>
<th>Trainable Feature-Transform / Classifier</th>
<th>Trainable Feature-Transform / Classifier</th>
<th>Trainable Feature-Transform / Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learned Internal Representations</strong></td>
<td><strong>Learned Internal Representations</strong></td>
<td><strong>Learned Internal Representations</strong></td>
</tr>
</tbody>
</table>
Deep Convolutional Neural Networks
Convolutions

- Images typically have invariant patterns
  - E.g., directional gradients are translational invariant:

- Apply convolution to local sliding windows
Convolution Filters

- Applies to an image patch $x$
  - Converts local window into single value
  - Slide across image

$$x \otimes W = \sum_{ij} W_{ij} x_{ij}$$

Left-to-Right Edge Detector

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>+1</td>
</tr>
</tbody>
</table>
Gabor Filters

- Most common low-level convolutions for computer vision

Example $W$:

```
-1  0  +1
-1  0  +1
-1  0  +1
```

http://en.wikipedia.org/wiki/Gabor_filter
Gaussian Blur Filters

- Weights decay according to Gaussian Distribution
  - Variance term controls radius

Example W:
Apply per RGB Channel

- Black = 0
- White = Positive

http://en.wikipedia.org/wiki/Gaussian_blur
Convolutional Neural Networks
Convolusion Layer

32x32x3 image

32 height

32 width

3 depth
Convolution Layer

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

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

consider a second, green filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation maps

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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.

```
3 32
CONV, ReLU e.g. 6 5x5x3 filters
6 28
```

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
Preview

[From recent Yann LeCun slides]

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)

---

**example 5x5 filters**

(32 total)

one filter =>

one activation map
Preview

[Image of a car with a neural network diagram showing layers of convolution (CONV), rectified linear unit (RELU), and pooling (POOL), with output labels for 'car', 'truck', 'airplane', 'ship', and 'horse'.]
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
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:)
\[(N - F) / \text{stride} + 1\]
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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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 (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
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.

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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 $p = 0$).

$W = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \end{pmatrix}$
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 $p_1 = p_2 = 1$.

Image credit: Vincent Dumoulin and Francesco Visin
Examples
time:

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

Output volume size: ?
Examples

time:

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

Output volume size:
$(32 + 2 \times 2 - 5)/1 + 1 = 32$ spatially, so
$32 \times 32 \times 10$
Examples

time:

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

Number of parameters in this layer?
Examples

time:

Input volume: \(32\times32\times3\)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has \(5\times5\times3 + 1 = 76\) params (+1 for bias)

\[76\times10 = 760\]
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:

\( K = \) (powers of 2, e.g. 32, 64, 128, 512)

- \( F = 3, \ S = 1, \ P = 1 \)
- \( F = 5, \ S = 1, \ P = 2 \)
- \( F = 5, \ S = 2, \ P = ? \) (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)
Example: CONV layer in Torch

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 $P$

SpatialConvolution

```python
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kw, kh, [dw], [dh], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The input tensor in forward(input) is expected to be a 3D tensor $(nInputPlane \times height \times width)$.

The parameters are the following:

- `nInputPlane`: The number of expected input planes in the image given into forward().
- `nOutputPlane`: The number of output planes the convolution layer will produce.
- `kw`: The kernel width of the convolution
- `kh`: The kernel height of the convolution
- `dw`: The step of the convolution in the width dimension. Default is 1.
- `dh`: The step of the convolution in the height dimension. Default is 1.
- `padW`: The additional zeros added per width to the input planes. Default is 0, a good number is $(kw-1)/2$.
- `padH`: The additional zeros added per height to the input planes. Default is $padW$, a good number is $(kh-1)/2$.

Note that depending on the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor $nInputPlane \times height \times width$, the output image size will be $nOutputPlane \times oheight \times owidth$ where

$$
owidth = \text{floor}\left(\frac{width + 2 \times padW - kw}{dw} + 1\right)
$$

$$
oheight = \text{floor}\left(\frac{height + 2 \times padH - kh}{dh} + 1\right)
$$
Example: CONV layer in Caffe

```c
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96
    kernel_size: 11    # each filter is 11x11
    stride: 4         # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian"    # initialize the filters from a Gaussian
      std: 0.01           # distribution with std 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant"    # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

**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 $P$. 

Example: CONV layer in Lasagne

```
class lasagne.layers.Conv2DLayer(
    incoming, num_filters, filter_size, stride=(1, 1), pad=0,
    untie biases=False, W=lasagne.init.GlorotUniform(), b=lasagne.init.Constant(0),
    nonlinearity=lasagne.nonlinearities.rectify, flip_filters=True, convolution=theano.tensor.nnet.conv2d,
    **kwargs)
```

2D convolutional layer

Performs a 2D convolution on its input and optionally adds a bias and applies an elementwise nonlinearity.

**Parameters:**
- **incoming:** a `Layer` instance or a tuple
  - The layer feeding into this layer, or the expected input shape. The output of this layer should be a 4D tensor, with shape `(batch_size, num_input_channels, input_rows, input_columns)`.
- **num_filters:** int
  - The number of learnable convolutional filters this layer has.
- **filter_size:** int or iterable of int
  - An integer or a 2-element tuple specifying the size of the filters.
- **stride:** int or iterable of int
  - An integer or a 2-element tuple specifying the stride of the convolution operation.
- **pad:** int, iterable of int, 'full', 'same' or 'valid' (default: 0)
  - By default, the convolution is only computed where the input and the filter fully overlap (a valid convolution). When `stride=1`, this yields an output that is smaller than the input by `filter_size - 1`. The pad argument allows you to implicitly pad the input with zeros, extending the output size.

A single integer results in symmetric zero-padding of the given size on all borders, a tuple of two integers allows different symmetric padding per dimension.

- **'full'** pads with one less than the filter size on both sides. This is equivalent to computing the convolution wherever the input and the filter overlap by at least one position.
- **'same'** pads with half the filter size (rounded down) on both sides. When `stride=1`, this results in an output size equal to the input size. Even filter size is not supported.
- **'valid'** is an alias for 0 (no padding / a valid convolution).

**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 $P$.
  - the stride $S$.
  - the amount of zero padding $P$. 

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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)
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. $5\times 5\times 3 = 75$-dimensional dot product)

It’s just a neuron with local connectivity...
The brain/neuron view of CONV Layer

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
Activation Functions
Activation Functions

Sigmoid

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

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

ReLU  \[ \text{max}(0, x) \]
Sigmoid

- 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
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:
Max Pooling

Single depth slice

max pool with 2x2 filters and stride 2

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
• 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:

- 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

F = 2, S = 2
F = 3, S = 2
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*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: 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: ZFNet [Zeiler and Fergus, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

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*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

(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 ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Note:
Most memory is in early CONV
Most params are in late FC
Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: ResNet \cite{he2015deep}
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)

Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)
VGG, 19 layers (ILSVRC 2014)
ResNet, 152 layers (ILSVRC 2015)

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]

Spatial dimension only 56x56!
Case Study Bonus: DeepMind’s AlphaGo
The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a $23 \times 23$ image, then convolves $k$ filters of kernel size $5 \times 5$ with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a $21 \times 21$ image, then convolves $k$ filters of kernel size $3 \times 3$ with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size $1 \times 1$ with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128, 256$ and 384 filters.

**Policy network:**

[19x19x48] Input

CONV1: 192 5x5 filters, stride 1, pad 2 => [19x19x192]

CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]

CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] (probability map of promising moves)
Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
  \[(\text{CONV-RELU})*N-\text{POOL}?]*M-(\text{FC-RELU})*K,\text{SOFTMAX}\]
  where $N$ is usually up to $\sim 5$, $M$ is large, $0 \leq K \leq 2$.
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm
Understanding ConvNets
Visualizing CNN (Layer 1)

Visualizing CNN (Layer 2)

Part that Triggered Filter

Top Image Patches

Visualizing CNN (Layer 3)

Part that Triggered Filter

Top Image Patches

Visualizing CNN (Layer 4)

<table>
<thead>
<tr>
<th>Part that Triggered Filter</th>
<th>Top Image Patches</th>
</tr>
</thead>
</table>

Visualizing CNN (Layer 5)

Part that Triggered Filter

Top Image Patches

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis

Jason Yosinski  Jeff Clune  Anh Nguyen  Thomas Fuchs  Hod Lipson

Cornell University  University of Wyoming  NASA Jet Propulsion Laboratory
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 $256 \times 256$ 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]
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

I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. Two guys run into and outlandish monster and have to escape before he devours them. Things are further complicated when they pick up a skeptical young student. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chills/suspense scenes. Nice pacing and some timing make this movie more than possible for the home/after work.经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬经纬
Today ConvNets are everywhere

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010
Today ConvNets are everywhere

Image Captioning

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

reddit.com/r/deepdream
Frameworks

- Torch [http://torch.ch/](http://torch.ch/) Very efficient. But you must LIKE Lua … Google and Facebook love it
- Theano [http://deeplearning.net/software/theano/](http://deeplearning.net/software/theano/) Compiled from Python. Not as efficient as Torch
- CXXNet [https://github.com/dmlc/cxxnet](https://github.com/dmlc/cxxnet) Simpler than Caffe. More efficient
- Parameter Server bindings to [https://github.com/dmlc/minerva](https://github.com/dmlc/minerva), Caffe, CXXNet, …