Lectures 7 and 8: Convolutional Neural Networks and Spatial Localization and Detection

Thursday February 16, 2017
Announcements!

- HW #2 due **next Friday Feb 24**

- Read **AlexNet paper** for next class

- Post paper summaries and discussion questions to class blog by **Mon Feb 20 11:59pm**

- These are easy points. Don’t miss them.

- Final project teams will be posted to webpage this weekend.
Python/Numpy of the Day

- **enumerate(<iterable object>)**
- returns iterator not generator, but use case behavior is similar
- no ‘yeild’

```python
for ind, thing in enumerate(list_of_things):
    print 'index: {} item: {}'.format(ind, thing)
```

output:
index: 0 item: thing0
index: 1 item: thing1
...

- np.full(shape, fill_val) and np.full_like(ex_array, fill_val)
Mini-batch SGD

Loop:
1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient
Parameter updates

We covered:
sgd,
momentum,
nag,
adagrad,
rmsprop,
adam (not in this vis),
we did not cover adadelta

Image credits: Alec Radford
Dropout

Forces the network to have a redundant representation.

- has an ear
- has a tail
- is furry
- has claws
- mischievous look

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Convolutional Neural Networks

[LeNet-5, LeCun 1980]

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Convolutional Neural Networks
Review from linear filters

**Sharpening filter**
- Accentuates differences with local average

![Original image](image1)

![Sharpened image](image2)

**Sobel filter**
- Vertical Edges

![Sobel filter](image3)

Source: D. Lowe
Convolution Layer

32x32x3 image

32 height

32 width

3 depth

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer - the convolution is in Fourier space

Let $\mathcal{F}$ denote the Fourier transform operator, so $\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$ are the Fourier transforms of $f$ and $g$, respectively.

Then

$$\mathcal{F}\{f \ast g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

where $\cdot$ denotes point-wise multiplication. It also works the other way around:

$$\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} \ast \mathcal{F}\{g\}$$

By applying the inverse Fourier transform $\mathcal{F}^{-1}$, we can write:

$$f \ast g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

and:

$$f \cdot g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \ast \mathcal{F}\{g\}\}$$
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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
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. $5\times5\times3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

comp150dl
Convolution Layer

32x32x3 image
5x5x3 filter

consider a second, green 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 Convolution Layers, interspersed with activation functions.
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Activation Maps from Filters at different layers of AlexNet

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
A closer look at spatial dimensions:

- **32x32x3 image**
- **5x5x3 filter**

convolve (slide) over all spatial locations

activation map

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
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. \(N = 7, F = 3:\)
\[\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\]
\[\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\]
\[\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33 :\]
In practice: Common to zero pad the border

0 0 0 0 0 0 0
0
0
0
0
0

Example: 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

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>
<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>
</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.

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Examples time:

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

Output volume size: ?
Examples time:

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

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so 32x32x10
Examples time:

Input volume: $32 \times 32 \times 3$
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)
=> \(76 \times 10 = 760\)
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)
Example: CONV layer in Torch

SpatialConvolution

```
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 x height x 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 of 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 x height x width, the output image size will be nOutputPlane x oheight x owidth where

```
owidth = floor((width + 2*padW - kw) / dW + 1)
oheight = floor((height + 2*padH - kh) / dH + 1)
```
Example: CONV layer in Caffe

```plaintext
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 # learn 96 filters
    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 stddev 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 $P$.  

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Pooling and FC Layers

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:
Single depth slice

max pool with 2x2 filters and stride 2

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Summary of Pooling Layer

- 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
Summary of Pooling Layer

- 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:

- $F = 2$, $S = 2$
- $F = 3$, $S = 2$
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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
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:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th>Filters/Size</th>
<th>Stride/Pad</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td></td>
<td>[227x227x3]</td>
<td></td>
</tr>
<tr>
<td>CONV1</td>
<td>96 11x11 filters</td>
<td>at stride 4, pad 0</td>
<td></td>
</tr>
<tr>
<td>MAX POOL1</td>
<td>3x3 filters</td>
<td>at stride 2</td>
<td></td>
</tr>
<tr>
<td>NORM1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONV2</td>
<td>256 5x5 filters</td>
<td>at stride 1, pad 2</td>
<td></td>
</tr>
<tr>
<td>MAX POOL2</td>
<td>3x3 filters</td>
<td>at stride 2</td>
<td></td>
</tr>
<tr>
<td>NORM2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONV3</td>
<td>384 3x3 filters</td>
<td>at stride 1, pad 1</td>
<td></td>
</tr>
<tr>
<td>CONV4</td>
<td>384 3x3 filters</td>
<td>at stride 1, pad 1</td>
<td></td>
</tr>
<tr>
<td>CONV5</td>
<td>256 3x3 filters</td>
<td>at stride 1, pad 1</td>
<td></td>
</tr>
<tr>
<td>MAX POOL3</td>
<td>3x3 filters</td>
<td>at stride 2</td>
<td></td>
</tr>
<tr>
<td>FC6</td>
<td>4096 neurons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC7</td>
<td>4096 neurons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC8</td>
<td>1000 neurons</td>
<td>(class scores)</td>
<td></td>
</tr>
</tbody>
</table>
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

*Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n*

[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 (not counting biases)

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

* Original slides borrowed from Andrej Karpathy
* and Li Fei-Fei, Stanford cs231n
INPUT: [224x224x3] memory: $224^2 224^2 3 = 150K$ params: 0

CONV3-64: [224x224x64] memory: $224^2 224^2 64 = 3.2M$ params: $(3^3 3^3) 64 = 1,728$

CONV3-64: [224x224x64] memory: $224^2 224^2 64 = 3.2M$ params: $(3^3 64) 64 = 36,864$

POOL2: [112x112x64] memory: $112^2 112^2 64 = 800K$ params: 0

CONV3-128: [112x112x128] memory: $112^2 112^2 128 = 1.6M$ params: $(3^3 64) 128 = 73,728$

CONV3-128: [112x112x128] memory: $112^2 112^2 128 = 1.6M$ params: $(3^3 128) 128 = 147,456$

POOL2: [56x56x128] memory: $56^2 56^2 128 = 400K$ params: 0

CONV3-256: [56x56x256] memory: $56^2 56^2 256 = 800K$ params: $(3^3 128) 256 = 294,912$

CONV3-256: [56x56x256] memory: $56^2 56^2 256 = 800K$ params: $(3^3 256) 256 = 589,824$

CONV3-256: [56x56x256] memory: $56^2 56^2 256 = 800K$ params: $(3^3 256) 256 = 589,824$

POOL2: [28x28x256] memory: $28^2 28^2 256 = 200K$ params: 0

CONV3-512: [28x28x512] memory: $28^2 28^2 512 = 400K$ params: $(3^3 256) 512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28^2 28^2 512 = 400K$ params: $(3^3 512) 512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28^2 28^2 512 = 400K$ params: $(3^3 512) 512 = 2,359,296$

POOL2: [14x14x512] memory: $14^2 14^2 512 = 100K$ params: 0

CONV3-512: [14x14x512] memory: $14^2 14^2 512 = 100K$ params: $(3^3 512) 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14^2 14^2 512 = 100K$ params: $(3^3 512) 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14^2 14^2 512 = 100K$ params: $(3^3 512) 512 = 2,359,296$

POOL2: [7x7x512] memory: $7^2 7^2 512 = 25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7^2 7^2 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096^2 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096^2 1000 = 4,096,000$

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*128)*128 = 147,456

**POOL2:** [56x56x128] memory: 56*56*128=400K params: 0

**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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Note:

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

(comp150dl) Tufts University
Case Study: GoogLeNet

[Szegedy et al., 2014]

ILSVRC 2014 winner (6.7% top 5 error)

Inception module
## Case Study: GoogLeNet

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/stride</th>
<th>output size</th>
<th>depth</th>
<th>#1x1</th>
<th>#3x3 reduce</th>
<th>#3x3</th>
<th>#5x5 reduce</th>
<th>#5x5</th>
<th>pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7x7/2</td>
<td>112x112x64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>56x56x64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td>3x3/1</td>
<td>56x56x192</td>
<td>2</td>
<td>64</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>32</td>
<td>112K</td>
<td>360M</td>
<td></td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>28x28x192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28x28x256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>159K</td>
<td>128M</td>
<td></td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28x28x480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>64</td>
<td>380K</td>
<td>304M</td>
<td></td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>14x14x480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14x14x512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>64</td>
<td>364K</td>
<td>73M</td>
<td></td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14x14x512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
<td></td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14x14x512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>463K</td>
<td>100M</td>
<td></td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14x14x528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
<td></td>
</tr>
<tr>
<td>inception (4e)</td>
<td>14x14x832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
<td>840K</td>
<td>170M</td>
<td></td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>7x7x832</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (5a)</td>
<td>7x7x832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
<td>1072K</td>
<td>54M</td>
<td></td>
</tr>
<tr>
<td>inception (5b)</td>
<td>7x7x1024</td>
<td>2</td>
<td>384</td>
<td>192</td>
<td>384</td>
<td>48</td>
<td>128</td>
<td>128</td>
<td>1388K</td>
<td>71M</td>
<td></td>
</tr>
<tr>
<td>avg pool</td>
<td>7x7/1</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout (40%)</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1x1x1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>softmax</td>
<td>1x1x1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Fun features:
- Only 5 million params! (Removes FC layers completely)

### Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)
Case Study: ResNet \[\text{He et al., 2015}\]
ILSVRC 2015 winner (3.6% top 5 error)

Slide from Kaiming He’s ICCV 2015 presentation  [https://www.youtube.com/watch?v=1PGLj-uKT1w](https://www.youtube.com/watch?v=1PGLj-uKT1w)
Revolution of Depth

152 layers

ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

ImageNet Classification top-5 error (%)


(slide from Kaiming He's ICCV 2015 presentation)
CIFAR-10 experiments

CIFAR-10 plain nets

- 56-layer
- 44-layer
- 32-layer
- 20-layer

solid: test
dashed: train

CIFAR-10 ResNets

- 20-layer
- 32-layer
- 44-layer
- 56-layer
- 110-layer

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
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 ICCV 2015 presentation)
Case Study: ResNet

[He et al., 2015]

224x224x3

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

- Plain net
  - any two stacked layers
  - $x \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow H(x)$

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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Case Study: ResNet  [He et al., 2015]

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used
- ResNet architecture can be thought of as large ensemble of relatively shallow networks. [Veit et al. NIPS 2016]
Intro to CNNs 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
  \[(\text{CONV-RELU})^N \text{-POOL?}^M \text{-}(\text{FC-RELU})^K \text{-SOFTMAX}\]
  where \(N\) is usually up to \(~5\), \(M\) is large, \(0 \leq K \leq 2\).
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm
Spatial Localization and Detection

Results from Faster R-CNN, Ren et al 2015

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Computer Vision Tasks

**Classification**

CAT

**Classification + Localization**

CAT

**Object Detection**

CAT, DOG, DUCK

**Instance Segmentation**

CAT, DOG, DUCK

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Computer Vision Tasks

- Classification
- Classification + Localization
- Object Detection
- Instance Segmentation

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Classification + Localization: Task

**Classification**: C classes
- **Input**: Image
- **Output**: Class label
- **Evaluation metric**: Accuracy

**Localization**:
- **Input**: Image
- **Output**: Box in the image \((x, y, w, h)\)
- **Evaluation metric**: Intersection over Union

**Classification + Localization**: Do both

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Localization as Regression

**Input:** image

![Image of a cat]

**Neural Net**

**Output:** Box coordinates (4 numbers)

**Correct output:** Box coordinates (4 numbers)

**Loss:** L2 distance

* Only one object, simpler than detection

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Simple Recipe for Classification + Localization

**Step 1**: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)
Simple Recipe for Classification + Localization

Step 2: Attach new fully-connected “regression head” to the network

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Simple Recipe for Classification + Localization

**Step 3**: Train the regression head only with SGD and L2 loss
Simple Recipe for Classification + Localization

**Step 4:** At test time use both heads
Per-class vs class agnostic regression

Assume classification over C classes:

**Classification head:**
C numbers (one per class)

**Class agnostic:**
4 numbers (one box)

**Class specific:**
C x 4 numbers (one box per class)

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Where to attach the regression head?

- **After conv layers:**
  - Overfeat, VGG

- **After last FC layer:**
  - DeepPose, R-CNN

### Diagram:
- **Image**
- **Convolution and Pooling**
- **Final conv feature map**
- **Fully-connected layers**
- **Class scores**
- **Softmax loss**
Aside: Localizing multiple objects

Want to localize exactly \( K \) objects in each image

(e.g. whole cat, cat head, cat left ear, cat right ear for \( K=4 \))
Sliding Window: Overfeat

Winner of ILSVRC 2013 localization challenge

Image: 3 x 221 x 221

Convolution + pooling

Feature map: 1024 x 5 x 5

Class scores: 1000

4096

FC

4096

FC

4096

FC

4096

FC

Feature maps:

Class scores:

1024

Boxes: 1000 x 4

Euclidean loss

Softmax loss

Winner of ILSVRC 2013 localization challenge


* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores: P(cat)

0.5
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores: P(cat)  
0.5  
0.75
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores:
P(cat)

0.5 0.75
0.6
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores:
P(cat)

<table>
<thead>
<tr>
<th>0.5</th>
<th>0.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores:
P(cat)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>0.6</td>
<td>0.8</td>
</tr>
</tbody>
</table>

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n*
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

Classification score:
P(cat)

Greedily merge boxes and scores (details in paper)
Sliding Window: Overfeat

In practice use many sliding window locations and multiple scales

Window positions + score maps
Box regression outputs
Final Predictions

Efficient Sliding Window: Overfeat

Image: 3 x 221 x 221

Convolution + pooling

Feature map: 1024 x 5 x 5

4096 → FC

1024 → FC

4096 → FC

Class scores: 1000

Boxes: 1000 x 4

4096 → FC

1024 → FC

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Efficient Sliding Window: Overfeat

Efficient sliding window by converting fully-connected layers into convolutions

Image: 3 x 221 x 221

Convolution + pooling

Feature map: 1024 x 5 x 5

Convolution

4096 x 1 x 1

1 x 1 conv

1 x 1 conv

Box coordinates: (4 x 1000) x 1 x 1

1024 x 1 x 1

1 x 1 conv

Class scores: 1000 x 1 x 1

1024 x 1 x 1

1 x 1 conv

4096 x 1 x 1
Summary: Sliding Window

- Run classification + regression network at multiple locations on a high-resolution image

- Convert fully-connected layers into convolutional layers for efficient computation

- Combine classifier and regressor predictions across all scales for final prediction
ImageNet Classification + Localization (1 object per image)

AlexNet: Localization method not published

Overfeat: Multiscale convolutional regression with box merging

VGG: Same as Overfeat, but fewer scales and locations; simpler method, gains all due to deeper features

ResNet: Different localization method (RPN) and much deeper features

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Computer Vision Tasks

<table>
<thead>
<tr>
<th>Classification</th>
<th>Classification + Localization</th>
<th>Object Detection</th>
<th>Instance Segmentation</th>
</tr>
</thead>
</table>

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Computer Vision Tasks

- Classification
- Classification + Localization
- Object Detection
- Instance Segmentation

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Detection Metrics - COCO Challenge

**Average Precision (AP):**

- AP
- $AP_{IoU=.50}$
- $AP_{IoU=.75}$

% AP at IoU=.50:.05:.95 (determines challenge winner)
% AP at IoU=.50 (PASCAL VOC metric)
% AP at IoU=.75 (strict metric)

**AP Across Scales:**

- $AP_{small}$
- $AP_{medium}$
- $AP_{large}$

% AP for small objects: area < 32^2
% AP for medium objects: 32^2 < area < 96^2
% AP for large objects: area > 96^2

**Average Recall (AR):**

- $AR_{max=1}$
- $AR_{max=10}$
- $AR_{max=100}$

% AR given 1 detection per image
% AR given 10 detections per image
% AR given 100 detections per image

**AR Across Scales:**

- $AR_{small}$
- $AR_{medium}$
- $AR_{large}$

% AR for small objects: area < 32^2
% AR for medium objects: 32^2 < area < 96^2
% AR for large objects: area > 96^2
Detection Metrics

**Challenges Score: AP**

- AP is averaged over multiple IoU values between 0.5 and 0.95.
- More comprehensive metric than the traditional AP at a fixed IoU value (0.5 for PASCAL).
Detection Metrics

<table>
<thead>
<tr>
<th>AP Across Scales:</th>
<th>AP(_\text{small})</th>
<th>% AP for small objects: area &lt; 32(^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP(_\text{medium})</td>
<td>% AP for medium objects: 32(^2) &lt; area &lt; 96(^2)</td>
<td></td>
</tr>
<tr>
<td>AP(_\text{large})</td>
<td>% AP for large objects: area &gt; 96(^2)</td>
<td></td>
</tr>
</tbody>
</table>

Other Scores: Size AP

- AP is averaged over instance size:
  - small (A < 32 x 32)
  - medium (32x 32 < A < 96 x 96)
  - large (A > 96 x 96)
Other Scores: AR

- Measures the maximum recall over a fixed number of detections allowed in the image of 1, 10, 100.

- AR is averaged over small ($A < 32 \times 32$), medium ($32 \times 32 < A < 96 \times 96$) and large ($A > 96 \times 96$) instances of objects.
Detection Ambiguity

Which one is better?

IoU = 0.5

IoU = 0.7

IoU = 0.95
Detection as Regression?

DOG, (x, y, w, h)
CAT, (x, y, w, h)
CAT, (x, y, w, h)
DUCK (x, y, w, h)

= 16 numbers
Detection as Regression?

DOG, (x, y, w, h)
CAT, (x, y, w, h)
= 8 numbers

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Detection as Regression?

CAT, \((x, y, w, h)\)

\(\ldots\)

CAT, \((x, y, w, h)\)

\(=\) many numbers

Need variable sized outputs
Detection as Classification

CAT? NO
DOG? NO
Detection as Classification

CAT? YES!

DOG? NO
Detection as Classification

CAT? NO
DOG? NO
Detection as Classification

**Problem:** Need to test many positions and scales

**Solution:** If your classifier is fast enough, just do it
Detection as Classification

**Problem:** Need to test many positions and scales, and use a computationally demanding classifier (CNN)

**Solution:** Only look at a tiny subset of possible positions
Region Proposals

- Find “blobby” image regions that are likely to contain objects
- “Class-agnostic” object detector
- Look for “blob-like” regions
Region Proposals: Selective Search

Bottom-up segmentation, merging regions at multiple scales


* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Region Proposals: Many other choices

<table>
<thead>
<tr>
<th>Method</th>
<th>Approach</th>
<th>Outputs Segments</th>
<th>Outputs Score</th>
<th>Control #proposals</th>
<th>Time (sec.)</th>
<th>Repeatability</th>
<th>Recall Results</th>
<th>Detection Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bing [18]</td>
<td>Window scoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.2</td>
<td>**</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>CPMC [19]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>250</td>
<td>-</td>
<td>**</td>
<td>-</td>
</tr>
<tr>
<td>EdgeBoxes [20]</td>
<td>Window scoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.3</td>
<td>**</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Endres [21]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>100</td>
<td>-</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Geodesic [22]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
<td>*</td>
<td>***</td>
<td>**</td>
</tr>
<tr>
<td>MCG [23]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>30</td>
<td>*</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Objectness [24]</td>
<td>Window scoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3</td>
<td>*</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>Rahtu [25]</td>
<td>Window scoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3</td>
<td>-</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>RandomizedPrims [26]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>**</td>
</tr>
<tr>
<td>Rantalankila [27]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>10</td>
<td>**</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>Rigor [28]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>10</td>
<td>*</td>
<td>***</td>
<td>**</td>
</tr>
<tr>
<td>SelectiveSearch [29]</td>
<td>Grouping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>**</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Gaussian</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>*</td>
</tr>
<tr>
<td>SlidingWindow</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>0</td>
<td>***</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Superpixels</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>1</td>
<td>*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Uniform</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Hosang et al, “What makes for effective detection proposals?”, PAMI 2015
Putting it together: R-CNN


Slide credit: Ross Girshick
R-CNN Training

**Step 1:** Train (or download) a classification model for ImageNet (AlexNet)
R-CNN Training

**Step 2:** Fine-tune model for detection
- Instead of 1000 ImageNet classes, want PASCAL 20 object classes + *background*
- Throw away final fully-connected layer, reinitialize from scratch
- Keep training model using positive / negative regions from detection images

---

![Diagram](image_url)

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

comp150dl
R-CNN Training

**Step 3: Extract features**
- Extract region proposals for all images
- For each region: warp to CNN input size, run forward through CNN, save pool5 features to disk
- Have a big hard drive: features are ~200GB for PASCAL dataset!
R-CNN Training

**Step 4:** Train one binary SVM per class to classify region features

- Training image regions
- Cached region features
- Positive samples for cat SVM
- Negative samples for cat SVM
R-CNN Training

**Step 4**: Train one binary SVM per class to classify region features

- **Training image regions**
- **Cached region features**
- **Negative samples for dog SVM**
- **Positive samples for dog SVM**
R-CNN Training

**Step 5** (bbox regression): For each class, train a linear regression model to map from cached features to offsets to GT boxes to make up for "slightly wrong" proposals.

Training image regions

Cached region features

Regression targets
(dx, dy, dw, dh)
Normalized coordinates

(0, 0, 0, 0)  Proposal is good
(.25, 0, 0, 0) Proposal too far to left
(0, 0, -0.125, 0) Proposal too wide
## Object Detection: Datasets

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of classes</td>
<td>20</td>
<td>200</td>
<td>80</td>
</tr>
<tr>
<td>Number of images (train + val)</td>
<td>~20k</td>
<td>~470k</td>
<td>~120k</td>
</tr>
<tr>
<td>Mean objects per image</td>
<td>2.4</td>
<td>1.1</td>
<td>7.2</td>
</tr>
</tbody>
</table>
R-CNN Problems

1. Slow at test-time: need to run full forward pass of CNN for each region proposal

2. SVMs and regressors are post-hoc: CNN features not updated in response to SVMs and regressors

3. Complex multistage training pipeline
R-CNN Problem #1:
Slow at test-time due to independent forward passes of the CNN

Solution:
Share computation of convolutional layers between proposals for an image
R-CNN Problem #2: Post-hoc training: CNN not updated in response to final classifiers and regressors

R-CNN Problem #3: Complex training pipeline

Solution: Just train the whole system end-to-end all at once!

Slide credit: Ross Girshick
Fast R-CNN: Region of Interest Pooling

**Hi-res input image:** 3 x 800 x 600 with region proposal

**Hi-res conv features:** C x H x W with region proposal

**Problem:** Fully-connected layers expect low-res conv features: C x h x w
Fast R-CNN: Region of Interest Pooling

Hi-res input image:
3 x 800 x 600
with region proposal

Hi-res conv features:
C x H x W
with region proposal

Project region proposal
onto conv feature map

Problem: Fully-connected layers expect low-res conv features: C x h x w
Fast R-CNN: Region of Interest Pooling

Hi-res input image: 3 x 800 x 600 with region proposal

Hi-res conv features: C x H x W with region proposal

Problem: Fully-connected layers expect low-res conv features: C x h x w

Convolution and Pooling

Divide projected region into h x w grid

Fully-connected layers

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Fast R-CNN: Region of Interest Pooling

Hi-res input image: 3 x 800 x 600 with region proposal

Hi-res conv features: C x H x W with region proposal

Max-pool within each grid cell

RoI conv features: C x h x w for region proposal

Fully-connected layers expect low-res conv features: C x h x w

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Fast R-CNN: Region of Interest Pooling

Hi-res input image: 3 x 800 x 600 with region proposal

Convolution and Pooling

Hi-res conv features: C x H x W with region proposal

Can back propagate similar to max pooling

RoI conv features: C x h x w for region proposal

Fully-connected layers expect low-res conv features: C x h x w

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
## Fast R-CNN Results

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Time</td>
<td>84 hours</td>
<td>9.5 hours</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>8.8x</td>
</tr>
</tbody>
</table>

Faster!

Using VGG-16 CNN on Pascal VOC 2007 dataset

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
## Fast R-CNN Results

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training Time:</strong></td>
<td>84 hours</td>
<td>9.5 hours</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>8.8x</td>
</tr>
<tr>
<td><strong>Test time per image</strong></td>
<td>47 seconds</td>
<td>0.32 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>146x</td>
</tr>
</tbody>
</table>

Using VGG-16 CNN on Pascal VOC 2007 dataset
## Fast R-CNN Results

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Faster!</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Time:</td>
<td>84 hours</td>
<td>9.5 hours</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>8.8x</td>
</tr>
<tr>
<td><strong>FASTER!</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test time per image</td>
<td>47 seconds</td>
<td>0.32 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>146x</td>
</tr>
<tr>
<td><strong>Better!</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mAP (VOC 2007)</td>
<td>66.0</td>
<td>66.9</td>
</tr>
</tbody>
</table>

Using VGG-16 CNN on Pascal VOC 2007 dataset
Fast R-CNN Problem:

Test-time speeds don’t include region proposals

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time per image</td>
<td>47 seconds</td>
<td>0.32 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>146x</td>
</tr>
<tr>
<td>Test time per image with Selective Search</td>
<td>50 seconds</td>
<td>2 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>25x</td>
</tr>
</tbody>
</table>
Fast R-CNN Problem Solution:

Test-time speeds don’t include region proposals
Just make the CNN do region proposals too!

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time per image</td>
<td>47 seconds</td>
<td>0.32 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>146x</td>
</tr>
<tr>
<td>Test time per image with Selective Search</td>
<td>50 seconds</td>
<td>2 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>25x</td>
</tr>
</tbody>
</table>
Faster R-CNN:

Insert a **Region Proposal Network (RPN)** after the last convolutional layer

RPN trained to produce region proposals directly; no need for external region proposals!

After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN


Slide credit: Ross Girshick

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Faster R-CNN: Region Proposal Network

Slide a small window on the feature map

Build a small network for:
• classifying object or not-object, and
• regressing bbox locations

Position of the sliding window provides localization information with reference to the image

Box regression provides finer localization information with reference to this sliding window

Slide credit: Kaiming He
Faster R-CNN: Region Proposal Network

Use \textbf{N anchor boxes} at each location

Anchors are \textbf{translation invariant}: use the same ones at every location

Regression gives offsets from anchor boxes

Classification gives the probability that each (regressed) anchor shows an object
## Faster R-CNN: Results

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
<th>Faster R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time per</td>
<td>50 seconds</td>
<td>2 seconds</td>
<td>0.2 seconds</td>
</tr>
<tr>
<td>image (with</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>proposals)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>25x</td>
<td>250x</td>
</tr>
<tr>
<td>mAP (VOC 2007)</td>
<td>66.0</td>
<td>66.9</td>
<td>66.9</td>
</tr>
</tbody>
</table>
Object Detection State-of-the-art: ResNet 101 + Faster R-CNN + some extras

<table>
<thead>
<tr>
<th>training data</th>
<th>COCO train</th>
<th>COCO trainval</th>
</tr>
</thead>
<tbody>
<tr>
<td>test data</td>
<td>COCO val</td>
<td>COCO test-dev</td>
</tr>
<tr>
<td>mAP</td>
<td>@.5</td>
<td>@[.5,.95]</td>
</tr>
<tr>
<td>baseline Faster R-CNN (VGG-16)</td>
<td>41.5</td>
<td>21.2</td>
</tr>
<tr>
<td>baseline Faster R-CNN (ResNet-101) + box refinement</td>
<td>48.4</td>
<td>27.2</td>
</tr>
<tr>
<td></td>
<td>49.9</td>
<td>29.9</td>
</tr>
<tr>
<td>+context</td>
<td>51.1</td>
<td>30.0</td>
</tr>
<tr>
<td>+multi-scale testing</td>
<td>53.8</td>
<td>32.5</td>
</tr>
<tr>
<td>ensemble</td>
<td>59.0</td>
<td>37.4</td>
</tr>
</tbody>
</table>

YOLO: You Only Look Once
Detection as Regression

Divide image into $S \times S$ grid

Within each grid cell predict:
- B Boxes: 4 coordinates + confidence
- Class scores: C numbers

Regression from image to $7 \times 7 \times (5 \times B + C)$ tensor

Direct prediction using a CNN

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Object Detection code links:

**R-CNN**
(Caffe + MATLAB): [https://github.com/rbgirshick/rcnn](https://github.com/rbgirshick/rcnn)
Probably don’t use this; too slow

**Fast R-CNN**
(Caffe + MATLAB): [https://github.com/rbgirshick/fast-rcnn](https://github.com/rbgirshick/fast-rcnn)

**Faster R-CNN**
(Caffe + MATLAB): [https://github.com/ShaoqingRen/faster_rcnn](https://github.com/ShaoqingRen/faster_rcnn)
(Caffe + Python): [https://github.com/rbgirshick/py-faster-rcnn](https://github.com/rbgirshick/py-faster-rcnn)

**YOLO**
(To be presented in class)