CS 4803 / 7643: Deep Learning

Topics:
- (Finish) Modern CNNs
- Detection & Segmentation architectures

Zsolt Kira
Georgia Tech
Administrivia

• PS2/HW2 out today!
  – Due: 03/03, 11:55pm
  – Less involved than PS1/HW1

• Please send project proposal!
  – Get feedback before next week
Last Time
Handwriting Recognition Example

filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh

Curved manifold

Flatter manifold
Translation Invariance
Some Rotation Invariance
Some Scale Invariance
Case Studies

- There are several generations of ConvNets
  - 2012 – 2014: AlexNet, ZNet, VGGNet
    - Conv-Relu, Pooling, Fully connected, Softmax
    - Deeper ones (VGGNet) tend to do better
  - 2014
    - Fully-convolutional networks for **semantic segmentation**
    - Matrix outputs rather than just one probability distribution
  - 2014-2016
    - Fully-convolutional networks for **classification**
    - Less parameters, faster than comparable Gen1 networks
    - GoogleNet, ResNet
  - 2014-2016
    - Detection layers (proposals)
    - Caption generation (combine with RNNs for language)
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%
INPUT: [224×224×3] memory: 224×224×3 = 150K params: 0 (not counting biases)

CONV3-64: [224×224×64] memory: 224×224×64 = 3.2M params: (3×3×64)×64 = 1,728

CONV3-64: [224×224×64] memory: 224×224×64 = 3.2M params: (3×3×64)×64 = 36,864

POOL2: [112×112×64] memory: 112×112×64 = 800K params: 0

CONV3-128: [112×112×128] memory: 112×112×128 = 1.6M params: (3×3×64)×128 = 73,728

CONV3-128: [112×112×128] memory: 112×112×128 = 1.6M params: (3×3×128)×128 = 147,456

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

CONV3-256: [56×56×256] memory: 56×56×256 = 800K params: (3×3×128)×256 = 294,912

CONV3-256: [56×56×256] memory: 56×56×256 = 800K params: (3×3×256)×256 = 589,824

CONV3-256: [56×56×256] memory: 56×56×256 = 800K params: (3×3×256)×256 = 589,824

POOL2: [28×28×256] memory: 28×28×256 = 200K params: 0

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

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

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

POOL2: [14×14×512] memory: 14×14×512 = 100K params: 0

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

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

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

POOL2: [7×7×512] memory: 7×7×512 = 25K params: 0

FC: [1×1×4096] memory: 4096 params: 7×7×512×4096 = 102,760,448

FC: [1×1×4096] memory: 4096 params: 4096×4096 = 16,777,216

FC: [1×1×1000] memory: 1000 params: 4096×1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~ = 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Case Study: GoogLeNet

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: GoogLeNet

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/stride</th>
<th>output size</th>
<th>depth</th>
<th>#1 x 1</th>
<th>#3 x 3 reduce</th>
<th>#3 x 3</th>
<th>#5 x 5 reduce</th>
<th>#5 x 5</th>
<th>pool proj</th>
<th>param</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7 x 7/2</td>
<td>112 x 112 x 64</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>271K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3 x 3/2</td>
<td>56 x 56 x 64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td>3 x 3/1</td>
<td>56 x 56 x 192</td>
<td>2</td>
<td>64</td>
<td>192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>max pool</td>
<td>3 x 3/2</td>
<td>28 x 28 x 192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (1a)</td>
<td>28 x 28 x 256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>15</td>
<td>32</td>
<td>32</td>
<td>96</td>
<td>64</td>
<td>380K</td>
</tr>
<tr>
<td>inception (1b)</td>
<td>28 x 28 x 480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
<td></td>
</tr>
<tr>
<td>max pool</td>
<td>3 x 3/2</td>
<td>14 x 14 x 480</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (1a)</td>
<td>14 x 14 x 512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>15</td>
<td>48</td>
<td>64</td>
<td>384K</td>
<td>78M</td>
<td></td>
</tr>
<tr>
<td>inception (1b)</td>
<td>14 x 14 x 512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>15</td>
<td>48</td>
<td>64</td>
<td>384K</td>
<td>78M</td>
<td></td>
</tr>
<tr>
<td>inception (1c)</td>
<td>14 x 14 x 512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>453K</td>
<td>100M</td>
<td></td>
</tr>
<tr>
<td>inception (1d)</td>
<td>14 x 14 x 512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>453K</td>
<td>100M</td>
<td></td>
</tr>
<tr>
<td>inception (1e)</td>
<td>14 x 14 x 512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>453K</td>
<td>100M</td>
<td></td>
</tr>
<tr>
<td>max pool</td>
<td>3 x 3/2</td>
<td>7 x 7 x 1024</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>840K</td>
<td>170M</td>
</tr>
<tr>
<td>inception (1a)</td>
<td>7 x 7 x 1024</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 (1b)</td>
<td>7 x 7 x 1024</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>7 x 7/1</td>
<td>1 x 1 x 1024</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout (40%)</td>
<td>1 x 1 x 1024</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1 x 1 x 1000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1000K</td>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>softmax</td>
<td>1 x 1 x 1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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%)

AlexNet: 60M params
ZNet: 75M
VGG: 138M
GoogleNet: 5M

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Case Study: ResNet [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)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC’15 ResNet, ILSVRC’14 GoogleNet, ILSVRC’14 VGG, ILSVRC’13, ILSVRC’12 AlexNet, ILSVRC’11, ILSVRC’10

3.57, 6.7, 7.3, 11.7, 16.4, 25.8, 28.2

152 layers

22 layers, 19 layers, 8 layers, 8 layers

(shadow from Kaiming He’s recent presentation)
Importance of Depth

- After a while, adding depth *decreases* performance
- At first, vanishing/exploding gradients
  - normalized initialization
  - Batch normalization
  - 2\textsuperscript{nd} order methods
- Then, optimization limitation
  - Deeper network should be able to mimic shallow ones

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Case Study: ResNet  [He et al., 2015]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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
Case Study: ResNet

[He et al., 2015]

(this trick is also used in GoogLeNet)
### Case Study: ResNet \cite{he2015deep}

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td></td>
<td></td>
<td>7×7, 64, stride 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2_x</td>
<td>56×56</td>
<td>[3×3, 64]×2</td>
<td>[3×3, 64]×3</td>
<td>1×1, 64</td>
<td>1×1, 64</td>
<td>1×1, 64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</td>
</tr>
<tr>
<td>conv3_x</td>
<td>28×28</td>
<td>[3×3, 128]×2</td>
<td>[3×3, 128]×4</td>
<td>1×1, 128</td>
<td>1×1, 128</td>
<td>1×1, 128</td>
</tr>
<tr>
<td>conv4_x</td>
<td>14×14</td>
<td>[3×3, 256]×2</td>
<td>[3×3, 256]×6</td>
<td>1×1, 256</td>
<td>1×1, 256</td>
<td>1×1, 256</td>
</tr>
<tr>
<td>conv5_x</td>
<td>7×7</td>
<td>[3×3, 512]×2</td>
<td>[3×3, 512]×3</td>
<td>1×1, 512</td>
<td>1×1, 512</td>
<td>1×1, 512</td>
</tr>
<tr>
<td>1×1</td>
<td></td>
<td></td>
<td></td>
<td>average pool, 1000-d fc, softmax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.8×10^9</td>
<td>3.6×10^9</td>
<td>3.8×10^9</td>
<td>7.6×10^9</td>
<td>11.3×10^9</td>
<td></td>
</tr>
</tbody>
</table>

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Localization and Detection

Results from Faster R-CNN, Ren et al 2015
So far: Image Classification

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Fully-Connected: 4096 to 1000

Vector: 4096

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Computer Vision Tasks

- **Classification**
  - Single object
  - CAT

- **Classification + Localization**
  - Single object
  - CAT

- **Object Detection**
  - Multiple objects
  - CAT, DOG, DUCK

- **Instance Segmentation**
  - Multiple objects
  - CAT, DOG, DUCK

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Computer Vision Tasks

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Classification + Localization

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
CLS - ImageNet

1000 classes (same as classification)

Each image has 1 class, at least one bounding box

~800 training images per class

Algorithm produces 5 (class, box) guesses

Example is correct if at least one one guess has correct class AND bounding box at least 0.5 intersection over union (IoU)

<izhevsky et. al. 2012>
Idea 1: Localization as Regression

**Input:** image

- Only one object, simpler than detection

**Neural Net**

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

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

**Loss:**
- L2 distance

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Step 1: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)
Step 2: Attach new fully-connected “regression head” to the network
Step 3: Train the regression head only with SGD and L2 loss
Step 4: At test time use both heads
Per-Class vs. Class Agnostic

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)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Where to attach?

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Multiple Objects

Want to localize **exactly** $K$ objects in each image

(e.g. whole cat, cat head, cat left ear, cat right ear for $K=4$)

- Convolution and Pooling
- Final conv feature map
- Fully-connected layers
- Class scores
  - Fully-connected layers
  - Box coordinates
- $K \times 4$ numbers (one box per object)
Human Pose Estimation

Represent a person by K joints

Regress \((x, y)\) for each joint from last fully-connected layer of AlexNet

(Details: Normalized coordinates, iterative refinement)

Sliding Window: Overfeat

Winner of ILSVRC 2013 localization challenge

Convolution + pooling

Image: 3 x 221 x 221

Feature map: 1024 x 5 x 5

Class scores: 1000

Softmax loss

Euclidean loss

Boxes: 1000 x 4


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Sliding Window: Overfeat

Network input: 3 x 221 x 221
Larger image: 3 x 257 x 257
Classification scores: P(cat)
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores: P(cat)
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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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 | 0.8  |
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 0.8

Why aren’t boxes across grid?

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Sliding Window: Overfeat

Greedily merge boxes and scores (details in paper)

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification score: P (cat)
Sliding Window: Overfeat

In practice use many sliding window locations and multiple scales


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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

5 x 5 conv

4096 x 1 x 1

5 x 5 conv

1024 x 1 x 1

1 x 1 conv

4096 x 1 x 1

1 x 1 conv

1024 x 1 x 1

1 x 1 conv

Class scores: 1000 x 1 x 1

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Efficient Sliding Window: Overfeat

**Training time**: Small image, 1 x 1 classifier output

**Test time**: Larger image, 2 x 2 classifier output, only extra compute at yellow regions

Sermanet et al, "Integrated Recognition, Localization and Detection using Convolutional Networks", ICLR 2014
**ImageNet Classification + Localization**

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Localization Error (Top 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>2012</td>
<td>34.2</td>
</tr>
<tr>
<td>Overfeat</td>
<td>2013</td>
<td>29.9</td>
</tr>
<tr>
<td>VGG</td>
<td>2014</td>
<td>25.3</td>
</tr>
<tr>
<td>ResNet</td>
<td>2015</td>
<td>9</td>
</tr>
</tbody>
</table>

- **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
Computer Vision Tasks

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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, \((x, y, w, h)\)

\[= 8 \text{ numbers}\]
Need variable sized outputs
Detection as Classification

CAT? NO

DOG? NO

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Detection as Classification

CAT? YES!

DOG? NO
Detection as Classification

CAT? NO

DOG? NO

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Detection as Classification

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

**Solution:** If your classifier is fast enough, just do it
Histogram of Oriented Gradients

- Compute HOG of the whole image at multiple resolutions
- Score every subwindow of the feature pyramid
- Apply non-maxima suppression

Dalal and Triggs, “Histograms of Oriented Gradients for Human Detection”, CVPR 2005
Slide credit: Ross Girshick
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
R-CNN

Girshick et al., "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014

Slide credit: Ross Girshick

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Step 1: Train (or download) a classification model for ImageNet (AlexNet)
Step 2: Fine-tune model for detection
- Instead of 1000 ImageNet classes, want 20 object classes + background
- Throw away final fully-connected layer, reinitialize from scratch
- Keep training model using positive / negative regions from detection images
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!
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**
  - [Image of kitten]
  - [Image of cat]
  - [Image of dog]
  - [Image of cat]
  - [Image of puppy]

- **Cached region features**

  - [Image of kitten]
  - [Image of cat]
  - [Image of dog]
  - [Image of cat]
  - [Image of puppy]

  ![Negative samples for dog SVM](Image)
  ![Positive samples for dog SVM](Image)
**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.
## 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</td>
<td>~20k</td>
<td>~470k</td>
<td>~120k</td>
</tr>
<tr>
<td>(train + val)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean objects per</td>
<td>2.4</td>
<td>1.1</td>
<td>7.2</td>
</tr>
<tr>
<td>image</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We use a metric called “mean average precision” (mAP).

Compute average precision (AP) separately for each class, then average over classes.

A detection is a true positive if it has IoU with a ground-truth box greater than some threshold (usually 0.5) (mAP@0.5).

Combine all detections from all test images to draw a precision / recall curve for each class; AP is area under the curve.

TL;DR mAP is a number from 0 to 100; high is good.
R-CNN Results

![Bar chart showing R-CNN results](chart.png)
R-CNN Results

Big improvement compared to pre-CNN methods

<table>
<thead>
<tr>
<th>Method</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM (2011)</td>
<td>33.7</td>
<td>29.6</td>
</tr>
<tr>
<td>Regionlets (2013)</td>
<td>41.7</td>
<td>39.7</td>
</tr>
<tr>
<td>R-CNN (2014, AlexNet)</td>
<td>54.2</td>
<td>50.2</td>
</tr>
<tr>
<td>R-CNN + bbox reg (AlexNet)</td>
<td>58.5</td>
<td>53.7</td>
</tr>
<tr>
<td>R-CNN (VGG-16)</td>
<td>66</td>
<td>62.9</td>
</tr>
</tbody>
</table>
R-CNN Results

Bounding box regression helps a bit

<table>
<thead>
<tr>
<th>Model</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM (2011)</td>
<td>33.7</td>
<td>29.6</td>
</tr>
<tr>
<td>Regionlets (2013)</td>
<td>41.7</td>
<td>39.7</td>
</tr>
<tr>
<td>R-CNN (2014, AlexNet)</td>
<td>54.2</td>
<td>50.2</td>
</tr>
<tr>
<td>R-CNN + bbox reg (AlexNet)</td>
<td>58.5</td>
<td>53.7</td>
</tr>
<tr>
<td>R-CNN (VGG-16)</td>
<td>66.6</td>
<td>62.9</td>
</tr>
</tbody>
</table>
R-CNN Results

Features from a deeper network help a lot

<table>
<thead>
<tr>
<th>Model</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM (2011)</td>
<td>33.7</td>
<td>29.6</td>
</tr>
<tr>
<td>Regionlets (2013)</td>
<td>41.7</td>
<td>39.7</td>
</tr>
<tr>
<td>R-CNN (2014, AlexNet)</td>
<td>54.2</td>
<td>50.2</td>
</tr>
<tr>
<td>R-CNN + bbox reg (AlexNet)</td>
<td>58.5</td>
<td>53.7</td>
</tr>
<tr>
<td>R-CNN (VGG-16)</td>
<td>66</td>
<td>62.9</td>
</tr>
</tbody>
</table>
R-CNN Results

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
Fast R-CNN (test time)

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Fast R-CNN (training)

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Region of Interest (ROI) 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
Region of Interest (ROI) 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
Region of Interest (ROI) 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

Divide projected region into h x w grid

Problem: Fully-connected layers expect low-res conv features: C x h x w
Region of Interest (ROI) 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
Region of Interest (ROI) 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

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

Fully-connected layers expect low-res conv features: C x h x w
## Fast R-CNN Results

<table>
<thead>
<tr>
<th>Faster!</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>
## 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>
<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>
</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 Results

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 Results

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</td>
<td>50 seconds</td>
<td>2 seconds</td>
</tr>
<tr>
<td>with Selective Search</td>
<td></td>
<td></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
Mask R-CNN: overview
Mask R-CNN

Also used for pose estimation

https://www.youtube.com/watch?v=KYNDzIcQMWA
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

Redmon et al, "You Only Look Once:
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation

Single object

Multiple objects

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Semantic Segmentation

Label each pixel in the image with a category label.

Don’t differentiate instances, only care about pixels.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Semantic Segmentation Idea: Sliding Window

Full image

Extract patch

Classify center pixel with CNN

Cow

Cow

Grass

Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Semantic Segmentation Idea: Sliding Window

Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al., “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014
Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input: 3 x H x W

Convolutions: D x H x W

Scores: C x H x W

Predictions: H x W

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input: $3 \times H \times W$

Convolutions: $D \times H \times W$

Scores: $C \times H \times W$

Predictions: $H \times W$

Problem: convolutions at original image resolution will be very expensive ...

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Dilated Convolutions

Figure Credit: Dumoulin and Visin, https://arxiv.org/pdf/1603.07285.pdf
Dilated Convolutions (Examples)
Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3 \times 3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7 \times 7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15 \times 15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.
Different Idea: Still Classify, Change Test Time

• Can run on an image of any size!

---

convolution

---

H × W  H/4 × W/4  H/8 × W/8  H/16 × W/16  H/32 × W/32

Figure Credit: [Long, Shelhamer, Darrell CVPR15]
Better Semantic Segmentation Idea: Fully Convolutional with Bottleneck

Design network as a bunch of convolutional layers, with down sampling and up sampling inside the network!

Input: $3 \times H \times W$

High-res: $D_1 \times H/2 \times W/2$

Med-res: $D_2 \times H/4 \times W/4$

Low-res: $D_3 \times H/4 \times W/4$

High-res: $D_1 \times H/2 \times W/2$

Predictions: $H \times W$


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**
Pooling, strided convolution

**Upsampling:**
???

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

- **Input:** $3 \times H \times W$
- **High-res:** $D_1 \times H/2 \times W/2$
- **Med-res:** $D_2 \times H/4 \times W/4$
- **Low-res:** $D_3 \times H/4 \times W/4$
- **Predictions:** $H \times W$


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
In-Network upsampling: “Unpooling”

**Nearest Neighbor**

<table>
<thead>
<tr>
<th>Input: 2 x 2</th>
<th>Output: 4 x 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>3 4</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td></td>
<td>3 3 4 4</td>
</tr>
<tr>
<td></td>
<td>3 3 4 4</td>
</tr>
</tbody>
</table>

**“Bed of Nails”**

<table>
<thead>
<tr>
<th>Input: 2 x 2</th>
<th>Output: 4 x 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1 0 2 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>3 4</td>
<td>3 0 4 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

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

Input: 4 x 4

Max Unpooling
Use positions from pooling layer

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Output: 2 x 2

Rest of the network

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Input: 2 x 2

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

**Recall:** Typical 3 x 3 convolution, stride 1 pad 1

Input: 4 x 4

Output: 4 x 4

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1

Input: 4 x 4

Output: 4 x 4

Dot product between filter and input

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

**Recall**: Normal 3 x 3 convolution, stride 1 pad 1
Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, **stride 2** pad 1

Input: 4 x 4

Output: 2 x 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, **stride** 2 pad 1

Input: 4 x 4  
Dot product between filter and input  
Output: 2 x 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, *stride* 2 pad 1

- Dot product between filter and input
- Filter moves 2 pixels in the input for every one pixel in the output
- Stride gives ratio between movement in input and output

Input: 4 x 4  
Output: 2 x 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

3 × 3 transpose convolution, stride 2 pad 1

- Input gives weight for filter
- Output: 4 × 4
- Sum where output overlaps
- Filter moves 2 pixels in the output for every one pixel in the input
- Stride gives ratio between movement in output and input

Input: 2 × 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Other names:
- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

Input gives weight for filter

Sum where output overlaps

Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Input: 2 x 2

Output: 4 x 4

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Transposed Convolution: 1D Example

Input: a, b

Filter:

- x
- y
- z

Output:

- ax
- ay
- az
- bx
- by
- bz

Output contains copies of the filter weighted by the input, summing at where at overlaps in the output.

Need to crop one pixel from output to make output exactly 2x input.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Transposed Convolution

- [https://distill.pub/2016/deconv-checkerboard/](https://distill.pub/2016/deconv-checkerboard/)
Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**
Pooling, strided convolution

**Upsampling:**
Unpooling or strided transpose convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

- **Input:** $3 \times H \times W$
- **High-res:** $D_1 \times H/2 \times W/2$
- **Med-res:** $D_2 \times H/4 \times W/4$
- **Low-res:** $D_3 \times H/4 \times W/4$
- **Predictions:** $H \times W$

skip layers

end-to-end, joint learning of semantics and location

Slide credit: Jonathan Long
Next Time

• Guest lecture by Zhile Ren (Dhruv Batra post-doc) on 3D