Convolutional Neural Networks for Object Recognition

Marius Leordeanu
Overview

• What is Computer Vision?

• Convolutional Neural Networks

• Convolutional Networks for Visual Object Recognition

Based on the course materials and slides by Fei-Fei Li, Andrej Karpathy and Justin Johnson at Stanford University
http://cs231n.stanford.edu/syllabus.html
Learning to See
From Eyes ... to Vision
What is vision?

What we see

What a computer sees
Vision is an inference problem

*it is a way of thinking*

Many different 3D scenes could have given rise to the same 2D picture.
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 $F$,
  - the stride $S$,
  - the amount of zero padding $P$.

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2015]
Pooling
Case Studies

**LeNet**
(1998)

**AlexNet**
(2012)

**ZFNet**
(2013)
Case Studies

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>16 weight layers</td>
<td>19 weight layers</td>
<td></td>
</tr>
<tr>
<td>conv3-32</td>
<td>conv3-64</td>
<td></td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64</td>
<td></td>
</tr>
<tr>
<td>conv3-128</td>
<td>conv3-128</td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>FC-4096</td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-1000</td>
<td></td>
</tr>
<tr>
<td>FC-1000</td>
<td>softmax</td>
<td></td>
</tr>
</tbody>
</table>

Revolution of Depth

ImageNet Classification: top-5 error (%)
Localization and Detection

Results from Faster R-CNN, Ren et al 2015
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation

Single object

Multiple objects
Computer Vision Tasks

Classification
Classification + Localization
Object Detection
Instance Segmentation
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
Idea #1: Localization as Regression

**Input:** image

**Neural Net**

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

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

**Loss:**
- L2 distance

Only one object, simpler than detection
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

Image -> Convolution and Pooling -> Final conv feature map -> Fully-connected layers -> Class scores

Fully-connected layers -> Box coordinates

“Classification head”

“Regression head”
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)
Where to attach the regression head?

After conv layers: Overfeat, VGG

After last FC layer: DeepPose, R-CNN

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 $K=4$)

![Diagram of network with convolution and pooling, followed by fully-connected layers generating class scores and box coordinates, resulting in $K \times 4$ numbers for each object.](image)
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation
Detection as Regression?

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

= many numbers

Need variable sized outputs
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

Convert regions to boxes

## 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>RandomizedPrim’s [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>10</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 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!
R-CNN Training

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

Training image regions

Cached region features

![Cat images](image1.png) ![Cat images](image2.png) ![Cat images](image3.png) ![Cat images](image4.png) ![Cat images](image5.png)

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
  - $(0.25, 0, 0, 0)$ Proposal too far to left
  - $(0, 0, -0.125, 0)$ Proposal too wide
R-CNN Results

Big improvement compared to pre-CNN methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Average Precision (mAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM (2011)</td>
<td>33.7</td>
</tr>
<tr>
<td>Regionlets (2013)</td>
<td>41.7</td>
</tr>
<tr>
<td>R-CNN (2014, AlexNet)</td>
<td>54.2</td>
</tr>
<tr>
<td>R-CNN + bbox reg (AlexNet)</td>
<td>58.5</td>
</tr>
<tr>
<td>R-CNN (VGG-16)</td>
<td>66</td>
</tr>
</tbody>
</table>

VOC 2007

VOC 2010
ImageNet Detection 2013 - 2015

ImageNet Detection (mAP)

- ResNet single (2015): 56.86
- Faster R-CNN single (2015): 53.67
- GoogLeNet ensemble (2014): 42.94
- NUS ensemble (2014): 43.63
- SPP ensemble (2014): 37.21
- UJIETvision (2013): 22.56
- Overfeat (2013): 19.4
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

YOLO: You Only Look Once
Detection as Regression

Faster than Faster R-CNN, but not as good

<table>
<thead>
<tr>
<th>Real-Time Detectors</th>
<th>Train</th>
<th>mAP</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100Hz DPM [30]</td>
<td>2007</td>
<td>16.0</td>
<td>100</td>
</tr>
<tr>
<td>Fast YOLO</td>
<td>2007+2012</td>
<td>52.7</td>
<td>155</td>
</tr>
<tr>
<td>YOLO</td>
<td>2007+2012</td>
<td>63.4</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Less Than Real-Time</th>
<th>Train</th>
<th>mAP</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fastest DPM [37]</td>
<td>2007</td>
<td>30.4</td>
<td>15</td>
</tr>
<tr>
<td>R-CNN Minus R [20]</td>
<td>2007</td>
<td>53.5</td>
<td>6</td>
</tr>
<tr>
<td>Fast R-CNN [14]</td>
<td>2007+2012</td>
<td>70.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Faster R-CNN VGG-16[27]</td>
<td>2007+2012</td>
<td>73.2</td>
<td>7</td>
</tr>
<tr>
<td>Faster R-CNN ZF [27]</td>
<td>2007+2012</td>
<td>62.1</td>
<td>18</td>
</tr>
</tbody>
</table>

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/ShaogqingRen/faster_rcnn](https://github.com/ShaogqingRen/faster_rcnn)
(Caffe + Python): [https://github.com/rbgirshick/py-faster-rcnn](https://github.com/rbgirshick/py-faster-rcnn)

**YOLO**
Maybe try this for projects?
Computational Frameworks for ConvNets

• Caffe
  http://caffe.berkeleyvision.org/

• Torch
  http://torch.ch/

• TensorFlow
  https://www.tensorflow.org/versions/r0.9/tutorials/deep_cnn/index.html

• Matconvnet
  http://www.vlfeat.org/matconvnet/
What is vision?

We learn patterns from past visual experiences and recognize them now, to create our present visual world.