Lecture 10:

Optimizing Object Detection: A Case Study of R-CNN, Fast R-CNN, and Faster R-CNN
Today’s task: object detection

Image classification: what is the object in this image?  
**tennis racket**

Object detection: where is the tennis racket in this image? (if there is one at all?)
Krizhevsky (AlexNet) image classification network

[Krizhevsky 2012]

Input: fixed size image

Output: probability of label (for 1000 class labels)

Network assigns input image one of 1000 potential labels.
VGG-16 image classification network

[Simonyan 2015]

Input:
fixed size image

Output:
probability of label
(for 1000 class labels)

Network assigns input image one of 1000 potential labels.
Today: three object detection papers

- R-CNN [Girshick 2014]
- Fast R-CNN [Girshick 2015]
- Faster R-CNN [Ren, He, Girshick, Sun 2015]

- Each paper improves on both the wall-clock performance and the detection accuracy of the previous
Using AlexNet as a “subroutine in object detection

[Gi2014]

Search over all regions of the image for objects.
(“Sliding window” over image, repeated for multiple potential object scales)

for all region top-left positions (x,y):
    for all region sizes (w,h):
        cropped = image_crop(image, bbox(x,y,w,h))
        resized = image_resize(227,227)
        label = detect_object(resized)
        if (label != background)
            // region defined by bbox(x,y,w,h) contains object
            // of class ‘label’
Optimization 1: filter detection work via object proposals

Selective search [Uijlings IJCV 2013]

Input: image
Output: list of regions (various scales) that are likely to contain objects
Idea: proposal algorithm filters parts of the image not likely to contain objects
Object detection pipeline executed only on proposed regions

[Giśhick 2014]
Object detection performance on Pascal VOC

Example training data

<table>
<thead>
<tr>
<th>VOC 2010 test</th>
<th>aero</th>
<th>bike</th>
<th>bird</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chair</th>
<th>cow</th>
<th>table</th>
<th>dog</th>
<th>horse</th>
<th>mbike</th>
<th>person</th>
<th>plant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tv</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM v5 [18]†</td>
<td>49.2</td>
<td>53.8</td>
<td>13.1</td>
<td>15.3</td>
<td>35.5</td>
<td>53.4</td>
<td>49.7</td>
<td>27.0</td>
<td>17.2</td>
<td>28.8</td>
<td>14.7</td>
<td>17.8</td>
<td>46.4</td>
<td>51.2</td>
<td>47.7</td>
<td>10.8</td>
<td>34.2</td>
<td>20.7</td>
<td>43.8</td>
<td>38.3</td>
<td>33.4</td>
</tr>
<tr>
<td>UVA [34]</td>
<td>56.2</td>
<td>42.4</td>
<td>15.3</td>
<td>12.6</td>
<td>21.8</td>
<td>49.3</td>
<td>36.8</td>
<td>46.1</td>
<td>12.9</td>
<td>32.1</td>
<td>30.0</td>
<td>36.5</td>
<td>43.5</td>
<td>52.9</td>
<td>32.9</td>
<td>15.3</td>
<td>41.1</td>
<td>31.8</td>
<td>47.0</td>
<td>44.8</td>
<td>35.1</td>
</tr>
<tr>
<td>Regionlets [36]</td>
<td>65.0</td>
<td>48.9</td>
<td>25.9</td>
<td>24.6</td>
<td>24.5</td>
<td>56.1</td>
<td>54.5</td>
<td>51.2</td>
<td>17.0</td>
<td>28.9</td>
<td>30.2</td>
<td>35.8</td>
<td>40.2</td>
<td>55.7</td>
<td>43.5</td>
<td>14.3</td>
<td>43.9</td>
<td>32.6</td>
<td>54.0</td>
<td>45.9</td>
<td>39.7</td>
</tr>
<tr>
<td>SegDPM [16]†</td>
<td>61.4</td>
<td>53.4</td>
<td>25.6</td>
<td>25.2</td>
<td>35.5</td>
<td>51.7</td>
<td>50.6</td>
<td>50.8</td>
<td>19.3</td>
<td>33.8</td>
<td>26.8</td>
<td>40.4</td>
<td>48.3</td>
<td>54.4</td>
<td>47.1</td>
<td>14.8</td>
<td>38.7</td>
<td>35.0</td>
<td>52.8</td>
<td>43.1</td>
<td>40.4</td>
</tr>
<tr>
<td>R-CNN</td>
<td>67.1</td>
<td>64.1</td>
<td>46.7</td>
<td>32.0</td>
<td>30.5</td>
<td>56.4</td>
<td>57.2</td>
<td>65.9</td>
<td>27.0</td>
<td>47.3</td>
<td>40.9</td>
<td>66.6</td>
<td>57.8</td>
<td>65.9</td>
<td>53.6</td>
<td>26.7</td>
<td>56.5</td>
<td>38.1</td>
<td>52.8</td>
<td>50.2</td>
<td>50.2</td>
</tr>
</tbody>
</table>

“Fine tuned” DNN weights obtained by “pretraining” for object classification on ImageNet for the 20 VOC categories (+ 1 “background” category)
Optimization 2: region of interest pooling

RGB input image: (of any size)

“Fully convolutional network”: sequence of convolutions and pooling steps: output size dependent on input size

Idea: the output of early convolutional layers of network on downsampled input region is approximated by resampling output of fully-convolutional implementation of conv layers.

Performance optimization: can evaluate convolutional layers once on large input, then reuse intermediate output many times to approximate response of a subregion of image.
Optimization 2: region of interest pooling

Form of “approximate common subexpression elimination”

for all proposed regions \((x, y, w, h)\):  
\[
\text{cropped} = \text{image\_crop(image, bbox(x, y, w, h))}
\]
\[
\text{resized} = \text{image\_resize(227, 227)}
\]
\[
\text{label} = \text{detect\_object(resized)}
\]

\[
\text{conv5\_response} = \text{evaluate\_conv\_layers(image)}
\]

for all proposed regions \((x, y, w, h)\):
\[
\text{region\_conv5} = \text{roi\_pool(conv5\_response, bbox(x, y, w, h))}
\]
\[
\text{label} = \text{evaluate\_fully\_connected\_layers(region\_conv5)}
\]

Redundant work (many regions overlap, so responses at lower network layers are computed many times)

Computed once per image
Fast R-CNN pipeline [Girshick 2015]

Evaluation speed: 146x faster than R-CNN (47sec/img → 0.32 sec/img)
[This number excludes cost of proposals]

Training speed: 9x faster than R-CNN
Training mini-batch: pick N images, pick 128/N boxes from each image (allows sharing of conv-layer pre-computation for multiple image-box training samples)
Simultaneously train class predictions and bbox predictions: joint loss = class label loss + bbox loss
Note: training updates weights in BOTH fully connected/softmax layers AND cons layers

| method         | train set | aero | bike | bird | boat | bottle | bus | car  | cat  | chair | cow  | table | dog  | horse | mbike | persn | plant | sheep | sofa  | train | tv    | mAP  |
|----------------|-----------|------|------|------|------|--------|-----|------|------|------|------|-------|------|-------|-------|-------|-------|-------|-------|-------|-------|
| R-CNN BB [10]  | 12        | 79.3 | 72.4 | 63.1 | 44.0 | 44.4   | 64.6| 66.3 | 84.9 | 38.8 | 67.3 | 48.4  | 82.3 | 75.0  | 65.7  | 35.8  | 66.2 | 54.8  | 69.1  | 58.8  | 62.9  |
| FRCN [ours]    | 12        | 80.1 | 74.4 | 67.7 | 49.4 | 41.4   | 74.2| 68.8 | 87.8 | 41.9 | 70.1 | 50.2  | 86.1 | 81.1  | 70.4  | 33.3  | 67.0 | 63.3  | 77.2  | 60.0  | 66.1  |
Problem: bottleneck is now generating proposals

Input image: (of any size) → Object Proposal generator → List of proposed regions (~2000)

DNN (conv layers only!) → Response maps

ROI pooling layer → Pixel region (of canonical size) → Fully-connected layers

for each proposed region

class-label softmax
bbox regression softmax
bbox
object label

Selective search [Uijlings 13] ~ 10 sec/image on CPU
EdgeBoxes [Zitnick 14] ~ 0.2 sec/image on CPU

Idea: why not predict regions from the convolutional feature maps that must be computed for detection anyway? (share computation between proposals and detection)
Faster R-CNN using a region proposal network (RPN)

[Ren 2015]

Input image: (of any size)

DNN (conv layers only!)

Response maps

Region proposal network

List of proposed regions

for each proposed region

ROI pooling layer
Faster R-CNN using a region proposal network (RPN)

Input image:
(of any size)

→ DNN
(conv layers only!)

Response maps
WxHx512

512 3x3 conv filters
(3x3x512x512 weights)

→ 1x1 conv
(2-way softmax)
512 x (9*2) weights

objectness score
(for 9 boxes)

→ 1x1 conv
(bbox regressor)
512 x (9x4) weights

bbox offset
(for 9 boxes)

3x3 conv projects into 512-element vector per spatial position (assuming VGG input conv layers, receptive field for each output is ~228x228 pixels)

At each point assume 9 “anchor boxes” of various aspect ratios and scales

Given 512-element vector predict “objectness score” of each anchor + bbox correction to anchor
Training faster R-CNN

Goal: want to jointly learn
- Region prediction network weights
- Object classification network weights
- While constraining initial conv layers to be the same (for efficiency)
Alternating training strategy

- Train RPN
- Then use trained RPN to train Fast R-CNN
- Use conv layers from R-CNN to initialize RPN
- Fine-tune RPN
- Use updated RPN to fine tune Fast R-CNN
- Repeat...

Notice: solution learns to predict boxes that are “good for object-detection task”

- “End-to-end” optimization for object-detection task
- Compare to using off-the-shelf object-proposal algorithm
## Faster R-CNN results

### Specializing region proposals for object-detection task yields better accuracy.

SS = selective search

<table>
<thead>
<tr>
<th>method</th>
<th># proposals</th>
<th>data</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>2000</td>
<td>12</td>
<td>65.7</td>
</tr>
<tr>
<td>SS</td>
<td>2000</td>
<td>07++12</td>
<td>68.4</td>
</tr>
<tr>
<td>RPN+VGG, shared†</td>
<td>300</td>
<td>12</td>
<td>67.0</td>
</tr>
<tr>
<td>RPN+VGG, shared‡</td>
<td>300</td>
<td>07++12</td>
<td><strong>70.4</strong></td>
</tr>
</tbody>
</table>

### Shared convolutions improve algorithm performance:

**Times in ms**

<table>
<thead>
<tr>
<th>model</th>
<th>system</th>
<th>conv</th>
<th>proposal</th>
<th>region-wise</th>
<th>total</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG</td>
<td>SS + Fast R-CNN</td>
<td>146</td>
<td>1510</td>
<td>174</td>
<td>1830</td>
<td>0.5 fps</td>
</tr>
<tr>
<td>VGG</td>
<td>RPN + Fast R-CNN</td>
<td>141</td>
<td><strong>10</strong></td>
<td>47</td>
<td>198</td>
<td>5 fps</td>
</tr>
</tbody>
</table>
Summary

- Detailed knowledge of algorithm and properties of DNN used to gain algorithmic speedups
  - Not just “tune the schedule of the loops”

- Key insight: sharing results of convolutional layer computations:
  - Between different proposed regions
  - Between region proposal logic and detection logic

- Push for “end-to-end” training
  - Clean: back-propagate through entire algorithm to train all components at once
  - Better accuracy: globally optimize the various parts of the algorithm to be optimal for task (here: how to propose boxes learned simultaneously with detection logic)
  - Can constrain learning to preserve performance characteristics (conv layer weights must age shared across RPN and detection task)
Emerging theme

(from today’s lecture and the Inception, SqueezeNet, and related readings)

- Computer vision practitioners are “programming” via low-level manipulation of DNN topology
  - See shift from reasoning about individual layers to writing up of basic “microarchitecture” modules (e.g., Inception module)

- What programming model constructs or “automated compilation” tools could help raise the level of abstraction?