Lecture 10:

Optimizing Object Detection:
A Case Study of R-CNN, Fast R-CNN, and Faster R-CNN and Single Shot Detection

Visual Computing Systems
Stanford CS348V, Winter 2018
Today’s task: object detection

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

Object detection involves localization: where is the tennis racket in this image? (if there is one at all?)
Quick review

- Why did we say that DNNs learn “good features”?
Krizhevsky (AlexNet) image classification network

[Krizhevsky 2012]

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: several 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

- Also Single Shot Detection (SSD) [Liu 2016]
- And Mask-RCNN for instance segmentation [He 2017]
Using classification network as a “subroutine for object detection

[Girshick 2014]

Search over all regions of the image and all region sizes 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

**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

**Selective search [Uijlings IJCV 2013]**

Image 1: 
- **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

---

**Figure 2:** Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales, while on the right we necessarily find the objects at different scales as the girl is contained by the TV.

---

**2.3 Other Sampling Strategies**

This work uses multiple complementary strategies to deal with as many image conditions as possible, thereby severely reducing the number of windows evaluated by class-specific object detectors. We compare our method with their work.

---

**3 Selective Search**

The goal of selective search is to yield a small set of class-independent object locations. Instead of focusing on the segmentation of objects, we use the image structure to sample boxes as evidence for the possible object locations for use in a practical object recognition and present a variety of diversification strategies to deal with as many image conditions as possible. A selective search is subject to the following considerations:

- **Fast to Compute.**
- **Capture All Scales.**
- **Diversification.**

In this section we detail our selective search algorithm for object recognition and present a variety of diversification strategies to deal with as many image conditions as possible. The creation of this set should not become a computational bottleneck, hence our algorithm should be reasonably fast.

---

**References**

- Maji and Malik [23]
- Vedaldi et al.
- Alexe et al.
- Uijlings et al.
Object detection pipeline executed only on proposed regions

[Girshick 2014]
Object detection performance on Pascal VOC

### Example training data

<table>
<thead>
<tr>
<th>Object</th>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplanes</td>
<td><img src="image1.png" alt="Example image" /></td>
<td><img src="image2.png" alt="Example image" /></td>
<td><img src="image3.png" alt="Example image" /></td>
</tr>
<tr>
<td>cow</td>
<td><img src="cow1.png" alt="Example image" /></td>
<td><img src="cow2.png" alt="Example image" /></td>
<td><img src="cow3.png" alt="Example image" /></td>
</tr>
</tbody>
</table>

### Table 1: Detection average precision (%) on VOC 2010 test.

<table>
<thead>
<tr>
<th>Method</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>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>
</tr>
<tr>
<td>UVA [34]</td>
<td>56.2</td>
<td>49.3</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>
</tr>
<tr>
<td>Regionlets [36]</td>
<td>65.0</td>
<td>56.1</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>
</tr>
<tr>
<td>SegDPM [16]</td>
<td>61.4</td>
<td>51.7</td>
<td>50.6</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>
</tr>
<tr>
<td>R-CNN</td>
<td>67.1</td>
<td>56.4</td>
<td>57.2</td>
<td>65.9</td>
<td>57.8</td>
<td>47.3</td>
<td>40.9</td>
<td>66.6</td>
<td>57.8</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>
</tr>
</tbody>
</table>

### DNN weights “pre-trained” using object classification on ImageNet (lots of data, different task)

DNN weights “fine-tuned” for the 20 VOC categories + 1 “background” category (task-specific data)
Optimization 2: region of interest pooling

RGB input image: (of any size)

“Fully convolutional network”: sequence of convolutions and pooling steps: output size is 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

This is a form of “approximate common subexpression elimination”

for all proposed regions \((x,y,w,h)\):  // 1000’s of regions/image
cropped = image_crop(image, bbox(x,y,w,h))
resized = image_resize(227,227)
label = detect_object(resized)

conv5_response = evaluate_conv_layers(image)
for all proposed regions \((x,y,w,h)\):
    region_conv5 = roi_pool(conv5_response, bbox(x,y,w,h))
    label = evaluate_fully_connected_layers(region_conv5)
Fast R-CNN pipeline [Girshick 2015]

Input image: (of any size)

Object Proposal generator

List of proposed regions (~2000)

DNN (conv layers only!)

Response maps

for each proposed region

Pixel region (of canonical size)

Fully-connected layers

class-label softmax

bbox regression softmax

bbox

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 conv layers

<table>
<thead>
<tr>
<th>method</th>
<th>train set</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>persn</th>
<th>plant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tv</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-CNN BB [10]</td>
<td>12</td>
<td>79.3</td>
<td>72.4</td>
<td>63.1</td>
<td>44.0</td>
<td>44.4</td>
<td>64.6</td>
<td>66.3</td>
<td>84.9</td>
<td>38.8</td>
<td>67.3</td>
<td>48.4</td>
<td>82.3</td>
<td>75.0</td>
<td>76.7</td>
<td>65.7</td>
<td>35.8</td>
<td>66.2</td>
<td>54.8</td>
<td>69.1</td>
<td>58.8</td>
<td>62.9</td>
</tr>
<tr>
<td>FRCN [ours]</td>
<td>12</td>
<td>80.1</td>
<td>74.4</td>
<td>67.7</td>
<td>49.4</td>
<td>41.4</td>
<td>74.2</td>
<td>68.8</td>
<td>87.8</td>
<td>41.9</td>
<td>70.1</td>
<td>50.2</td>
<td>86.1</td>
<td>81.1</td>
<td>70.4</td>
<td>33.3</td>
<td>67.0</td>
<td>63.3</td>
<td>77.2</td>
<td>60.0</td>
<td>66.1</td>
<td></td>
</tr>
</tbody>
</table>
Problem: bottleneck is now generating proposals

Selectieve 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]
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 region proposal network (RPN)
  - Using loss based on ground-truth object bounding boxes
  - Positive example: intersection over union with ground truth box above threshold
  - Negative example: intersection over union less than threshold
- Then use trained RPN to train Fast R-CNN
  - Using loss based on detections and bbox regression
- Use conv layers from R-CNN to initialize RPN
- Fine-tune RPN
  - Using loss based on ground-truth boxes
- Use updated RPN to fine tune Fast R-CNN
  - Using loss based on detections and bbox regression
- 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 for object proposals

<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>70.4</td>
</tr>
</tbody>
</table>

Shared convolutions improve algorithm performance:
Values are 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>10</td>
<td>47</td>
<td>198</td>
<td>5 fps</td>
</tr>
</tbody>
</table>
Summary

- Knowledge of algorithm and properties of DNN used to gain algorithmic speedups
  - Not just “modify the schedule of the loops”

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

- Example of “end-to-end” training
  - 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 given task (Faster R-CNN: how to propose boxes learned simultaneously with detection logic)
  - Can constrain learning to preserve performance characteristics (Faster R-CNN: conv layer weights shared across RPN and detection task)
Extending to instance segmentation

[Image credit: He et al. 2017]
Mask RCNN

- Extend Faster R-CNN to also emit a segmentation per box
  - Previously: box and class emitted in parallel
  - Now: box, class, and segmentation emitted in parallel

Pixel region of canonical size (7x7)
Output of resampling the region generated by the Faster R-CNN region proposal network

Faster R-CNN w/ ResNet [19]

per-class scores (80)
bbox adjustments

binary masks for 80 output classes

ROI pooling layer (ROIAlign)
Mask R-CNN for human pose

- Loss based on bitmapped with hot pixels at joint keypoint locations rather than segmentation masks
An alternative approach to object detection

Recall structure of algorithms so far: (reduce detection to classification)

for all proposed regions \((x,y,w,h)\):
    \[
    \text{cropped} = \text{image\_crop}(\text{image}, \text{bbox}(x,y,w,h))
    \]
    \[
    \text{resized} = \text{image\_resize}(\text{classifier\_width}, \text{classifier\_height})
    \]
    \[
    \text{label} = \text{classify\_object}(\text{resized})
    \]
    \[
    \text{bbox\_adjustment} = \text{adjust\_bbox}(\text{resized})
    \]

New approach to detection:

for each level \(l\) of network:
    for each \((x,y)\) position in output:
        use region around \((l,x,y)\) to directly predict which anchor boxes centered at \((x,y)\) are valid and class score for that box

If there are \(B\) anchor boxes and \(C\) classes, then...
At each \((l,x,y)\), prediction network has \(B(C + 4)\) outputs

For each anchor \(B\), there are \(C\) class probabilities + 4 values to adjust the anchor box
SSD: Single shot multi box detector

multibox detectors operating on different scales of features

If feature maps have P channels (e.g., P=512 and 256 below)
Each classifier is a $3 \times 3 \times P$ filter
$(C + 4)$ filters for one anchor bbox
(assume one of the C categories is “background”)

If feature maps have P channels (e.g., P=512 and 256 below)
Each classifier is a $3 \times 3 \times P$ filter
$(C + 4)$ filters for one anchor bbox
(assume one of the C categories is “background”)

Deep feature hierarchy of fully convolutional layers

Note: diagram shows only the feature maps
SSD anchor boxes

(a) Image with GT boxes
(b) 8 × 8 feature map
(c) 4 × 4 feature map

Anchor boxes at each feature map level are of different sizes

Intuition: receptive field of cells at higher levels of the network (lower resolution feature maps) is a larger fraction of the image, have information to make predictions for larger boxes
## Object detection performance

### 600x600 input images

### COCO-trained models

<table>
<thead>
<tr>
<th>Model name</th>
<th>Speed (ms)</th>
<th>COCO mAP[^1]</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssd_mobilenet_v1_coco</td>
<td>30</td>
<td>21</td>
<td>Boxes</td>
</tr>
<tr>
<td>ssd_inception_v2_coco</td>
<td>42</td>
<td>24</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_v2_coco</td>
<td>58</td>
<td>28</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet50_coco</td>
<td>89</td>
<td>30</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet50_lowproposals_coco</td>
<td>64</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>rfcn_resnet101_coco</td>
<td>92</td>
<td>30</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet101_coco</td>
<td>106</td>
<td>32</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet101_lowproposals_coco</td>
<td>82</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_resnet_v2_atrous_coco</td>
<td>620</td>
<td>37</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco</td>
<td>241</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_nas</td>
<td>1833</td>
<td>43</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_nas_lowproposals_coco</td>
<td>540</td>
<td></td>
<td>Boxes</td>
</tr>
</tbody>
</table>

[^1]: COCO mAP score

[Credit: Tensorflow detection model zoo]
Discussion

- Why did we say that DNNs learn “good features”?
- Consider Mask R-CNN
Discussion

- Today we saw our first examples of end-to-end learning
  - Idea: globally optimize all parts of a topology for specific task at hand
  - Empirically: often enables better accuracy (and also better performance)

- Interesting to consider effects on interpretability of these models (modularity is typically a favorable property of software)
Emerging theme
(from today’s lecture and the Inception, MobileNet, 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)
  - Differentiable programming

- Interesting question: what programming model constructs or “automated compilation” tools could help raise the level of abstraction?