Lecture 21: Faster RCNN

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2021

Outline

Review: How to train a neural network

1. Find a **training dataset** that contains n examples showing the desired output, $\vec{y}_i$, that the NN should compute in response to input vector $\vec{x}_i$:

$$\mathcal{D} = \{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_n, \vec{y}_n)\}$$

2. Randomly **initialize** the weights and biases, $W^{(1)}$, $\vec{b}^{(1)}$, $W^{(2)}$, and $\vec{b}^{(2)}$.
3. Perform **forward propagation**: find out what the neural net computes as $\hat{y}_i$ for each $\vec{x}_i$.
4. Define a **loss function** that measures how badly $\hat{y}$ differs from $\vec{y}$.
5. Perform **back propagation** to improve $W^{(1)}$, $\vec{b}^{(1)}$, $W^{(2)}$, and $\vec{b}^{(2)}$.
6. Repeat steps 3-5 until convergence.

Review: Fully-connected and Convolutional Neural Networks

- Fully-connected layers: forward-prop is a matrix multiplication, back-prop is multiplication by the transposed matrix, weight gradient is a vector outer product.
- Convolutional layers: forward-prop is a convolution, back-prop is a correlation, weight gradient is a convolution.
- Max pooling: back-prop just propagates the derivative to the pixel that was chosen by forward-prop.

Error Metrics Summarized

- Use MSE to achieve $\hat{y} \rightarrow E\left[\vec{y}|\vec{x}\right]$. That's almost always what you want.
- For a binary classifier with a sigmoid output, BCE loss gives you the MSE result without the vanishing gradient problem.
- For a multi-class classifier with a softmax output, CE loss gives you the MSE result without the vanishing gradient problem.
- After you're done training, you can make your cell phone app more efficient by throwing away the uncertainty:
  - Replace softmax output nodes with max
  - Replace logistic output nodes with unit-step
  - Replace tanh output nodes with signum

Outline

Object Recognition vs. Object Detection

- **Object Recognition**
  - The task: Decide which objects are present in an image.
  - SOTA solution: very deep convolutional neural nets.
- **Object Detection**
  - The task: Figure out where the object is in the image.
  - SOTA solution: RPN w.r.t. anchors fixed w.r.t. ROI.

Object Detection Example

WIDER FACE Dataset (Yang, Luo, Loy & Tang, 2016)

- Dataset published 2015 w/13k images, 300k+ faces.
- Bounding box for each face given as (x,y,w,h).
- Metadata: blur, expression, illumination, occlusion, pose.
- In the example at right, I've eliminated all faces with nonzero blur, illumination, or occlusion.

Object Detection as Classification

Suppose that we are given a region of interest, $ROI = (x, y, w, h)$, and asked to decide whether the ROI is an object. We can do this by training a neural network to estimate the classifier output:

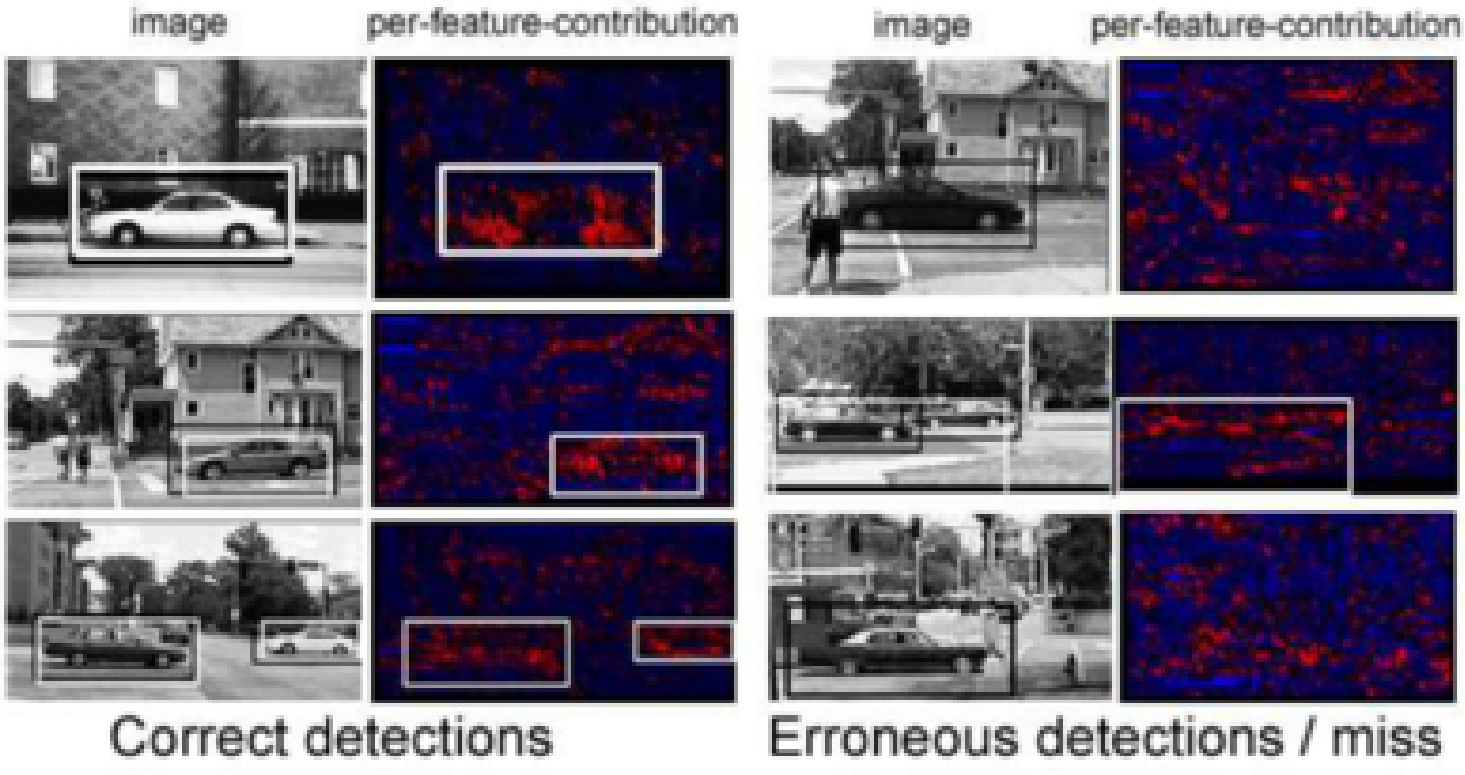
$$y_c(ROI) = \begin{cases} 1 & \text{ROI contains an object} \\ 0 & \text{ROI does not contain an object} \end{cases}$$

A neural net trained with MSE or CE will then compute

$$\hat{y}_c = \Pr(\text{ROI contains an object})$$

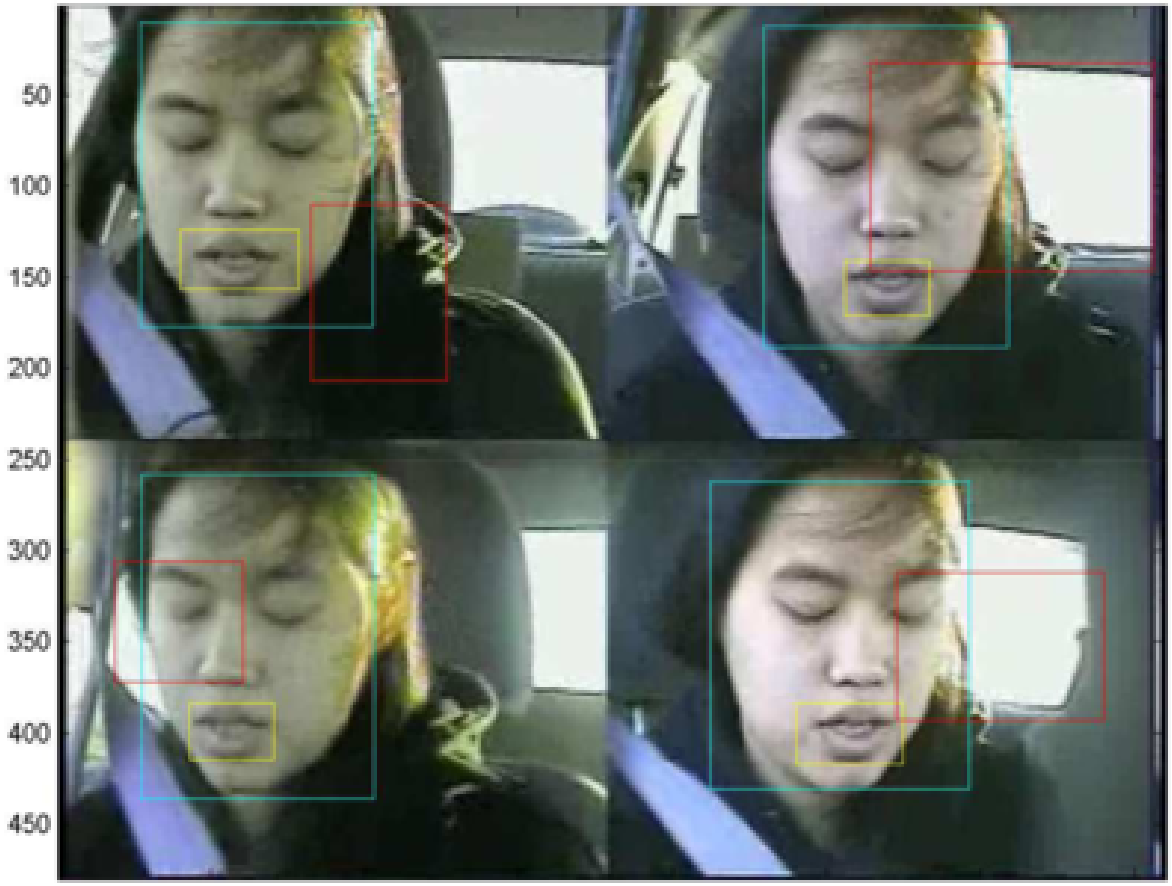
Training a network for object detection

Back-prop to the individual pixels can show the degree to which each pixel contributes to the detection probability. Here's an example based on Gaussian supervectors (Zhuang et al., "Efficient Object Localization with Gaussianized Vector Representation," 2009):



What about partial overlap?

Real networks need to deal with situations of partial overlap, e.g.,



Lee, Hasegawa-Johnson, Goudeseune, Kamdar, Borys, Liu & Huang (2004)

Intersection over union (IOU)

We deal with partial-overlap by putting some sort of threshold on the intersection-over-union measure. Suppose the hypothesis is $(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI})$, and the reference is $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$, then IOU is

$$IOU = \frac{I}{U} = \frac{\text{number of pixels in both ROI and REF}}{\text{number of pixels in either ROI or REF}},$$

where the intersection between REF and ROI is:

$$I = (\min(x_{REF} + w_{REF}, x_{ROI} + w_{ROI}) - \max(x_{REF}, x_{ROI})) \times (\min(y_{REF} + h_{REF}, y_{ROI} + h_{ROI}) - \max(y_{REF}, y_{ROI})),$$

and their union is:

$$U = w_{REF} h_{REF} + w_{ROI} h_{ROI} - I$$

Arbitrary Thresholds on IOU

We could use IOU as a soft-measure, or could we put some sort of arbitrary threshold, like:

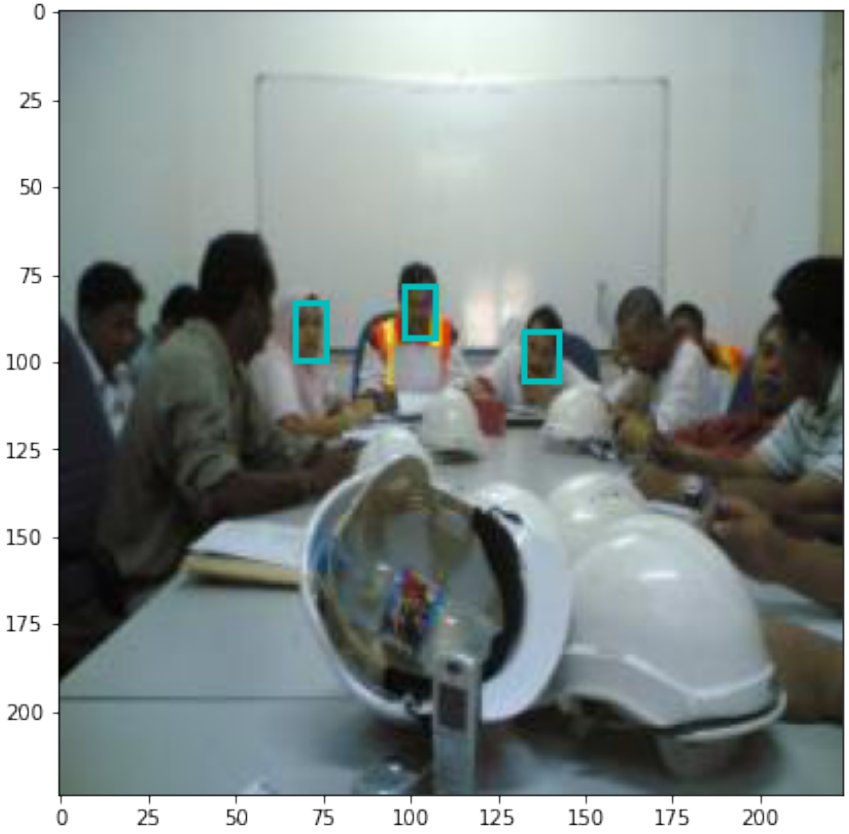
$$y_c(ROI) = \begin{cases} 1 & IOU > 0.7 \\ 0 & \text{otherwise} \end{cases}$$

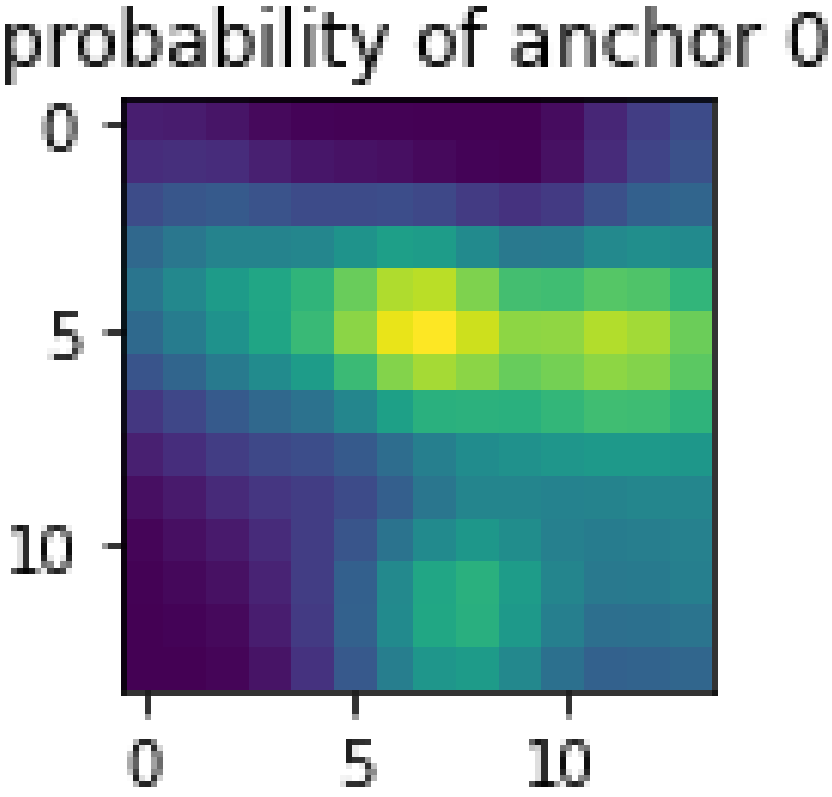
Then we get:

$$\hat{y}_c = \Pr(IOU > 0.7)$$

Training a network for object detection

Here is one of the MP3 object detectors:





Outline

Why Object Detection is Hard: Too Many Rectangles

- Suppose the image is $N \times N$, e.g., $N \approx 1000$.
- A bounding-box rectangle is (x, y, w, h), so there are $O\left\{N^4\right\} \approx 10^{12}$ rectangles to evaluate.
- If it takes the classifier $100\mu s$ to evalute one rectangle, then it takes 10^8 seconds $= 3.17$ years to evaluate all of the rectangles in an image.

Object Detection: Solutions

1. Very fast classifiers: e.g., Viola-Jones Adaboost.
2. Region proposal network (RPN): category-independent object proposals.
3. Fast RCNN: RPN computed as a nonlinear regression, w.r.t. a predefined ROI.
4. Faster-RCNN: RPN computed as a nonlinear regression, w.r.t. a predefined anchor, which is defined w.r.t. a predefined ROI.

Solution #1: Very Fast Classifiers

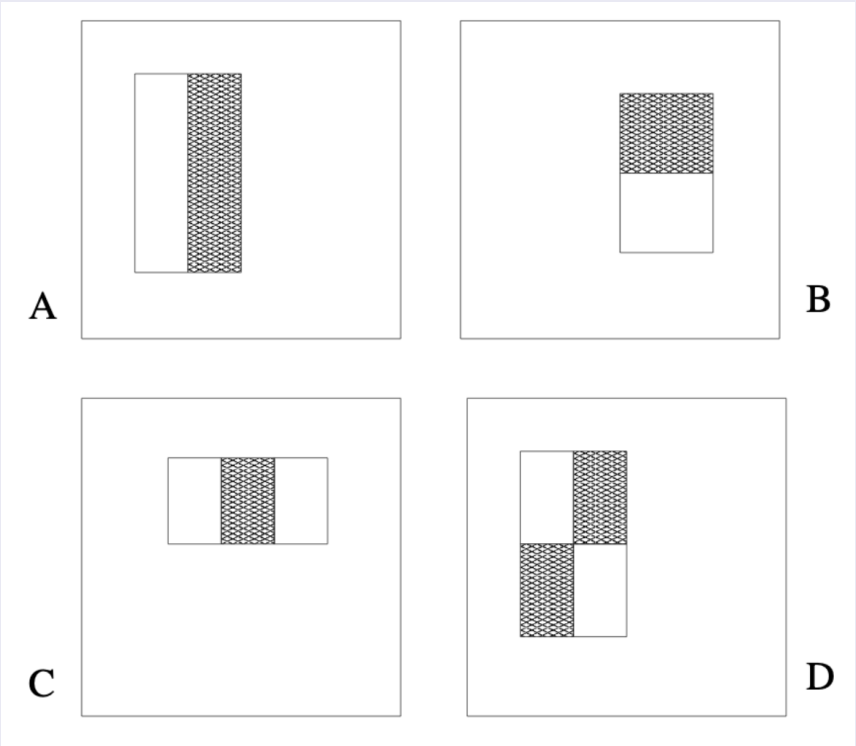


Image copyright Viola & Jones, 2001

"Rapid Object Detection using a Boosted Cascade of Simple Features," Viola and Jones, 2001

- Each weak classifier evaluates just one Haar feature (features shown at left), which can be computed using only ~ 6 additions/rectangle.
- Most rectangles eliminated after a cascade of just two weak classifiers (so: nanoseconds, not microseconds).

Solution #2: "Category-Independent Object Proposals," Endres & Hoiem, 2010

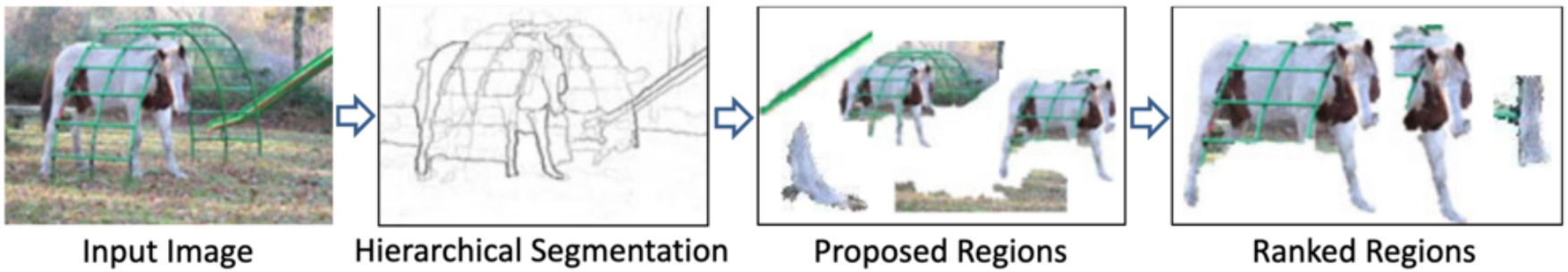


Image copyright Endres & Hoiem, 2010

- Pixels accumulated into candidate regions-of-interest (ROI) based on similarity of texture, color, etc.
- Candidate ROIs ranked by a neural net.
- Neural net trained to decide whether an ROI contains a nameable object or not, regardless of what type of object it is.

Solution #3: "Fast RCNN," Girshick, 2015

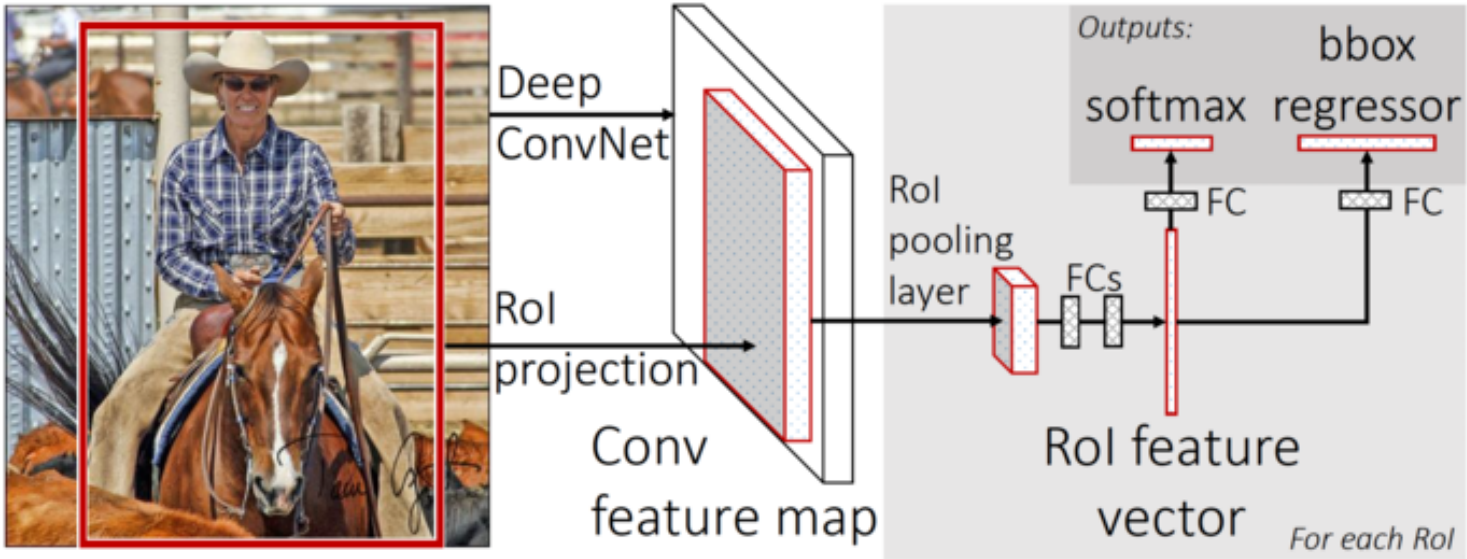


Image copyright Girshick, 2015

- Start with a small set of candidate ROIs (a few hundred per image)
- Each ROI feeds a neural net whose output is a 4-vector specifying the (x,y,w,h) of the nearest object.

ROI: Variable vs. Fixed

- Previous object detectors, up through RCNN, computed ROI candidates in a bottom-up fashion, so that different images would have different ROI candidates.
- Fast RCNN proposed using fixed ROI candidates, that are exactly the same in every image.
  - That way, you don't have to waste time figuring out where the ROIs are.
  - Girchick's proposal: just use the last convolutional layer of an object recognizer like VGG16.

VGG16: "Very Deep Convolutional Networks for Large-Scale Image Classification"

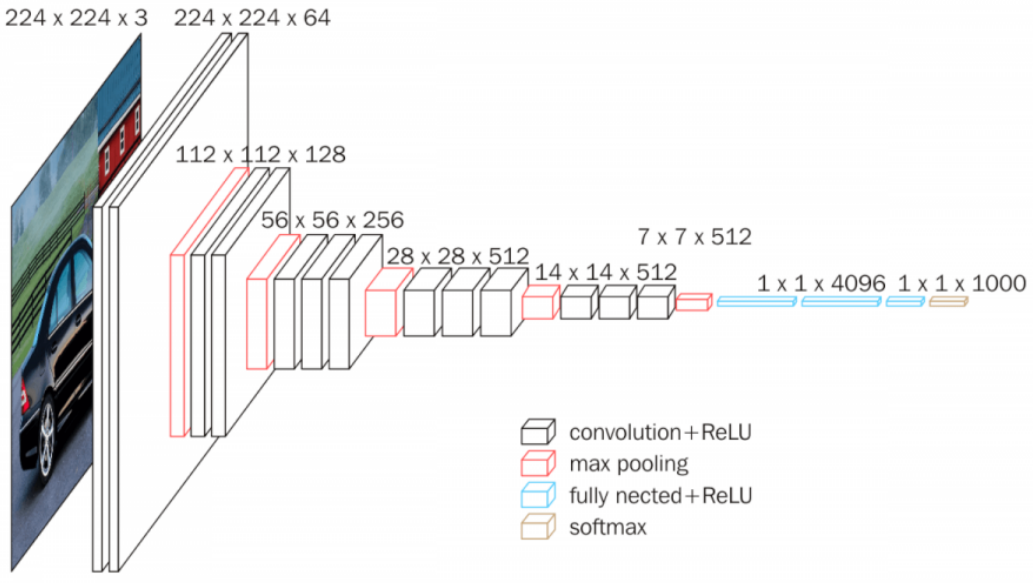
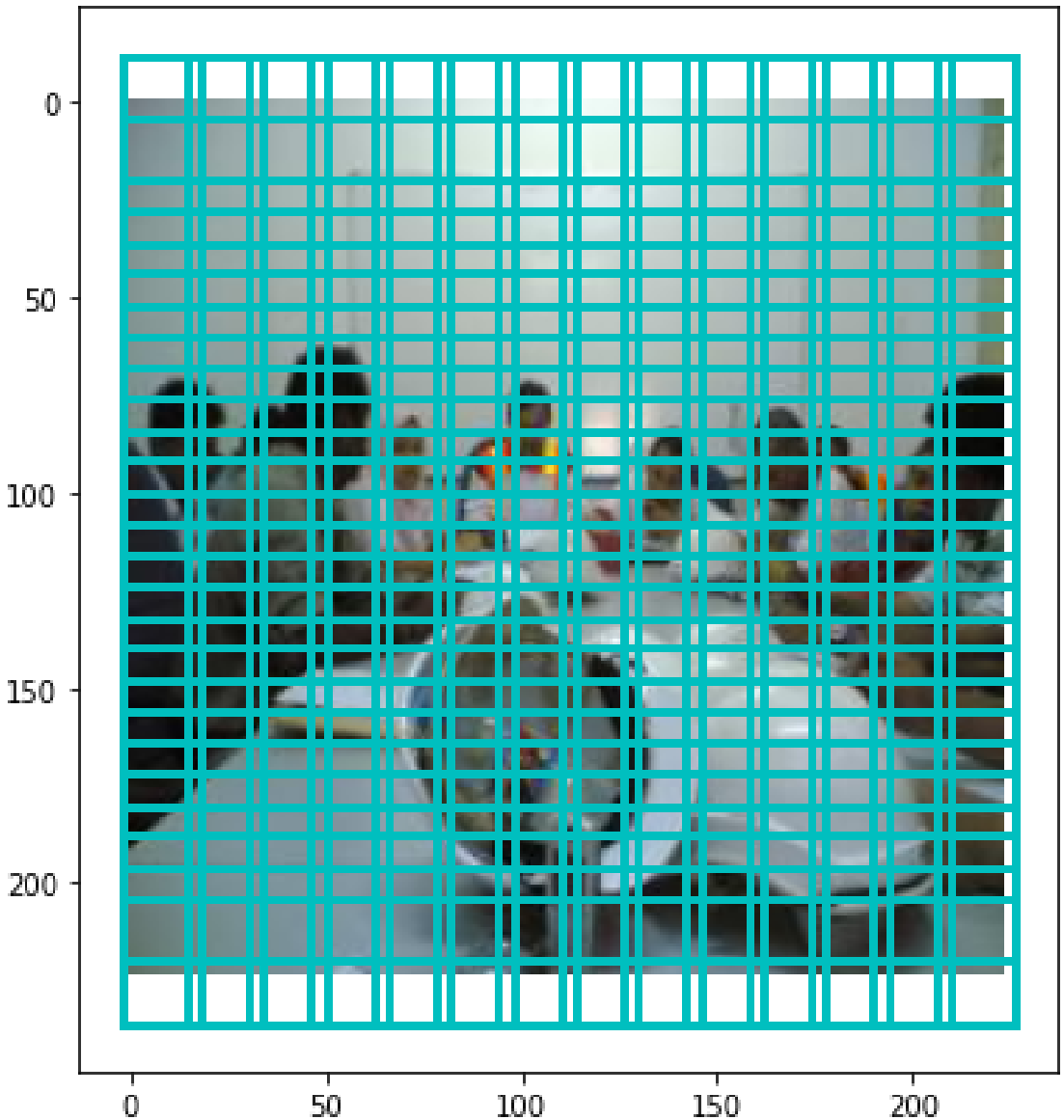


Image copyright Simonyan & Zisserman, 2015

- Input normalized to 224×224 pixels, 3 color channels.
- Last convolutional layer is 14×14 pixels, 512 channels. Call this $\vec{f}[m, n]$, where $\vec{f} \in \Re^{512}$, $0 \leq (m, n) \leq 13$.
- Output FCN trained for object recognition: 1000 different object types.

Last conv layer contains $14 \times 14 = 196$ ROIs



ROI = 3×3 grid of VGG16 feature vectors

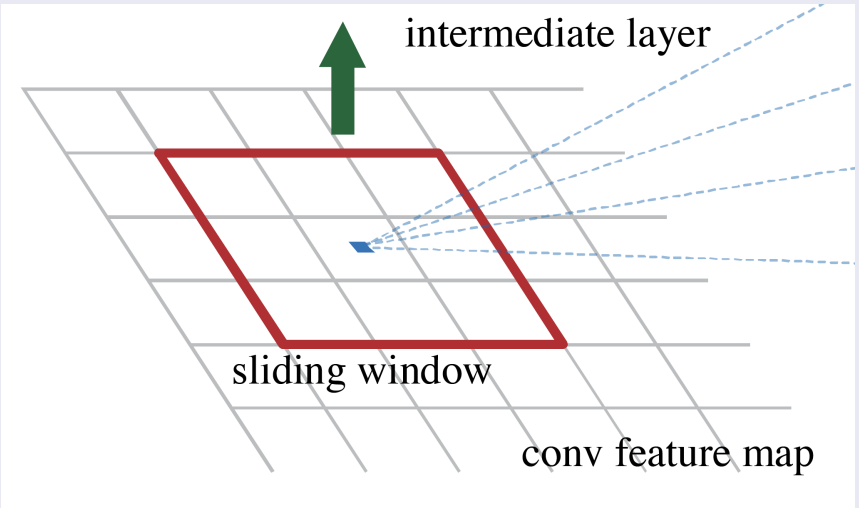


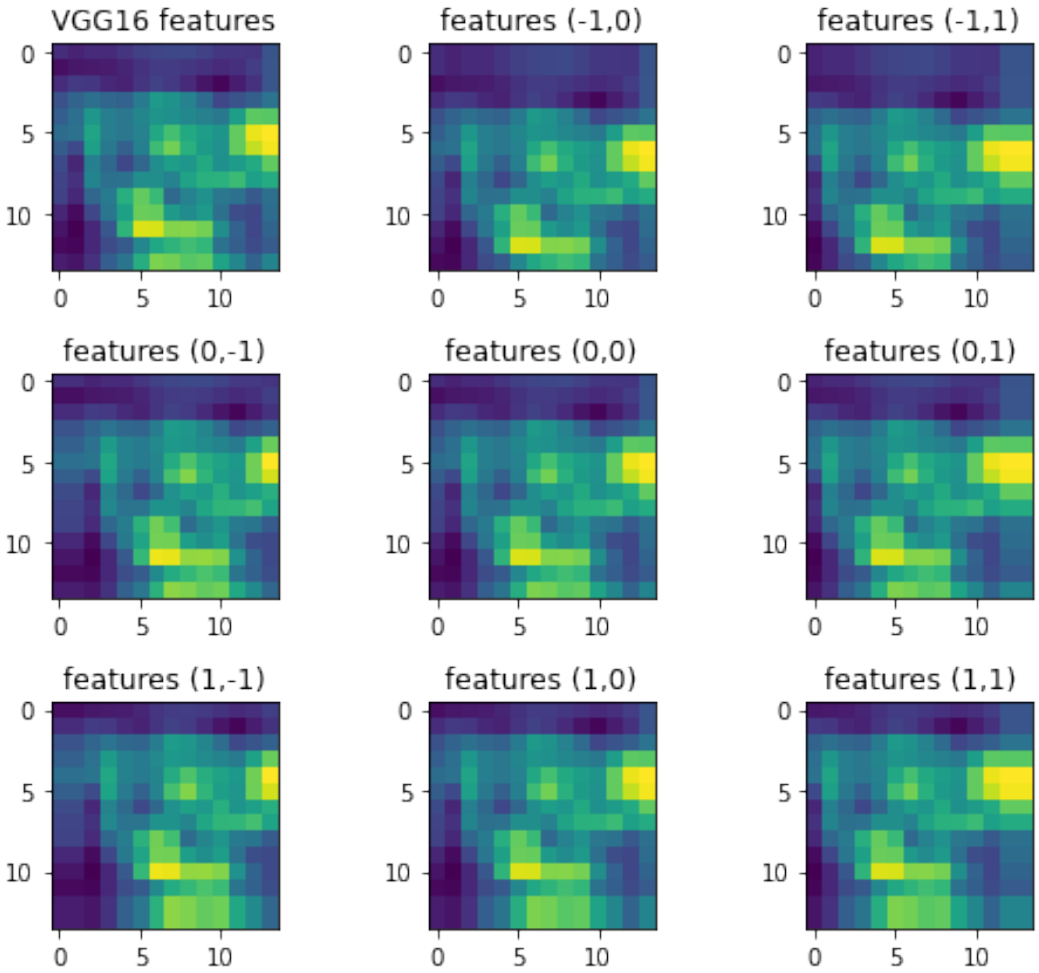
Image copyright Ren, He, Girshick & Sun, 2016

The region proposal network takes, as input, the concatenation of nine neighboring feature vectors from the VGG16 layer:

$$\vec{x}_{m,n} = \begin{bmatrix} \vec{f}[m-1, n-1] \\ \vec{f}[m-1, n] \\ \vdots \\ \vec{f}[m+1, n+1] \end{bmatrix}$$

Notice, we could think of this as another convolutional layer, but Fast RCNN just treats it as a fully-connected network.

ROI $= 3 \times 3$ grid of VGG16 feature vectors



$$\vec{x}_{m,n} = [\vec{f}[m-1, n-1], \vec{f}[m-1, n], \ldots, \vec{f}[m+1, n+1]]$$

Outline

What pixels are covered by the ROI called $\vec{f}_{m,n}$?

The $(m, n)^{\text{th}}$ feature vector, $\vec{f}_{m,n}$, covers a particular block of pixels in the input image:

$$(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI}) = (76n, 76m, 228, 228)$$

- Each $\vec{x}[m, n]$ covers 76×76 input pixels.
- Each $\vec{f}_{m,n}$ is $(3 \cdot 76) \times (3 \cdot 76) = 228 \times 228$.
- $m \rightarrow y$ is the vertical axis, $n \rightarrow x$ horizontal.

What pixels **should** be covered?

Suppose the nearest true object is in rectangle $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$. We want to somehow encode the difference between where we are now $(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI})$ and where we want to be $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$. Fast RCNN does this using the following target vector, $\vec{y}_r$, for the neural network:

$$\vec{y}_r = \begin{bmatrix} \frac{x_{REF} - x_{ROI}}{w_{ROI}} \\ \frac{y_{REF} - y_{ROI}}{h_{ROI}} \\ \ln\left(\frac{w_{REF}}{w_{ROI}}\right) \\ \ln\left(\frac{h_{REF}}{h_{ROI}}\right) \end{bmatrix}$$

The neural net is trained to find a $\hat{y}_r$ that is as close as possible to $\vec{y}_r$ (minimum MSE).

Training a bbox regression network

The network is now trained with two different outputs, $\hat{y}_c$ and $\hat{y}_r$. The total loss is

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_r$$

where $\mathcal{L}_c$ is BCE for the classifier output:

$$\mathcal{L}_c = -\frac{1}{n}\sum_{i=1}^{n}\left(y_{c,i}\ln\hat{y}_{c,i} + (1-y_{c,i})\ln(1-\hat{y}_{c,i})\right)$$

and $\mathcal{L}_r$ is zero if $y_c = 0$ (no object present), and MSE if $y_c = 1$:

$$\mathcal{L}_r = \frac{1}{2n}\sum_{i=1}^{n} y_{c,i}\|\vec{y}_{r,i} - \hat{y}_{r,i}\|^2$$

Outline

"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Ren, He, Girshick & Sun, 2016

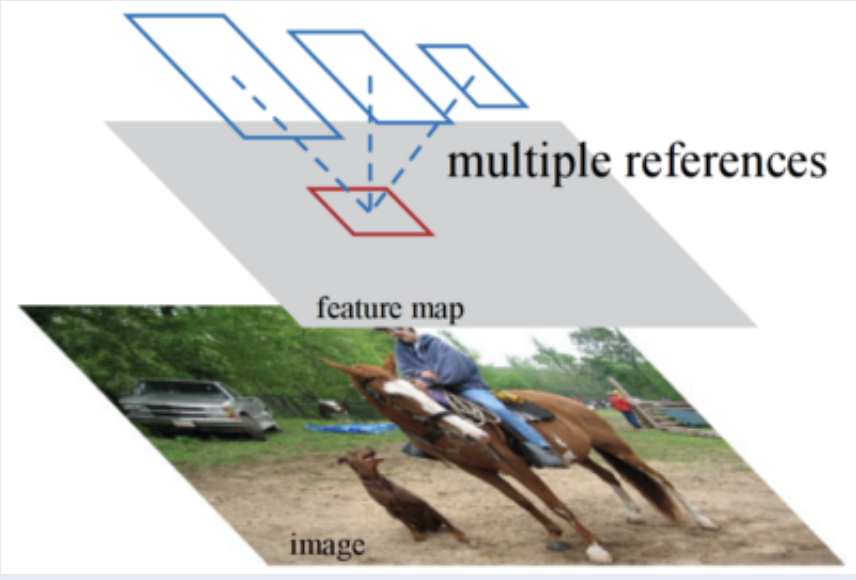


Image copyright Ren, He, Girchick & Sun, 2016

- Each candidate bounding box computes 9 different regression outputs, each of which is a 4-vector (x,y,w,h)
- The 9 different regression outputs from each bbox are w.r.t. 9 different "anchor" rectangles, each offset from the input ROI. Thus:

$$\text{anchor} = \text{ROI} + \text{known shift}$$

$$\text{object} = \text{anchor} + \text{regression}$$

What pixels **should** be covered?

- The ROI is $(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI})$.
- The anchor is (x_a, y_a, w_a, h_a).
- The true object is located at $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$.
- The regression target is:

$$\vec{y}_r = \begin{bmatrix} \frac{x_{REF} - x_a}{w_a} \\ \frac{y_{REF} - y_a}{h_a} \\ \ln\left(\frac{w_{REF}}{w_a}\right) \\ \ln\left(\frac{h_{REF}}{h_a}\right) \end{bmatrix}$$

3 sizes, 3 aspect ratios

The Faster RCNN paper described 9 anchors per ROI:

- 3 different anchor sizes: 128×128, 256×256, and 512×512.
- 3 different aspect ratios: $1:2$, $1:1$, and $2:1$

9 anchors per ROI

Outline

Summary

- An ROI network has a 4608d input, corresponding to a 3×3 grid of 512d feature vectors from the last conv layer of a VGG16 object recognizer.
- Faster-RCNN defines 9 different anchors centered on each ROI.
- W.r.t. each anchor, we define the classification target $y_c = 1$ if $IOU > 0.7$, otherwise $y_c = 0$.
- If $y_c = 1$, then we define a regression target $\vec{y}_r$, specifying how much the REF bbox differs from the anchor.