Lecture 22
Deep Learning:
Object Detection and Segmentation

COS 429: Computer Vision

Thanks: most of these slides shamelessly adapted from Stanford CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li, Andrej Karpathy, Justin Johnson
http://cs231n.stanford.edu/
Step 1: Preprocess the data

(Assume X [NxD] is data matrix, each example in a row)
Define Network Architecture
Fully Connected layer

\[ B_j = \sum_i (W_{ij} * A_i) + b_j \]
**Convolution Layer**

- 32x32x3 image
- 5x5x3 filter
- Convolve (slide) over all spatial locations
- Activation maps

32 32 3
5x5x3 filter
28 28
Single depth slice

max pool with 2x2 filters and stride 2
Activation Layer

**Sigmoid**

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

\[ \text{tanh} \quad \text{tanh}(x) \]

\[ \text{ReLU} \quad \text{max}(0, x) \]

**Leaky ReLU**

\[ \text{max}(0.1x, x) \]

**Maxout**

\[ \text{max}(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**

\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
Batch Normalization

**Input:** Values of \( x \) over a mini-batch: \( \mathcal{B} = \{x_1 \ldots m\} \);
Parameters to be learned: \( \gamma, \beta \)

**Output:** \( \{y_i = \text{BN}_{\gamma, \beta}(x_i)\} \)

\[
\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}
\]

\[
\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \quad \text{// mini-batch variance}
\]

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \quad \text{// normalize}
\]

\[
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad \text{// scale and shift}
\]

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe
Weight Initialization

input layer had mean 0.086501 and std 0.999444
hidden layer 1 had mean 0.562480 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545676 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826982
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587183 and std 0.869035
hidden layer 7 had mean 0.596667 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567490 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523

He et al., 2015
(note additional /2)

\[ W = \text{np.random.randn}(\text{fan\_in}, \text{fan\_out}) / \text{np.sqrt}(\text{fan\_in}/2) \]  # layer initialization

\[
\frac{1}{2} \sum_{i} \text{Var}[w_i] = 1 \quad \text{ours}
\]

\[
\frac{1}{2} \sum_{i} \text{Var}[w_i] = 1 \quad \text{Xavier}
\]
Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”

-Srivastava et al., 2014-
Regularization: **DisturbLabel**

“randomly change ground truth label of small % of examples”

=> Improves generalization, reduces need for dropout
Case Study: GoogLeNet [Szegedy et al., 2014]

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Mini-batch SGD

Loop:
1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient
CIFAR-10 experiments

CIFAR-10 plain nets

- 56-layer
- 44-layer
- 32-layer
- 20-layer

solid: test
dashed: train

CIFAR-10 ResNets

- 20-layer
- 32-layer
- 44-layer
- 56-layer
- 110-layer
Inception Resnet V2 Network

Compressed View

Convolution, MaxPool, AvgPool, Concat, Dropout, Fully Connected, Softmax, Residual
ImageNet Classification Error (Top 5)

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>AlexNet</td>
<td>16.4</td>
</tr>
<tr>
<td>2013</td>
<td>ZF</td>
<td>11.7</td>
</tr>
<tr>
<td>2014</td>
<td>VGG</td>
<td>7.3</td>
</tr>
<tr>
<td>2014</td>
<td>GoogLeNet</td>
<td>6.7</td>
</tr>
<tr>
<td>2015</td>
<td>ResNet</td>
<td>3.57</td>
</tr>
<tr>
<td></td>
<td>GoogLeNet-v4</td>
<td>3.08</td>
</tr>
</tbody>
</table>

Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation

CAT

CAT

CAT, DOG, DUCK

CAT, DOG, DUCK

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
Classification + Localization: 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 guess has correct class AND bounding box at least 0.5 intersection over union (IoU)

Krizhevsky 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

**Input:** image

- 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
Simple Recipe for Classification + Localization

**Step 3:** Train the regression head only with SGD and L2 loss
Step 4: At test time use both heads

Simple Recipe for Classification + Localization

Convolution and Pooling

Image

Final conv feature map

Fully-connected layers

Class scores

Box coordinates

Fully-connected layers
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**: \( C \times 4 \) numbers (one box per class)
Where to attach the regression head?

- **After conv layers:** Overfeat, VGG
- **After last FC layer:** DeepPose, R-CNN

Diagram:
- 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 $K=4$)
Aside: 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)

Idea #2: Sliding Window

- Run classification + regression network at multiple locations on a high-resolution image

- Convert fully-connected layers into convolutional layers for efficient computation

- Combine classifier and regressor predictions across all scales for final prediction
Sliding Window: Overfeat

Image: 3 x 221 x 221

Convolution + pooling

Feature map: 1024 x 5 x 5

4096

Class scores: 1000

Softmax loss

4096

Euclidean loss

4096

1024

Boxes: 1000 x 4


Winner of ILSVRC 2013 localization challenge
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

Classification scores:
P(cat) = 0.5
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
Sliding Window: Overfeat

Network input: 3 x 221 x 221

Larger image: 3 x 257 x 257

Classification scores:
P(cat)

<table>
<thead>
<tr>
<th>0.5</th>
<th>0.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

Classification scores:
P(cat)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>0.6</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Sliding Window: Overfeat

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

Classification scores:
P(cat)
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) = 0.8
Sliding Window: Overfeat

In practice use many sliding window locations and multiple scales

Window positions + score maps

Box regression outputs

Final Predictions

Efficient Sliding Window: Overfeat

Image: 3 x 221 x 221

Convolution + pooling

Feature map: 1024 x 5 x 5

Class scores: 1000

Boxes: 1000 x 4

4096 → FC → 4096 → FC

4096 → FC → 1024 → FC
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

4096 x 1 x 1 → 1024 x 1 x 1

5 x 5 conv

1 x 1 conv

5 x 5 conv

1 x 1 conv

4096 x 1 x 1 → 1024 x 1 x 1

Class scores: 1000 x 1 x 1

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

1 x 1 conv

1 x 1 conv

1 x 1 conv
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

ImageNet Classification + Localization

- **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

![Localization Error (Top 5)](chart)

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Error</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>
Computer Vision Tasks

- Classification
- Classification + Localization
- Object Detection
- Instance Segmentation
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation
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 numbers
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

CAT? NO
DOG? NO
Detection as Classification

CAT? YES!
DOG? NO
Detection as Classification

CAT? NO
DOG? NO
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
Deformable Parts Model (DPM)

Aside: Deformable Parts Models are CNNs?

Girschick et al, “Deformable Part Models are Convolutional Neural Networks”, CVPR 2015
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

Region Proposals: Many other choices

<table>
<thead>
<tr>
<th>Method</th>
<th>Approach</th>
<th>Outputs</th>
<th>Outputs</th>
<th>Control</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>Rantalanikila [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)

- **Image**
- **Convolution and Pooling**
- **Final conv feature map**
- **Fully-connected layers**
- **Class scores 1000 classes**
- **Softmax loss**
**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

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
**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.
Object Detection: 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 (train + val)</td>
<td>~20k</td>
<td>~470k</td>
<td>~120k</td>
</tr>
<tr>
<td>Mean objects per image</td>
<td>2.4</td>
<td>1.1</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Object Detection: Evaluation

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

R-CNN Results

Big improvement compared to pre-CNN methods

Mean Average Precision (mAP)

<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>66.0</td>
<td>62.9</td>
</tr>
</tbody>
</table>
R-CNN Results

Bounding box regression helps a bit

<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>66.0</td>
<td>62.9</td>
</tr>
</tbody>
</table>
R-CNN Results

Features from a deeper network help a lot

![Bar Chart]

<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>66.0</td>
<td>62.9</td>
</tr>
</tbody>
</table>
R-CNN Problems

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)

Regions of Interest (RoIs) from a proposal method

Softmax classifier
Linear + softmax
Linear
Bounding-box regressors

Fully-connected layers
“RoI Pooling” (single-level SPP) layer
“conv5” feature map of image
Forward whole image through ConvNet

ConvNet
Input image

Slide credit: Ross Girschick
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
Fast R-CNN (training)

Log loss + smooth L1 loss

Multi-task loss

Linear + softmax

Linear

FCs

Trainable

ConvNet

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
Fast R-CNN: Region of Interest 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
Fast R-CNN: Region of Interest 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

Project region proposal onto conv feature map

Problem: Fully-connected layers expect low-res conv features: C x h x w

Convolution and Pooling

Fully-connected layers
Fast R-CNN: Region of Interest Pooling

Hi-res input image: 3 x 800 x 600 with region proposal

Convolution and Pooling

Hi-res conv features: C x H x W with region proposal

Divide projected region into h x w grid

Fully-connected layers

Problem: Fully-connected layers expect low-res conv features: C x h x w
Fast R-CNN: Region of Interest 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: Region of Interest 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
- Convolution and Pooling
- Can back propagate similar to max pooling
- Fully-connected layers
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>

Using VGG-16 CNN on Pascal VOC 2007 dataset
<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>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>
<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>Better!</td>
<td>mAP (VOC 2007)</td>
<td>66.0</td>
</tr>
</tbody>
</table>

Using VGG-16 CNN on Pascal VOC 2007 dataset
Fast R-CNN Problem:

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 Problem Solution:

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 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>
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
Slide a small window on the feature map

Build a small network for:
- classifying object or not-object, and
- regressing bbox locations

Position of the sliding window provides localization information with reference to the image

Box regression provides finer localization information with reference to this sliding window

Slide credit: Kaiming He
Faster R-CNN: Region Proposal Network

Use **N anchor boxes** at each location

Anchors are **translation invariant**: use the same ones at every location

Regression gives offsets from anchor boxes

Classification gives the probability that each (regressed) anchor shows an object
Faster R-CNN: Training

In the paper: Ugly pipeline
- Use alternating optimization to train RPN, then Fast R-CNN with RPN proposals, etc.
- More complex than it has to be

Since publication: Joint training!
One network, four losses
- RPN classification (anchor good / bad)
- RPN regression (anchor -> proposal)
- Fast R-CNN classification (over classes)
- Fast R-CNN regression (proposal -> box)
### Faster R-CNN: Results

<table>
<thead>
<tr>
<th></th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
<th>Faster R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time per image (with proposals)</td>
<td>50 seconds</td>
<td>2 seconds</td>
<td>0.2 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>25x</td>
<td>250x</td>
</tr>
<tr>
<td>mAP (VOC 2007)</td>
<td>66.0</td>
<td>66.9</td>
<td>66.9</td>
</tr>
</tbody>
</table>
Object Detection State-of-the-art: ResNet 101 + Faster R-CNN + some extras

<table>
<thead>
<tr>
<th>training data</th>
<th>COCO train</th>
<th>COCO trainval</th>
</tr>
</thead>
<tbody>
<tr>
<td>test data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mAP</td>
<td>@ .5</td>
<td>@ [.5, .95]</td>
</tr>
<tr>
<td>baseline Faster R-CNN (VGG-16)</td>
<td>41.5</td>
<td>27.2</td>
</tr>
<tr>
<td>baseline Faster R-CNN (ResNet-101)</td>
<td>48.4</td>
<td>29.9</td>
</tr>
<tr>
<td>+box refinement</td>
<td>49.9</td>
<td></td>
</tr>
<tr>
<td>+context</td>
<td>51.1</td>
<td>30.0</td>
</tr>
<tr>
<td>+multi-scale testing</td>
<td>53.8</td>
<td>32.5</td>
</tr>
<tr>
<td>ensemble</td>
<td>59.0</td>
<td>37.4</td>
</tr>
</tbody>
</table>

ImageNet Detection 2013 - 2015

ImageNet Detection (mAP)

mAP

- NeoNet ensemble (2015): 53.57
- Faster R-CNN single (2015): 42.94
- GoogleNet ensemble (2014): 43.93
- NUS ensemble (2014): 37.21
- SPP ensemble (2014): 35.11
- UvA-Eurvision (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>
Computer Vision Tasks

**Classification**

- Single object
  - CAT

**Classification + Localization**

- Single object
  - CAT

**Object Detection**

- Multiple objects
  - CAT, DOG, DUCK

**Segmentation**

- Multiple objects
  - CAT, DOG, DUCK
Computer Vision Tasks

Classification

+ Localization

Object Detection

Segmentation

Lecture 8
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Segmentation

Today
Semantic Segmentation

Label every pixel!

Don’t differentiate instances (cows)

Classic computer vision problem

Figure credit: Shotton et al, “TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context”, IJCV 2007
Instance Segmentation

Detect instances, give category, label pixels

“simultaneous detection and segmentation” (SDS)

Lots of recent work (MS-COCO)

Figure credit: Dai et al, "Instance-aware Semantic Segmentation via Multi-task Network Cascades", arXiv 2015