Computer Vision
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley
So far...

Convolutional networks: map image to output value

e.g., semantic category ("bicycle")
Standard computer vision problems

object classification

object localization

object detection

semantic segmentation
a.k.a. scene understanding
Object localization setup

Before: \( \mathcal{D} = \{(x_i, y_i)\} \)
- image
- class label (categorical)

Now: \( \mathcal{D} = \{(x_i, y_i)\} \)
- image
- \( y_i = (l_i, x_i, y_i, w_i, h_i) \)
Measuring localization accuracy

Did we get it right?

Intersection over Union (IoU)

IoU = I / U

Different datasets have different protocols, but one reasonable one is: **correct if IoU > 0.5**

If also outputting class label (usually the case): **correct if IoU > 0.5 and class is correct**

This is **not** a loss function! Just an evaluation standard
Object localization as regression

\[ \mathcal{D} = \{(x_i, y_i)\} \quad y_i = (\ell_i, x_i, y_i, w_i, h_i) \]

- Very simple design
- Can either train jointly (multi-task), or train with classification first, then train regression head
- More or less works OK
- By itself, this is **not** the way it’s usually done!
  - We’ll see why shortly

**cross-entropy loss**

**regression loss**

(e.g., Gaussian log-likelihood, MSE)
What if we classify every patch in the image?

\[ \mathcal{D} = \{(x_i, y_i)\} \quad y_i = (\ell_i, x_i, y_i, w_i, h_i) \]
Sliding windows

\[ \mathcal{D} = \{(x_i, y_i)\} \quad y_i = (\ell_i, x_i, y_i, w_i, h_i) \]

could just take the box with the highest class probability

more generally: non-maximal suppression
A practical approach: OverFeat

\[ \mathcal{D} = \{ (x_i, y_i) \} \quad y_i = (\ell_i, x_i, y_i, w_i, h_i) \]


➢ Pretrain on just classification
➢ Train regression head on top of classification features
➢ Pass over different regions at different scales
➢ “Average” together the boxes to get a single answer

provides a little “correction” to sliding window
A practical approach: OverFeat

Sliding window **classification** outputs at each scale/position (*yellow* = bear)

Predicted box x, y, w, h at each scale/position (*yellow* = bear)

Final combined bounding box prediction (*yellow* = bear)

Sermanet et al. “**OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.**” 2013
Sliding windows & reusing calculations

**Problem:** sliding window is very expensive! (36 windows = 36x the compute cost)

This looks a lot like convolution...

Can we just **reuse** calculations across windows?

"Convolutional classification"

Before: 2x fully connected layers with 4096 units

Now: 2x **convolutional** layers with 1x1x4096 filters
Sliding windows & reusing calculations

This kind of calculation reuse is extremely powerful for localization problems with conv nets.

We’ll see variants of this idea in every method we’ll cover today!

Summary

**Building block:** conv net that outputs class and bounding box coordinates

**Evaluate** this network at multiple scales and for many different crops, each one producing a probability and bounding box

**Implement** the sliding window as just another convolution, with 1x1 convolutions for the classifier/regressor at the end, to save on computation
Object detection architectures
The problem setup

Before

\((x_i,\) 

number of objects \(n_i\) different for each image \(x_i\)!

Now

\[ \text{“cat”: 0.21} \]

\[ (x, y, w, h) \quad ??? \]
How do we get multiple outputs?

**Sliding window:** each window can be a different object

Instead of selecting the window with the highest probability (or merging windows), just output an object in each window above some threshold

**Big problem:** a high-scoring window probably has other high-scoring windows nearby

**Non-maximal suppression:** (informally) kill off any detections that have other higher-scoring detections of the same class nearby

**Actually output multiple things:** output is a list of bounding boxes

**Obvious problem:** need to pick number, usually pretty small

```
{ "cat": 0.21, "dog": 0.54 }
```

```
(x_1, y_1, w_1, h_1, x_2, y_2, w_2, h_2)
```
Case study: you only live once (YOLO) look

Actually, you look a few times (49 times to be exact...)

different output for each of the 7x7 (49) grid cells (a bit like sliding window)

use the same trick as before to reuse computation (cost is not 49x higher!)

for each cell, output:
- \( (x, y, w, h) \) (confidence)
- \( \text{IoU} \) (class label)
- \( \ell \) (class label)

zero if no object

output \( B \) of these

some training details:
- need to assign which output is “responsible” for each true object during training
- just use the “best-fit” object in that cell (i.e., the one with highest IoU)

What if we have too many objects?

Well, nothing... we just miss them

CNNs + Region proposals

A smarter “sliding window”: region of interest proposals

This is really slow

But we already know how to fix this!

CNNs + Region proposals

A smarter “sliding window”: region of interest proposals

Compare this to evaluating every location:

CNNs + Region proposals

How to train region of interest proposals?

Very similar design to what we saw before (e.g., OverFeat, YOLO), but now for predicting if any object is present around that location.

Ren et al. “Faster R-CNN.” 2015
Suggested readings

  - Just regress to different bounding boxes in each cell
  - A few follow-ups (e.g., YOLO v5) that work better
  - Uses region of interest proposals instead of sliding window/convolution
- Ren et al. “Faster R-CNN.” 2015
  - Same as above with a few improvements, like region of interest proposal learning
- Liu et al. SSD: Single Shot MultiBox Detector. 2015
  - Directly “classifies” locations with class and bounding box shape
Segmentation architectures
The problem setup

Before

Now

Simple solution:
“per pixel” classifier

Label every single pixel with its class

Actually simpler in some sense:

• No longer variable # of outputs
• Every pixel has a label

Problem:

We want the output to have the same resolution as the input!

Not hard if we never downsample (i.e., zero padding, stride 1, no pooling), but that is very expensive
The problem setup

Classify every point with a class

Don’t worry for now about instances
(e.g., two adjacent cows are just one “cow blob,” and that’s OK for some reason)

The challenge: design a network architecture that makes this “per-pixel classification” problem computationally tractable
Fully convolutional networks

Slide borrowed from Fei-Fei Li, Justin Johnson, Serena Yeung
Up-sampling/transpose convolution

**Normal convolutions:** reduce resolution with **stride**

Stride = 2

- Input: $H_f \times W_f \times C_{in}$
- Output: $1 \times 1 \times C_{out}$
- Filter: $H_f \times W_f \times C_{in} \times C_{out}$

**Transpose convolutions:** increase resolution with **fractional “stride”**

Stride = 1/2

- Input: $1 \times 1 \times C_{in}$
- Output: $H_f \times W_f \times C_{out}$
- Filter: $C_{in} \times H_f \times W_f \times C_{out}$

We have two sets of values here! Just average them.
Un-pooling

Max Pooling
Remember which element was max!

Input: 4 x 4

Output: 2 x 2

Max Unpooling
Use positions from pooling layer

Input: 2 x 2

Output: 4 x 4

Rest of the network

Corresponding pairs of downsampling and upsampling layers

Slide borrowed from Fei-Fei Li, Justin Johnson, Serena Yeung
Bottleneck architecture

U-Net architecture


concatenate activations from conv layers to upsampling layers
Standard computer vision problems

- Object classification
- Object localization
- Object detection
- Semantic segmentation
  a.k.a. scene understanding