Deep Learning for Visual Computing

Object Detection

Christopher Pramerdorfer
Computer Vision Lab, TU Wien

Image from reddit, created using midjourney

Topics

Beyond image classification

Object detection

- Introduction & performance metrics

Deep object detection

- Sliding window approach
- Two-stage detectors (R-CNN)
- One-stage detectors (YOLO)
- Feature pyramid networks

Beyond Image Classification

We have focused on image classification

- Fundamental computer vision task
- We now know how to achieve good performance

Rest of lecture will focus on more challenging tasks

Beyond Image Classification

Recall our ingredients for image classification

- A suitable dataset
- A network with a suitable final layer (linear layer)
- A suitable loss function $L(\boldsymbol{\theta})$ (cross-entropy)
- An algorithm for computing $\nabla L(\boldsymbol{\theta})$ (backpropagation)
- An algorithm for updating $\boldsymbol{\theta}$ on this basis (Adam)
- An intuitive performance metric (accuracy)

Beyond Image Classification

These ingredients are universal

Steps 4 and 5 are not task-specific and can be reused

So when tackling a new problem, ask yourself

- Do I have enough (labeled) data?
- What is a suitable network output?
- What is a suitable loss function?
- What is a good metric for evaluation?

Beyond Image Classification

Answer to first question is almost always “no”

- Networks are usually pre-trained on ImageNet
- First learn what different objects look like
- Then adapt for other tasks (transfer learning)

PyTorch provides such models directly

- `net = models.resnet34(pretrained=True)`

Object Detection
Task Definition

Given an image and C class labels (e.g. {bird, cat, dog})

- Draw bounding boxes around all instances of all classes
- Assign correct class label to each bounding box

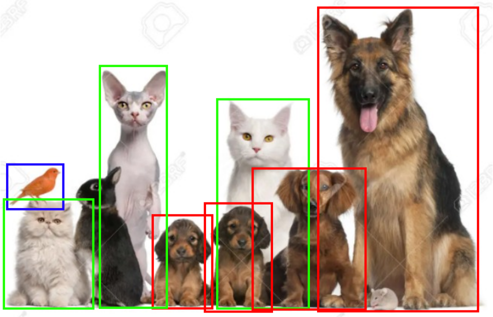


Image adapted from 123rf.com

More challenging than image classification

- Same basic challenges (see lecture 1)
- More complex task
- Harder to implement efficiently

Many useful applications such as

- Face detection (e.g. surveillance)
- Autonomous driving (e.g. road sign detection)

COCO is the most popular dataset with detection labels

Image from cocodataset.org

Object Detection
Progress

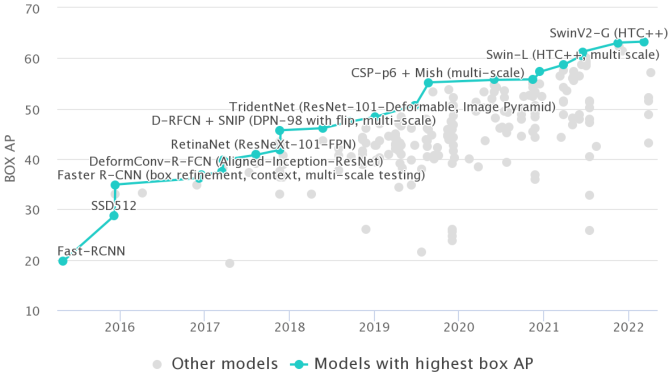


Image from paperswithcode.com

What is a suitable object detector output?

Most detectors predict the following

- N bounding boxes $\mathcal{B}_n$
- N corresponding class labels c_n
- N corresponding confidence scores p_n

Two tasks at once

- Predict accurate bounding boxes (regression)
- Assign them correct class labels (classification)

We thus usually have $L(\boldsymbol{\theta}) = L_{\mathsf{loc}}(\boldsymbol{\theta}) + L_{\mathsf{clf}}(\boldsymbol{\theta})$

- First term measures bounding box accuracy (e.g. L1 loss)
- Second term is classification loss (e.g. cross-entropy loss)

Intersection over union (IoU) base measure

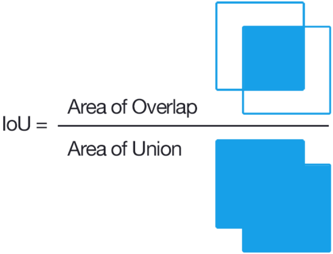


Image from Wikipedia

Ignoring class labels and given a

- Single ground-truth bounding box $\mathcal{B}'$
- Single predicted bounding box $\mathcal{B}$
- IoU threshold t_{iou}

$\mathcal{B}$ is a

- True positive (TP) if $\mathrm{iou}(\mathcal{B}', \mathcal{B}) \geq t_{iou}$
- False positive (FP) otherwise

In second case there is also one false negative (FN)

- No $\mathcal{B}$ with sufficient IoU with $\mathcal{B}'$

Ignoring class labels and given multiple bounding boxes

- Precision = TPs / (TPs + FPs)
- Recall = TPs / (TPs + FNs)

In case of multiple $\mathcal{B}$ for a given $\mathcal{B}'$

- Count $\mathcal{B}$ with highest p as TP, others als FPs

Both metrics are in $[0, 1]$

- Ideally precision $=$ recall $= 1$

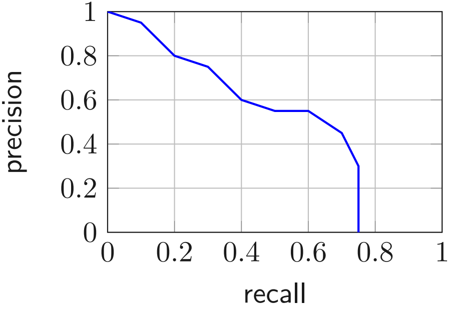
Use confidence threshold t_p to balance recall vs. precision

Increasing t_p results in fewer detections

- Usually increases precision
- Usually decreases recall

Precision vs. recall curves highlight this behavior

- Plotting precision and recall over $t_p \in [0, 1]$
- Area under curve is called average precision (AP)



AP is most popular base metric

- Concise measure of detector performance
- For single t_{iou} and single class

Mean average precision (mAP) extends this to C classes

$$\mathsf{mAP}_{t_{iou}} = \frac{1}{C} \sum_{c}^{C} \mathsf{AP}_c$$

And further to multiple t_{iou}, e.g.

$$\text{mAP} = \frac{1}{10}(\text{mAP}_{0.50} + \cdots + \text{mAP}_{0.95})$$

Confusingly mAP is sometimes named just AP in papers

- Like in COCO performance chart above

Deep Object Detection

Many interesting approaches and methods exist

- See survey papers such as [1] for an overview

Virtually all use a classification network as backbone

- Transfer learning as mentioned earlier

We will cover three approaches

- Naive sliding window approach
- Two-stage detectors (R-CNN)
- Single-stage detectors (YOLO)

Training

- Train a classifier for $C + 1$ classes
- Additional "background" class

Detection

- Slide fixed-size window over image
- Predict class-scores for every window
- Perform non-maximum suppression

Deep Object Detection
Sliding Window Approach

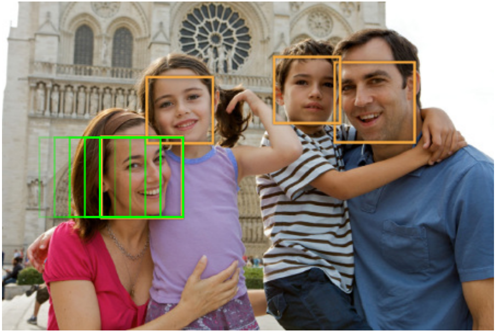


Image adapted from apple.com

Inefficient

- Many windows to classify

Single fixed-size window (no scale invariance)

- Must process image at multiple scales (more inefficient)
- Inaccurate bounding boxes (fixed aspect ratio)

Cannot handle multiple objects in same window (softmax)

Deep Object Detection
Two-Stage Detectors

Improve efficiency by

- First generating many region proposals
- Classifying these proposals

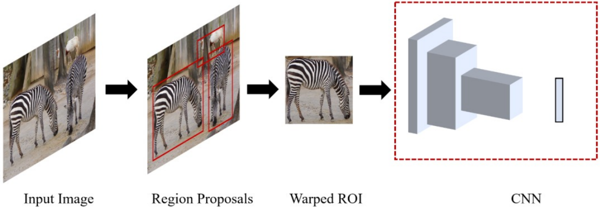


Image adapted from [1]

Region proposals have different size and aspect ratio

- Solves window-related problems
- Warp to common size to support minibatch predictions

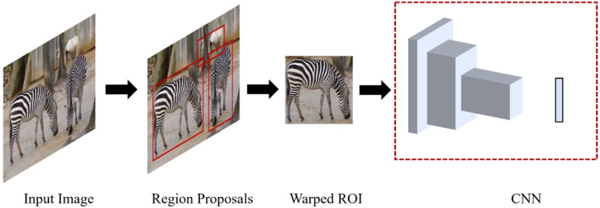


Image adapted from [1]

Deep Object Detection
Two-Stage Detectors

Approach of R-CNN

- First successful deep-learning-based object detector
- Clunky design and training, see [2] if interested

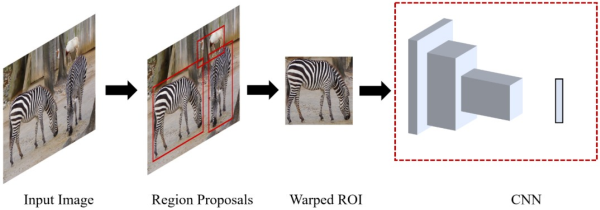


Image adapted from [1]

Approach still inefficient

- Got to classify many region proposals

More efficient extension

- Process complete (high-resolution) image once
- Project region proposals onto suitable conv layer output
- Classify each resulting feature map region

Approach was introduced by Fast R-CNN [3]

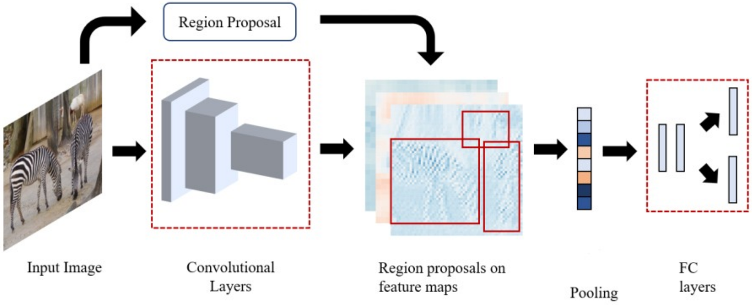


Image adapted from [1]

Deep Object Detection
Two-Stage Detectors

Region proposal step remains a bottleneck

- Generic algorithms (e.g. selective search [4])
- Not integrated into network
- Usually quite slow

Image adapted from [4]

There are improvements that address this

- Faster R-CNN is an example

R-CNN variants are two-stage detectors

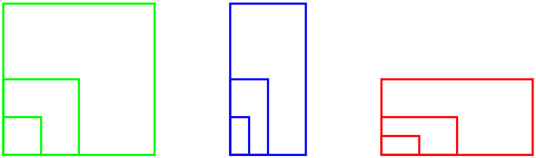
- First generate r region proposals
- Then analyze these proposals

Modern detectors are one-stage detectors

- Combine both stages, inference time independent of r

Most one-stage detectors utilize anchor boxes

- Fixed set of reference bounding boxes
- At different scales and aspect ratios



YOLO is a popular anchor-based detector familiy

- High detection performance
- Runs in real-time

Many versions exist

- We will cover v2, and v3
- Overall approach (detector head) identical

Simplified overall approach

- Divide input image into coarse $S \times S$ grid
- For each grid cell and anchor box, predict class labels (if any)
- Perform non-maximum suppression

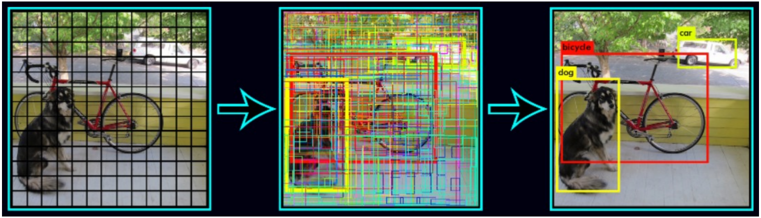


Image from pjreddie.com

More precisely, given an anchor box we seek to predict

- The IoU overlap p with the most relevant object
- The offset and scale of the box (x, y, w, h)
- C class scores

To constrain the number of detections

- Divide the image into grid (e.g. 7×7)
- Align each anchor box with the center of each grid cell
- Predict the above for every box and every cell

Assuming k anchor boxes and a 7×7 grid

- Output is $k \cdot (5 + C) \times 7 \times 7$
- Can be implemented using conv and pooling layers

Backbone pools down to desired grid size

- For instance $224 \mapsto 112 \mapsto 56 \mapsto 28 \mapsto 14 \mapsto 7$

Detector head consists of conv layers only

- Last one has $k \cdot (5 + C)$ channels
- YOLO v2 uses 4 layers in total (implementation detail)

Network has no linear layers

- Fully convolutional network (FCN) architecture
- Supports varying input (image) resolution

Pretrain backbone, then fine-tune for detection

For every training image, grid cell, and bounding box label

- Find the anchor box with the highest IoU
- Compute and assign $(p, x, y, h, w, o_1, \ldots, o_C)$
- Assign $(0, \ldots)$ to all unassigned anchor boxes

Weighted sum of three L2 losses per anchor box

- Confidence loss (loss on p)
- Classification loss (loss on class scores) *
- Localization loss (loss on x, y, w, h) *

(*) Only if IoU with ground-truth box is sufficient

Deep Object Detection
YOLO v2 – Inference

Network always predicts $k \cdot S \cdot S$ detections

- But most have $p \approx 0$
- Perform per-class non-maximum suppression

Image adapted from Youtube

Issues with

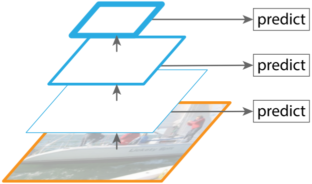
- Small objects
- Predicting tight bounding boxes

Because backbone feature tensor has low resolution of $S \times S$

- Late in network so semantically strong features
- But limited amount of spatial information due to pooling

Could address this by duplicating detector heads

- Attach to different conv layers



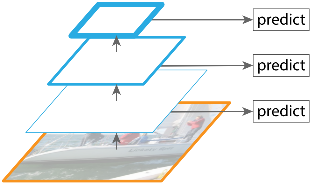
(c) Pyramidal feature hierarchy

Image from [6]

Deep Object Detection
Feature Pyramid Networks

Earlier layers have higher resolution but are semantically weaker

- We want both

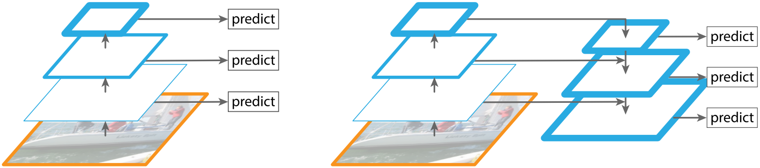


(c) Pyramidal feature hierarchy

Image from [6]

Feature pyramid networks (FPNs) achieve this by combining

- High-resolution but semantically weaker features
- Lower-resolution but semantically stronger features



(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Image from [6]

Accomplished by

- Upsampling lower-resolution features
- Combining features using sum operation

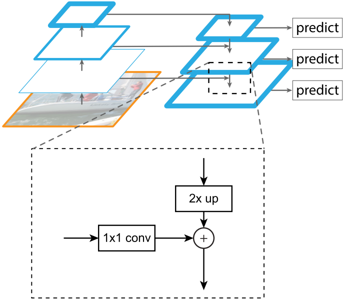


Image from [6]

Requires matching number of channels

- Fixed number of channels (e.g. 256)
- Implemented using pointwise convolutions

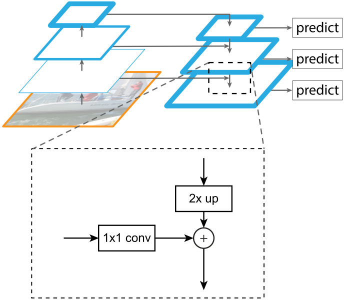


Image from [6]

FPNs aggregate and refine features

- Independent of base network architecture and task

In object detection context

- FPN (or alternatives) forms the neck of the detector
- Independent of backbone and head network architectures

YOLO v3 uses a FPN neck with 3 scales

- One YOLO head per scale
- In addition to other improvements, see [7] if interested

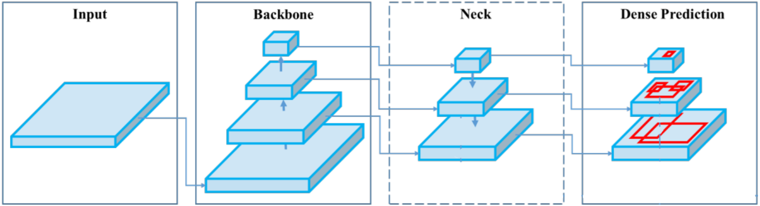


Image adapted from [7]

Several improved variants of YOLO exist, such as [8]

- Better backbone network
- Improved neck architecture
- Better training data augmentation
- Overall approach (detector head) largely identical

YOLO is a good candidate for object detection

- Modern variants perform well (COCO mAP around 55%)
- Have great speed vs. performance trade-off

Speed vs. performance on COCO [8]

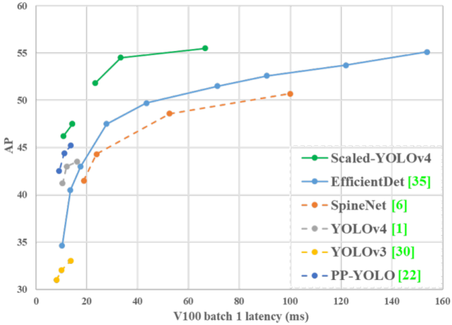


Image from [8]

Deep Object Detection

YOLO – Showcase

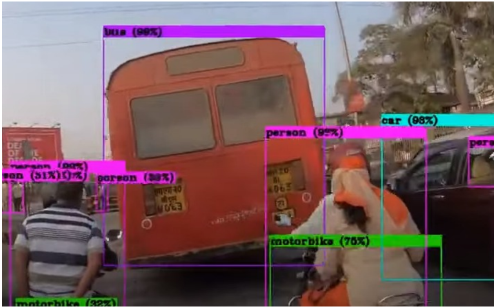


Image from youtube

Bibliography

[1] Zaidi et al. Deep Object Detection Survey. 2021

[2] Girshick et al. R-CNN. 2013

[3] Girshick. Fast R-CNN. 2015

[4] Uijlings et a. Selective Search. 2013

[5] Redmon & Farhadi. YOLO v2. 2016

[6] Lin et al. Feature Pyramid Networks. 2017

[7] Redmon & Farhadi. YOLO v3. 2017

[8] Wang et al. Scaled-YOLO v4. 2021