CAP 4453
Robot Vision
Dr. Gonzalo Vaca-Castaño
gonzalo.vacacastano@ucf.edu
Administrative details

• Issues submitting homework
Credits

• Some slides comes directly from:
  • Kristen Grauman
  • A. Zisserman
  • Ross B. Girshick
Histogram of Oriented Gradients (HOG)
Robot Vision

13. Object detection I
Outline

• **Overview: What is Object detection?**
• Top methods for object detection
• Object detection with Sliding Window and Feature Extraction (HoG)
  • Sliding Window technique
  • HOG: Gradient based Features
  • Machine Learning
    • Support Vector Machine (SVM)
    • Non-Maxima Suppression (NMS)
• Implementation examples
• Deformable Part-based Model (DPM)
• Object detection using deep learning
What is object detection

Classification

Classification + Localization

Object Detection

Instance Segmentation

CAT

CAT

CAT, DOG, DUCK

CAT, DOG, DUCK

Single object

Multiple objects
Object detection

- Multiple outputs
  - Bounding box
  - Label
  - Score
Detection Competitions

Pascal VOC
COCO
ImageNet ILSVRC

VOC: 20 classes

COCO: 200 classes

http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html#introduction
Valid detection

- **Groundtruth:**
  - *Bounding box*
  - *Label*

- **Possible detection**
  - *Bounding box*
  - *Label*
  - *score*

Different criteria to declare detections:

- **Pascal criteria**
  \[ \text{score}_{iou} > 0.5 \]

- All of these:
  \[
  \begin{align*}
  \text{score}_{iou} & > 0.5 \\
  \text{score}_{iou} & > 0.55 \\
  \text{score}_{iou} & > 0.6 \\
  \text{score}_{iou} & > 0.65 \\
  \text{score}_{iou} & > 0.7 \\
  \text{score}_{iou} & > 0.75 \\
  \text{score}_{iou} & > 0.8 \\
  \text{score}_{iou} & > 0.9 \\
  \text{score}_{iou} & > 0.95
  \end{align*}
  \]

**score**\(_{iou}\) = \(\frac{\text{Intersected Area}}{\text{Union BB area}}\)
Terms

Recall
Precision
mAP
IoU

Possible detection
Bounding box
Label
score
Terms
Recall
Precision
mAP
IoU

Possible detection
Bounding box
Label
score

Average precision (AP): Area under curve
Terms

Recall
Precision
mAP
IoU

mAP is simply all the AP values averaged over different classes/categories

Box Average Precision (AP@[0.5:0.95]): sums IOUs between 0.5 and 0.95 and divides the sum by the number of the IOU values

Possible detection
Bounding box
Label
score

Average precision (AP): Area under curve
Outline

• Overview: What is Object detection?
• **Top methods for object detection**
  • Object detection with Sliding Window and Feature Extraction (HoG)
    • Sliding Window technique
    • HOG: Gradient based Features
    • Machine Learning
      • Support Vector Machine (SVM)
      • Non-Maxima Suppression (NMS)
• Implementation examples
• Deformable Part-based Model (DPM)
Popular algorithms for object detection

• Pre-DeepLearning
  • HOG + SVM (Dalal, Triggs)
  • Deformable Part-based Model (DPM)

• Deep learning
  • Fast R-CNN
  • Faster R-CNN
  • Region-based Convolutional Neural Networks (R-CNN)
  • Region-based Fully Convolutional Network
  • Single Shot Detector (SSD)
  • YOLO (You Only Look Once)
PASCAL VOC 2005-2012

**20 object classes**

- Classification: person, motorcycle
- Detection
- Person
- Motorcycle

**22,591 images**

**Action: riding bicycle**

Object detection progress
PASCAL VOC

mean Average Precision (mAP)

Before CNNs

Using CNNs

Source: R. Girshick
Large Scale Visual Recognition Challenge (ILSVRC) 2010-2014

- **20 object classes**
  - 200 object classes
  - 1000 object classes

- **22,591 images**
  - 517,840 images
  - 1,431,167 images

http://image-net.org/challenges/LSVRC/
ILSVRC detection in 2014 (Deep learning)

1.9x increase in object detection average precision in one year

~3% due to more data

~18% due to better methods

Microsoft COCO: Common Objects in Context

COCO - Common Objects in Context (cocodataset.org)
State of the art methods
State of the art methods

Do you still need the old methods?
Outline

• Overview: What is Object detection?
• Top methods for object detection
  • **Object detection with Sliding Window and Feature Extraction (HoG)**
    • Sliding Window technique
    • HOG: Gradient based Features
    • Machine Learning
      • Support Vector Machine (SVM)
      • Non-Maximum Suppression (NMS)
  • Implementation examples
• Deformable Part-based Model (DPM)
Histograms of Oriented Gradients for Human Detection
Navneet Dalal and Bill Triggs
INRIA Rhône-Alps, 655 avenue de l’Europe, Montbonnot 38334, France

Abstract
We study the question of feature sets for robust visual object recognition, adopting linear SVM based human detection as a test case. After reviewing existing edge and gradient based descriptors, we show experimentally that grids of Histograms of Oriented Gradient (HOG) descriptors significantly outperform existing feature sets for human detection. We study the influence of each stage of the computation on performance, concluding that fine-scale gradients, fine orientation binning, relatively coarse spatial binning, and high-quality local contrast normalization in overlapping descriptor blocks are all important for good results. The new approach gives near-perfect separation on the original MIT pedestrian database, so we introduce a more challenging dataset containing over 1800 annotated human images with a large range of pose variations and backgrounds.

2 Previous Work
We briefly discuss previous work on human detection in §2, give an overview of our method §3, describe our data sets in §4 and give a detailed description and experimental evaluation of each stage of the process in §5–6. The main conclusions are summarized in §7.

1 Introduction

• CVPR 2005
Sliding Window Technique

• Classification problem:
  • Score for a category
Sliding Window Technique

• Score every subwindow
  • extract features from the image window
  • classifier decides based on the given features.

• It is a brute-force approach
Window-based detection: strengths

Pros

• Sliding window detection and global appearance descriptors:
  • Simple detection protocol to implement
  • Good feature choices critical
  • Past successes for certain classes

Cons

• High computational complexity
  • For example: 250,000 locations x 30 orientations x 4 scales = 30,000,000 evaluations!
  • If training binary detectors independently, means cost increases linearly with number of classes

• With so many windows, false positive rate better be low
Cons (continued)

- Not all objects are “box” shaped
Limitations (continued)

- If considering windows in isolation, context is lost

![Sliding window vs Detector’s view](image)

Figure credit: Derek Hoiem

Slide: Kristen Grauman
Outline

• Overview: What is Object detection?
• Top methods for object detection
• **Object detection with Sliding Window and Feature Extraction (HoG)**
  • Sliding Window technique
  • **HOG: Gradient based Features**
  • Machine Learning
    • Support Vector Machine (SVM)
    • Non-Maximum Suppression (NMS)
• Implementation examples
• **Deformable Part-based Model (DPM)**
Let’s examine possible feature vectors

• Pixel based (as a vector)
  • Sensitive to small shifts

• Color based
  • Color-based representations are sensitive to color (illumination)
Gradient-based representations

• summarize the local distribution of gradients with histograms
• invariance to small shifts and rotations
• offers more spatial information compared to a single global histogram
• Includes contrast normalization
  • reduce the impact of variable illumination (color)
Histograms of Oriented Gradients (HOG)

• Step 1: Extract a square window (called “block”) of some size around the pixel location of interest.
• Step 2: Divide block into a square grid of sub-blocks (called “cells”) (2x2 grid in our example, resulting in four cells).
• Step 3: Compute orientation histogram of each cell.
• Step 4: Concatenate the four histograms.
• Step 5: normalize v using one of the three options:
  • Option 1 (L2): Divide v by its Euclidean norm.
  • Option 2 (L1): Divide v by its L1 norm (the L1 norm is the sum of all absolute values of v).
  • Option 3 (L2-Hys):
    • Divide v by its Euclidean norm.
    • In the resulting vector, clip any value over 0.2
    • Then, renormalize the resulting vector by dividing again by its Euclidean norm
Histogram of Oriented Gradients (HOG)

• Angles range from 0 to 180 or from 0 to 360 degrees?
  • In the Dalal & Triggs paper, a range of 0 to 180 degrees is used

• Number of orientation bins.
  • Usually 9 bins, each bin covering 20 degrees.

• Cell size.
  • Cells of size 8x8 pixels are often used. (64 → 9)

• Block size.
  • Blocks of size 2x2 cells (16x16 pixels) are often used.

• HOG feature has 36 dimensions.
  • 4 cells * 9 orientation bins.
Calculate HOG Descriptor vector

• The 16×16 window then moves by 8 pixels and a normalized 36×1 vector is calculated over this window and the process is repeated for the image.

• To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector.

• Example: an input picture of size 64×64
  • The 16×16 block has 7 positions horizontally and 7 position vertically.
  • In one 16×16 block we have 4 histograms which after normalization concatenate to form a 36×1 vector.
  • This block moves 7 positions horizontally and vertically totalling it to 7×7 = 49 positions.
  • we concatenate them all into one giant vector we obtain a 36×49 = 1764 dimensional vector.
Outline

• Overview: What is Object detection?
• Top methods for object detection
  • Object detection with Sliding Window and Feature Extraction (HoG)
    • Sliding Window technique
    • HOG: Gradient based Features
    • Machine Learning
      • Support Vector Machine (SVM)
      • Non-Maximum Suppression (NMS)
  • Implementation examples
• Deformable Part-based Model (DPM)
Support vector machines
Support vector machines

• When the data is linearly separable, there may be more than one separator (hyperplane)

Which separator is best?
Linear classifiers

A linear classifier has the form

\[ f(x) = w^\top x + b \]

- in 2D the discriminant is a line
- \( w \) is the normal to the line, and \( b \) the bias
- \( w \) is known as the weight vector
What is the best $w$?

- **maximum margin** solution: most stable under perturbations of the inputs
Support vector machines

- Find hyperplane that maximizes the margin between the positive and negative examples

\[
\begin{align*}
    &\text{x positive (y = 1):} & \text{x} \cdot \text{w} + b & \geq 1 \\
    &\text{x negative (y = -1):} & \text{x} \cdot \text{w} + b & \leq -1 \\
\end{align*}
\]

For support vectors, \( \text{x} \cdot \text{w} + b = \pm 1 \)

Distance between point and hyperplane:
\[
\left| \text{x} \cdot \text{w} + b \right| / \| \text{w} \|
\]

Therefore, the margin is \( 2 / \| \text{w} \| \)

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|w\|$

2. Correctly classify all training data:
   - $x_i$ positive ($y_i = 1$): $x_i \cdot w + b \geq 1$
   - $x_i$ negative ($y_i = -1$): $x_i \cdot w + b \leq -1$

- **Quadratic optimization problem:**

  \[
  \min_{w, b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1
  \]

Linear separability again: What is the best $w$?

- the points can be linearly separated but there is a very narrow margin

- but possibly the large margin solution is better, even though one constraint is violated

In general there is a trade off between the margin and the number of mistakes on the training data
Introduce “slack” variables

\[ \xi_i \geq 0 \]

- for \( 0 < \xi \leq \frac{1}{||w||} \) point is between margin and correct side of hyperplane. This is a **margin violation**
- for \( \xi > \frac{1}{||w||} \) point is **misclassified**

\[ w^T x + b = 1 \]
\[ w^T x + b = 0 \]
\[ w^T x + b = -1 \]
SVM training in general

- Separable data:
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i (w \cdot x_i + b) \geq 1
  \]
  
  Maximize margin

- Non-separable data:
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w \cdot x_i + b))
  \]
  
  Maximize margin
  Minimize classification mistakes

Classify training data correctly
SVM training in general

\[
\min_{w, b} \, \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w \cdot x_i + b))
\]

• Demo: [http://cs.stanford.edu/people/karpathy/svmjs/demo](http://cs.stanford.edu/people/karpathy/svmjs/demo)
Linear separability

- Linearly separable
- Not linearly separable
Nonlinear SVMs

• General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable
Nonlinear SVMs

• Linearly separable dataset in 1D:

• Non-separable dataset in 1D:

• We can map the data to a higher-dimensional space:

Slide credit: Andrew Moore
The kernel trick

- Linear SVM decision function:

\[
\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b
\]

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998
The kernel trick

• Linear SVM decision function:

\[ \mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \]

• Kernel SVM decision function:

\[ \sum_i \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}) + b = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \]

• This gives a nonlinear decision boundary in the original feature space

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998
The kernel trick

• Instead of explicitly computing the lifting transformation $\phi(x)$, define a kernel function $K$ such that
  
  $$K(x, y) = \phi(x) \cdot \phi(y)$$

• (to be valid, the kernel function must satisfy Mercer’s condition)
Polynomial kernel:

\[ K(x, y) = (c + x \cdot y)^d \]
Gaussian kernel

• Also known as the radial basis function (RBF) kernel:

\[ K(x, y) = \exp\left( -\frac{1}{\sigma^2} \|x - y\|^2 \right) \]
Gaussian kernel

- Demo: [http://cs.stanford.edu/people/karpathy/svmjs/demo](http://cs.stanford.edu/people/karpathy/svmjs/demo)
SVMs: Pros and cons

• Pros
  • Kernel-based framework is very powerful, flexible
  • Training is convex optimization, globally optimal solution can be found
  • Amenable to theoretical analysis
  • SVMs work very well in practice, even with very small training sample sizes

• Cons
  • No “direct” multi-class SVM, must combine two-class SVMs (e.g., with one-vs-others)
  • Computation, memory (esp. for nonlinear SVMs)
Person detection with HoG’s & linear SVM’s (so far)

- Histogram of oriented gradients (HoG): Map each grid cell in the input window to a histogram counting the gradients per orientation.

- Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

Dalal & Triggs, CVPR 2005
The Dalal & Triggs detector

Image pyramid
The Dalal & Triggs detector

1. Compute HOG of the whole image at multiple resolutions!
The Dalal & Triggs detector

1. Compute HOG of the whole image at multiple resolutions!

2. Score every window of the feature pyramid

\[ \text{score}(I, p) = w \cdot \phi(I, p) \]

FROM TRAINING
The Dalal & Triggs detector

1. Compute HOG of the whole image at multiple resolutions!

2. Score every window of the feature pyramid

\[ \text{score}(I, p) = w \cdot \phi(I, p) \]

FROM TRAINING

3. Apply non-maximal suppression (NMS)
Outline

• Overview: What is Object detection?
• Top methods for object detection
• Object detection with Sliding Window and Feature Extraction (HoG)
  • Sliding Window technique
  • HOG: Gradient based Features
  • Machine Learning
    • Support Vector Machine (SVM)
  • Non-Maximum Suppression (NMS)
• Implementation examples
• Deformable Part-based Model (DPM)
Non-Maximum Suppression

Before non-max suppression

After non-max suppression

Non-Max Suppression
Non-Maximum Suppression

Algorithm 1: Non-Max Suppression

1: procedure NMS(B, c)
2: \[ B_{nms} \leftarrow \emptyset \] Initialize empty set
3: for \( b_i \in B \) do \(
4: \quad \text{discard} \leftarrow False \) Iterate over all the boxes
5: \quad \text{Start another loop to compare with } b_i \)
6: \quad \text{if same}(b_i, b_j) > \lambda_{nms} \text{ then}
7: \quad \quad \text{if score}(c, b_j) > \text{score}(c, b_i) \text{ then}
8: \quad \quad \quad \text{discard} \leftarrow True \)
9: \quad \quad \text{if not discard} then
10: \quad \quad \quad \quad B_{nms} \leftarrow B_{nms} \cup b_i \)
11: return \( B_{nms} \)

Do the same procedure for remaining boxes and return the final list

CAP4453
Outline

• Overview: What is Object detection?
• Top methods for object detection
• **Object detection with Sliding Window and Feature Extraction (HoG)**
  • Sliding Window technique
  • HOG: Gradient based Features
  • Machine Learning
    • Support Vector Machine (SVM)
    • Non-Maximum Suppression (NMS)
• **Implementation examples**
• Deformable Part-based Model (DPM)
Implementation example (car detector)
Get the data. UIUC Car Database

• 550 positive images
• 500 negatives
Implementation example (car detector)

• Extract features

```python
print("Calculating the descriptors for the positive samples and saving them")
for im_path in glob.glob(os.path.join(pos_im_path, "*")):
    im = imread(im_path)
    #print(im.shape)
    if des_type == "HOG":
        fd = hog(im, orientations, pixels_per_cell, cells_per_block, visualize=visualize, block_norm='L2-Hys')
    fd_name = os.path.split(im_path)[1].split(".")[0] + ".feat"
    fd_path = os.path.join(pos_feat_ph, fd_name)
    joblib.dump(fd, fd_path)

print("Positive features saved in {}\n.format(pos_feat_ph))

print("Calculating the descriptors for the negative samples and saving them")
for im_path in glob.glob(os.path.join(neg_im_path, "*")):
    im = imread(im_path)
    if des_type == "HOG":
        fd = hog(im, orientations, pixels_per_cell, cells_per_block, visualize=visualize, block_norm='L2-Hys')
    fd_name = os.path.split(im_path)[1].split(".")[0] + ".feat"
    fd_path = os.path.join(neg_feat_ph, fd_name)
    joblib.dump(fd, fd_path)

print("Negative features saved in {}\n.format(neg_feat_ph))
```

VladKha/object_detector: Object detector from HOG + Linear SVM framework (github.com)
Implementation example (car detector)

- Train SVM with HOG features
Implementation example (car detector)

Test

- Load image
- Loop over different pyramid images
  - loop the window position
    - Compute HOG for each window
    - Compute score

```python
from skimage.transform import pyramid_gaussian
from skimage.io import imread
from skimage.feature import hog
import joblib
import cv2
import argparse as ap
from nms import nms
from config import *
import numpy as np

# Downsacle the image and iterate
for im_scaled in pyramid_gaussian(im, downscale=downscale):
    print('im_scaled.shape')
    # This list contains detections at the current scale
    cd = []
    # If the width or height of the scaled image is less than
    # the width or height of the window, then end the iterations.
    if im_scaled.shape[0] < min_wdw_sz[1] or im_scaled.shape[1] < min_wdw_sz[0]:
        break
    for (x, y, im_window) in sliding_window(im_scaled, min_wdw_sz, step_size):
        print(x, y)
        # Calculate the HOG features
        if visualize:
            fd, imgVis = hog(im_window, orientations, pixels_per_cell, cells_per_block, visualize=True, block_norm='L2-Hys')
            cv2.imshow('HOG input', imgVis)
            cv2.waitKey(30)
        else:
            fd = hog(im_window, orientations, pixels_per_cell, cells_per_block, visualize=False, block_norm='L2-Hys')

        fd = fd[np.newaxis, :]

        print(fd.shape)
        pred = clf.predict(fd)

        if pred == 1:
            print("Detection: Location = (" + str(x) + " , " + str(y) + ")")
            # Compute score
            print("Score : ", clf.decision_function(fd))
            detections.append((x, y, clf.decision_function(fd),
                               int(min_wdw_sz[0])*(downscale**scale),
                               int(min_wdw_sz[1])*(downscale**scale)))
            cd.append(detections[-1])
```
Implementation example (car detector)

Test

- Load image
- Loop over different pyramid images
  - loop the window position
    - Compute HOG for each window
- Compute score
- Perform NMS
Testing (different pyramid levels)
NMS

Before NMS

After NMS
Outline

• Overview: What is Object detection?
• Top methods for object detection
• Object detection with Sliding Window and Feature Extraction (HoG)
  • Sliding Window technique
  • HOG: Gradient based Features
  • Machine Learning
    • Support Vector Machine (SVM)
    • Non-Maxima Suppression (NMS)
• Implementation examples
• Deformable Part-based Model (DPM)
• Object detection using deep learning
Object detection

- R-CNN
- Fast R-CNN
- Faster R-CNN
- SSD
- YOLO — You Only Look Once
  - Multiple versions
Object detection
R-CNN (2013)

Selective Search:
1. Generate initial sub-segmentation, we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals

Problems with R-CNN

• It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
• It cannot be implemented real time as it takes around 47 seconds for each test image.
• The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Object detection
FAST R-CNN (2014)

- We feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map
- We identify the region of proposals and warp them into squares
- Using a RoI pooling layer we reshape them into a fixed size
- they can be fed into a fully connected layer

Object detection

FAST R-CNN (2014)

Object detection
FASTER CNN (2015)

• At each location, the original paper uses:
  • 3 kinds of anchor boxes for scale
    • 128×128, 256×256 and 512×512.
  • it uses three aspect ratios
    • 1:1, 2:1 and 1:2.

• So, In total at each location, we have 9 boxes on which RPN predicts the probability of it being background or foreground.

Object detection
FASTER CNN (2015)

R-CNN Test-Time Speed

<table>
<thead>
<tr>
<th>Model</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-CNN</td>
<td>49</td>
</tr>
<tr>
<td>SPP-Net</td>
<td>4.3</td>
</tr>
<tr>
<td>Fast R-CNN</td>
<td>2.3</td>
</tr>
<tr>
<td>Faster R-CNN</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Object detection
YOLO (2015)

• an image and split it into an SxS grid,
• within each of the grid we take B bounding boxes.
• For each of the bounding box,
  • the network outputs a class probability and offset values for the bounding box.
  • The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts $B$ bounding boxes, confidence for those boxes, and $C$ class probabilities. These predictions are encoded as an $S \times S \times (B \times 5 + C)$ tensor.

Object detection

YOLO (2015)

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts $B$ bounding boxes, confidence for those boxes, and $C$ class probabilities. These predictions are encoded as an $S \times S \times (B \times 5 + C)$ tensor.

Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 $\times$ 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ($224 \times 224$ input image) and then double the resolution for detection.

Object detection

YOLO (2015)

• The limitation of YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints of the algorithm.

Object detection
SSD (2016)

- Multi-scale feature maps for detection (handle scale)
- uses anchor boxes at various aspect ratio similar to Faster-RCNN
- learns the off-set rather than learning the box

Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300 × 300 input size significantly outperforms its 448 × 448 YOLO counterpart in accuracy on VOC2007 test while also improving the speed.

https://arxiv.org/pdf/1512.02325
SSD Loss

\[ L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \]

\[ L_{loc}(x, l, g) = \sum_{i \in Pos} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k \text{smooth}_L}(l_{i}^{m} - \hat{g}_{j}^{m}) \]

\[ \hat{g}_{j}^{cx} = (g_{j}^{cx} - d_{i}^{cx})/d_{i}^{w} \]
\[ \hat{g}_{j}^{cy} = (g_{j}^{cy} - d_{i}^{cy})/d_{i}^{h} \]
\[ \hat{g}_{j}^{cw} = \log \left( \frac{g_{j}^{cw}}{d_{i}^{w}} \right) \]
\[ \hat{g}_{j}^{ch} = \log \left( \frac{g_{j}^{ch}}{d_{i}^{h}} \right) \]

\[ L_{conf}(x, c) = -\sum_{i \in Pos} x_{ij}^{p} \log(\hat{c}_{i}^{p}) - \sum_{i \in Neg} \log(\hat{c}_{i}^{0}) \]
where \[ \hat{c}_{i}^{p} = \frac{\exp(c_{i}^{p})}{\sum_{p} \exp(c_{i}^{p})} \]
SSD training details

• Hard Negative Mining
• Data Augmentation
  • entire original input image
  • Sample a patch so that the overlap with objects is 0.1, 0.3, 0.5, 0.7 or 0.9.
  • Randomly sample a patch
• Atrous Convolution
Object detection

COMPARISON OF THESE CLASSICAL METHODS

![Accuracy vs Speed Graph]

- Faster RCNN
- SSD
- YOLO

Accuracy

- YoloV2
- SSD
- Faster-RCNN

Small Objects | Medium Objects | Large Objects

0 | 20 | 40

60
Modern object detectors (single stage)

Model Backbone
The backbone is a pre-trained network used to extract rich feature representation for images. This helps reducing the spatial resolution of the image and increasing its feature (channel) resolution.

Model Neck
The model neck is used to extract feature pyramids. This helps the model to generalize well to objects on different sizes and scales.

Model Head
The model head is used to perform the final stage operations. It applies anchor boxes on feature maps and render the final output: classes, objectness scores and bounding boxes.
Yolo V5
DenseNet
Backbone: Cross Stage Partial Network

it uses residual and dense blocks: overcome the vanishing gradient problem.

CSP network preserves the advantage of DenseNet's feature reuse characteristics and helps reducing the excessive amount of redundant gradient information by truncating the gradient flow.
YOLOv5 employs CSPNet strategy to partition the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy.

Applying this strategy comes with big advantages to YOLOv5, since it helps reducing the number of parameters and helps reducing an important amount of computation (less FLOPS) which lead to increasing the inference speed that is crucial parameter in real-time object detection models.
Spatial Pyramid Pooling (SPP)
Head of the network

YOLOv5 uses the same head as YOLOv3 and YOLOv4, three convolution layers that predicts the location of the bounding boxes \((x,y,\text{height},\text{width})\), the scores and the objects classes. The equation to compute the target coordinates for the bounding boxes have changed from previous versions, the difference is shown in the figure bellow.

\[
\begin{align*}
    b_x &= \sigma(t_x) + c_x & \text{(a)} \\
    b_y &= \sigma(t_y) + c_y \\
    b_w &= p_w \cdot e^{t_w} \\
    b_h &= p_h \cdot e^{t_h} \\
    b_x &= (2 \cdot \sigma(t_x) - 0.5) + c_x \\
    b_y &= (2 \cdot \sigma(t_y) - 0.5) + c_y \\
    b_w &= p_w \cdot (2 \cdot \sigma(t_w))^2 \\
    b_h &= p_h \cdot (2 \cdot \sigma(t_h))^2
\end{align*}
\]
**Activation Function**

**SiLU** stands for Sigmoid Linear Unit and it is also called the **swish** activation function. It has been used with the convolution operations in the **hidden layers**.

**Sigmoid** activation function has been used with the convolution operations in the **output layer**.
Loss function

- YOLOv5 returns three outputs:
  - the **classes** of the detected objects,
  - their **bounding boxes**
  - **objectness scores**.
- it uses **BCE** (Binary Cross Entropy) to compute the **classes loss** and the **objectness loss**.
- **CloU** (Complete Intersection over Union) **loss** to compute the **location loss**.

\[
Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc}
\]
Yolo V5

• **Real-time performance:** excels at real-time object detection, achieving high frame rates on even modest hardware.

• **High accuracy:** YOLOv5 also delivers impressive accuracy. Different versions like “s,” “m,” “l,” and “x” offer a trade-off between speed and accuracy.

• **Flexible and customizable:** The model is open-source and readily customizable for specific tasks and datasets.

• **Easy to use:** YOLOv5 is built with user-friendliness in mind, featuring clear documentation and readily available pre-trained models.
Yolo v8

• **State-of-the-Art**: Delivers cutting-edge accuracy and speed, competing with other top models like Efficient and DETR.

• **Multi-Task Capable**: Handles diverse tasks like object detection, instance segmentation, and image classification within one framework.

• **Anchor-Free Detection**: Eliminates the need for pre-defined anchor boxes, simplifying the architecture and improving accuracy.

• **Streamlined Design**: Easy to use and customize, with readily available pre-trained models and a vibrant community.

• **Open-Source and Scalable**: Freely available under the GNU General Public License and adaptable to various platforms, from edge devices to cloud AI.
Changes compared to YOLOv5:

• Replace the C3 module with the C2f module
• Replace the first 6x6 Conv with 3x3 Conv in the Backbone
• Delete two Convs (No.10 and No.14 in the YOLOv5 config)
• Replace the first 1x1 Conv with 3x3 Conv in the Bottleneck
• Use decoupled head and delete the objectness branch
# Yolov5 vs Yolov8

<table>
<thead>
<tr>
<th>Feature</th>
<th>YOLOv5</th>
<th>YOLOv8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Anchor-based</td>
<td>Anchor-free</td>
</tr>
<tr>
<td>Neck Module</td>
<td>Convolutional connection layers present</td>
<td>Convolutional connection layers removed</td>
</tr>
<tr>
<td>Head Module</td>
<td>Single head for class and bounding box predictions</td>
<td>Split head for class and bounding box predictions</td>
</tr>
<tr>
<td>Objectness Prediction</td>
<td>Outputs objectness score</td>
<td>No objectness output, directly predicts center point and size of bounding boxes</td>
</tr>
<tr>
<td>Loss Function</td>
<td>Focal Loss + IOU Loss</td>
<td>TAL (Tangent-Aided Loss) + DFL (Dynamic Focal Loss)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>YOLOv5</th>
<th>YOLOv8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (mAP50)</td>
<td>Varies depending on model size and dataset</td>
<td>Generally higher than YOLOv5 for similar model sizes</td>
</tr>
<tr>
<td>Speed (FPS)</td>
<td>Varies depending on model size and hardware</td>
<td>Generally faster than YOLOv5 for similar model sizes</td>
</tr>
<tr>
<td>Model Size</td>
<td>Generally larger than YOLOv8</td>
<td>More compact, requires fewer parameters</td>
</tr>
<tr>
<td>User Interface/Experience (UI/UX)</td>
<td>Less user-friendly</td>
<td>Significantly improved UI/UX, easier to use and customize</td>
</tr>
<tr>
<td>Training Ease</td>
<td>More complex training regime</td>
<td>Simpler training process, often converges faster</td>
</tr>
<tr>
<td>Community Support</td>
<td>Large and active community</td>
<td>Growing community, but not as large as YOLOv5 yet</td>
</tr>
</tbody>
</table>
Yolo v8?

**Speed comparisons:**
- **Benchmark results:** Some benchmarks show YOLOv8 as slightly faster for certain model sizes, particularly on image inference. However, for video and live camera applications, YOLOv5 often holds the edge in speed.
- **Individual results may vary:** Hardware, dataset size, and specific model configurations can significantly affect performance. What’s faster for one person might not be faster for another.

**Beyond speed:**
- **Accuracy:** YOLOv8 generally demonstrates slightly higher accuracy on object detection tasks compared to YOLOv5.
- **Model size:** YOLOv8 models tend to be smaller and have fewer parameters, potentially leading to faster training times and lower deployment memory requirements.
- **Ease of use:** Both frameworks are user-friendly with extensive documentation and community support.
Questions?