Object detection

Course:
INFO-6152 Deep Learning with Tensorflow & Keras 2

Developed by:
Mohammad Noorchenarboo

October 20, 2025

Current Section

Object Detection vs Image Classification

Real-World Motivation: Autonomous vehicles must not only recognize what is in a scene (cars, pedestrians, traffic lights), but also **where** each object is located. This requirement leads to the distinction between **image classification** and **object detection**.

Concept and Key Formulas

Image Classification: Assigns a single label to an entire image. $\hat{y} = \arg\max_c P(c|I)$, where I is the image and c is the class label.

Object Detection: Identifies both the class and the position of each object in the image. It outputs:

$$\{(c_i, b_i)\}_{i=1}^{N}, \quad b_i = (x_i, y_i, w_i, h_i)$$

where c_i is the class label and b_i is the bounding box defined by its top-left coordinates (x_i, y_i), width w_i, and height h_i.

Loss functions combine both classification and localization errors:

$$L = L_{\text{cls}} + \lambda L_{\text{bbox}}$$

Object Detection vs Image Classification

Numerical Example

- Suppose an image contains one cat located at coordinates $(x = 50, y = 60, w = 120, h = 100)$.
- **Image Classification:** Model outputs "cat" as the label.
- **Object Detection:** Model outputs:

$$(c_1 = \text{cat},\ b_1 = (50, 60, 120, 100))$$

- If a dog also appears in the same image, detection model outputs:

$$\{(\text{cat}, b_1), (\text{dog}, b_2)\}$$

where b_2 could be $(150, 80, 110, 90)$.

Get more info $\rightarrow$ **yolov8**

Summary: Object Detection vs Image Classification

Concept / Aspect	**Key Explanation**
Image Classification	Predicts one label for the entire image, no spatial localization.
Object Detection	Predicts both label and bounding box (x, y, w, h) for each object.
Output Type	Classification $\rightarrow$ class label; Detection $\rightarrow$ set of labels and boxes.
Typical Model Examples	Classification: ResNet, VGG; Detection: Faster R-CNN, SSD, YOLO.
Key Formula	$L = L_{\text{cls}} + \lambda L_{\text{bbox}}$ (combines class and box losses).
Real-World Use Case	Self-driving cars, security systems, medical image localization.

Current Section

R-CNN(Region-Based Convolutional Neural Network)

Real-World Motivation: Imagine a self-driving car that must detect pedestrians, vehicles, and traffic lights in one image. Traditional classifiers predict only one label per image, not multiple localized objects. R-CNN solved this by combining **region proposals** with **CNN-based feature extraction**.

Concept and Process Overview

Goal: Detect and classify multiple objects by analyzing candidate regions.
Steps:

- Generate about 2000 region proposals using **Selective Search**.
- For each region R_i, extract features using a **CNN** (e.g., AlexNet).
- Classify each region and adjust bounding boxes with regression.

Mathematical Summary:

$$f_i = \mathrm{CNN}(R_i), \quad \hat{y}_i = \arg\max_c P(c|f_i)$$

$$b_i' = b_i + \Delta b_i, \quad \Delta b_i = W_b f_i + b_b$$

The total loss combines classification and localization:

$$L = L_{\mathrm{cls}} + \lambda L_{\mathrm{bbox}}$$

R-CNN(Region-Based Convolutional Neural Network)

Simplified Example

- Selective Search generates $N = 2000$ regions.
- Each region is resized to 227×227 and passed through CNN.
- CNN outputs a 4096-dimensional feature vector for each region.
- Each vector is classified into object categories (e.g., car, person, dog).
- Bounding box regression fine-tunes location (x, y, w, h).

Common Mistakes

- Wrong: CNN runs once per image. Correct: R-CNN runs CNN **for every region proposal**, making it slow.
- Wrong: R-CNN is end-to-end. Correct: It has multiple training stages: CNN pretraining, classifier training, and bounding box regression.
- Wrong: R-CNN is real-time. Correct: Processing takes several seconds per image.

R-CNN(Region-Based Convolutional Neural Network)

Caveats

- High computation cost due to thousands of CNN evaluations.
- Requires large storage for saving extracted features.
- Not suitable for real-time applications.

Analogy

R-CNN acts like a careful inspector using a magnifying glass. Selective Search points to possible areas, and CNN checks each area one by one. This approach is thorough but slow.

R-CNN(Region-Based Convolutional Neural Network)

Evolution of R-CNN Models

R-CNN → Fast R-CNN → Faster R-CNN

- **R-CNN:** Runs CNN on each proposed region separately. Accurate but very slow (2000 forward passes per image).
- **Fast R-CNN:** Runs CNN once per image to create a shared feature map. Uses ROI (Region of Interest) Pooling to extract region features quickly. Trains classification and bounding box regression jointly.
- **Faster R-CNN:** Adds a Region Proposal Network (RPN) that learns to generate region proposals automatically. Eliminates Selective Search and makes detection nearly end-to-end.

Interactive example → link

R-CNN(Region-Based Convolutional Neural Network)

Key Differences at a Glance

- **Region Proposals:** R-CNN uses Selective Search, Fast R-CNN still uses Selective Search, Faster R-CNN replaces it with RPN.
- **Computation:** R-CNN computes CNN features for each region. Fast/Faster R-CNN compute CNN features once per image.
- **Training:** R-CNN trains in multiple stages. Fast and Faster R-CNN train nearly end-to-end.
- **Speed:** R-CNN $\approx$ 47 seconds per image. Fast R-CNN $\approx$ 2 seconds per image. Faster R-CNN $\approx$ 0.2 seconds per image (near real-time).

Get more info $\rightarrow$ **Original R-CNN Paper (Girshick et al., 2014)**

Get more info $\rightarrow$ **R-CNN Overview**

Summary: R-CNN

Aspect	Explanation
Main Idea	Combines region proposals with CNN feature extraction for object detection.
Region Proposal	Generated by Selective Search (about 2000 per image).
Feature Extraction	Each region is processed separately through CNN.
Classification	Uses a simple classifier on extracted CNN features.
Bounding Box Regression	Adjusts region coordinates for accurate localization.
Training Type	Multi-stage (CNN $\rightarrow$ Classifier $\rightarrow$ Bounding Box Regression).
Weaknesses	High computation time, heavy storage, not end-to-end.
Successors	Fast R-CNN (shared feature maps) and Faster R-CNN (Region Proposal Network).
Connection to YOLO	YOLO replaces region proposals with direct prediction over the full image, explained next.

Current Section

Bounding Box Filtering using NMS

Object detection models like YOLO predict multiple overlapping bounding boxes for the same object. The problem: how can we filter redundant boxes and keep only the most confident one?

Concept and Formula of Non-Maximum Suppression (NMS)

Definition: Non-Maximum Suppression (NMS) removes overlapping bounding boxes that predict the same object, keeping only the box with the highest confidence score.

Key Formulas:

$$\text{IoU} = \frac{A_{\text{intersection}}}{A_{\text{union}}} = \frac{\text{Intersection Area}}{\text{Union Area}}$$

$$\text{Keep box}_i \text{ if } \text{IoU}(\text{box}_i, \text{box}_j) \leq \text{IoU}_{\text{threshold}} \quad \forall j \in \text{Higher Confidence Boxes}$$

Algorithm Steps:

- Step 1: Filter boxes by confidence threshold (remove all boxes with confidence < 0.5).
- Step 2: Sort boxes by descending confidence.
- Step 3: Keep highest confidence box, remove all with IoU greater than threshold.
- Step 4: Repeat until no boxes remain.

Bounding Box Filtering using NMS

Numerical Example

Consider four detected boxes for the same object:

- Box A: confidence = 0.9, coordinates (10, 10, 100, 100)
- Box B: confidence = 0.8, coordinates (20, 20, 100, 100)
- Box C: confidence = 0.6, coordinates (200, 200, 100, 100)
- Box D: confidence = 0.4, coordinates (250, 250, 100, 100)

Step 1: Confidence Filtering Box D (0.4) is removed because it is below the threshold (0.5).

Step 2: Compute IoU between Box A and Box B

$$A_{\text{intersection}} = 80 \times 80 = 6400$$

$$A_{\text{union}} = 10000 + 10000 - 6400 = 13600$$

$$\text{IoU}(A, B) = \frac{6400}{13600} = 0.47$$

If $\text{IoU}_{\text{threshold}} = 0.45$, Box B will be suppressed, and Boxes A and C will remain.

Interactive example → link

Bounding Box Filtering using NMS

TensorFlow Example: Apply NMS with Confidence Filtering

```python
import tensorflow as tf

# Define bounding boxes [y1, x1, y2, x2]
boxes = tf.constant([
    [0.1, 0.1, 0.5, 0.5],      # Box A (0.9)
    [0.15, 0.15, 0.55, 0.55], # Box B (0.8)
    [0.6, 0.6, 0.9, 0.9],      # Box C (0.6)
    [0.7, 0.7, 0.95, 0.95]     # Box D (0.4) -> should be filtered out
], dtype=tf.float32)

# Confidence scores for each box
scores = tf.constant([0.9, 0.8, 0.6, 0.4])

# Step 1: Filter boxes with confidence < 0.5
conf_threshold = 0.5
mask = scores >= conf_threshold
filtered_boxes = tf.boolean_mask(boxes, mask)
filtered_scores = tf.boolean_mask(scores, mask)

# Step 2: Apply Non-Maximum Suppression
selected_indices = tf.image.non_max_suppression(
    filtered_boxes, filtered_scores,
    max_output_size=4, iou_threshold=0.45
)

# Step 3: Gather selected boxes
selected_boxes = tf.gather(filtered_boxes, selected_indices)

print("Kept box indices:", selected_indices.numpy())
print("Kept boxes:\n", selected_boxes.numpy())
# Expected: Keeps Box A and Box C, removes Box B and Box D
```

Summary: Bounding Box Filtering using NMS

Aspect	Key Explanation
Purpose of NMS	Removes redundant overlapping detections, keeping only the most confident box
IoU (Intersection over Union)	Measures overlap between two boxes, defined as $\frac{\text{Intersection}}{\text{Union}}$
IoU Threshold	Controls how much overlap triggers suppression (typical range: 0.3–0.5)
Confidence Threshold	Filters low-confidence boxes before running NMS (e.g., 0.5)
Example with 4 Boxes	Box D (0.4) removed before NMS, Box B (IoU 0.47) suppressed, Boxes A and C kept
Sorting Step	Boxes are sorted by confidence score before suppression
TensorFlow Function	`tf.image.non_max_suppression(boxes, scores, max_output_size, iou_threshold)`
Typical Result in YOLO	Only one box per object class remains after NMS

Get more info → TensorFlow NMS API

Get more info → YOLO Object Detection Reference

Current Section

YOLO (You Only Look Once) Model

Real-World Motivation: Autonomous vehicles and real-time surveillance require instant, accurate object detection. Two-stage detectors like R-CNN are accurate but too slow for real-time systems. YOLO addresses this by predicting all bounding boxes and class probabilities in a single network pass.

YOLO Core Concept (Compact)

Key Idea: YOLO reformulates detection as a single regression problem, directly predicting bounding box coordinates and class probabilities.
Grid-Based Prediction:

- Image divided into $S \times S$ grid cells.
- Each cell predicts B boxes and C class probabilities.
- Each box:
$$(x, y, w, h, C)$$
- (x, y) = center coordinates (relative to grid cell).
- (w, h) = normalized width and height of box.
- $C = P(Object) \times IoU_{pred}^{truth}$.

Final Class Confidence:

$$P(Class_i) = P(Class_i | Object) \times C$$

Thus, YOLO outputs both class and location in one forward pass.

YOLO (You Only Look Once) Model

YOLO Loss Function and $\sqrt{w}$, $\sqrt{h}$ Explanation

The YOLO loss combines localization, confidence, and classification errors:

$$L = \lambda_{coord} \sum 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+\lambda_{coord} \sum 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+\sum 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

Why use $\sqrt{w}$ and $\sqrt{h}$:

- Large boxes can dominate the loss because width/height differences are larger in magnitude.
- Taking the square root normalizes the error scale, making small and large boxes contribute more evenly.
- This stabilizes training and prevents large objects from overshadowing smaller ones.

Parameters:

- λ_{coord} emphasizes localization accuracy.
- λ_{noobj} reduces penalties for background boxes.

Interactive example → link

YOLO (You Only Look Once) Model

Numerical Example

- 3×3 grid, each cell predicts $B = 2$ boxes.
- One cell prediction:

$$Box_1 = (x = 0.4, y = 0.5, w = 0.3, h = 0.4, C = 0.9)$$

$$Box_2 = (x = 0.6, y = 0.5, w = 0.2, h = 0.3, C = 0.6)$$

- Class probabilities:

$$P(person|obj) = 0.7, \quad P(car|obj) = 0.3$$

- Final confidence:

$$P(person) = 0.7 \times 0.9 = 0.63, \quad P(car) = 0.3 \times 0.9 = 0.27$$

YOLO selects “person” with 0.63 confidence as the detected object.

Common Mistakes

- Wrong: YOLO uses region proposals. Correct: YOLO predicts bounding boxes directly.
- Wrong: Coordinates are absolute. Correct: They are relative to the grid cell and normalized.
- Wrong: Confidence equals class probability. Correct: Confidence = $P(Object) \times IoU_{pred}^{truth}$.

YOLO (You Only Look Once) Model

Caveats

- YOLO struggles with small or overlapping objects.
- Grid size limits spatial precision.
- Accuracy decreases in crowded scenes.

Analogy

YOLO is like a human taking one quick glance at an image and identifying all objects instantly, without scanning piece by piece.

Get more info → YOLO Original Paper (Redmon et al., 2016)

Get more info → YOLO v1: Unified Real-Time Object Detection

Summary: YOLO (You Only Look Once) Model

Concept / Aspect	Key Explanation
Main Idea	Single network predicts all boxes and classes in one step.
Image Division	Split image into $S \times S$ grid; each predicts B boxes and C class scores.
Output Format	(x, y, w, h, C) per box; $P(Class_i \| Object)$ per cell.
Confidence Formula	$P(Class_i) = P(Class_i \| Object) \times P(Object) \times IoU_{pred}^{truth}$.
Loss Function	Combines localization, confidence, and classification losses.
Why $\sqrt{w}$, $\sqrt{h}$	Reduces dominance of large boxes, stabilizes training, balances loss for all object sizes.
Strength	Real-time detection (45+ FPS) and end-to-end training.
Weakness	Struggles with small, overlapping, or densely packed objects.
Applications	Drones, autonomous driving, robotics, security cameras.

Get more info → YOLO Framework

Get more info → YOLO v1 Paper