Image Segmentation

- Image segmentation is the process of dividing an image into distinct regions or segments that share similar attributes such as color, intensity, or texture.
- Each segment ideally represents a different object or a part of an object within the image.
- It helps in identifying and locating objects and boundaries (lines, curves, etc.) within an image.

Image Segmentation

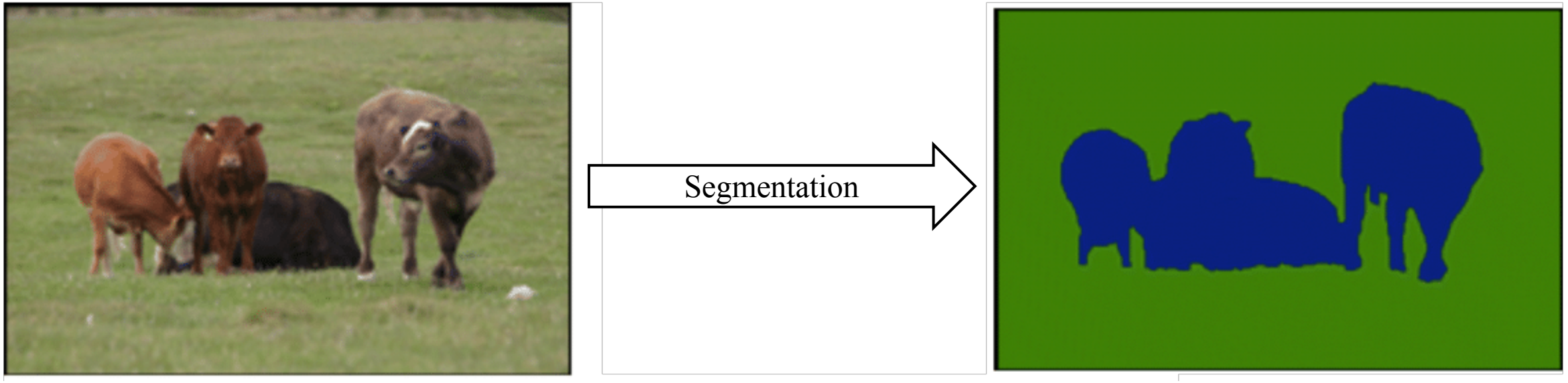
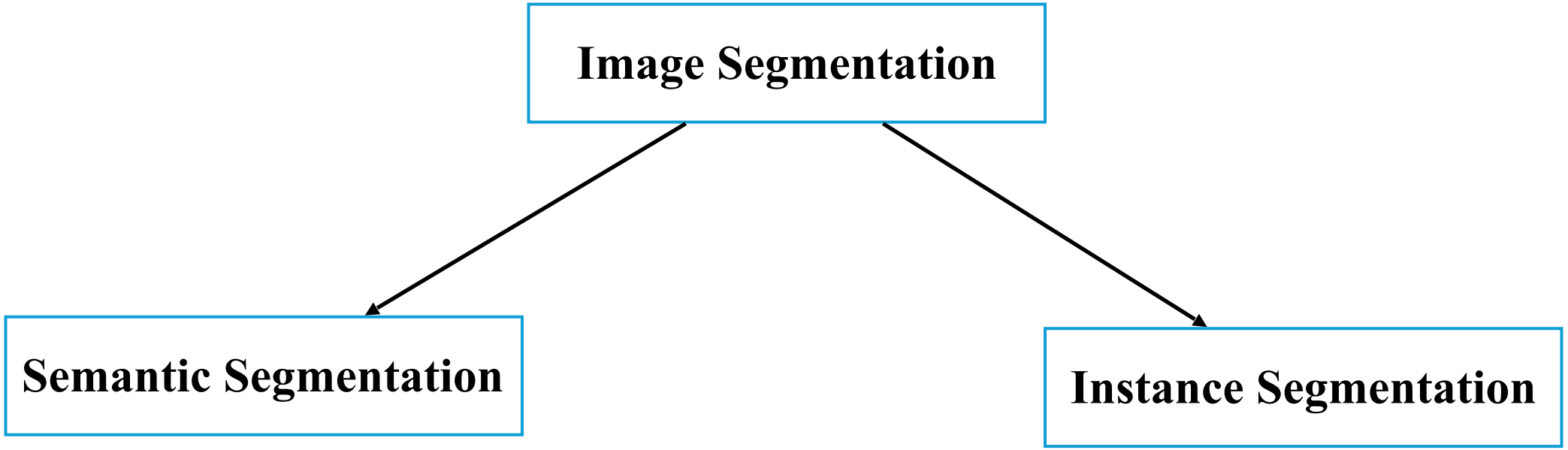


Image **Segmented Mask**

Source: https://www.researchgate.net/figure/Example-of-clustering-based-image-segmentation-a-Original-image-b-Segmented-image_fig2_349192395



Source: https://www.researchgate.net/figure/Example-of-clustering-based-image-segmentation-a-Original-image-b-Segmented-image_fig2_349192395

Semantic Segmentation

Semantic segmentation classifies pixels based on their semantic meaning, treating all objects within a category as one entity.

Source: https://keylabs.ai/blog/semantic-segmentation-vs-object-detection-a-comparison/

Instance Segmentation

Instance segmentation, distinguishes between different instances of the same class, allowing for more precise object identification and differentiation.

Source: https://www.researchgate.net/figure/Illustration-of-differences-in-segmentation-a-object-detection-b-semantic_fig3_350334710

Types of Image Segmentation

- **Thresholding-Based Segmentation:** Segments an image by converting it into a binary image, distinguishing objects from the background based on intensity values.

- **Edge-Based Segmentation:** Detects object boundaries by identifying significant changes in intensity, effectively highlighting the edges of objects.

- **Region-Based Segmentation:** Groups pixels into regions based on predefined criteria of similarity, such as intensity, color, or texture.

- **Deep Learning-Based Segmentation:** Utilizes neural networks, particularly convolutional neural networks (CNNs), for more advanced and precise segmentation tasks (e.g., U-Net, Mask R-CNN).

Thresholding-Based Segmentation

- **Input:** A gray-scale image I of size $M \times N$
- **Initialize Parameter:** Set a threshold value T. This value can be chosen manually or automatically (e.g., using Otsu's method).
- **Iterate Over Each Pixel**: For each pixel (i, j) in the image:
  Retrieve the intensity value $I(i, j)$
- **Apply Threshold:** Compare the pixel intensity value $I(i, j)$ with the threshold T:
  If $I(i, j) \geq T$: classify the pixel as **foreground** (set to 1 or white in the binary image)
  Else-If $I(i, j) < T$: classify the pixel as **background** (set to 0 or black in the binary image)
- **Construct the Binary Image**: Create a binary image B of the same size as the input image I.
  For each pixel (i,j):
  set $B(i, j) = 1$, if $I(i, j) \geq T$ **(foreground)**
  set $B(i, j) = 0$, if $I(i, j) \geq T$ **(background)**
- **Output**: The binary image B, where all pixels are either 0 (background) or 1 (foreground).

Thresholding-Based Segmentation

Python code:

```
def thresholding_segmentation(image, T):
    binary_image = np.zeros_like(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if image[i, j] >= T:
                binary_image[i, j] = 1
            else:
                binary_image[i, j] = 0
    return binary_image
```

Points to remember

Choosing the Threshold T:

- The threshold can be set manually if prior knowledge about the image's intensity distribution is available.
- Automatic methods like Otsu's method can be used to find an optimal threshold that maximizes the separation between the foreground and background.

Handling Noise:

- Thresholding can be sensitive to noise. Pre-processing steps like smoothing (using a Gaussian filter) can help reduce noise before thresholding.

Adaptive Thresholding:

- For images with varying lighting conditions, adaptive thresholding can be used. This method calculates a different threshold for each small region of the image, which allows for better segmentation in non-uniform illumination.

Example

Transformation

Original Image

Threshold and Segmented Image

Source: https://encord.com/blog/image-thresholding-image-processing/

Segmentation using U-Net

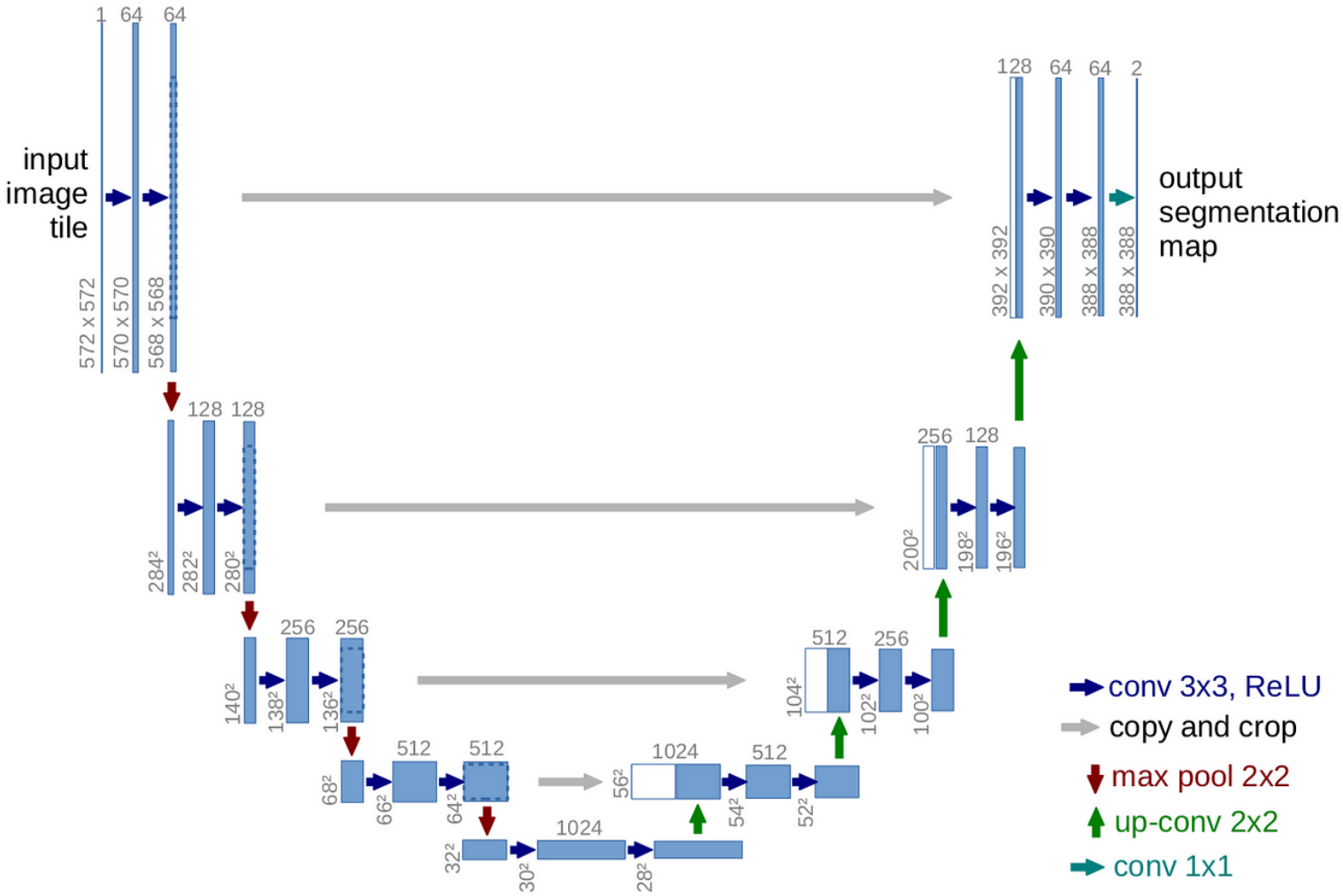
U-Net is a type of convolutional neural network (CNN) architecture designed specifically for image segmentation tasks, especially in medical imaging.

Architecture: U-Net has a symmetrical U-shaped structure consisting of two main parts:

Contracting Path (Encoder): This part is similar to a standard CNN. It captures context in the image by using repeated convolutional and max-pooling layers to down-sample the input image and extract features.

Expanding Path (Decoder): This part up-samples the feature maps using transposed convolutions, allowing precise localization and reconstruction of the image. It combines the high-resolution features from the contracting path via skip connections, which help retain spatial information.

U-Net Architecture



Source: https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5

Application of U-Net

- **Medical Imaging:**
  - **Organ Segmentation**: U-Net is extensively used to segment organs such as the liver, kidneys, heart, and lungs from medical scans (CT, MRI). Accurate organ segmentation is crucial for diagnostic purposes and treatment planning.
  - **Tumor Detection**: In oncology, U-Net can identify and delineate tumors or abnormal growths in medical images. This helps in assessing tumor size, shape, and growth patterns, which are important for diagnosis and therapy evaluation.
  - **Lesion and Disease Detection**: U-Net helps in detecting and segmenting lesions or pathological regions in images. For example, it can be used for segmenting multiple sclerosis lesions in brain MRI scans.

- **Satellite Imagery:**
  - **Land Cover Classification**: U-Net can classify different land cover types (e.g., forests, water bodies, urban areas) from satellite images. This is useful for environmental monitoring, urban planning, and disaster management.
  - **Change Detection**: It can be used to detect changes over time in satellite images, such as deforestation, urban expansion, or natural disasters.

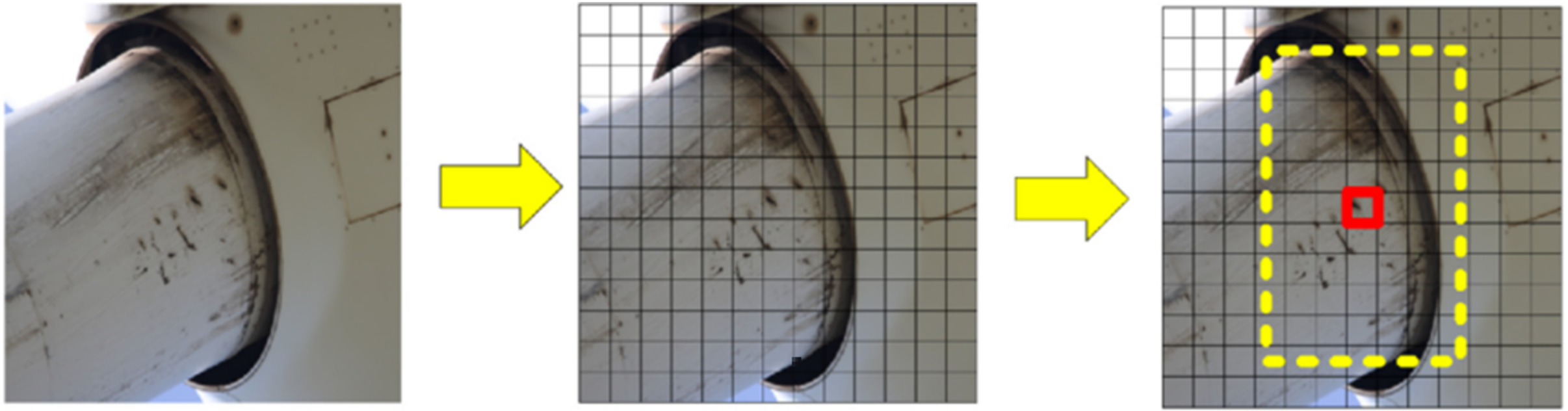
Application of U-Net

- **Agriculture:**
  - **Crop Monitoring**: U-Net can segment different crop types or detect crop health issues from aerial or satellite imagery. This helps farmers monitor crop growth and identify problems such as pest infestations or nutrient deficiencies.
  - **Weed Detection**: It helps in differentiating between crops and weeds, enabling targeted weed control and reducing the need for broad-spectrum herbicides.
- **Biology and Ecology:**
  - **Cell Segmentation**: U-Net is used to segment and analyze cells in microscopy images, aiding in biological research and understanding cellular structures and functions.
  - **Animal Tracking**: It can segment animals or specific features in ecological studies, assisting in tracking and monitoring wildlife populations.

Object Detection

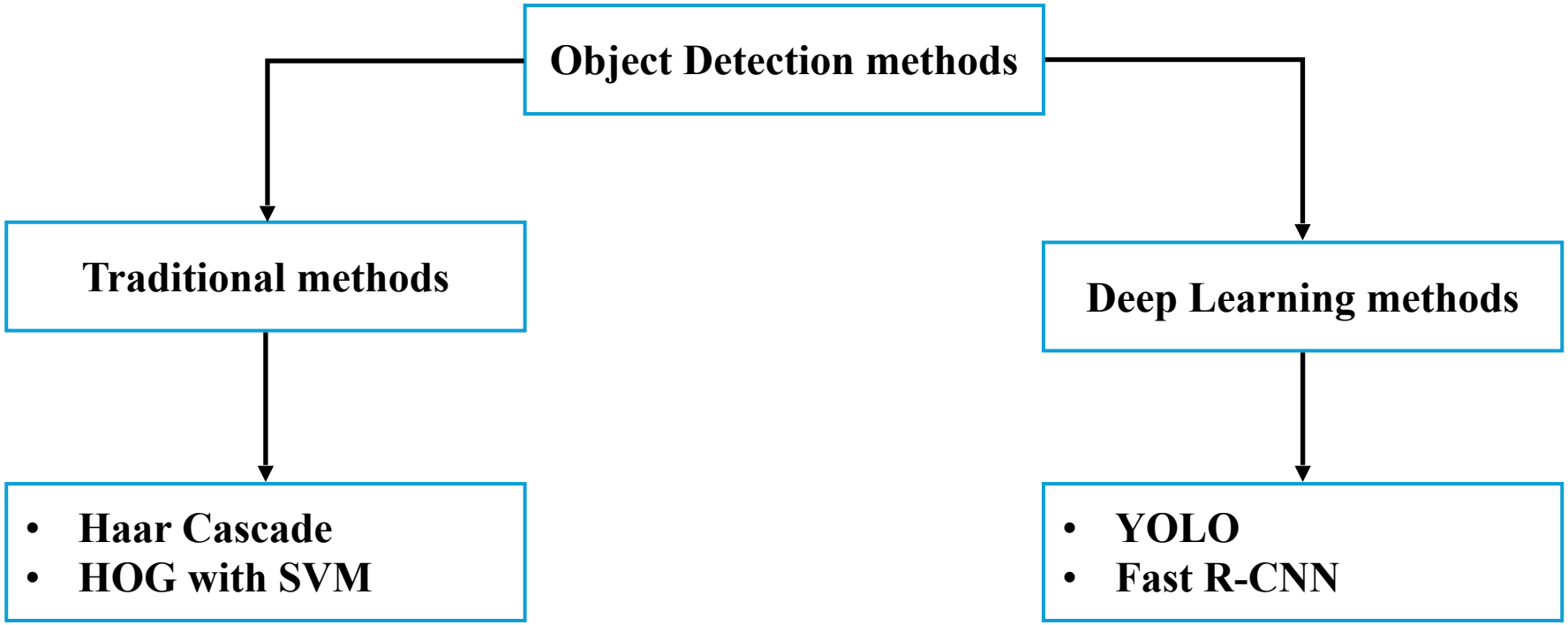
In image processing, object detection refers to the process of identifying and locating specific objects within an image. The goal is to not only determine which objects are present but also to find their exact locations, typically represented by bounding boxes or other geometric shapes.

Object Detection



Source: https://www.researchgate.net/figure/Decision-of-the-bounding-box-for-an-input-image-a-Original-image-b-13-13-grids_fig1_335003377

Object Detection



Haar Cascade method

- **Preprocessing:**
  - Load Image *I*
  - **Convert Image to Grayscale**
  - **Compute Integral Image** for efficient feature calculation

- **Feature Extraction:**
  - **Extract Haar Features** from the Integral Image for both positive and negative samples

- **Training:**
  - **Collect Dataset**: - Positive samples (images with the object)
    - Negative samples (images without the object)
  - **Extract Haar Features** from the dataset
  - **Train Weak Classifiers**: - For each Haar feature, train a weak classifier using the extracted features
  - **Combine Weak Classifiers** using AdaBoost to form a Strong Classifier
  - **Build Cascade Classifier**: - Arrange the Strong Classifiers into a cascade of stages

Haar Cascade method

- **Detection:**
  - **Initialize Sliding Window** with a starting position and scale
  - **For each window position** and **scale**:
    - **Apply Cascade Classifier** to the window
    - **If the window passes all stages**, mark it as a potential object detection
    - **Move Sliding Window** to the next position and scale

- **Post-Processing**:
  - **Apply Non-Maximum Suppression (NMS)** to remove overlapping bounding boxes - For each detected bounding box, suppress other boxes that have high overlap (Intersection over Union) with it

- **Output**:
  - **Return Detected Objects** with bounding boxes

YOLO (You Only Look Once) algorithm

- **Input Image**:
  - An image is resized to a fixed size (e.g., 416x416 pixels) suitable for the YOLO model.

- **Feature Extraction**:
  - The resized image is passed through a convolutional neural network (CNN), which extracts features from the image using a series of convolutional layers. This network acts as the backbone, capturing various spatial hierarchies and patterns.

- **Grid Division**:
  - The CNN's output is divided into a grid of cells (e.g., 13x13 or 19x19 cells).
  - Each cell is responsible for detecting objects whose center falls within the cell.

YOLO (You Only Look Once) algorithm

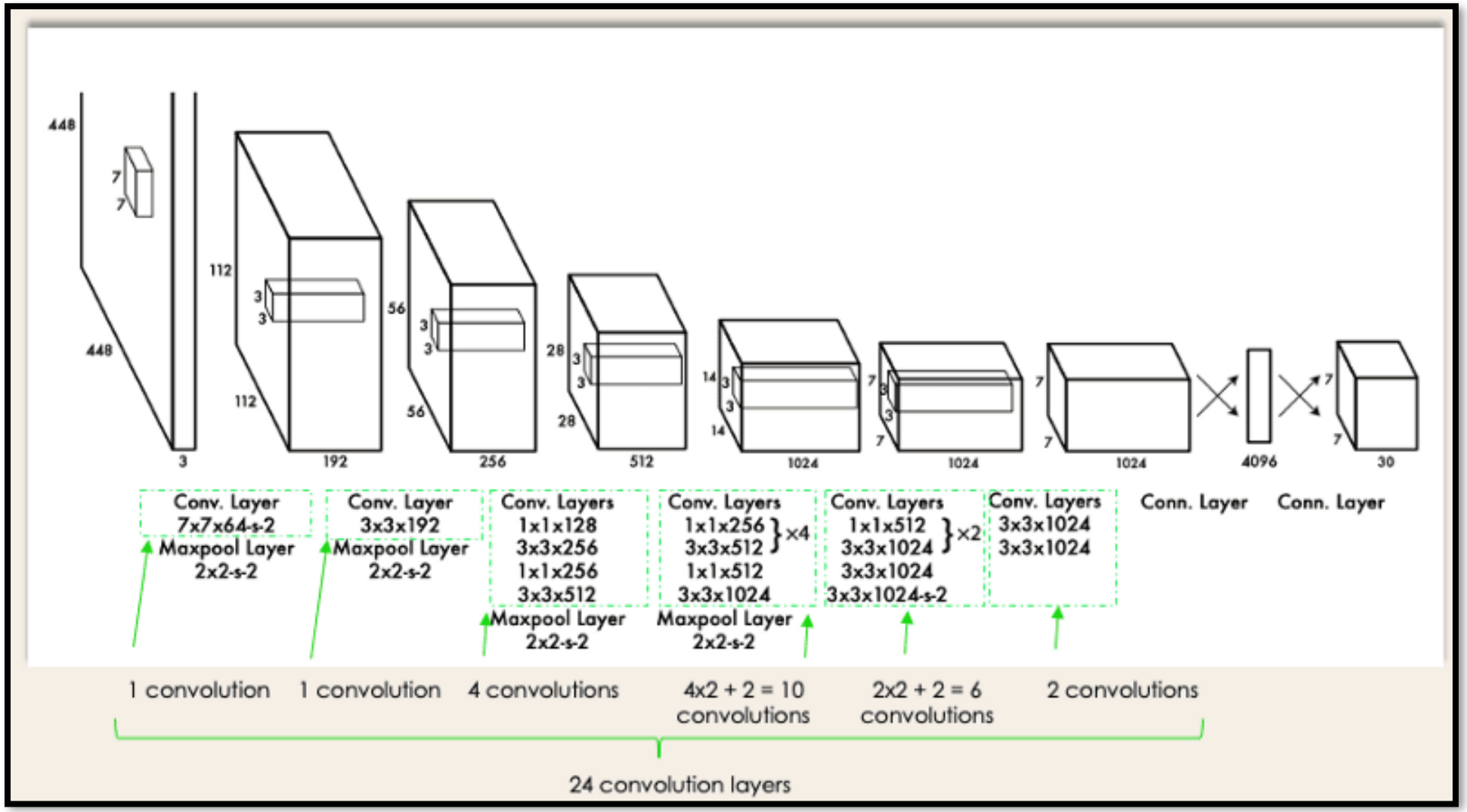
- **Bounding Box and Class Prediction**:
  - Each grid cell predicts a fixed number of bounding boxes (e.g., 3 boxes per cell).
  For each bounding box, it outputs:
    Bounding Box Coordinates: Center (x, y), width (w), and height (h) relative to the grid cell.
    Objectness Score: Confidence that the bounding box contains an object.
    Class Probabilities: Probabilities of each class, indicating which class the object belongs to.

- **Combine Predictions**:
  - The outputs from all grid cells are combined to form the final detections.
  - This includes bounding boxes, objectness scores, and class probabilities.

- **Post-Processing**:
  - **Non-Maximum Suppression (NMS)**: Applied to remove overlapping bounding boxes that have high Intersection over Union (IoU) with each other, keeping only the most confident detections.

- **Output**:
  - The final result is a list of detected objects with their bounding boxes and class labels.

YOLO architecture



Source: https://arxiv.org/pdf/1506.02640

Applications of YOLO

- **Medical Research**
  - **Cell Detection**: YOLO can be used to detect and classify different types of cells in microscopic images, aiding in research on cellular biology and disease.
  - **Surgical Instrument Detection**: Assisting in real-time detection and tracking of surgical instruments during operations to improve surgical precision.
- **Physics**
  - **Particle Detection**: YOLO can assist in identifying and tracking particles in experimental physics, such as detecting tracks of particles in high-energy physics experiments.
  - **Laboratory Experiment Monitoring**: Monitoring experimental setups and detecting anomalies or specific components in laboratory settings.

Source: https://arxiv.org/pdf/1506.02640

Applications of YOLO

○ Agriculture

- **Crop Monitoring**: YOLO can help in detecting crop diseases, pests, or growth stages from aerial or ground-level images, improving precision agriculture practices.
- **Weed Detection**: Identifying and classifying weeds in agricultural fields to optimize herbicide application and reduce crop competition.

○ Astronomy

- **Celestial Object Detection**: YOLO can be applied to detect and classify celestial objects like asteroids, comets, and stars in astronomical images or video data.
- **Space Debris Monitoring**: Detecting and tracking space debris in satellite imagery to prevent collisions.