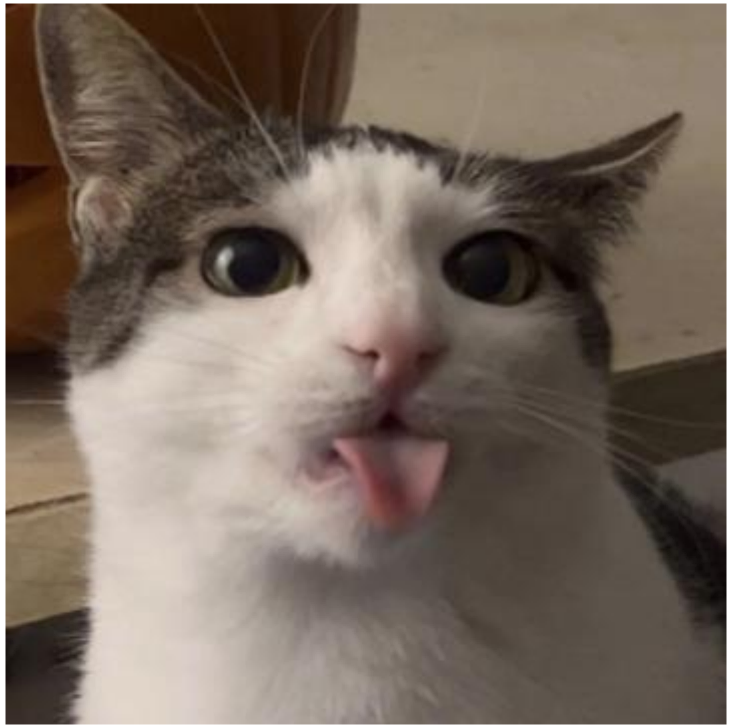
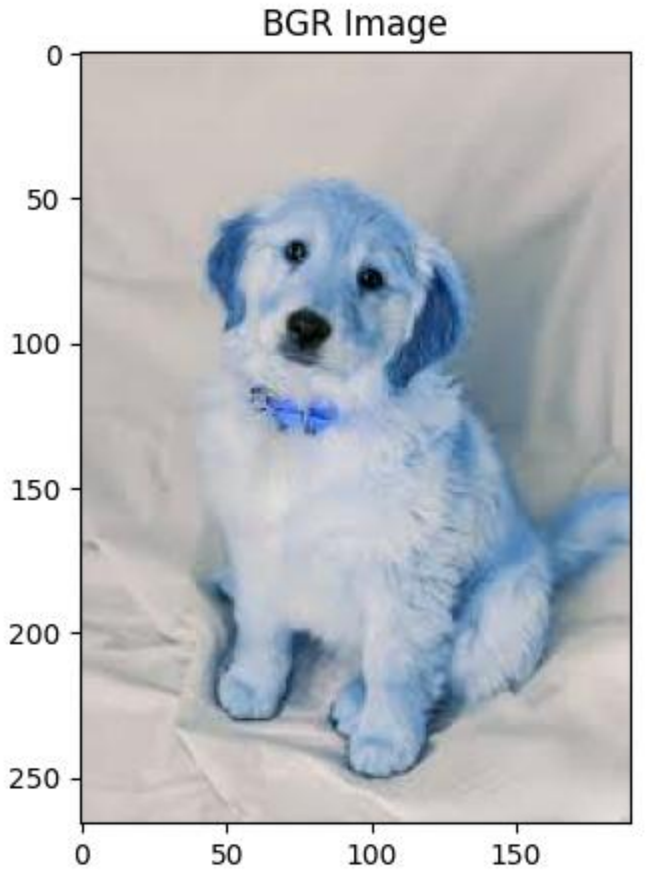
Object Detection & Segmentation with OpenCV + YOLO

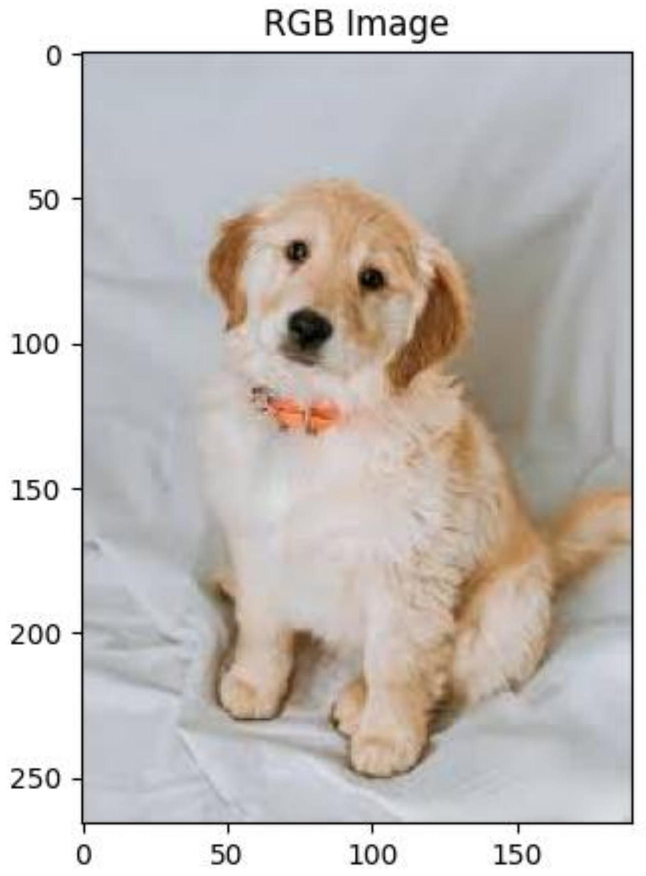
How a Computer "Sees"



- ...is just a 3-D array of numbers:
- **(Height × Width × 3 Channels)**
- Each pixel has three integers (0-255).
- For OpenCV, the order is **B**lue, **G**reen, **R**ed.

Color Spaces Matter





A correctly colored dog picture. (right) **Expected (RGB)**

A dog picture with swapped color channels (blueish tint). **What OpenCV Loads (BGR)**

The Fix: rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)

Pre-processing

- Preparing images for robust detection
- **Resize** — Speed & uniform input
- **Crop** — Focus on a region of interest
- **Blur / Denoise** — Reduce image noise
- **Edge Detection (Canny)** — Highlight shapes
- **Grayscale** — Simplify to one channel

Why Pre-process? Garbage In, Garbage Out

- **Standardize Inputs:** Machine learning models expect all data to be in a consistent format (e.g., same size, same value range).
- **Reduce Noise:** Real-world images have imperfections like camera grain, poor lighting, and blur. We can clean this up.
- **Enhance Features:** We can make important features, like edges or contours, more prominent for the model to "see".
- **Improve Speed & Efficiency:** Simpler, smaller images are processed much faster.

Standardizing the Canvas (Size & Color)

1. Resizing

- Changing the height and width of the image.
- Model Requirement: Most deep learning models are trained on and expect a fixed input size (e.g., 640x640 pixels).
- Computational Speed: A 4K image (3840x2160) has over 8 million pixels. A 640x640 image has about 400,000. Processing the smaller image is ~20x faster!
- **What is the trade-off of resizing? If you shrink an image of a crowded street too much to make it faster, what critical information might you lose?**

2. Grayscaling

- Converting a 3-channel BGR image into a 1-channel grayscale image.
- Why we do it:
- Simplicity: We reduce the amount of data by 66%! This makes subsequent operations much faster.
- Focus on Structure: It forces the algorithm to focus on shapes, textures, and brightness, not color.
- **When would grayscaling be a bad idea? Name a detection task where color is the most important feature. (e.g., identifying traffic light status, sorting ripe vs. unripe fruit).**

Seeing Through the Noise (Blurring)

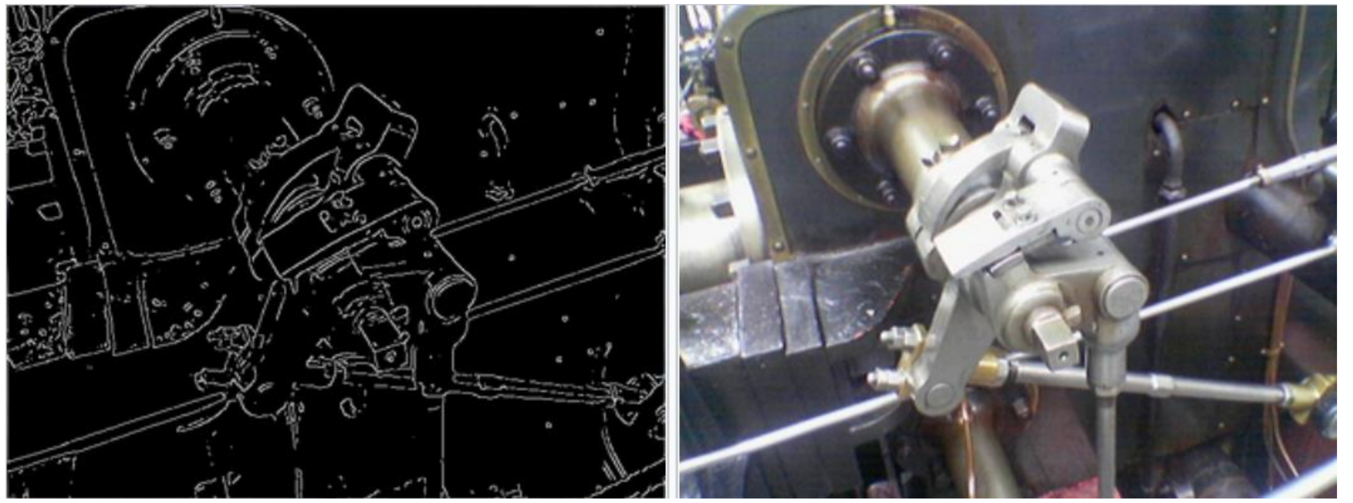
Applying a filter (like a Gaussian blur) that averages each pixel with its neighbors. This smooths the image.

- **Why we do it:**
  - It reduces minor, irrelevant details and random noise (e.g., sensor noise from a low-light photo).
  - This helps algorithms like edge detection perform better, as they won't be triggered by tiny, insignificant textures.
  - **Blurring intentionally destroys information. Imagine you are building a system to read license plates from a security camera. How could blurring be helpful? How could it be harmful if you apply too much?**

Thresholding & Edges

- **1. Thresholding (Binarization)**
- Converting a grayscale image into a pure black-and-white (binary) image. If a pixel's value is above a threshold, it becomes white (255); if below, it becomes black (0).
- It's the most aggressive way to separate a foreground object from the background, assuming there's good contrast.

- **2. Edge Detection (Canny)**
- A multi-step algorithm that finds and highlights sharp changes in intensity—the outlines or "edges" of objects.
- It reduces a complex image to just its structural contours. This is incredibly useful for shape analysis and was a cornerstone of "classic" computer vision.



Normalization

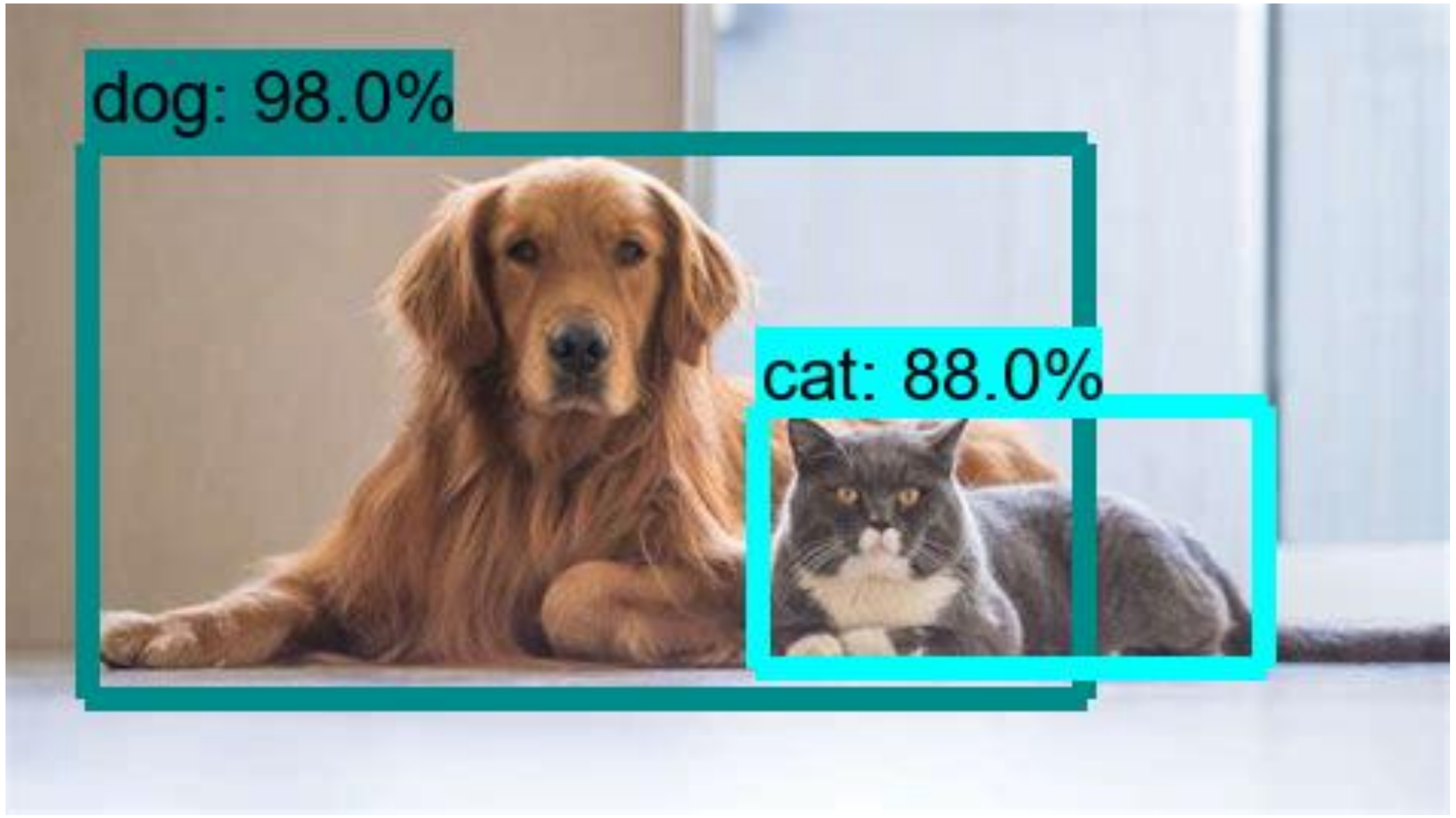
- This is a critical pre-processing step specifically for neural networks like YOLO.
- Scaling the pixel values from their original range of [0, 255] to a smaller range, typically [0.0, 1.0].
- # Simple normalization in NumPy
- normalized_image = image_as_float / 255.0
- Neural networks learn by adjusting internal weights based on the error they make. Large input numbers (like 255) can lead to massive, unstable adjustments, making the model learn slowly or not at all.
- It ensures that all input features (in this case, pixels) are on a similar scale, which is a fundamental assumption for many machine learning algorithms.

Mini-Exercise

- "Grayscale the image, then draw a red rectangle around the brightest 100x100 patch."
- **Block: Hints**
  Use np.argmax, array slicing, and cv2.rectangle.

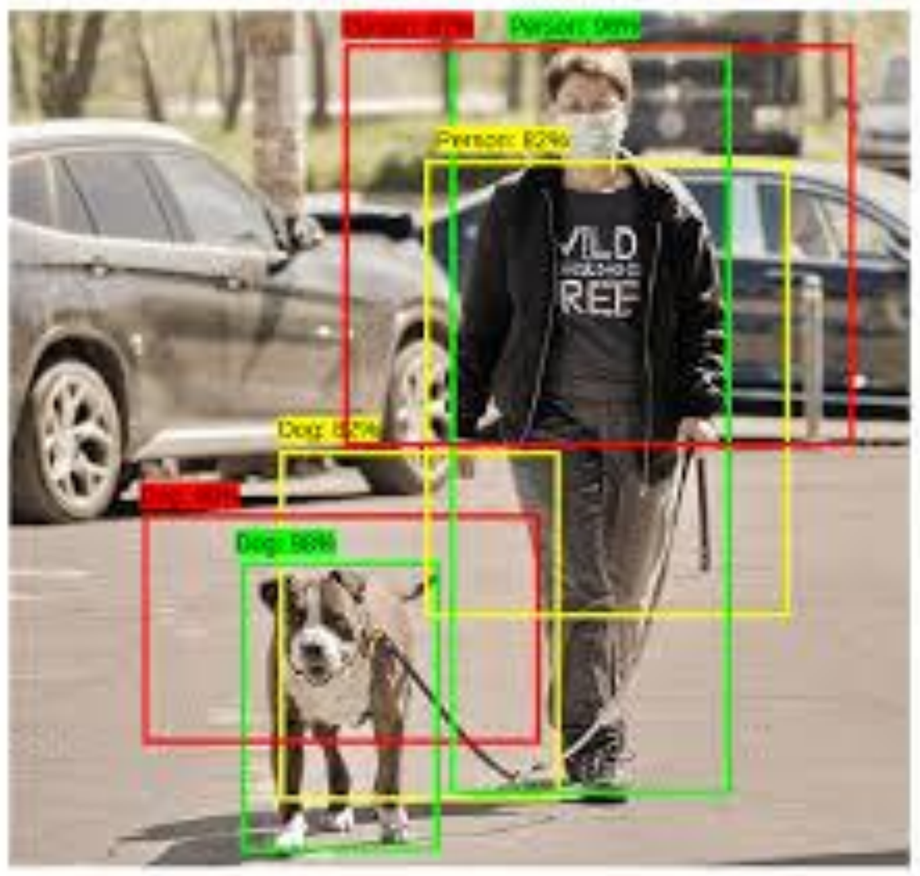
A bounding box localizes and classifies an object.

- **Box** = (x, y, w, h)
- **Class** = "cat", "dog", ...
- **Confidence** = 0.0 to 1.0



How do we measure if a predicted box is "good"?

- IoU measures the overlap between the ground truth and predicted boxes.
- **Formula:**
  IoU = (Area of Overlap) / (Area of Union)
- A common threshold for a "correct" detection is **IoU > 0.5**.

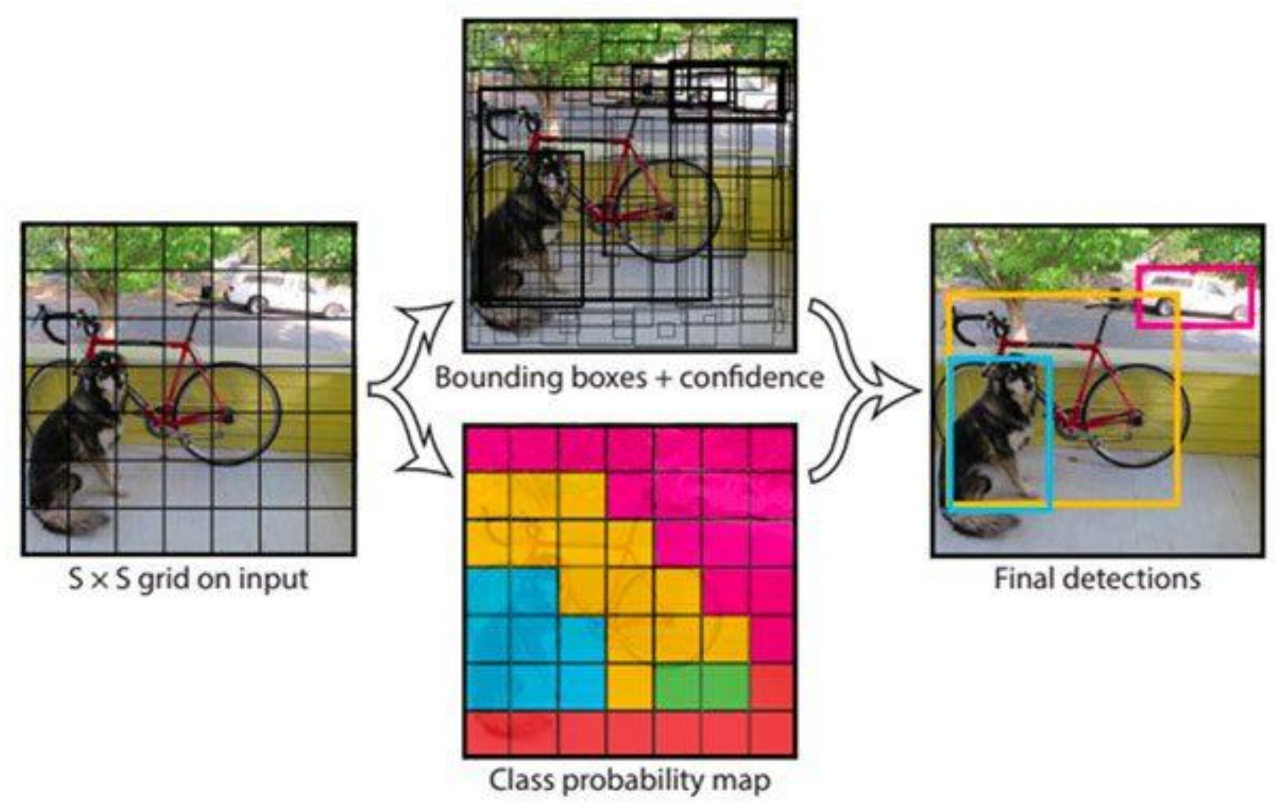


Classic vs. Deep Learning Vision

- **Classic Vision (Haar, HOG)**
  Relies on *hand-crafted features*. Brittle and requires manual tuning.

- **Deep Learning (YOLO)**
  Learns
  features *automatically* from data. Robust and fast.

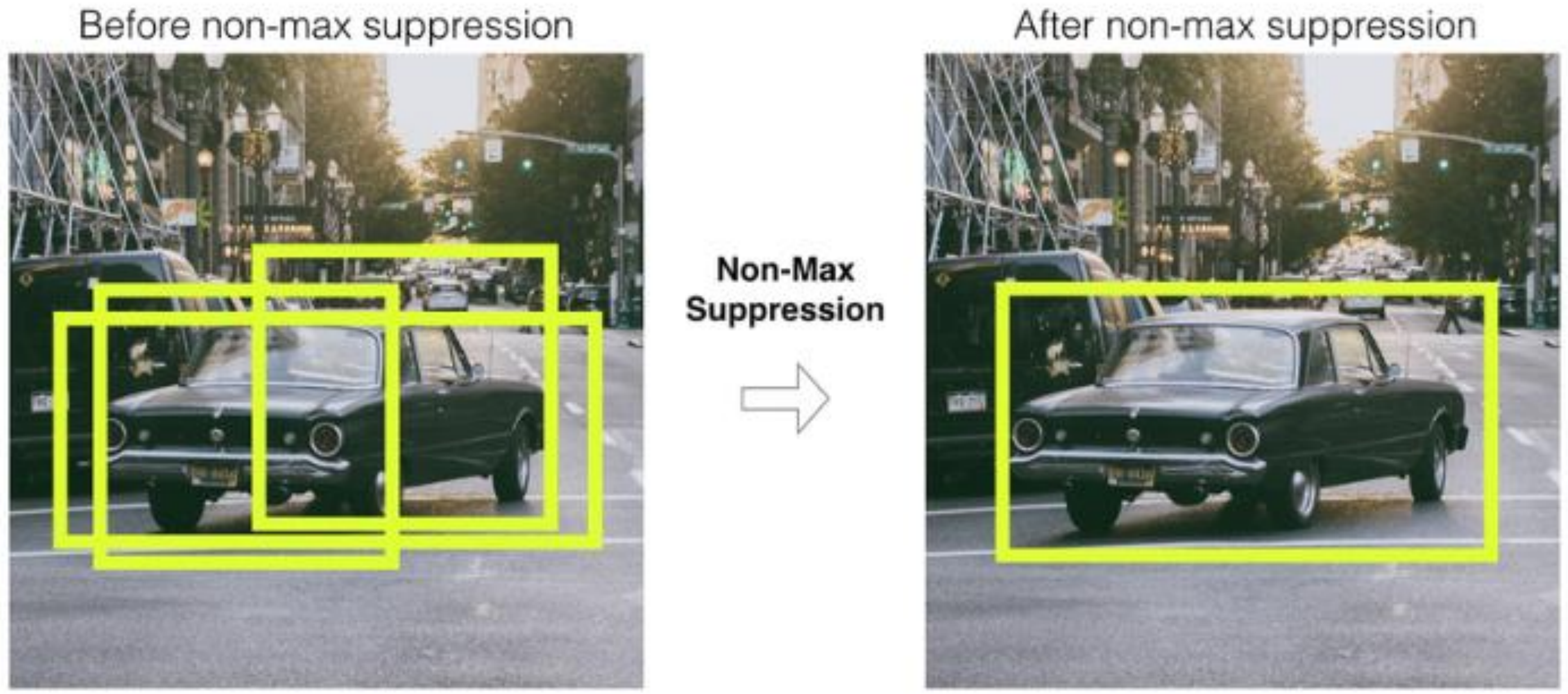
You Only Look Once



- Image is split into a grid.
- Each grid cell predicts boxes and classes for objects centered within it.
- Everything happens in **one single pass** of the network.

Cleaning Up: Non-Max Suppression (NMS)

NMS removes duplicate, overlapping boxes for the same object.



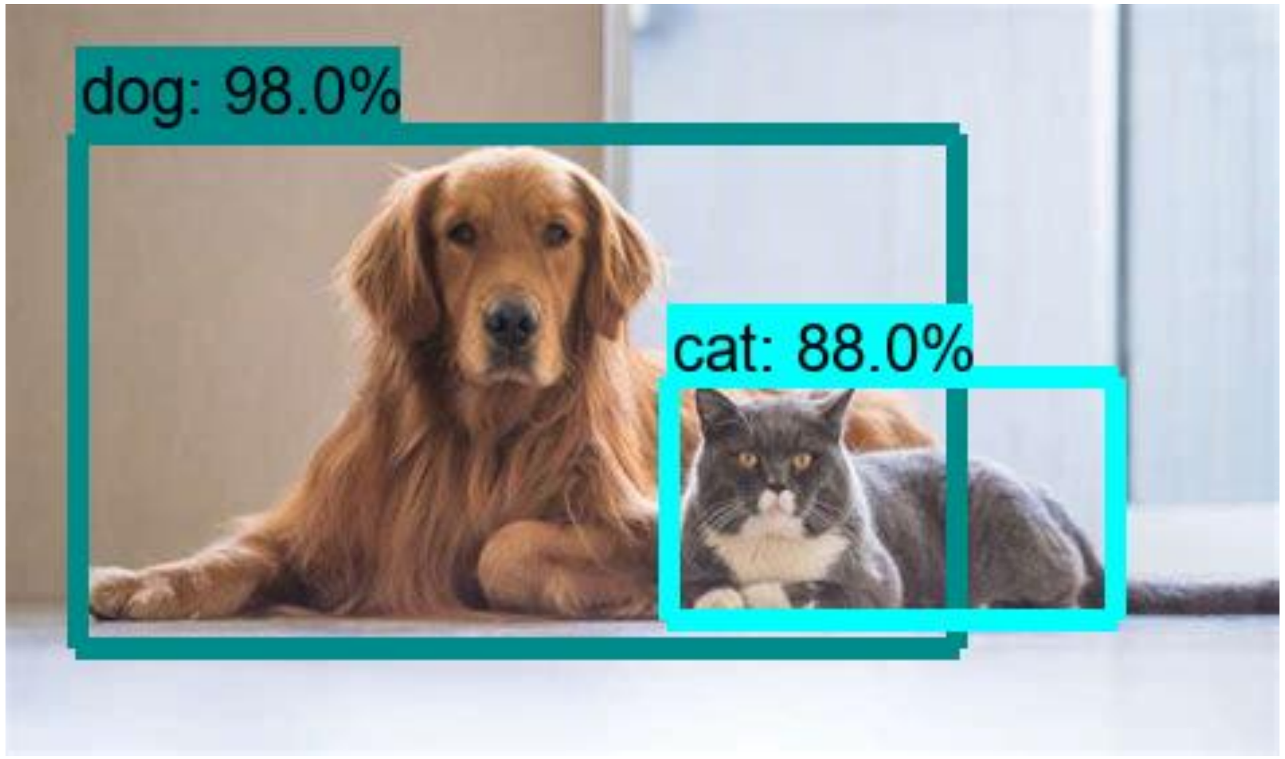
Code Setup

- pip install ultralytics opencv-python
- from ultralytics import YOLO
- model = YOLO('yolov8n.pt')

Single-Image Detection

- # Run detection on an image
- results = model('pets.jpg')

- # Plot the results with boxes
- results[0].plot()

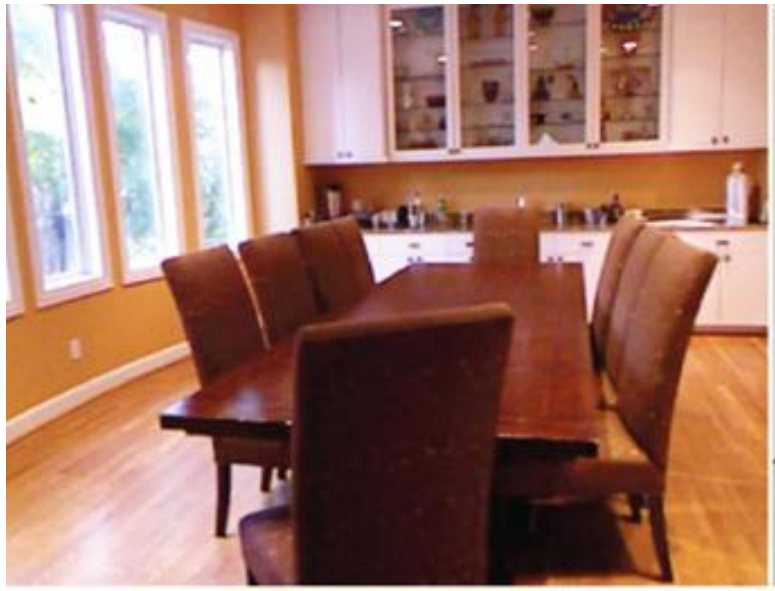


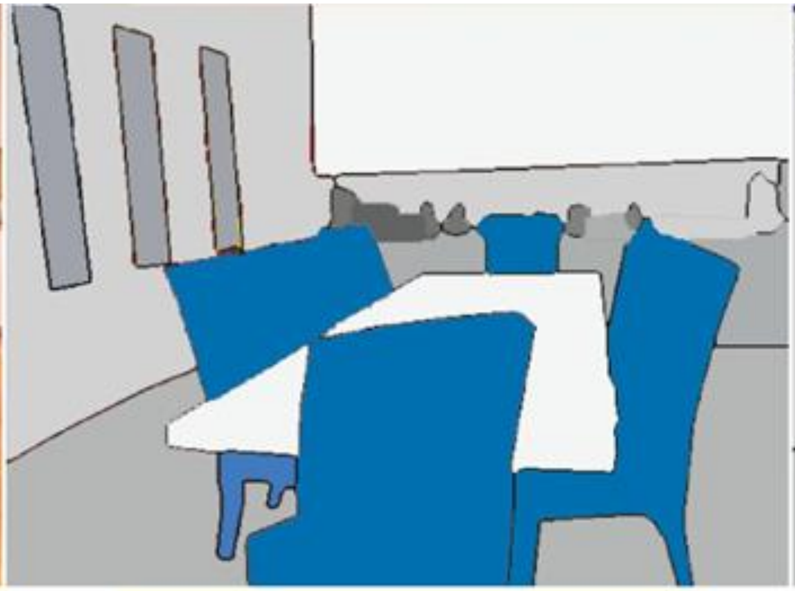
Video & Webcam

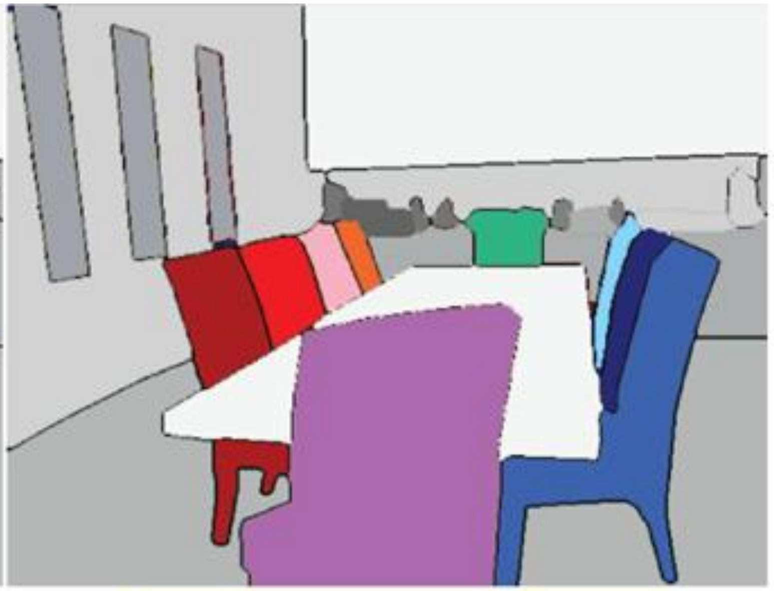
- **Video File**
- model.predict('traffic.mp4', save=True)

- **Live Webcam**
- model.predict(source=0, show=True) # Use camera 0

Segmentation


Input Image


Semantic Segmentation


Instance Segmentation

YOLOv8-Seg in Code

- from ultralytics import YOLO

- # Load a segmentation model
- model = YOLO('yolov8n-seg.pt')

- # Run inference and plot the masks
- results = model('street.jpg')
- results[0].plot()