Multiple Partial Object Detection in a Single Image using Segmentation and Elliptical Symmetry

Nooreen Fatima

May 4, 2025

Abstract

In my project, I aim to perform Multiple Partial Object Segmentation and Detection in a Single Image. The idea is to use a simple image processing angeometric operation to identify the location of objects in an image. This is done by segmenting all objects, doing image reconstruction and performing a detection algorithm on each. The entire project will be done in Python, using its libraries, which include: openCV, NumPy, sklearn/tensorflow, YOLO algorithm, and more. A practical application of this project is as follows. Imagine a cluttered room where you are in a hurry, cannot find something you need to head out. You can just take a picture of the room, upload it into the software, and it will find and locate the objects found in the image, whether or not they are fully visible.

1 Introduction

1.1 Inspiration

The idea of working on this project was inspired by my visit to a book store. I was looking for a particular book, which was supposedly in the stated aisle of the store according to the website and the librarian's database, yet it was nowhere to be found. I wish there was a way to take a picture of the shelves and have the software find the book for me. This led me to the idea of what if I could just take a picture of a messy room, feed it to a software and make it find whatever I am looking for, for example, my keys.

1.2 Main Idea

The main idea of the research is focused on finding multiple objects [**?**] at the same time in a single image and then label them, not just the labeling. I have used YOLO algorithm for the detection (which I will elaborate further later in the paper), however it has 2 major disadvantages: [gj]

- it cannot identify partially covered objects always.
- it cannot identify objects at an angle always.

These disadvantages are what I have attempted to solve using multiple Image Processing techniques along with developing a software with a real world application.

1.3 Problem Statement

In the above pictures, we can see that the limitation of the current YOLO algorithm. These deep learning models require vast amounts of labeled training data and significant computational resources, making them less practical in real-world applications. Additionally, they often struggle when objects in images are partially covered or fragmented, a challenge that's frequent in everyday scenes. This project proposes an alternative approach by using simple image processing techniques to detect and complete multiple objects in the same image, without relying on deep learning. Instead of needing extensive training, it uses geometric properties and simple image processing techniques. This makes it a lightweight, easy-to-apply solution for object detection, even in cluttered or incomplete scenes. By focusing on reconstructing objects based on their visible contours, the method offers a faster, data-efficient way to handle incompleteness and partial detections in images.

Figure 1: Not detectable by Existing Algorithm

Figure 2: Detectable by Algorithm - This Paper's Algorithm Processing: Segmentation and Object Completion

2 System Overview

The flow of the entire process is given below. The input is an image containing multiple objects, fully or partially visible. The output is a labeled image with the object names.

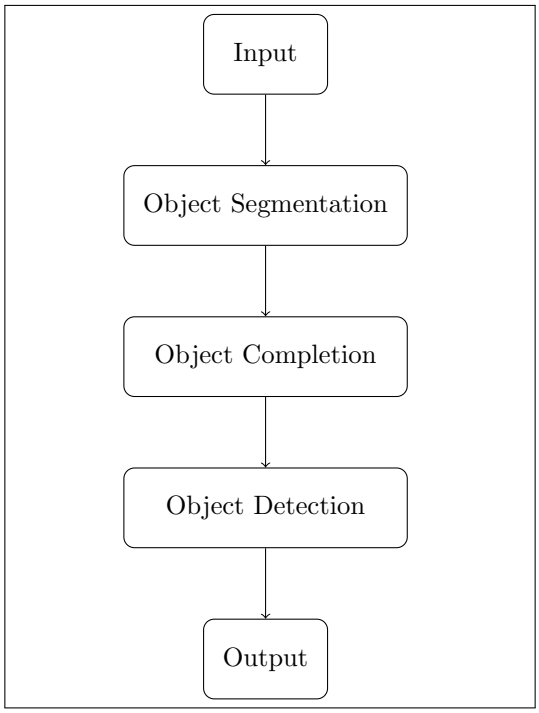


Figure 3: Flow of System Processes

3 Object Segmentation

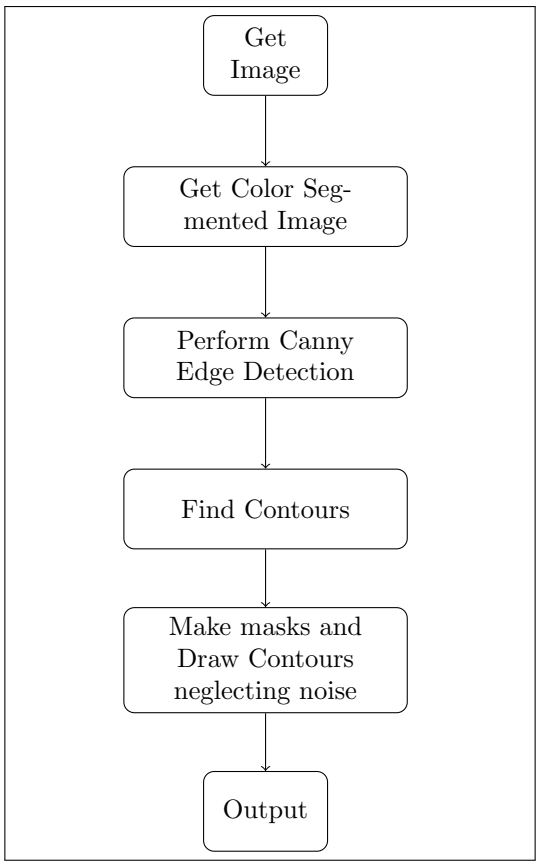


Figure 4: Flow of Object Segmentation

3.1 Color Filtered Image

Figure 5: Color Filtered Output

[OPE]

cv2.pyrMeanShiftFiltering() is an edge-preserving, noise-reducing filter that segments the image based on color and spatial proximity. It works by shifting each pixel toward the average color of its neighbors, creating smooth, region-based segmentation. This filter is particularly useful for preprocessing before contour detection or shape approximation.

This step is done for enhancing object boundaries while maintaining the overall structure, making it effective for visual segmentation tasks.

3.2 Edge Detection

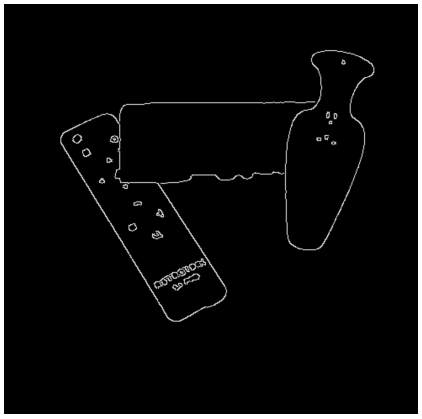


Figure 6: Canny Edge Detection Output

Canny Edge Detection is a popular technique to detect edges by finding areas of rapid intensity change. It starts by applying Gaussian blur to reduce noise in the image.Then, it computes the gradient magnitude and direction using Sobel operators. Non-maximum suppression is applied to thin out the edges by keeping only local maxima.Finally, double thresholding and hysteresis link strong and weak edges to form the final edge map.

This step is done so that we get the edges of the objects which is required in finding the contours of all the objects in the image.

3.3 Dilating the Edges

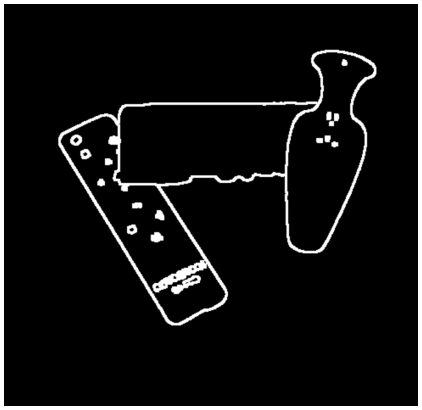


Figure 7: Edge Dilation Output

Image dilation is a morphological operation that expands or thickens a binary image's white regions (foreground). It's used to fill small gaps, connect disjoint components, or accentuate particular features in an image. In essence, dilation makes image features more prominent and easier for algorithms to process and analyze. [Mor23]

The edges produced by the Canny algorithm are sometimes disconnected and sparse, and have very less thickness. To join the sparse edges and to increase the thickness, we have to dilate them. I have used 3x3 kernel to do the same.

3.4 Finding the Contours

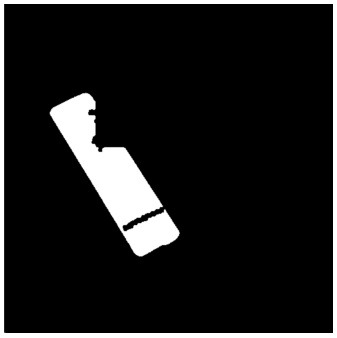


Figure 8: Contour of Object 1

Figure 9: Contour of Object 2)

Figure 10: Contour of Object 3

Contours are curves joining continuous points along a boundary with the same color or intensity. These are produced using the findContours OpenCV function and are stored as a list of points and can be drawn, analyzed, or processed further. Hierarchy information can also be preserved which describe the relationship between contours. I have used RETR_CCOMP: which is a contour retrieval mode in OpenCV's findContours function. It organizes contours into a two-level hierarchy: outer contours and holes inside them.The external boundary of an object is on the first level, and any holes (internal contours) are on the second. Each contour's relationship (parent or child) is recorded in the hierarchy output. This step is done so that overlapped objects can be found separately.

3.5 Rotating the Objects Upright

Figure 11: Segmented and Singled out Objects Output 1

Figure 12: Segmented and Singled out Objects Output 2

All the contours which are too small (noise) are filtered out. A mask is created with black pixels and the singled / segmented out objects are placed on the mask in the same position as the original image. Since YOLO has difficulty in detecting objects at an angle, they are made upright and then completed and sent for detection. Object 2 is not rotated since it is already upright.

4 Object Completion

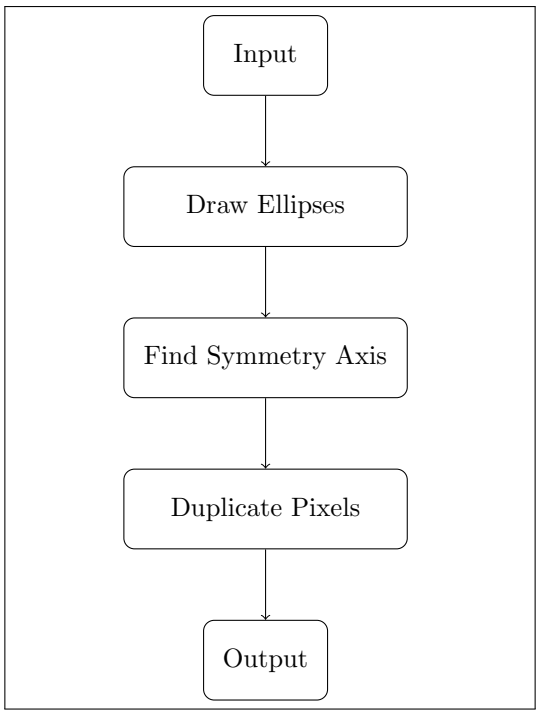


Figure 13: Flow of Object Completion

4.1 Draw Ellipses

Figure 14: Ellipse on Object 1

Figure 15: Ellipse on Object 2

Figure 16: Ellipse on Object 3

The axes of a fitted ellipse can be used to find a line of symmetry because they represent the principal directions of the object's shape - the major axis shows the longest spread, and the minor axis shows the shortest. If the object is symmetric, the line of symmetry will often align with one of these axes, as they correspond to the natural orientation and balance of the shape. These axes are mathematically stable and less sensitive to small contour noise, making them reliable for symmetry estimation even in imperfect objects. [DPM95] Ellipses are fitted to detected contours using methods like cv2.fitEllipse(), capturing the shape and orientation of partially visible objects. This step approximates complex or fragmented regions with smooth geometric shapes, useful for inferring missing parts. Drawing ellipses helps visualize the estimated object boundaries and guides further symmetry-based processing. [GAWW93] It also standardizes irregular shapes into analyzable forms, aiding in consistent symmetry axis estimation.

4.2 Find Symmetry Axes

The symmetry axis is typically defined as the major or minor axis of the fitted ellipse, depending on the object's structure. This axis serves as a reference for reflecting pixels to complete the occluded or missing side of an object. Geometric properties like the ellipse's center, angle, and axis lengths are used to calculate this symmetry line. Identifying the correct axis is crucial, as it determines the accuracy of the mirrored or completed shape. [GAWW93]

Figure 17: Mirrored Left (A)

Figure 18: Right (B)

Figure 19: A - B

Figure 20: Right (A)

Figure 21: Mirrored Left (B)

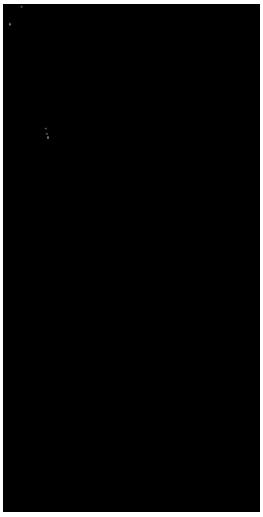


Figure 22: A - B

In order to find which side to mirror, matrix properties of subtraction, A-B is used. This basically implies that pixels which are there in left but not in right are turned white. Then mean of the image is calculated. Whichever parts are missing in B, all those pixels are turned into white which increases the mean value. By this logic, whichever has lower mean is the complete half of the object and that should be mirrored. From the above images, it can be seen that 28 has more white pixels, therefore A corresponding to that, .i.e. left side should be mirrored. [DPM95]

4.3 Duplicate Pixels

Figure 23: Symmetry: Object 1

Figure 24: Symmetry: Object 2

Figure 25: Symmetry: Object 3

Once the symmetry axis is known, pixel data from one side of the object is mirrored across the axis to generate the missing half. This duplication simulates how the object might appear if it were fully visible, aiding in its recognition and completion. Numpy operations like np.flip() or duplicating or affine transforms are commonly used to reflect pixel values efficiently. The goal is to preserve continuity and structure, enabling non-deep learning methods to restore covered regions.

5 Object Detection

Figure 26: Detection: Object 1

Figure 27: Detection: Object 2

Figure 28: Detection: Object 3

Object detection is a computer vision technique that aims to identify and locate objects within images or videos. It not only recognizes the objects' classes (e.g., cars, dogs, or people) but also determines their positions within the scene using bounding boxes. Object detection is a crucial component in various applications, including autonomous vehicles, security systems, and facial recognition. [SZL21]

5.0.1 YOLO Algorithm

I have used YOLO (You Only Look Once) Algorithm to achieve this. It is a single-stage object detector that uses a convolutional neural network (CNN) to predict the bounding boxes and class probabilities of objects in input images. [yol22]

Benefits of YOLO:

- Process frames at the rate of 45 fps (larger network) to 150 fps(smaller network) which is better than real-time.
- The network is able to generalize the image better.

Disadvantages of YOLO:

- Comparatively low recall and more localization error compared to Faster R_CNN. [RHGS15]
- Struggles to detect close objects because each grid can propose only 2 bounding boxes.
- Struggles to detect small objects.

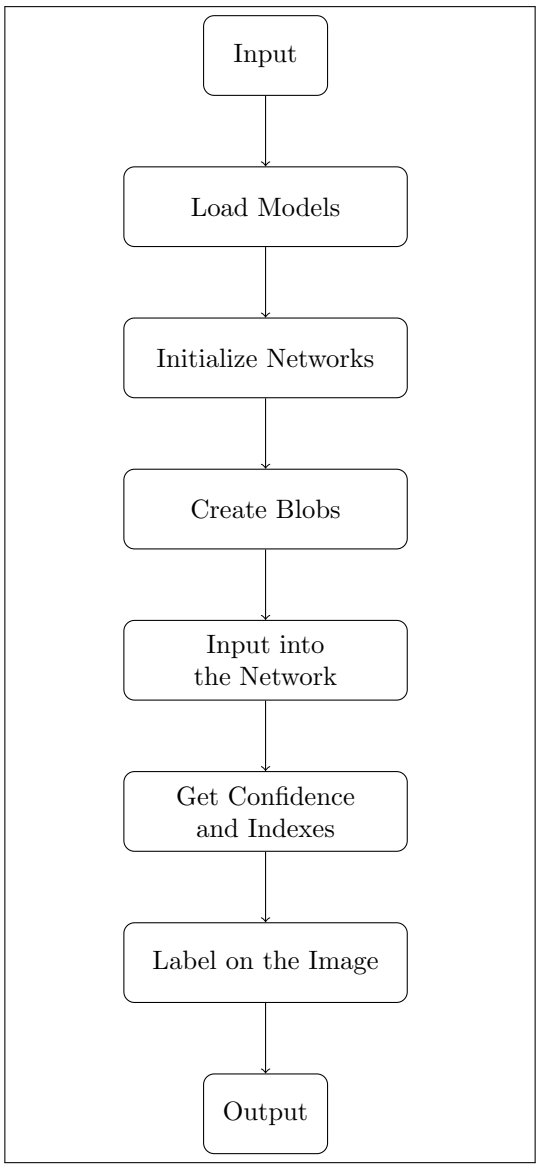
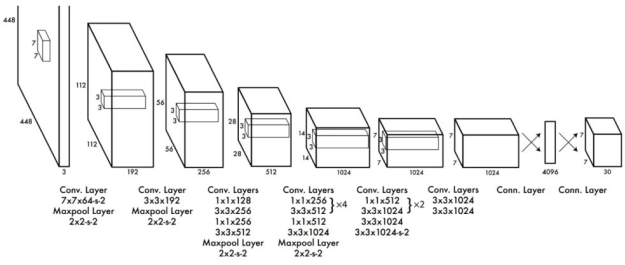


Figure 29: Flow of Object Detection using YOLO

5.0.2 Neural Networks of YOLO Algorithm



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

Figure 30: YOLO Architechture [yol22]

The first 20 convolution layers of the model are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer. Then, this pre-trained model is converted to perform detection since previous research showcased that adding convolution and connected layers to a pre-trained network improves performance. YOLO's final fully connected layer predicts both class probabilities and bounding box coordinates. [Kun02]

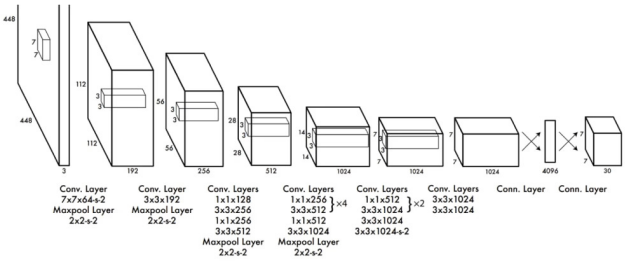
5.0.3 Creating Blobs

The process begins by converting the input image into a "blob" using a function like cv2.dnn.blobFromImage(), which standardizes the image size and format. The blob essentially bundles the image data into a multi-dimensional array, making it ready for batch processing in deep learning models. This conversion involves resizing, normalizing pixel values, and optionally swapping color channels to fit the network's expected input. Creating blobs ensures a consistent input structure for YOLO, regardless of the original image dimensions or color space.

5.0.4 Feed Forward Network

Once the blob is created, it is fed into the YOLO network through a forward pass using net.forward(), initiating the detection process. During this pass, the blob propagates through multiple convolutional and pooling layers, allowing the network to extract hierarchical features from the image. The feed forward operation is responsible for computing feature maps that encode both the object's appearance and spatial information. This step effectively transforms the raw image data into prediction maps where potential object detections are represented with bounding boxes and confidence scores.

5.0.5 Getting Confidences and Indexes



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

Figure 31: Confidence and Labeled Object [yol22]

The network's output is a set of predictions that include bounding boxes, confidence scores, and class probabilities for various regions in the image. Each predicted bounding box is associated with a confidence score indicating the likelihood of an object being present in that region. By evaluating these scores, the algorithm filters out low-confidence predictions, keeping only those detections that meet a certain threshold. Indexing the remaining predictions helps in organizing and later processing detections (often using non-max suppression) to finalize the output.

6 Back to Original Rotation and Image

Figure 32: Original: Object 1

Figure 33: Original: Object 3

7 Conclusion

Figure 34: Final Result on the Original Image

References

[DPM95] J. M. Brady D. P. Mukherjee, A. Zisserman. Shape from symmetry – detecting and exploiting symmetry in affine images. *Philosophical Transactions of the Royal Society of London*, 1995.

[GAWW93] Paul L. Rosin Geoffrey A. W. West. Using symmetry, ellipses, and perceptual groups for detecting generic surfaces of revolution in 2d images. *Proceedings Volume 1964, Applications of Artificial Intelligence 1993: Machine Vision and Robotics;*, 1993.

[gj] glenn jocher. https://github.com/ultralytics/yolov5/issues/12. Limitation of YOLO.

[Kun02] Rohit Kundu. https://www.v7labs.com/blog/yolo-object-detection. 202.

[Mor23] Jeremy Morgan. Dilation demystified: A complete guide to image dilation with opencv. 2023.

[OPE] OPENCV. https://docs.opencv.org/4.x/index.html. Documentation.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[SZL21] Shuang Song, Jian Zhang, and Zhengming Liu. A survey of self-supervised and few-shot object detection. *arXiv preprint arXiv:2110.14711*, 2021.

[yol22] https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/. 2022.