

Medical waste segregation using ‘Smart Robotic Arm’.

Prepared by:

Sourav Paul

Email: souravpaul.bio@gmail.com

Department : Information Technology - IEST, Shibpur

Arindam Mondal

Email: arindam32083@gmail.com

Department : Information Technology - IEST, Shibpur

Amisha Sadhukhan

Email: sadhukhanamisha@gmail.com

Department : Information Technology - IEST, Shibpur

Under the guidance of

Dr. Sunirmal Khatua

Department of Computer Science & Engineering

University of Calcutta (Technology Campus)

INDEX

Topic	page No
1)Description	3
2)Prerequisites	3-4
3)YOLO Explanations	5-13
4)Hardware Requirements and Connections	14-20
5)Code Explanation	20-25
6)Results	25
7)Extended versions - Modifications	26
8)Conclusions	26-27

1)Description:

The project aims to revolutionise the handling and disposal of medical waste in healthcare facilities by integrating advanced robotic technology. The initiative focuses on the development and deployment of a sophisticated robotic arm equipped with machine learning (ML) capabilities to accurately and efficiently segregate various types of medical waste.

The robotic arm identifies objects in front of it, and applies object detection algorithm to detect the type of object(waste). Once it's done the arm picks the object and drop it in proper place. Thus it segregates all the wastes.

2)Prerequisites

Following are the prerequisites of the project.

1) Python Programming : In Our project we have used python programming Language for object detection tasks and also for the entire system.

2) OpenCV : OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.The library has more than 2500 optimised

algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects. [Reference link](#)

3) YoloV8 : You Only Look Once is a popular algorithm for computer vision application, Useful for Object detection and Image Segmentation. There are many versions of YOLO. In our project we have used YOLOv8.

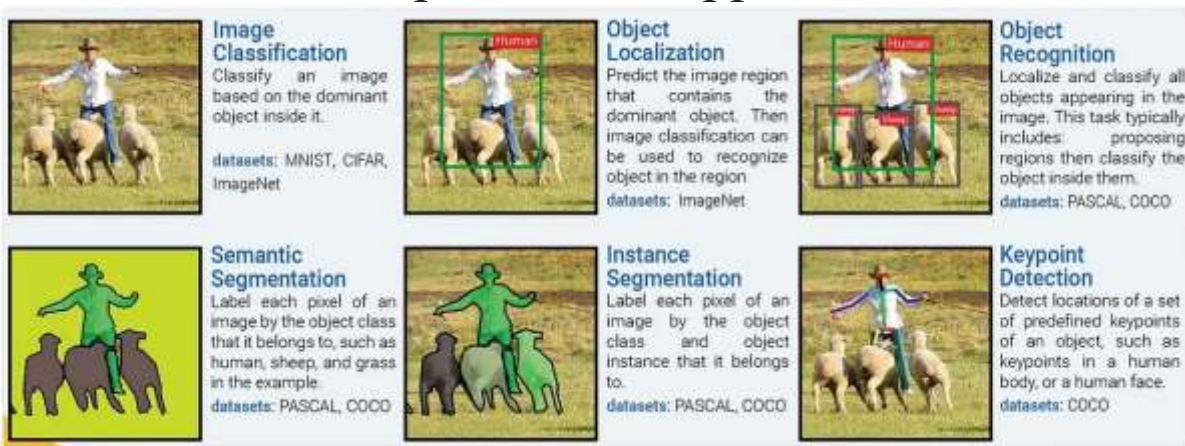
[Reference link \(Official Documentation\):](#)

4) Knowledge of Raspberry pi : Prior knowledge of raspberry pi will help to understand the project well. Although the necessary steps are discussed in section - 4.

3)YoloV8 Explanations

This module explains in details about YOLO algorithm.

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. Nowadays, task **classification, object detection, segmentation, and keypoint detection** are the main real-time computer vision applications.



SOLUTION: YOLO

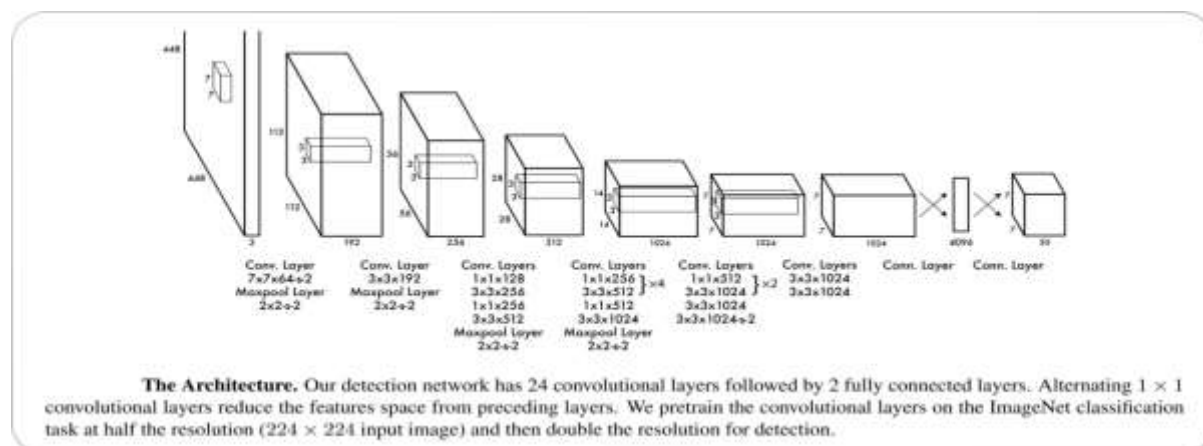
CNN-based Object Detectors are primarily applicable for recommendation systems. YOLO (You Only Look Once) models are used for Object detection with high performance. YOLO divides an image into a grid system, and each grid detects objects within itself. They can be used for real-time object detection based on the data streams. They require very few computational resources. YOLO, developed by Joseph Redmon et al., frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. It looks at the whole image at test time so its

predictions are informed by global context in the image. YOLO is known for its speed, making it suitable for real-time applications.

YOLO (You Only Look Once) models are real-time object detection systems that identify and classify objects in a single pass of the image.

YOLO Architecture

The YOLO algorithm employs a single Convolutional Neural Network (CNN) that divides the image into a grid. Each cell in the grid predicts a certain number of bounding boxes. Along with each bounding box, the cell also predicts a class probability, which indicates the likelihood of a specific object being present in the box. The architecture of YOLOv8 builds upon the previous versions of [YOLO](#) algorithms.



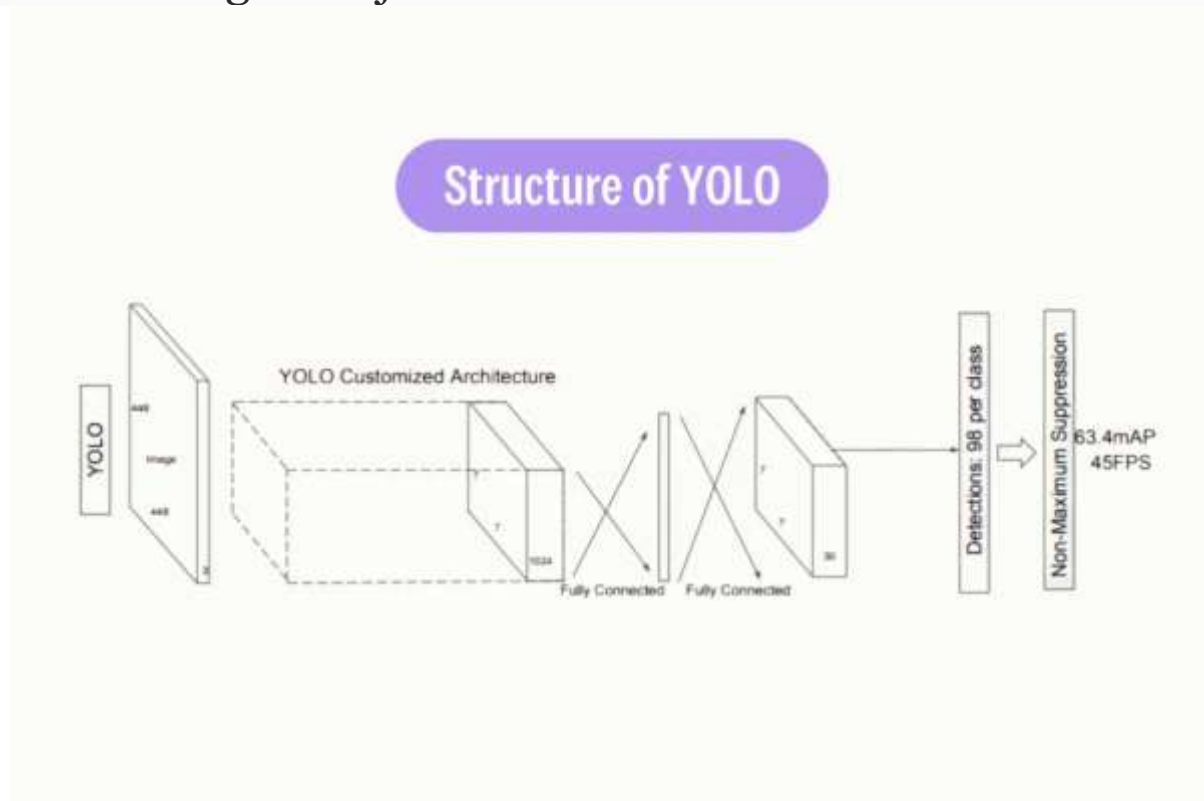
Bounding Box Recognition Process

The bounding box recognition process in YOLO involves the following steps:

- **Grid Creation:** The image is divided into an $S \times S$ grid. Each grid cell is responsible for predicting an object if the object's centre falls within it.
- **Bounding Box Prediction:** Each grid cell predicts B bounding boxes and confidence scores for those boxes. The confidence score reflects how certain the model is that a

box contains an object and how accurate it thinks the box is.

- **Class Probability Prediction:** Each grid cell also predicts C conditional class probabilities (one per class for the potential objects). These probabilities are conditioned on there being an object in the box.



Non-Max Suppression (NMS)

After the bounding boxes and class probabilities are predicted, post-processing steps are applied. One such step is Non-Max Suppression (NMS). NMS helps in reducing the number of overlapping bounding boxes. It works by eliminating bounding boxes that have a high overlap with the box that has the highest confidence score.

Vector Generalization

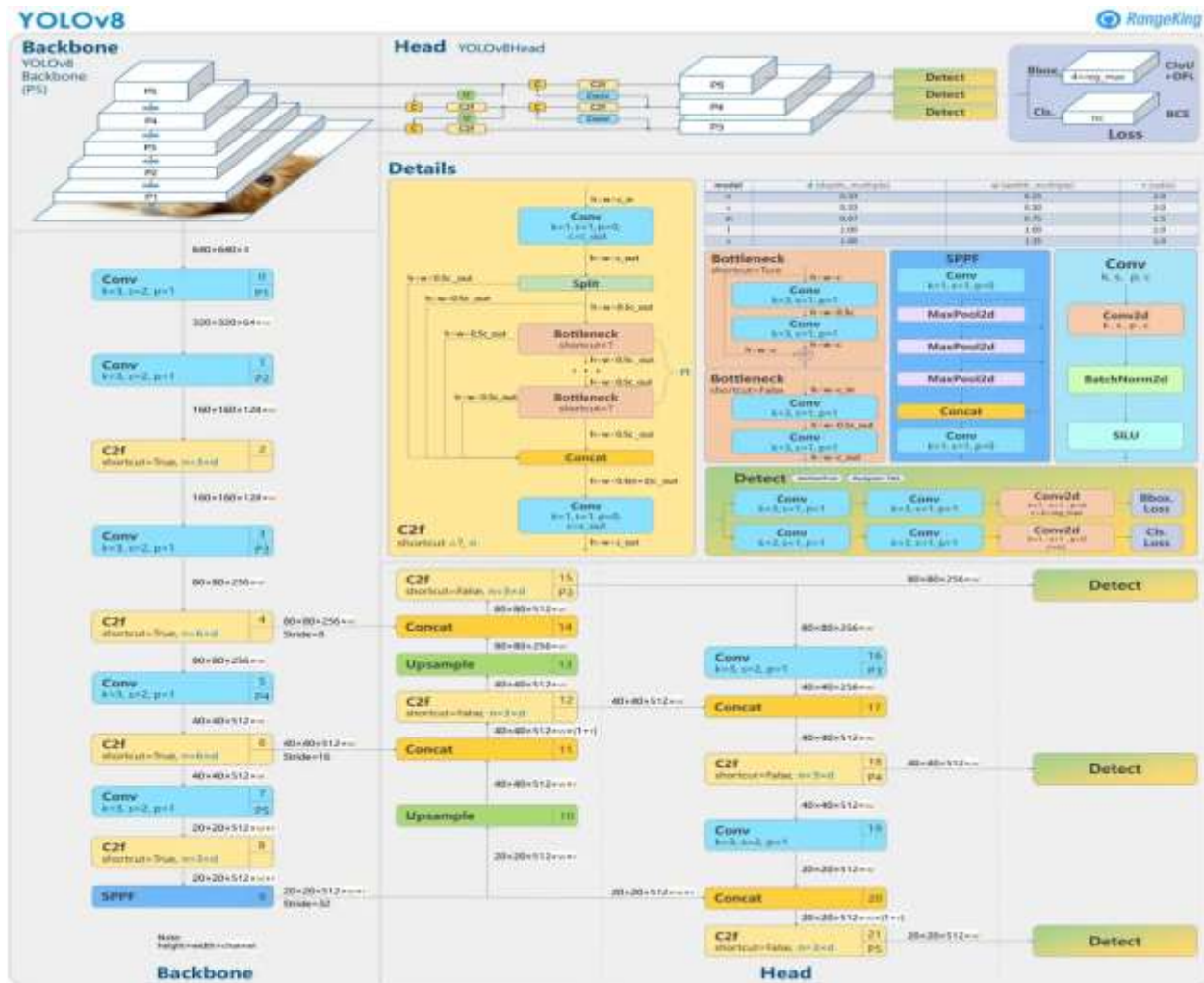
Vector generalisation is a technique used in the YOLO algorithm to handle the high dimensionality of the output. The output of the YOLO algorithm is a tensor that contains the bounding box coordinates, objectness score, and class probabilities. This high-dimensional tensor is flattened into a vector to make it easier to process. The vector is then passed through a softmax function to

convert the class scores into probabilities. The final output is a vector that contains the bounding box coordinates, objectness score, and class probabilities for each grid cell.

One key technique introduced in YOLOv8 is **multi-scale** object detection. This technique allows the model to detect objects of various sizes in an image. Another significant enhancement in YOLOv8 is the use of the ELU activation function. ELU, or Exponential Linear Unit, helps to speed up learning in deep neural networks by mitigating the vanishing gradient problem, leading to faster convergence.

YOLOv8 adopted the GIoU loss. GIoU, or Generalised Intersection over Union, is a more advanced version of the IoU (Intersection over Union) metric that takes into account the shape and size of the bounding boxes, improving the precision of object localization.

The YOLOv8 algorithm shows a 1.2% increase in average precision (AP) compared to the YOLOv7, which is a significant improvement. It has achieved this while reducing the model's weight file size by 80.6 M, making the model more efficient and easier to deploy in resource-constrained environments.



One notable improvement in YOLOv8 is its modular and scalable design. The model is divided into three main components: the backbone, neck, and head, everything will occur in these blocks. This architecture consists of 53 convolutional layers and employs cross-stage partial connections to improve information flow between the different layers.

Backbone:

Function: The backbone, also known as the feature extractor, is responsible for extracting meaningful features from the input.

Activities:

- Captures simple patterns in the initial layers, such as edges and textures.

- Can have multiple scales of representation as you go, capturing features from different levels of abstraction.
- Will provide a rich, hierarchical representation of the input.

Neck:

Function: The neck acts as a bridge between the backbone and the head, performing feature fusion operations and integrating contextual information. Basically the Neck assembles feature pyramids by aggregating feature maps obtained by the Backbone, in other words, the neck collects feature maps from different stages of the backbone.

Activities:

- Perform concatenation or fusion of features of different scales to ensure that the network can detect objects of different sizes.
- Integrates contextual information to improve detection accuracy by considering the broader context of the scene.
- Reduces the spatial resolution and dimensionality of resources to facilitate computation, a fact that increases speed but can also reduce the quality of the model.

Head:

Function: The head is the final part of the network and is responsible for generating the network's outputs, such as bounding boxes and confidence scores for object detection.

Activities:

- Generates bounding boxes associated with possible objects in the image.
- Assigns confidence scores to each bounding box to indicate how likely an object is present.
- Sorts the objects in the bounding boxes according to their categories.

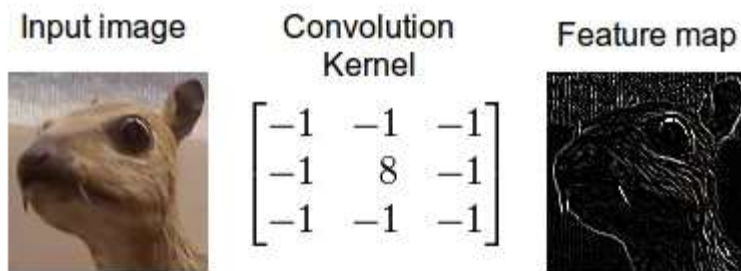
Features of YOLOV8:

- The head of YOLOv8 consists of multiple convolutional layers followed by a series of fully connected layers.
- These layers are responsible for predicting bounding boxes, objectness scores, and class probabilities for the objects detected in an image.
- One of the key features of YOLOv8 is the use of a **self-attention** mechanism in the head of the network.
- This mechanism allows the model to focus on different parts of the image and adjust the importance of different features based on their relevance to the task.
- Another important feature of YOLOv8 is its ability to perform multi-scaled object detection. The model utilises a feature pyramid network to detect objects of different sizes and scales within an image.

Conv:








The YOLO architecture adopts the local feature analysis approach instead of examining the image as a whole, the objective of this strategy is mainly to reduce computational effort and enable real-time detection. To extract feature maps, convolutions are used several times in the algorithm.

Convolution is a mathematical operation that combines two functions to create a third. In computer vision and signal processing, convolution is often used to apply filters to images or signals, highlighting specific patterns. In convolutional neural networks (CNNs), convolution is used to extract features from inputs such as images. Convolutions are structured by Kernels (K), Strides (s) and paddings (p).



Kernel:

The kernel, also known as the filter, is a small array of numbers that is slid across the input (image or signal) during the convolution operation. The goal is to apply local operations to the input to detect specific characteristics. Each element in the kernel represents a weight that is multiplied by the corresponding value in the input during convolution.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Stride:

Stride is the amount of displacement the kernel undergoes as it moves across the input during convolution. A stride of 1 means the kernel moves one position at a time, while a stride of 2 means the kernel skips two positions with each move. Stride directly influences the spatial dimensions of the convolution output. Larger strides can decrease the dimensionality of the output, while smaller strides retain more spatial information. Larger strides reduce computational effort, thereby increasing the speed of the operation, which can directly impact quality.

Padding:

“padding” refers to adding extra pixels around the edges of the input image (typically zeros) before applying convolution operations. This is done to ensure that information at the edges of the image is treated in the same way as information in the centre during convolution operations.

When a filter (kernel) is applied to an image, it typically goes through the image pixel by pixel. If no padding is applied, pixels at the edges of the image have fewer neighbours than pixels in the centre, which can lead to a loss of information in these regions. In figure 4 below, there is an example of padding.

0	0	0	0	0
0	1	4	7	0
0	8	6	2	0
0	1	9	5	0
0	0	0	0	0

INPUT
Padding

In terms of probability, the convolution operation can be understood as a weighted average or a weighted sum of random events, which can be interpreted as a probability distribution with two random variables X and Y with probability distributions $p_X(x)$ and $p_Y(y)$. The convolution of X and Y is given by the equation 1:

$$(X * Y)(z) = \int_{-\infty}^{\infty} p_X(x) \cdot p_Y(z - x) dx \quad (1)$$

4) Hardware Requirements and Connections

a) Steps to use the raspberry pi :

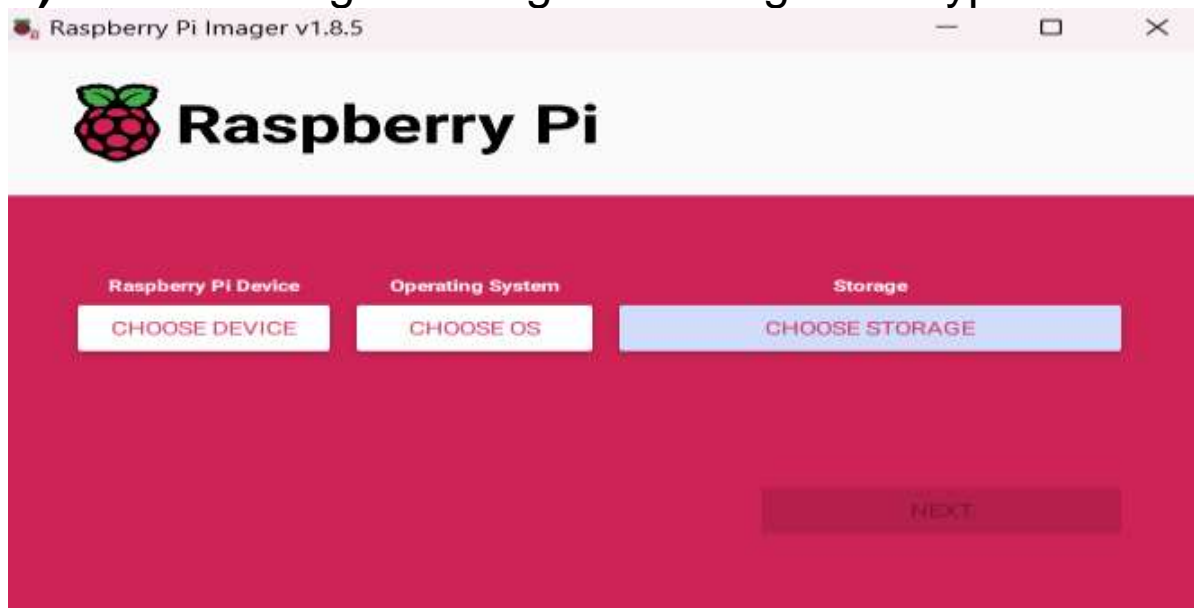
The Raspberry pi has no Operating System previously installed. We have to install it separately.

Components required for OS installation:

- 1) SD card(8 GB-32 GB)
- 2) SD card reader
- 3) Any Desktop or Laptop

1)Download the raspberry pi imager the website link is given here. [Link](#)

2)After installing the imager we will get this type of Interface.



3)CHOOSE DEVICE : Raspberry pi 4 .,as we have used raspberry pi 4 Model B for our project

CHOOSE OS : Raspberry pi OS(64 bit)

CHOOSE STORAGE : Choose the SD card (make sure that the SD card is connected with the laptop or computer via card reader.)

4) Click on next and wait until it's done. Once it's done insert the sd card in raspberry pi. There is a specific sd card slot in the raspberry pi.

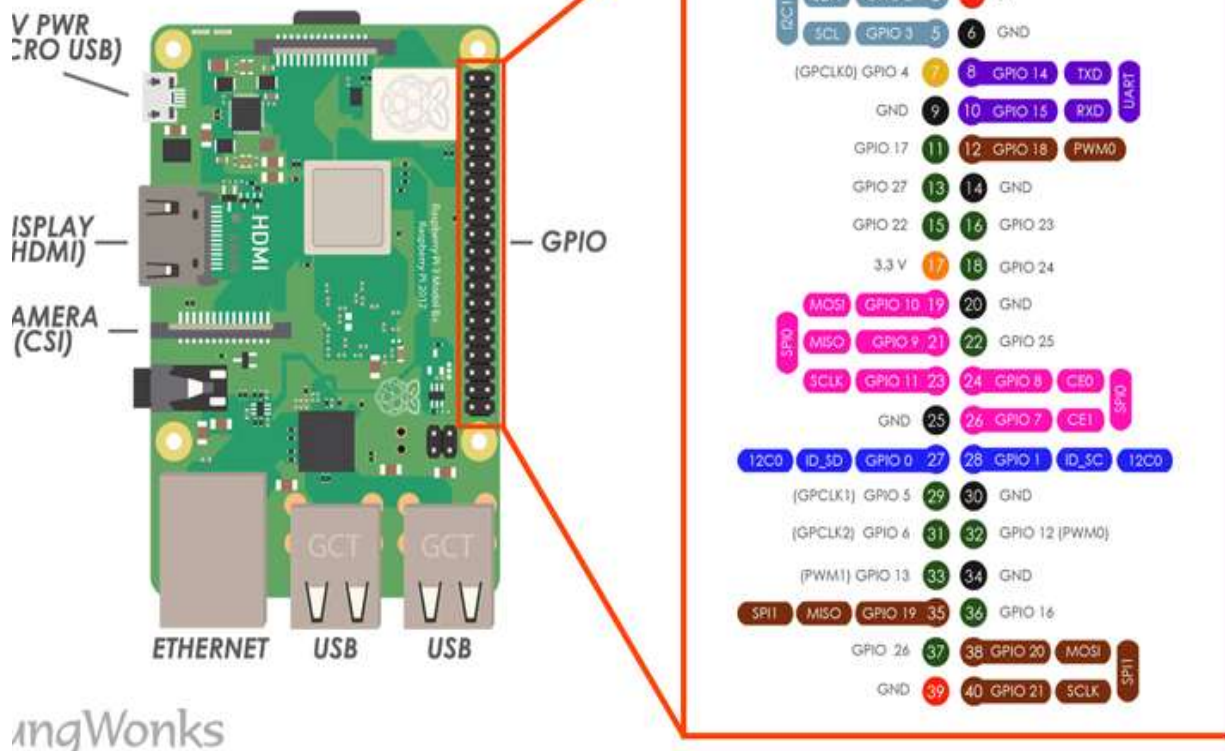
5) Now connect the monitor with the micro HDMI port of raspberry pi. It requires a micro HDMI to HDMI converter and a HDMI to VGA cable.

6)Give the power supply to the raspberry pi (in the type C port) and monitor.

After successful connection the raspberry pi screen will be visible on the monitor.

7)Connect a keyboard and mouse with the USB port of the raspberry pi.

8)Internet : we can use Ethernet with the raspberry pi using an ethernet cable, Ethernet port is there in it. or we can also use wifi.



Pin diagram of raspberry pi-4 model B

b) Servo Connections with raspberry pi

- We used GPIO 17,18,22,23,24,27

The signal cable of each servo is connected with raspberry pi as follows.

GPIO 17 = pin no 9 —> servo -1(Mg995 , base servo).

GPIO 18 = pin no 10 —> servo -2(Mg995 ,2nd servo from base).

GPIO 27 = pin no 11—> servo -3(Mg995 , 3rd servo from base).

GPIO 22 = pin no 15 —> servo -4(Mg90S , 4rd servo from base).

GPIO 23 = pin no 16 —> servo -5(Mg90S , 5th servo from base).

GPIO 24 = pin no 18 —> servo -6(Mg90S , 6th servo from base).

Using breadboard connect the pin-6(GND pin) of raspberry pi and the ground cable of all the servos and also the ground pin of power supply.

connect the positive terminal of power supply with the positive terminal of all servos.

c)How we used YOLO in raspberry pi

(Before using YOLO in raspberry pi directly , its better to use it in our own system(laptop or Desktop))

1)We have to install ultralytics in raspberry pi but directly we cannot install it using pip. To do so , we have to create a virtual Environment .

- use this command in raspberry pi terminal to create the virtual environment

>python -m venv virtualEnvironmentName

- Activate the virtual Environme

>source virtualEnvironmentName /bin/activate

2)Now install ultralytics. Use the following command

>pip install ultralytics

Now we can run YOLO programs in raspberry pi.

Following is a sample python code. In the Imread Function the path of the image is given. This program will identify the objects in the image and will create a bounding box around the object.

```
import cv2
from ultralytics import YOLO
model = YOLO('yolov8n.pt')

img = cv2.imread('/home/roboArm/Desktop/coder/mask4')

result = model(img)

modified = result[0].plot()

cv2.imshow('frame',modified)

cv2.waitKey(10)
cv2.destroyAllWindows()
```

Yolov8n.pt is the pretrained model which will not identify all the objects accurately(it can identify only 60-64 types of objects)

In our project We have done custom training of the pretrained model.which will be discussed later.

-

d)Structure and details of Robotic Arm

A robotic arm is a servo motor controlled mechanical arm which can be controlled by various microcontroller like Arduino, raspberry pi , Esp-32 etc.

We have used the following robotic arm for our project.



It is 6 DOF(degree of freedom) robotic arm.This arm requires 6 servo motors , three of them are MG995 servo motor and the remaining three are MG90s servo motor.

- These servo motors are individually controlled from a microcontroller (In our case it is Raspberry pi).
- Never give power supply from raspberry pi. It will affect the raspberry pi . Always use an external power supply. Constant 6V power supply is good for the servos.
- Connect the signal cable of servos with the GPIO pins of the raspberry pi.
- The Ground pin of All the servos,raspberry pi and the external power supply must be the same. Otherwise servos will not work.

Possible modifications of the robotic arm we used for better performance:

- 1) If possible, powerful servos can be used at the base for the better performance of the arm. like MG958 Servo.
- 2) If possible, use a better claw to hold the object more tightly.

Product Link : [claw for robotic arm](#) , [Mg958 servo](#)

Software Installations for servo control

In the virtual environment, install the following

1) gpiozero and 2) pigpio

run the following command in the virtual environment to install.

> **pip install gpiozero**

> **pip install pigpio**

5) Code Explanation

We focus primarily on two main aspects:

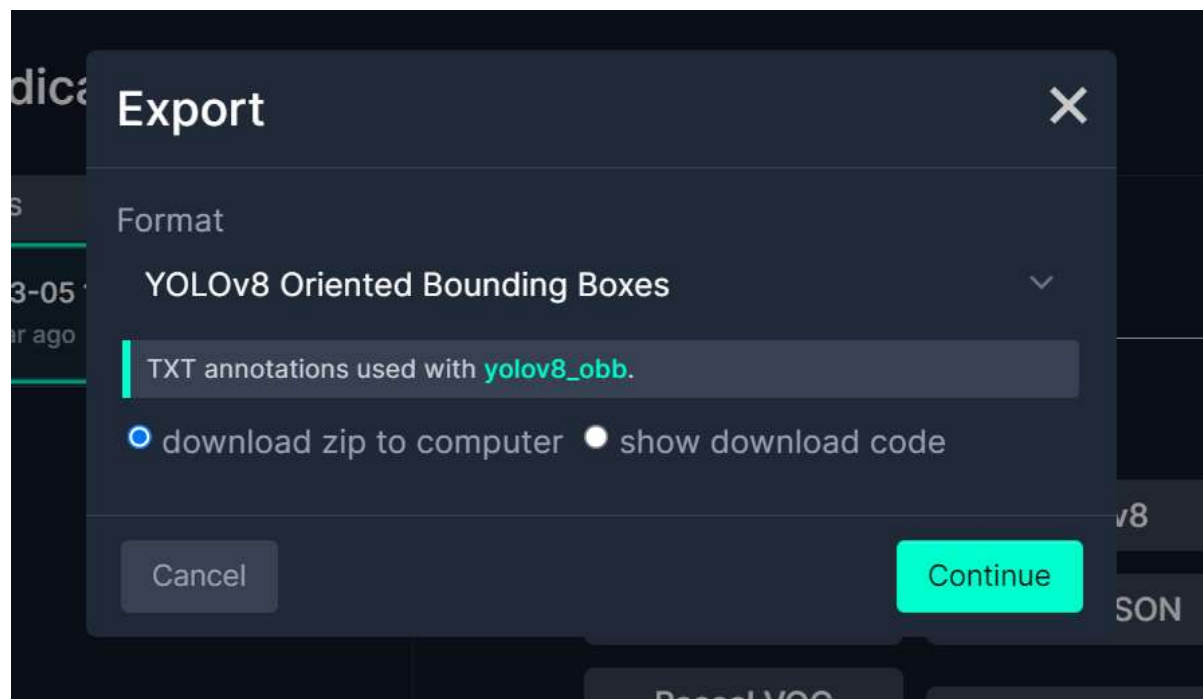
1. **Code for Model Training:** This involves implementing and optimising algorithms to train our YOLOv8 model on custom datasets, ensuring high accuracy and efficiency in object detection.

2. **Code for the Entire System:** This encompasses the development of the complete software stack required to interface with hardware components, integrate various modules, and ensure seamless and robust operation of the entire system.

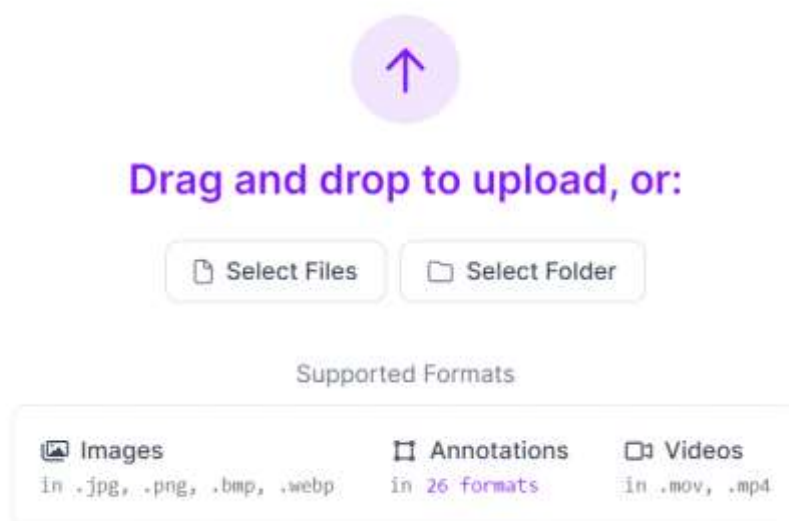
Before taking a deep dive into the coding section, we have to make a dataset using roboflow for custom model training.

Working with Roboflow

We downloaded some datasets on medical waste from Roboflow, which were annotated for our custom dataset. We selected YOLOv8 Oriented Bounding Boxes as the format to download the zip folder to our computer.

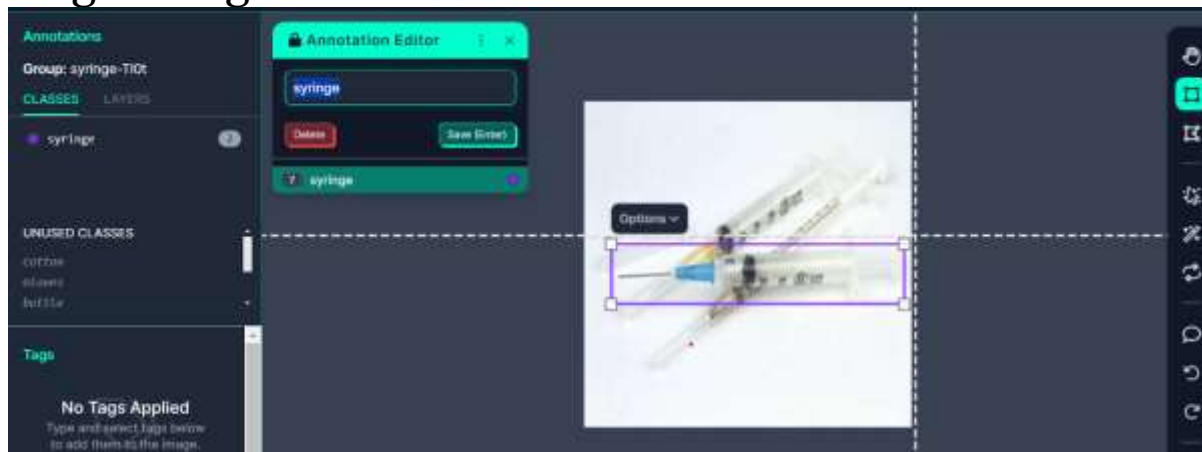


To start annotation we first upload the dataset folder in Roboflow.

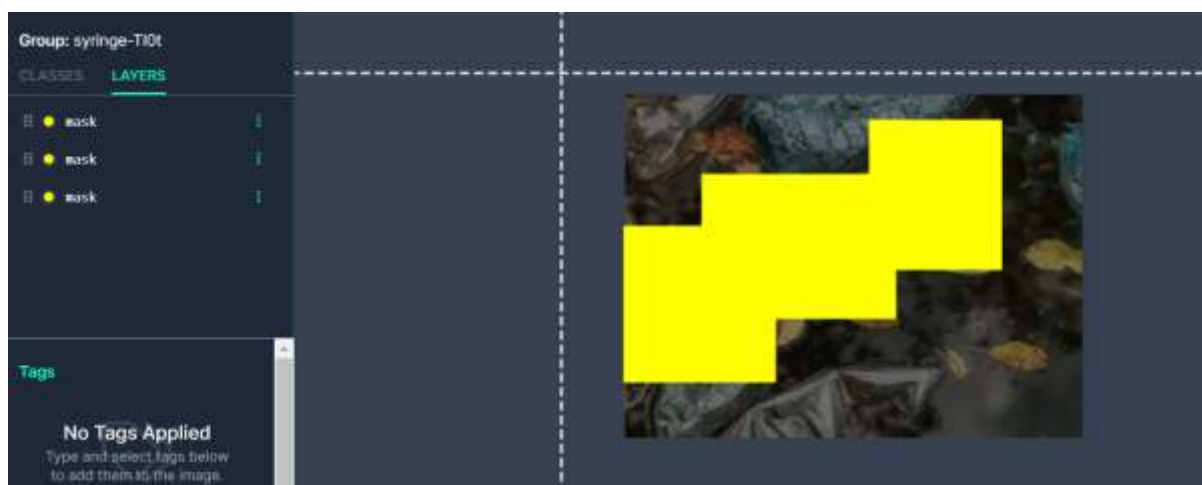


Below is a demonstration of the annotation procedure in Roboflow. After drawing the bounding box we annotate it with the respective class and proceed to the next image. There can be

multiple objects belonging to same or different classes in a single image.



The below image portrays the different layers in a single image representing multiple objects.



Altogether we have annotated 1388 images for our custom dataset.

Dataset

18 Jobs

 [See all 1388 images](#)

Now the next step involves generating the final dataset including all the annotated images in various jobs. The below image describes the process.

Create New Version

VERSIONS

2024-07-23 3:08pm
v9 · 22 minutes ago

3591 640x640

Stretch to

2024-06-27 8:34am
v8 · a month ago

3599 640x640

Stretch to

2024-06-27 7:30am
v7 · a month ago

3598 640x640

Stretch to

2024-06-26 5:21pm
v6 · a month ago

3606 640x640

Stretch to

2024-06-20 6:55am
v5 · a month ago

3602 640x640

Stretch to

2024-06-20 6:54am
v4 · a month ago

3593 640x640

Stretch to

2024-06-19 4:11pm
v3 · a month ago

3607 640x640

Stretch to

Creating New Version

Prepare your images and data for training by compiling them into a version.
Experiment with different configurations to achieve better training results.

✓

Source Images

Images: 1,388

Classes: 7

Unannotated: 0

✓

Train/Test Split

Training Set: 1.1k images

Validation Set: 134 images

Testing Set: 112 images

3

Preprocessing

What can preprocessing do?

Decrease training time and increase performance by applying image transformations to all images in this dataset.

Auto-Orient

Edit

×

Resize

Stretch to 640×640

Edit

×

+

Add Preprocessing Step

Continue

4

Augmentation

5

Create

Just do the continue, continue and you will get a code for directly use the dataset in colab.

1. Custom training of the YOLOv8 model using our specific dataset from Roboflow:

23 | Page


```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="SKoA61nzU2FBi8tqh3wI")
project = rf.workspace("myspace-hav7y").project("dataset1-mtnju")
version = project.version(2)
dataset = version.download("yolov8")

!pip install ultralytics -q

from ultralytics import YOLO
model = YOLO('yolov8x.pt')
model.train(data="/content/Dataset1-2/data.yaml", epochs=20)
```

```
model.train(data="/content/Dataset1-2/data.yaml", epochs=20)
```

`data="/content/Dataset1-2/data.yaml"`: Specifies the path to the dataset configuration file, which contains information about the dataset (e.g., classes, paths to images and annotations).

`epochs=20`: Specifies that the model should be trained for 20 epochs.

```
model.train(data = "/content/Dataset1-2/data.yaml", epochs = 40)
```

Finally we updated the training command to run for 40 epochs instead of 20 which will allow the model to train longer, potentially improving its performance.

2.Code for the Entire System

Processing an image using the trained YOLO model, displaying the detections and printing details about each detected object.

Robotic Arm Control and Image Capture with Raspberry Pi module.

Approach to determine whether any object is present or not : we have taken an image with no object present(A clear

image of the area covered by a raspberry pi camera without placing any object) ,we call it a standard frame. Now every time the camera is taking an image we are comparing the absolute difference of two frames(standard frame and new frame). If there is no object the absolute difference will be below a threshold value. In that case we will simply discard the frame(or image). If the difference between the standard frame and new frame is above the threshold value then surely any object is present, Then we use our model to classify the type of object.

After classification the robotic arm is instructed to pick the object and place it in the proper location.

Code for the two sections are here in this [github repo](#).

Carefully follow the instructions written in the code and run the files in raspberry pi terminal.

6)Results

- The Camera continuously takes images and if an object is present then the type of object will be detected by the raspberry pi .
-
- If any specific class of object is detected by the custom trained model , the robotic arm picks up the object and places it in the proper place(as mentioned in the code).
-
- The robotic arm can only pick up the object from a particular place. and comes back to the initial position.
-
- The camera again continuously takes images.
-
- In the absence of any object , No object detection algorithm is applied on the image, and the robotic arm remains static.

In the following link we attach an [explanation video](#) of the system.

7)Extended Versions - Modifications

1. In our project,the robotic arm can only pick up the object from a fixed position in front of it. As we have used YOLO which can also give us the position of the object in the image in the form of bounding boxes, we may try to modify it in such a way that the robotic arm can pick the object from any place in front of it.
2. The Algorithm takes little time to detect the type of object present. In that time if the object is removed, then the arm should not take any action. More hardware components(sensors) can be used to achieve this.
3. Future improvements of this project could involve enhancing the accuracy of the object detection model, optimising the robotic arm's movements for smoother and faster operation.It will help the system to handle a broader range of objects and more complex tasks.

8)Conclusions

Our entire project focuses on the application of machine learning in the real world. By combining IoT and machine learning, we developed a smart robotic arm capable of segregating medical wastes. This innovative system integrates the powerful YOLOv8 object detection model with a Raspberry Pi-based control mechanism to achieve precise and autonomous sorting of medical waste.

The robotic arm's functionality is enhanced by accurate image capture and classification capabilities, ensuring that various types of medical waste are correctly identified and segregated. This kind of smart robotic arm can be utilised in a variety of purposes beyond medical waste management, such as industrial automation, hazardous material handling, and precision tasks in manufacturing.

This project demonstrates the potential of integrating machine learning with robotics and IoT, paving the way for innovative solutions in various industries and applications.