ACKNOWLEDGEMENT

I would like to take this opportunity to express my gratitude to everyone who has supported me in completing this project report.

First and foremost, I would like to thank my project guide **Dr. Sarika Yadav**, for her invaluable guidance, feedback, and encouragement throughout the project. I would also like to thank my subject teachers who have constantly resolved my domain-specific queries throughout the project. Their expertise and experience have been instrumental in shaping my understanding of the subject and developing my research skills.

I am also grateful to my family and friends for their constant support and motivation, which has kept me going throughout the project. Their belief in me and my abilities has been a source of inspiration and encouragement.

I would like to extend my sincere thanks to everyone who has contributed to this project, directly or indirectly. Without their support, this project would not have been possible.

Table of Contents

INTRODUCTION

a. Background

This project uses a fast real-time object detector to identify and localize various objects present in a live feed through webcam. This will help segregate objects-of-concern from other objects. The real-time detection feature of this detector can also help in surveillance on multiple places at once.

b. Problem definition

In this project, two big worldwide problems are identified, and an attempt is made to propose a solution for them.

Poor waste management contributes to climate change and air pollution, and directly affects many ecosystems and species. Failing to segregate waste properly means that it will end up mixed in landfills. Waste items like food scraps, paper and liquid waste can mix and decompose, releasing run-off into the soil and harmful gas into the atmosphere.

On the other side, Plastic bags cause many minor and major ecological and environmental issues. In 2002, India banned the production of plastic bags below 20 μm in thickness to prevent plastic bags from clogging of the municipal drainage systems and to prevent the cows of India ingesting plastic bags as they confuse it for food. However, enforcement remains a problem. The Ministry of Environment, Forest and Climate Change has also passed regulation to ban all polythene bags less than 50 microns on 18 March 2016. Due to poor implementation of this regulation, regional authorities (states and municipal corporations), have had to implement their own regulation [source: Wikipedia].

c. Motivation

The most general issue with plastic bags is the amount of waste produced. Many plastic bags end up on streets and subsequently pollute major water sources, rivers, and streams. On the other hand, sorting waste makes it easier to understand how to reduce general waste output, identify items that can be reused and set aside items that should be recycled.

THEORETICAL BACKGROUND

a. Deep Learning Basics

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers (at least one input, one output and one hidden layer). These neural networks attempt to simulate the behavior of the human brain of the human brain, allowing it to "learn" from large amount of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. This technology lies behind everyday products and services that improve automation, performing analytical and physical tasks without human intervention.
Products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars) [4].

b. Computer Vision Basics

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.

Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image.

Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyse thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.

Two essential technologies are used to accomplish this: a type of machine learning called deep learning and a convolutional neural network (CNN) [5].

c. Convolutional Neural Networks

There are various types of neural nets, which are used for different use cases and data types. For example, recurrent neural networks are commonly used for natural language processing and speech recognition whereas convolutional neural networks (ConvNets or CNNs) are more often utilized for classification and computer vision tasks. Prior to CNNs, manual, time-consuming feature extraction methods were used to identify objects in images. However, convolutional neural networks now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train models.

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- Convolutional layer
- Pooling layer
- Fully connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colours and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object [6].

REQUIREMENTS ANALYSIS

a. Software requirements

Operating System: Windows 8 or above

Programming Language: Python 3.9

Associated Libraries: OpenCV, NumPy, OpenCV-contrib-python, playsound.

Integrated Development Environment (IDE): PyCharm (a dedicated Python IDE), google Colab (a web IDE for python).

Dataset: 2113 images of plastic bag (513), Tin can (800), Bottle (800) are collected using OIDv4_ToolKit from Google Open Images Dataset.

b. Hardware requirements

Processor: at least intel i5 8th Generation/ Ryzen 5500U

RAM: at least 8GB

I/O devices: webcam

Hard Disk: at least 500GB SSD

c. Algorithm Used

YOLO (You Only Look Once) algorithm is used in this project. It was proposed by R. Joseph in 2015. It was the first one-stage detector in deep learning era [2]. YOLO is extremely fast: a fast version of YOLO runs at 155 fps with VOC07 mAP=52.7%, while its enhanced version runs at 45 fps with VOC07 mAP=63.4% and VOC12 mAP=57.9% [1]. It has taken a different approach, instead of older paradigm of "proposal detection + verification", it applies a single neural network to the full image. Further improvements are made in YOLO, version v2 and v3 were released in subsequent years.

YOLO v3 is preferred in this project. In recent years, newer and better YOLO versions are released by different agencies namely, YOLO v5 in 2020 and YOLO v8 on January 10th, 2023, by *Ultralytics* and YOLOv4 in 2020 and YOLOv7 by Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao.

Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

How does the YOLO algorithm work?

YOLO algorithm works using the following three techniques:

• Residual blocks

• Bounding box regression

• Intersection Over Union (IOU)

Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S. (**Figure 1**.)

S × S grid on input

Bounding box regression

A bounding box is an outline that highlights an object in an image. Every bounding box in the image consists of the following attributes:

• Width (b_w)

• Height (b_h)

• Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.

• Bounding box center (b_x, b_y)

If an object center appears within a certain grid cell, then this cell will be responsible for detecting it. YOLO uses a single bounding box regression to predict the height, width, center, and class of objects.

Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly. Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box. YOLO ensures that the two bounding boxes are equal.

Combination of the three techniques

The three techniques are applied to produce the final detection results. First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object. For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single convolutional neural network. Intersection over union ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly [2,3].

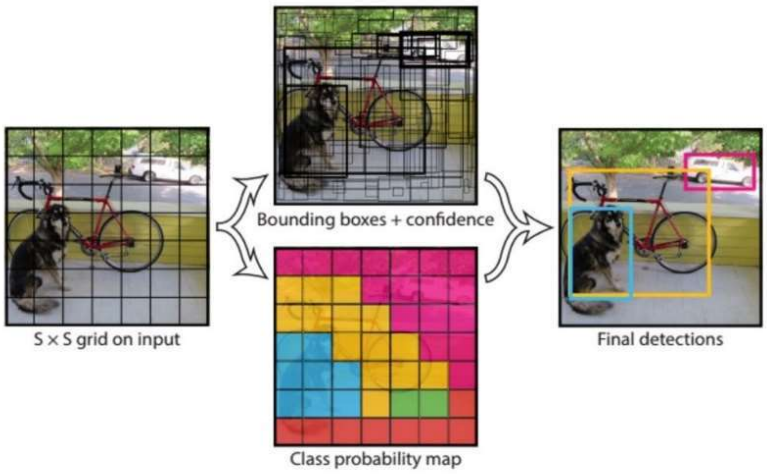

Figure 2. Each S x S grid image cell predicts B bounding boxes, confidence for these boxes, and C class probabilities. These predictions are encoded as an S x S x (B * 5 + C) tensor.

OBJECTIVES

- Adequate amount of data needs to be collected (images in our case). These images should correctly represent the objects which need to be classified.
- Data pre-processing – In our case, creating a text file for each image which stores the details of every instance of object and their spatial positions in YOLO format.
- Model Training – In our case, training will be done online utilizing GPU and Darknet framework, using a Jupyter notebook for different epochs (number of iterations). Weights, at different epochs, will be downloaded and used locally on a personal machine. Out of these weights, best weights will be selected.
- Model evaluation – Model will be evaluated on the basis several metrics and chart provided by Darknet framework and by running several inferences on the local machine.
- Finally, a module will be created which can be used on different machines. This module will save a short clip of approx. 6 seconds whenever it detects an object-of-concern. It will also generate an alert buzzing sound, if it detects an object for some fixed amount of time.

ARCHITECTURE

a. Data Flow Diagram

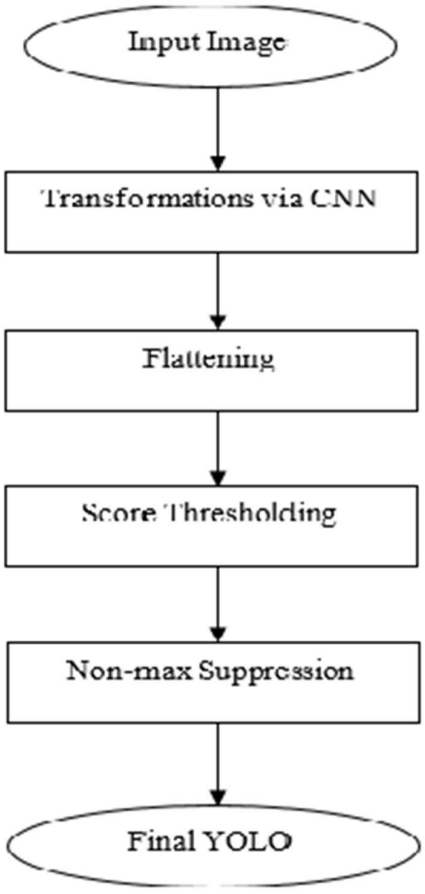

Figure 3. Data flow chart of YOLO algorithm

b. Activity Diagram

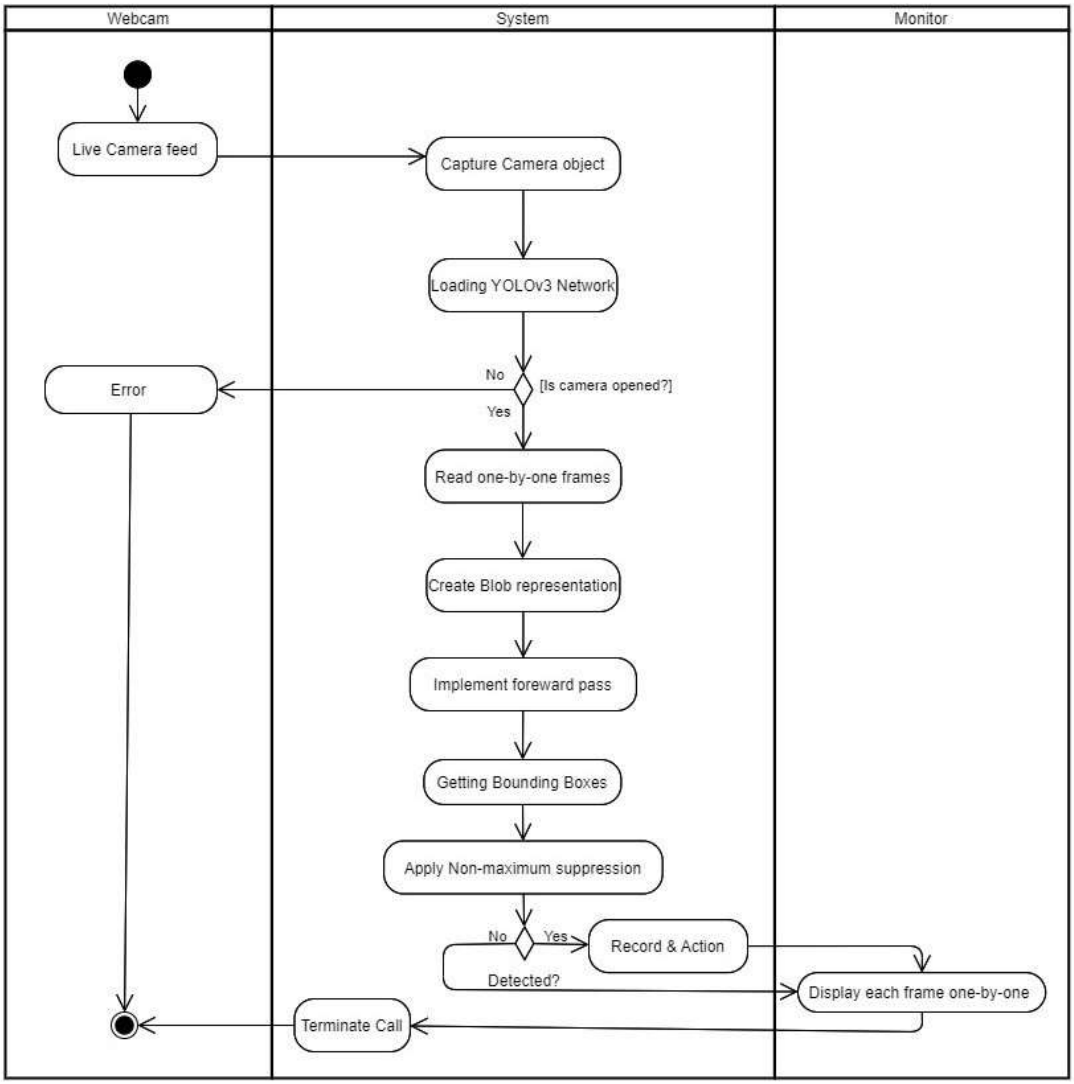

Figure 4. Activity Diagram

IMPLEMENTATION

1. Steps followed

1) Firstly, Images are gathered from Open Images dataset - 513 images of plastic bags, 800 images of bottles, and 800 images of tin cans (Note- Images are of jpg format only).

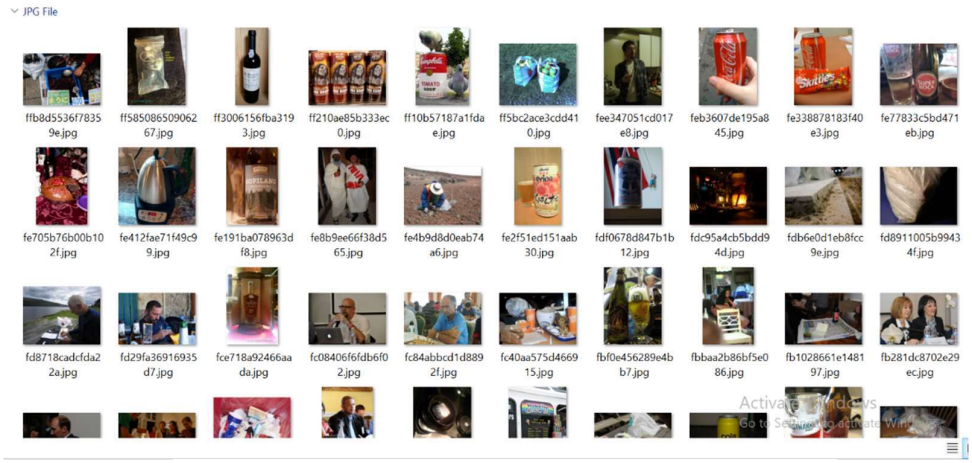

Figure 5. Images collected

2) Preprocessing / Annotation is performed. A text file is generated for each image. These files contain the location(s) of object instances in the images together with their class identities. Files contain these information in YOLO format (class id, object centers (x, y) , object width and object height). These numbers are normalized by real width and height of the images respectively. Text files are generated using a tool – labelImg.

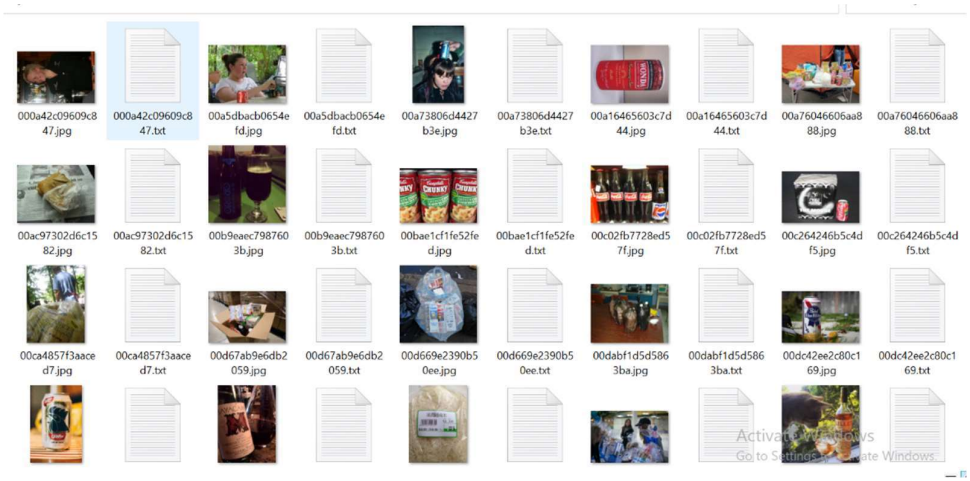

Figure 6. Annotated Images

3) Training is done on Google's colab. Online GPU is utilized to speed up the process. Further, advantage of pre-trained weights are taken and weights are downloaded and tested after every 2000 iterations. Overall, 6000 iterations are performed i.e., approximately 9 hours of training. Darknet framework is utilized for training purpose which is created also created by one of the contributors of YOLO algorithm – Joseph Redmon. This framework serves as a backbone or feature extractor. Images are split into 7:3 ratio for training and validation.

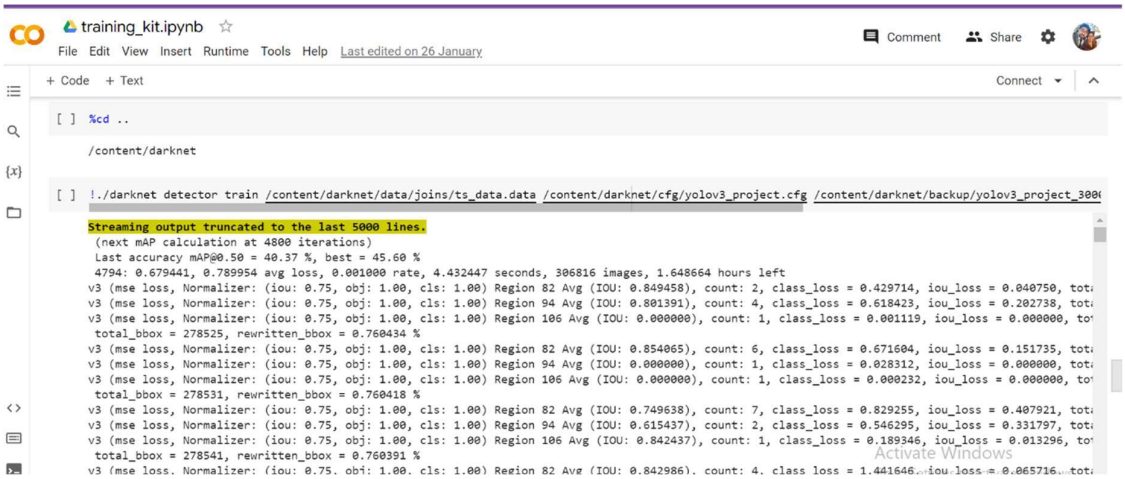

Figure 7. Training

4) Finally, model is evaluated by using charts provided by Darknet framework and tested over some real-time images / feed from webcam.

2. Coding

Jupyter Notebook

```
# Cloning darknet framework
!git clone https://github.com/AlexeyAB/darknet
```

```
# Changing directory
%cd darknet
```

```
# Necessary changes in Makefile to use GPU
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

```
!make
```

```
# Mounting google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Getting original weights
!wget https://pjreddie.com/media/files/darknet53.conv.74
```

```
# Initialize training and saving details on a log file
!./darknet detector train /content/darknet/data/joins/ts_data.data
/content/darknet/cfg/yolov3_project.cfg darknet53.conv.74 -dont_show -map |tee
/content/darknet/log.txt
```

Camera.py

```
# Importing needed libraries
import numpy as np
import cv2
from playsound import playsound

"""
Start of:
Reading stream video from camera
"""

# Defining 'VideoCapture' object
# and reading stream video from camera
camera = cv2.VideoCapture(0)

# Preparing variable for writer
# that we will use to write processed frames
writer = None

# Preparing variable for video count
video_count = 1

# Preparing variable for adding delay -- d_count , recording time --
active_count , locking the writer -- lock
d_count = 0
active_count = 0
lock = True

# Preparing variables for spatial dimensions of the frames
h, w = None, None

"""
End of:
Reading stream video from camera
"""

"""
Start of:
Loading YOLO v3 network
"""

# Loading COCO class labels from file
# Opening file
```

```
with open('..\\data\\plastic\\classes.names') as f:
    # Getting labels reading every line
    # and putting them into the list
    labels = []
    for line in f:
        labels += [line.strip()]

# Loading trained YOLO v3 Objects Detector
# with the help of 'dnn' library from OpenCV

network =
cv2.dnn.readNetFromDarknet('..\\data\\plastic\\yolov3_project.cfg',

'..\\data\\plastic\\yolov3_project_best3.weights')

#  unconnected output layers' names that we need from YOLO v3 algorithm
layers_names_output = ['yolo_82', 'yolo_94', 'yolo_106']

# Setting minimum probability to eliminate weak predictions
probability_minimum = 0.5

# Setting threshold for filtering weak bounding boxes
# with non-maximum suppression
threshold = 0.3

"""
End of:
Loading YOLO v3 network
"""

"""
Start of:
Reading frames in the loop
"""

# Defining loop for catching frames
while camera.isOpened():
    # Capturing frame-by-frame from camera
    _, frame = camera.read()
    # print("frame", frame.shape)

    # Getting spatial dimensions of the frame
    # we do it only once from the very beginning
    # all other frames have the same dimension
    if w is None or h is None:
        # Slicing from tuple only first two elements
        h, w = frame.shape[:2]
        # print(h, w)
    """
    Start of:
    Getting blob from current frame
    """

    # Getting blob from input image

    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
swapRB=True, crop=False)  # blob.shape - tuple =>
    # (no. of images, no. of channels, width, height)
    # print("blob", blob.shape)

    """
```

```
    End of:
    Getting blob from current frame
    """

    """
    Start of:
    Implementing Forward pass
    """

    # Implementing forward pass with our blob and only through output
layers
    network.setInput(blob)  # setting blob as input to the network
    output_from_network = network.forward(layers_names_output)

    """
    End of:
    Implementing Forward pass
    """

    """
    Start of:
    Getting bounding boxes
    """

    # Preparing lists for detected bounding boxes,
    # obtained confidences and class's number
    bounding_boxes = []
    confidences = []
    class_numbers = []
    # Going through all output layers after feed forward pass
    for result in output_from_network:
        # Going through all detections from current output layer
        for detected_objects in result:
            # Getting classes' probabilities for current detected object
            scores = detected_objects[5:]
            # Getting index of the class with the maximum value of
probability
            class_current = np.argmax(scores)
            # Getting value of probability for defined class
            confidence_current = scores[class_current]

            # print(detected_objects.shape)  # (85,)

            # Eliminating weak predictions with minimum probability
            if confidence_current > probability_minimum:
                # Scaling bounding box coordinates to the initial image
size
                box_current = detected_objects[0:4] * np.array([w, h, w,
h])

                # Now, from YOLO data format, we can get top left corner
coordinates
                # that are x_min and y_min
                x_center, y_center, box_width, box_height = box_current
                x_min = int(x_center - (box_width / 2))
                y_min = int(y_center - (box_height / 2))

                # Adding results into prepared lists
                bounding_boxes.append([x_min, y_min, int(box_width),
int(box_height)])
                confidences.append(float(confidence_current))
                class_numbers.append(class_current)
```

```
    """
    End of:
    Getting bounding boxes
    """

    """
    Start of:
    Non-maximum suppression
    """

    # It is needed to make sure that data type of the boxes is 'int'
    # and data type of the confidences is 'float'
    results = cv2.dnn.NMSBoxes(bounding_boxes, confidences,
                               probability_minimum, threshold)
    """
    End of:
    Non-maximum suppression
    """

    """
    Start of:
    Drawing bounding boxes and labels
    """

    # Defining counter for detected objects
    Plastic_bag = 0
    Tin_can = 0
    Bottle = 0

    # Checking if there is at least one detected object
    # after non-maximum suppression
    if len(results) > 0:

        # Going through indexes of results
        for i in results.flatten():

            if int(class_numbers[i]) == 0:
                Plastic_bag += 1
            elif int(class_numbers[i]) == 1:
                Tin_can += 1
            else:
                Bottle += 1

            # Getting current bounding box coordinates,
            # its width and height
            x_min, y_min = bounding_boxes[i][0], bounding_boxes[i][1]
            box_width, box_height = bounding_boxes[i][2],
bounding_boxes[i][3]

            # Drawing bounding box on the original current frame
            cv2.rectangle(frame, (x_min, y_min),
                          (x_min + box_width, y_min + box_height),
                          [0, 255, 0], 2)

            # Preparing text with label and confidence for current
bounding box
            text_box_current = '{}:
{:.2f}%'.format(labels[int(class_numbers[i])],
                                                 confidences[i] * 100)

            # Putting text with label and confidence on the original image
```

```
            cv2.putText(frame, text_box_current, (x_min, y_min - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, [255, 0, 0], 2)
            d_count += 1
            lock = False

    if lock is False:
        active_count += 1
        """
        Start of:
        Writing processed frame into the file
        """
        # Initializing writer
        # we do it only once from the very beginning
        # when we get spatial dimensions of the frames
        if writer is None:
            # Constructing code of the codec
            # to be used in the function VideoWriter
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')

            # Writing current processed frame into the video file
            writer =
cv2.VideoWriter(f'..\\surveillance\\video{video_count}.mp4', fourcc, 8,
                                    (frame.shape[1], frame.shape[0]),
True)

        # Write processed current frame to the file
        writer.write(frame)
        """
        End of:
        Writing processed frame into the file
        """
    print(d_count)

    # Incrementing video count to save next video
    if active_count > 50:
        active_count = 0
        lock = True
        writer = None
        video_count += 1

    if d_count > 50:
        d_count = 0
        # playsound('sound1.wav', block=False)  # playing a warning sound
    """
    End of:
    Drawing bounding boxes and labels
    """

    """
    Start of:
    Displaying count of objects
    """

    # Putting text with counter on the original image
    cv2.putText(frame, f'Plastic bag : {Plastic_bag}', (5, 30),
                cv2.FONT_HERSHEY_COMPLEX, 0.7, [0, 0, 255], 2)
    cv2.putText(frame, f'Tin can : {Tin_can}', (5, 65),
                cv2.FONT_HERSHEY_COMPLEX, 0.7, [0, 0, 255], 2)
    cv2.putText(frame, f'Bottle : {Bottle}', (5, 100),
                cv2.FONT_HERSHEY_COMPLEX, 0.7, [0, 0, 255], 2)

    """
```

```
    End of:
    Displaying count of objects
    """

    """
    Start of:
    Showing processed frames in OpenCV Window
    """

    # Showing results obtained from camera in Real Time

    # Showing current frame with detected objects
    # Giving name to the window with current frame
    # And specifying that window is resizable
    cv2.namedWindow('YOLO v3 Real Time Detections', cv2.WINDOW_NORMAL)
    cv2.imshow('YOLO v3 Real Time Detections', frame)

    # Breaking the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('p'):
        break

    """
    End of:
    Showing processed frames in OpenCV Window
    """

"""
End of:
Reading frames in the loop
"""

# Releasing camera
camera.release()
# Destroying all opened OpenCV windows
cv2.destroyAllWindows()
```

TESTING & FINDINGS

Firstly, Mean Average Precision(mAP) is used to evaluate model performance. The mean of average precision values is calculated over recall values from 0 to 1. It uses other sub metrics such as Confusion Matrix, Intersection over Union or Jaccard Index, Recall, and Precision. These values are computed by the Darknet framework after every 1000 iterations.

Secondly, YOLOv3 uses binary cross-entropy loss for each label and computes total loss to plot a chart of mAP and loss values for each iteration.

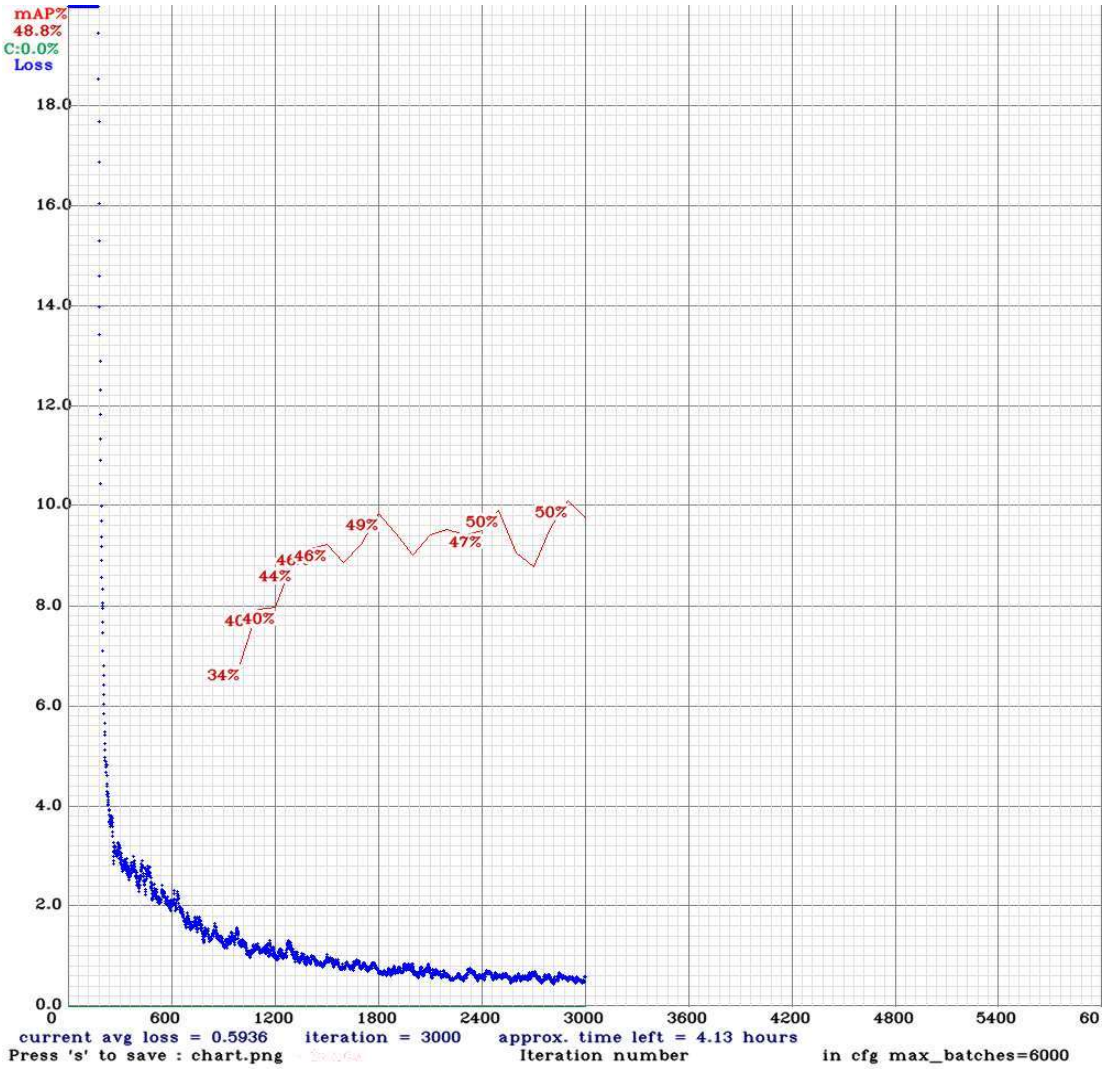

Figure 8. Chart showing loss and mAP after 3000 iterations.

Real-time testing

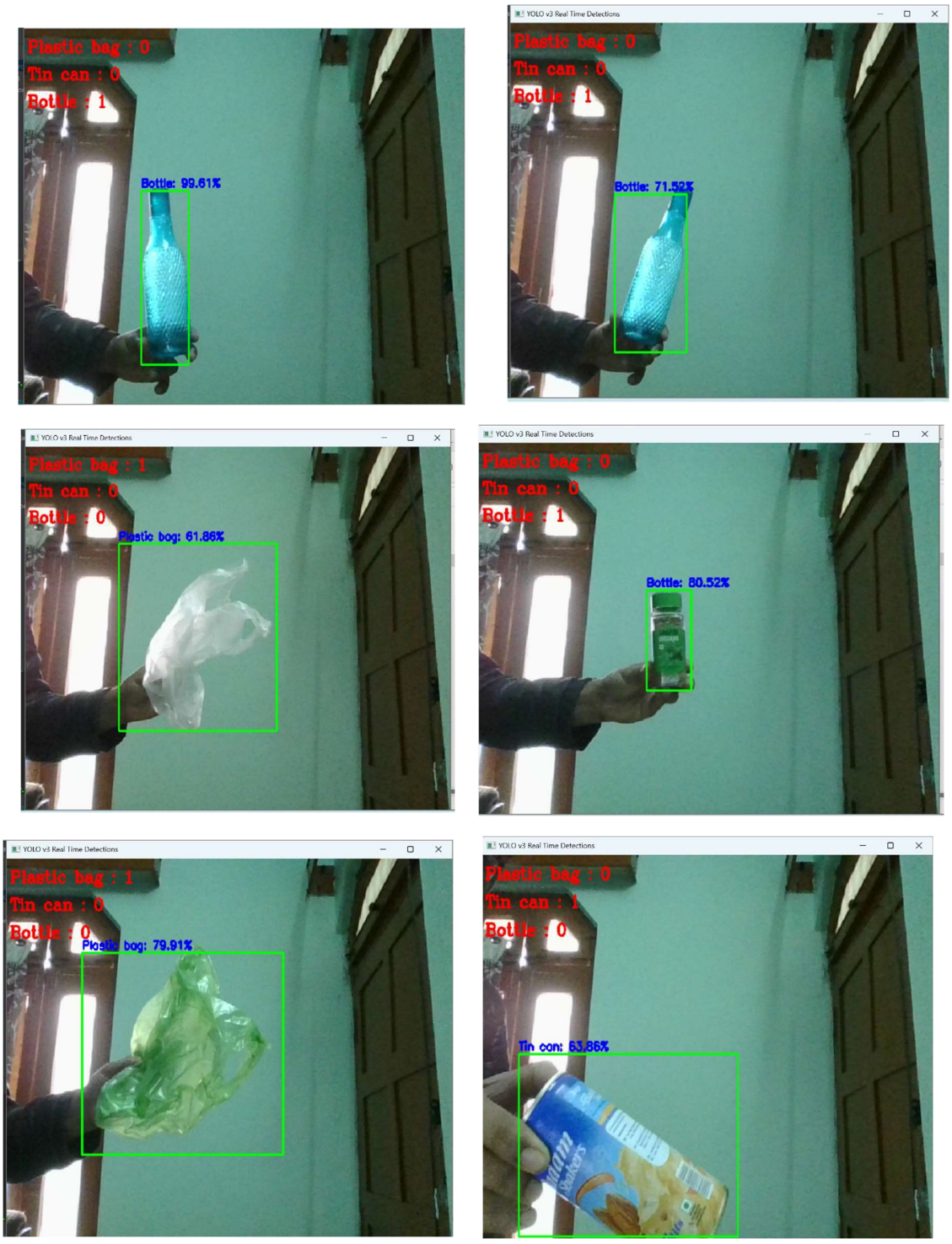

Figure 9. Object detected in real-time (True Positives).

Figure 10. Object detected in real-time (True Negatives).

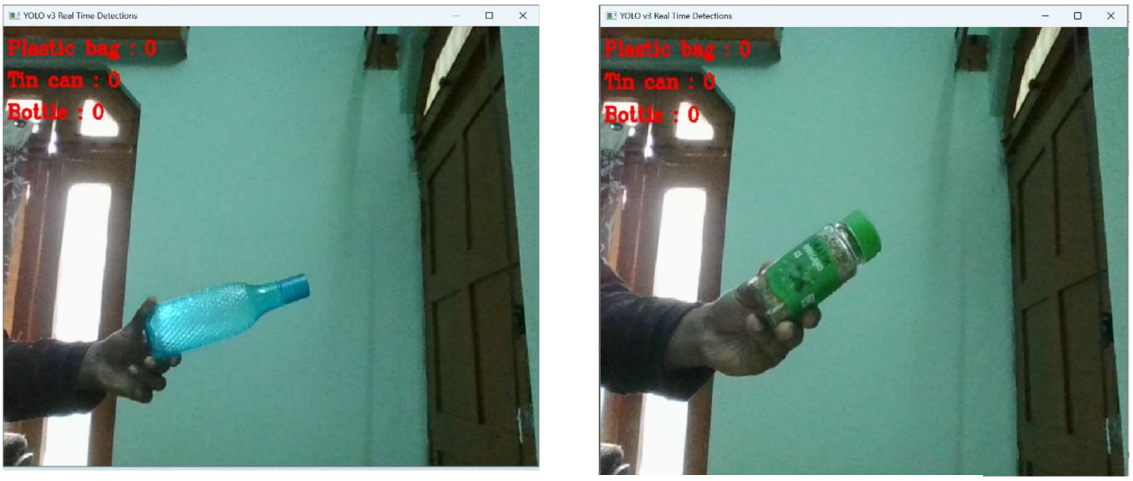

Figure 11. Object detected in real-time (False Negatives).

Figure 11. Object detected in real-time (False Positives).

CONCLUSION

By implementing YOLO algorithm, an efficient object detector is developed which can detect plastic bag, bottle, and tin can. This detection is very fast, and detections are made in almost real-time. As soon as, any object-of-concern is detect, a short clip is recorded and an alert sound is generated the model detect object for some fixed amount of time. This model can be used for implementing tighter ban or surveillance.

Yolo's architecture can be trained on multiple objects; thus, it can be scaled easily without hurting any other elements. Any new set of items / objects can be detected simply by using newly trained weights. However, receptive field and feature resolution will remain the factors of utmost concerns.

Increasing the size of dataset will increase the detection accuracy, but for scale robust detection data augmentation needs to be implemented.

Transfer learning is used to improve generalization ability and training speed but in some cases, divergence between original MS COCO dataset and the dataset used, and domain mismatch may increase false positive or true negative results.

Lastly, fixed size anchor boxes are scale variant leading to misses in case of unexpected changes.

FUTURE SCOPE

To address the problem of waste management and effective sorting of different items from a pool of waste, this detector will be implemented on a robotic hand which can work 24x7 to sort waste into different categories.

Efforts will be made retrospectively towards improving the training dataset quality and quantity. Use of methods like data augmentation, random initialization, and different image input sizes (320x320, 608x608, etc.).

REFERENCES

[1] Zhengxia Zou, Zhenwei Shi, *Member, IEEE,* Yuhong Guo, Jieping Ye, *Senior Member, IEEE,* "*Object detection in 20 years: A Survey*," arXiv preprint arXiv:1905.05055, 2019.

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "*You Only Look Once: Unified, Real-Time Object Detection,*" in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[3] Joseph Redmon, "*Yolov3: An incremental improvement*," arXiv preprint arXiv:1804.02767, 2018.

[4] https://www.ibm.com/in-en/topics/deep-learning

[5] https://www.ibm.com/in-en/topics/computer-vision

[6] https://www.ibm.com/in-en/topics/convolutional-neural-networks

[7] MIT video lectures series on Deep learning https://www.youtube.com/@AAmini

[8] Deep Learning by Simplilearn https://www.youtube.com/@SimplilearnOfficial