ROBUST HUMAN TARGET DETECTION

A Dissertation submitted to the Jawaharlal Nehru Technological University, Hyderabad

in partial fulfillment of the requirement for the award of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

P. AASRIJA (20B81A05J2)

P. ADHITHI (20B81A05J4)

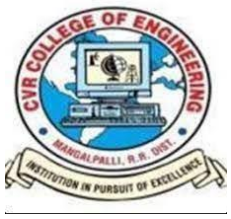
MANDULA SHRAVANI (20B81A05N5)

Under the guidance of

Ms. Sushma Gudivada

Sr. Assistant Professor

Department of CSE



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CVR COLLEGE OF ENGINEERING

(***An Autonomous institution, NBA, NAAC Accredited and Affiliated to JNTUH, Hyderabad***)

Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),

Rangareddy (D), Telangana- 501 510

2023-2024

CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH,
Accredited by NBA, and NAAC)
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),
Ranga Reddy (Dist.) - 501510, Telangana State.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**ROBUST HUMAN TARGET DETECTION**" being submitted by **P.Aasrija (20B81A05J2), P.Adhithi (20B81A05J4), Mandula Shravani (20B81A05N5)** in partial fulfilment for the award of **Bachelor of Technology** in **Computer Science and Engineering** to the CVR College of Engineering, is a record of bonafide work carried out by them under my guidance and supervision during the year 2023-2024.

The results embodied in this project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Project Guide

Ms. Sushma

Senior Assistant Professor

Department of CSE

Professor-in-charge

Dr. S. Suguna Mallika

Department of CSE

External Examiner

Professor and HOD, CSE

Dr.A.Vani Vasthala

DECLARATION

We hereby declare that the project report entitled "**Robust Human Target Detection**" is an original work done and submitted to CSE Department, CVR College of Engineering, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad in partial fulfilment of the requirement for the award of Bachelor of Technology in **Computer Science and Engineering,** and it is a record of bonafide project work carried out by us under the guidance of **Ms. Sushma Gudivada, Senior Assistant Professor, Department of Computer Science and Engineering.**

We further declare that the work reported in this project has not been submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other Institute or University.

_____

Signature of the Student

P.Aasrija

20B81A05J2

_____

Signature of the Student

P.Adhithi

20B81A05J4

_____

Signature of the Student

Mandula Shravani

20B81A05N5

ACKNOWLEDGEMENT

The satisfaction of completing this project would be incomplete without mentioning our gratitude towards all the people who have supported us. Constant guidance and encouragement have been instrumental in the completion of this project.

We offer our sincere gratitude to our project guide **Ms. Sushma Gudivada,** Senior Assistant Professor, Department of CSE, CVR College of Engineering for her immense support and guidance throughout the course of our project work.

We would like to thank the Head of Department, Professor, **Dr. A Vani Vathsala**, for accepting our project.

We are thankful to **Mr. K Giri Babu,** Project Coordinator, Department of CSE, for his supportive guidelines and for having provided the necessary help for carrying forward this project without any obstacles and hindrances.

We also thank the **Project Review Committee Members** for their valuable suggestions.

ABSTRACT

"Robust human target detection" refers to the process of identifying and tracking humans accurately and reliably in various environments, conditions and applications. This idea addresses the problem of detecting and tracking human targets in outdoor environments, even when the targets are partially visible. The system would be built using OpenCV, YOLOv3, and Python and typically use a combination of computer vision and machine learning techniques to detect and track human targets. Human target detection is a critical task in various applications, including surveillance, autonomous driving, and human-computer interaction. This abstract explores techniques and challenges in achieving robust human target detection. Traditional methods primarily rely on features such as colour, texture and shape. However, these approaches often struggle with variations in illumination, occlusions and complex backgrounds. To address these challenges, recent advancements incorporate deep learning models, which excel in learning discriminative features from an image or a video.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

YOLO- You Look Only Once

NMS- Non-Maximum Suppression

OS- Operating System

GUI- Graphical User Interface

CNN- Convolutional Neural Network

OpenCV- Open-source Computer Vision

R-CNN- Region-based Convolutional Neural Network

CHAPTER 1

INTRODUCTION

1.1 Motivation

The motivation behind implementing a robust human target detection serves several important purposes. One of the primary motivations for implementing human target detection is to enhance public safety and security. By accurately detecting and tracking human targets in various environments, such as crowded public spaces or sensitive areas, authorities can better monitor and respond to potential threats or security breaches. Human target detection is crucial for surveillance and monitoring applications. It enables organizations to monitor areas of interest, such as border crossings, airports, or public events, to ensure compliance with regulations, identify suspicious activities, and maintain public order. In emergency situations, such as natural disasters or accidents, human target detection systems can aid search and rescue operations by quickly identifying and locating individuals in need of assistance, even in challenging environments or low visibility conditions. Implementing human target detection is essential for the development of autonomous systems and robotics. It enables robots and autonomous vehicles to perceive and interact with humans in their environment safely and effectively, facilitating applications such as robotic assistance, navigation, and human-robot collaboration.

1.2 Problem Statement

Human target detection is a critical task in various applications, including surveillance, autonomous driving, and human-computer interaction. This abstract explores techniques and challenges in achieving robust human target detection. Traditional methods primarily rely on features such as colour, texture and shape. However, these approaches often struggle with variations in illumination, occlusions, and complex backgrounds. To address these challenges, recent advancements incorporate deep learning models, which excel in learning discriminative features from an image or a video.

1.3 Project Objectives

- **High Accuracy:** develop a detection system capable of achieving high accuracy in identifying human targets within images or video streams, minimising false positives and false negatives.
- **Robustness to Environmental Variations:** ensure the detection system performs consistently across divorce environmental conditions, including variations in lighting, weather and scene complexity, by employing techniques such as data augmentation and domain adaptation.
- **Real-time performance:** design the system to operate in real time or near real time to enable rapid detection and response in dynamic environments, optimising computational efficiency without sacrificing accuracy.

1.4 Project Report Organization

The report is organised as follows:

• The second chapter discusses about any existing works and their limitations in this domain of the project.

• The third chapter specifies the software and hardware requirements for developing the application.

• The fourth chapter describes the concepts or methods used to build this application and the technology used to implement them. This chapter also includes various UML diagrams to provide a way to visualize the design of the application.

• It is followed by the fifth chapter with implementation details and also discusses in detail about the various steps involved in the project development process.

• The final chapter concludes the report by examining the future scope of the application and possible enhancements that can be done in the future.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing work

A literature survey on existing work of robust human target detection reveals that several studies have been conducted to develop and improve such systems. Here are some key findings:

Features: Most human detection systems have features such as implementing human detection algorithms optimized for deployment on embedded systems with limited computational resources and human detection algorithms are tailored for integration into autonomous vehicles and advanced driver-assistance systems (ADAS), prioritizing accuracy, speed, and real-world deployment.

User Interface: User interface considerations may include minimalistic visualization to accommodate resource constraints of embedded platforms. Typically, visualization of detected human targets is done using OpenCV, which provides a user-friendly interface for displaying video streams with bounding boxes around detected humans.

Integration: Integration with lightweight deep learning architectures and optimization techniques for efficient inference on embedded hardware. Integration with other perception modules such as object tracking and motion prediction to enhance situational awareness and decision-making capabilities before taking any action.

Cloud-based systems: With the increase in cloud computing, many faculty management systems are moving to cloud-based systems. Cloud-based systems provide scalability, flexibility, and accessibility, making them an attractive option for universities and colleges.

Mobile Apps: Many faculty management systems have mobile applications that provide access to system features on the go. These mobile applications are designed to help faculty members manage academic activities anytime, anywhere.

Machine Learning and AI: Several studies have explored the use of deep learning and artificial intelligence in human detection systems. Lightweight deep learning architectures such as Tiny YOLO or MobileNet are utilized for human detection on resource-constrained embedded systems. These models are designed to balance accuracy and computational efficiency, enabling real-time inference. AI techniques are employed to optimize deep learning

models for deployment on embedded platforms, ensuring efficient execution without compromising detection performance.

2.2 Limitations of Existing work

While existing work on human target detection has made significant advancements, there are several limitations and challenges that researchers continue to address. Some common limitations include:

Performance in Challenging Conditions: existing models may struggle to maintain high performance under challenging conditions such as low lighting, occlusions, cluttered backgrounds or varying viewpoints. Detection accuracy may degrade when humans are partially obscured or when there are complex interactions with other objects in the scene.

Generalization to Diverse Environments: models trained on specific data sets or environments generalisation to diverse or unseen environments. They may exhibit reduced performance When deployed in real world settings the differ significantly from the training data, leading to poor detection accuracy and reliability.

Robustness to Adversarial Attacks: many deep learning models are suspectable to adversarial attacks, carefully crafted perturbations to input data can cause misclassification of false positives/negatives. Ensuring robustness against such attacks remains a challenging problem, especially in security critical applications.

Limited Dataset Diversity and Bias: human detection models trained on biased or limited data sets may exhibit biases in their predictions, leading to disparities in performance across demographic groups underrepresented scenarios. Ensuring data set diversity and fairness is crucial to mitigate these biases and improve model generalisation.

Computational Complexity and Resource Requirements: deep learning models are often regarded as black boxes, making it challenging to interpret the decisions or understand the factors influencing detection outcomes. Lack of interpretability may hinder trust and acceptance of these systems in critical applications set as law enforcement or healthcare.

Real-time Performance and Latency: while many human detection systems aim for real time performance, achieving low latency and high throughput without compromising accuracy remains a challenge, especially in applications requiring rapid decision making or response times.

Integration with Other Systems: integrating human detection systems with other components of larger systems, such as robotics, autonomous vehicles, or smart environments, post interoperability

challenges. Ensuring seamless integration and compatibility with existing frameworks and platforms is essential for practical deployment.

CHAPTER 3

SOFTWARE AND HARDWARE SPECIFICATIONS

3.1 Software Requirements

For a robust human target detection project built using YOLOv3, Python, OpenCV, and CNN algorithms, along with the specified libraries and packages (numpy, argparse, time, cv2, os), the following software requirements are necessary:

- **Python:** The project requires Python (v3.10.1), the programming language used for the implementation of this project to develop the code base and run the scripts.
- **OpenCV:** OpenCV (Open-Source Computer Vision Library) is essential for image and video processing tasks, including loading images, performing operations such as resizing and normalisation, and displaying results.
- **YOLOv3 model:** The YOLOv3 model architecture and pretend weights and necessary for object detection, including detecting human targets in images or video streams.
- **Numpy:** NumPy is a fundamental package first scientific computing in Python, providing support for array operations and numerical computations, which are commonly used in deep learning applications.
- **Argparse:** The argparse module is used for passing command line arguments allowing users to specify input parameters such as input image or video file paths, output directories, or model configurations.
- **Time:** the time module is utilised for measuring execution time and performance metrics, which may be useful for profiling and optimising the detection system.
- **CV2(OpenCV):** OpenCV (cv2) is an essential library for computer vision tasks, providing various functionalities for image and video processing, feature extraction, object detection, and more.
- **OS:** The OS module is used for interacting with the operating system, such as reading directories, accessing files, and managing paths, which may be necessary for loading data sets or saving detection results.

3.1.1 Functional Requirements:

- Loading Dataset
- Data cleaning
- Data Transformation
- Data Visualization
- Choosing Algorithm
- Create Model
- Train Dataset
- Validate Dataset
- Record Accuracy
- Detect

3.1.2 Non-Functional Requirements:

Describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Non-Functional requirements include quantitative constraints, such as response time (i.e. how fast the system reacts to user commands.) or accuracy (.e. how precise are the systems numerical answers.).

- Portability
- Reliability
- Usability
- Time Constraints
- Error messages
- Space Constraints
- Performance
- Standards
- Ethics
- Interoperability
- Scalability

3.2 Hardware Requirements

The specified hardware an operating system requirement (Intel Core i3 or above processor, 4 GB RAM or higher, Windows 11 OS) contribute to the robustness and performance of the human target detection system in several ways:

- **Processing power:** An intel Core i3 processor or above provides sufficient computational power to execute the detection algorithms efficiently. Human target detection, especially when using the planning models like YOLOv3, can we computationally intensive. A capable processor ensures that the system can handle complex computations in real-time on near-real-time, enabling swift direction of human targets.
- **Memory:** The minimum requirement of 4 GB RAM or higher ensures that the system has enough memory to load and process large images or video data sets. During inference, the model an intermediate result are stored in memory, and having an adequate amount of RAM prevents memory-related bottlenecks and allows for smoother execution.
- **Operating System:** Windows 11 OS provides a stable and modern platform for running the human target detection system. It offers compatibility with wide range of tools and libraries commonly used in computer vision and deep learning applications, including Python, OpenCV, and TensorFlow. Additionally, Windows 11 may offer optimizations and improvements in system performance and resource management, enhancing the overall efficiency of the detection system.

CHAPTER 4
PROPOSED SYSTEM DESIGN

4.0 Proposed Methods

The proposed method for robust human target detection, leveraging YOLOv3, Python, OpenCV, and CNN algorithms, along with the specified libraries and packages, can be outlined as follows:

- **Data Preprocessing:**

1. Load input images or video frames using OpenCV (cv2).
2. Normalize the input data to ensure consistent pixel values across different images.
3. Resize the images to the required input size for YOLOv3.

- **Model Initialization:**

1. Load the YOLOv3 model architecture and pre-trained weights using OpenCV (cv2.dnn.readNet).
2. Configure the model parameters, such as confidence threshold and NMS threshold.

- **Object detection:**

1. Pass the pre-processed images through the Yolo V3 model for object detection.
2. Utilise the forward pass functionality of OpenCV's DNN model (net.forward) to obtain building box coordinates and conference scores for detected objects, including humans.

- **Post-processing:**

1. Apply non-maximum suppression (NMS) to remove redundant bounding boxes with overlapping detections.
2. Filter out detections with confidence scores below certain threshold to improve detection precision.
3. Optionally, perform additional post processing steps such as bounding box refinement or size filtering.

- **Visualization:**

1. Overlay bounding boxes and class labels on the input images to visualize detected human targets.
2. Use OpenCV (cv2.rectangle, cv2.putText) to draw bounding boxes and labels on the images.

- **Performance Evaluation:**

1. Measure the execution time of the detection process using the time module to evaluate performance.

2. Optionally, compute additional metrics such as precision, recall, and accuracy for performance assessment.

- **Integration and Deployment:**

1. Integrate the detection pipeline into a larger application or system for real-world deployment.
2. Ensure compatibility and interoperability with other components of the system.
3. Handle exceptions and errors gracefully to ensure robustness and reliability in deployment scenarios.

- **Documentation and Testing:**

1. Document the implementation details, including code structure, parameter configurations, and usage instructions.
2. Conduct through testing to validate the functionality and performance of the human target detection system under various conditions and scenarios.

By following this proposed method, leveraging the specified tools and libraries, developers can build a robust human target detection system capable of accurately detecting and localising humans in images or video streams in real time or near real time. Additionally, the flexibility of Python and the extensive capabilities of OpenCV enable developers to customise and extend the functionality of the system to meet specific application requirements.

4.1 Use Case Diagram

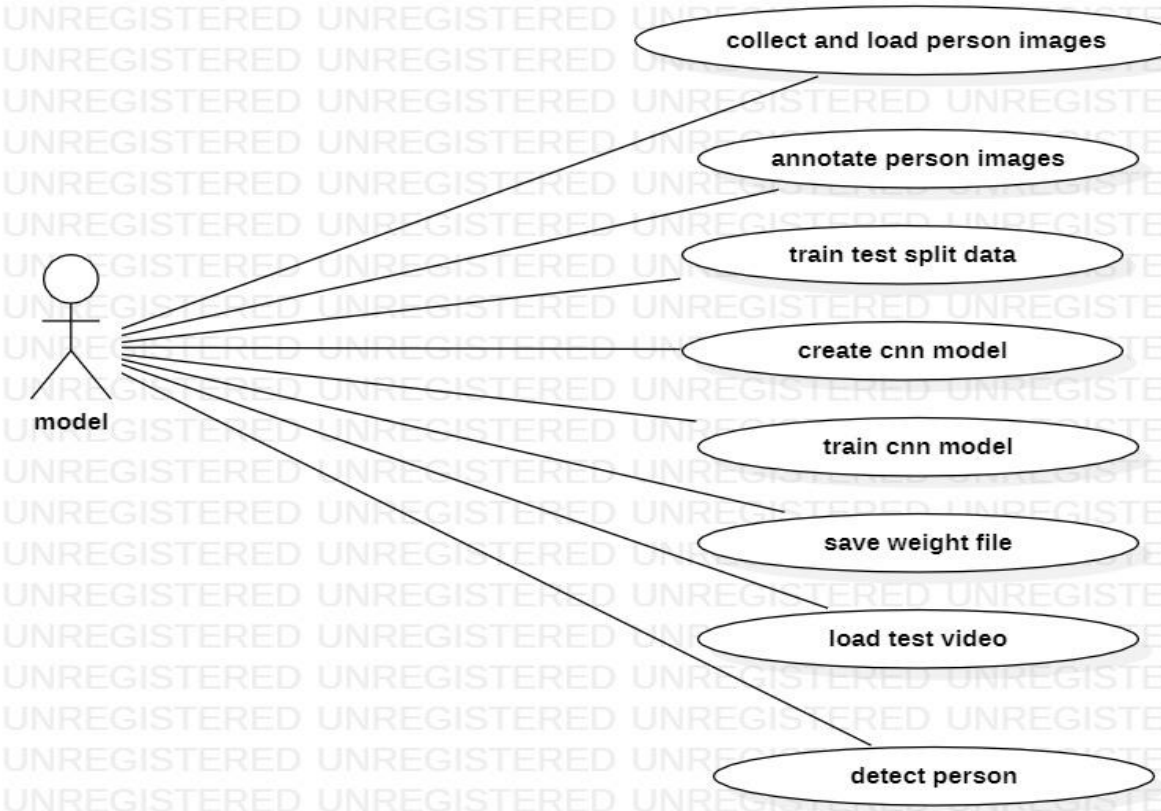


Figure 4.1 Use Case Diagram

From the above Use Case Diagram, you can clearly see that everything is done by the model. After collecting and loading the person images or videos, the model annotates the person images using a software called labelImg and does the bounding boxes only to the humans in an image. After this, the model has to train test split data. Later by creating cnn model, we train the cnn model, save the yolo weight files and load the test video and detect the person.

4.2 Activity Diagram

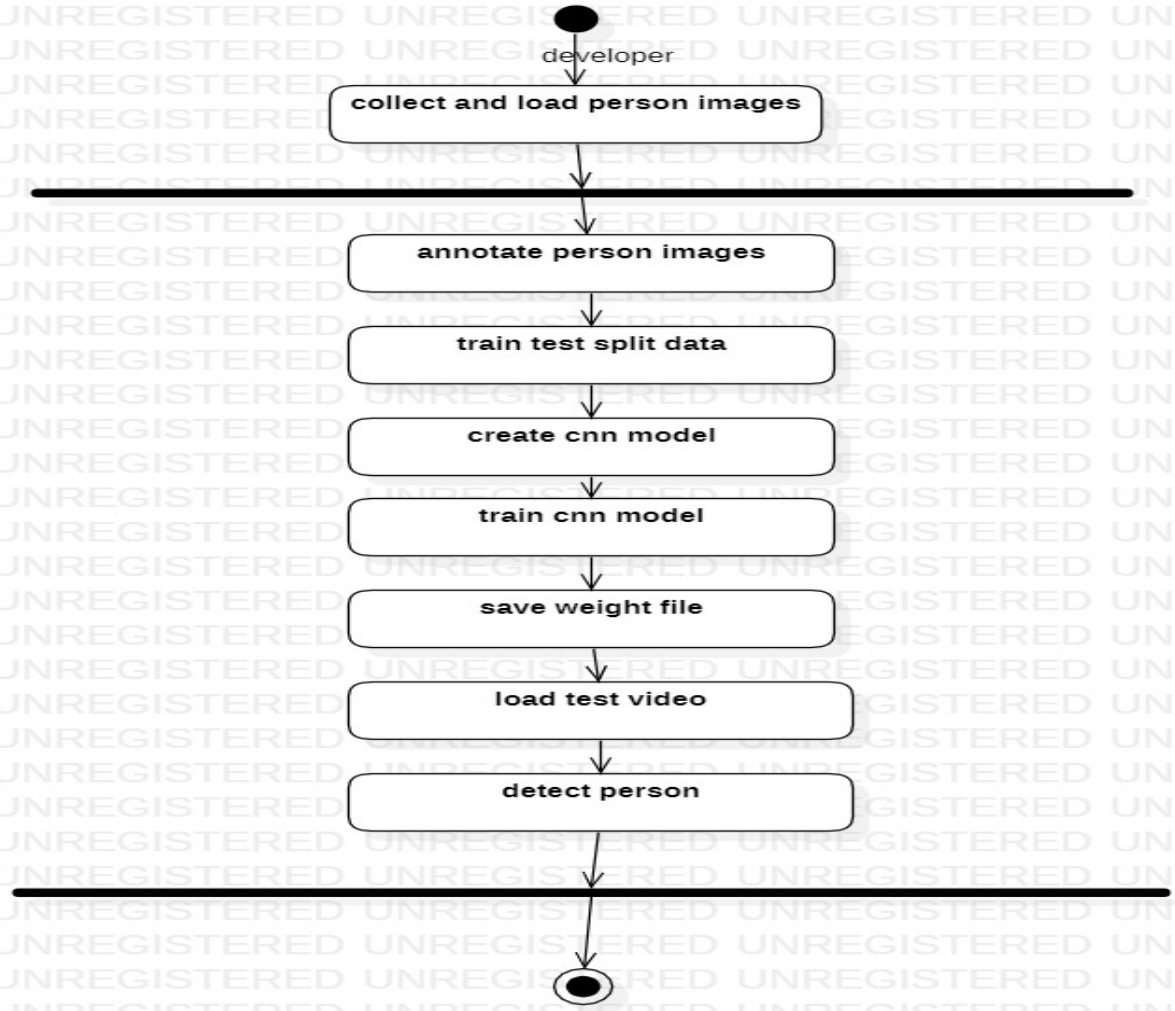


Figure 4.2 Activity Diagram

Through this activity diagram, we get to know that the developer does every activity right from collection of images or videos, training the model and detecting the person. The first activity that takes place is collecting and loading of person images, annotates those images, train and test split the data. Then comes the main activity, creation of a cnn model, training the model and saving the weight files, loading the test video and detecting the person.

4.3 Class Diagram

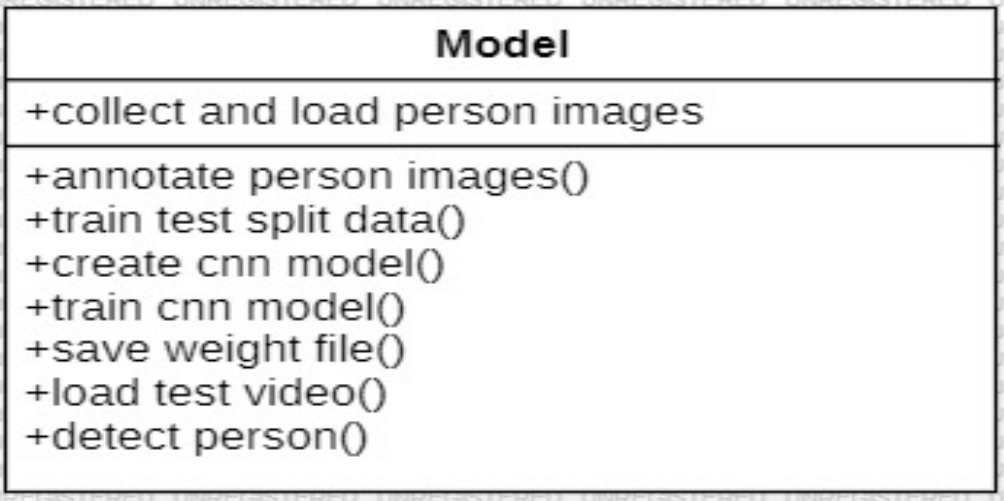


Figure 4.3 Class Diagram

From the above class diagram, it is clear that after collecting and loading person images, the other operations takes place. The first and foremost operation that must be done for detecting a person is annotating person images, train test split data and creating the cnn model. After the creation of the model, we train the model then save the weight files to load the test video and the main action takes place that is detecting the person in any environmental conditions.

4.4 Sequence Diagram

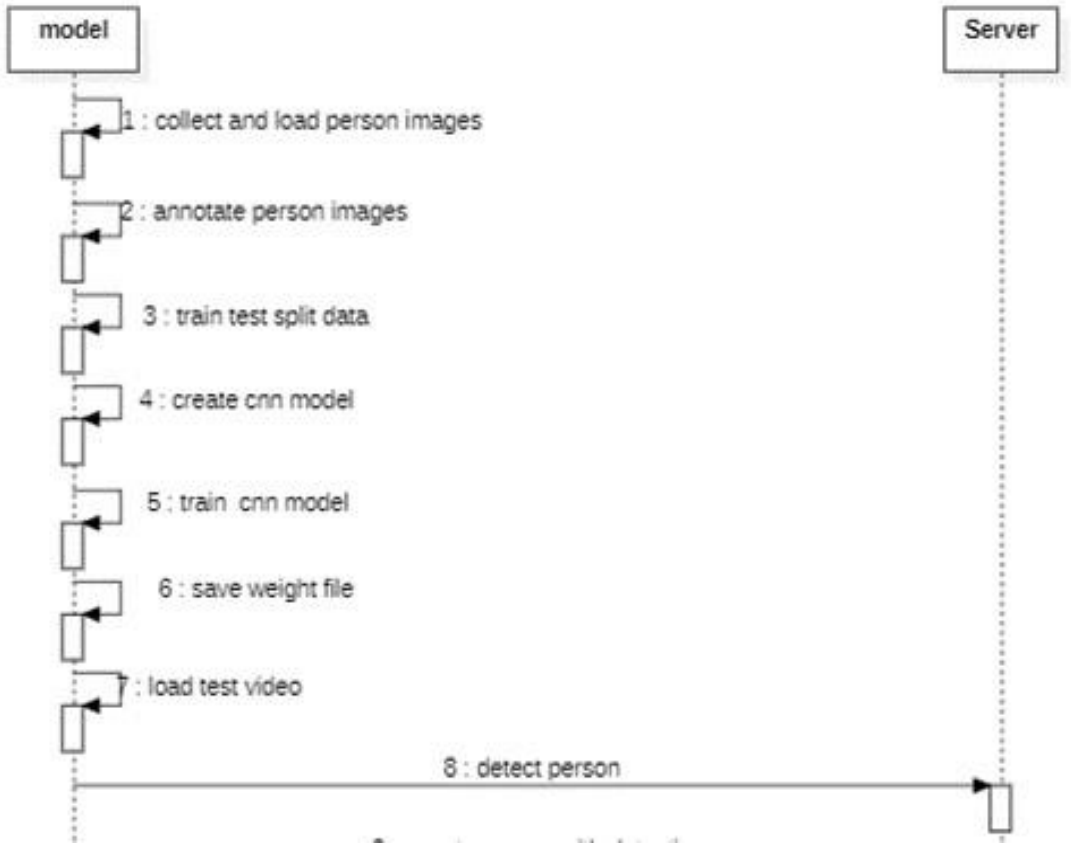


Figure 4.4 Sequence Diagram

In the above diagram, the model does everything so the model itself self-messages itself that it has to collect and load person images and then annotate them. Train and test split the data and create a cnn model that operates using CNN algorithm. Training of the model happens and weight files gets saved and loads the test video and detects the person which is then known to the user.

4.5 System Architecture

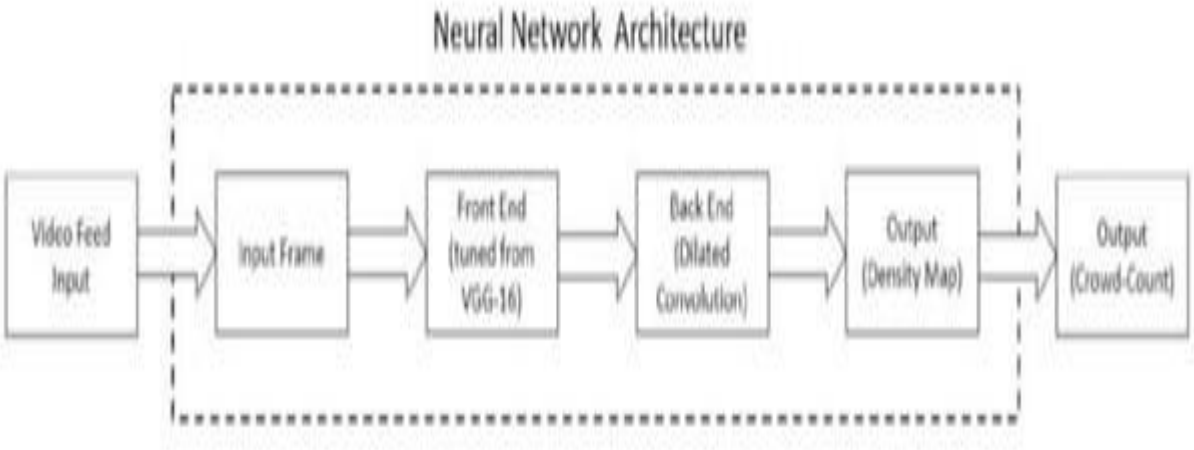


Figure 4.5 Architecture of the system

The system architecture of a robust human target action system built using a Convolutional Neural Network (CNN) deep learning algorithm typically consist of several interconnected components designed to process input data, extract features, perform detection, and provide output results. Below is a high-level overview of the system architecture:

- **Input Module:**

- **Input Data:** The system receives input data, which can be images, video frames, or streaming video.
- **Preprocessing:** Input data undergoes preprocessing steps such as resizing, normalization, and format conversion to prepare it for input to the CNN model.

- **Feature Extraction Module:**

- **CNN Layers:** The core of the system is a CNN model responsible for extracting meaningful features from the input data.
- **Convolutional Layers:** Convolutional layers extract hierarchical features from the input data through convolutions.
- **Pooling layers:** Pooling layers down sample feature maps, reducing spatial dimensions while preserving important features.

- **Activation Functions:** Non-linear activation functions such as ReLU (Rectified Linear Unit) introduce non-reality into the feature extraction process.
- **Normalization and Regularization:** Techniques like batch normalization and dropout are applied to improve training stability and generalization.

➢ **Detection Module:**

- **Output Layers:** The output layers of the CNN model generate predictions for object detection, including bounding box coordinates and confidence scores.
- **Non-Maximum Suppression (NMS):** Post-processing techniques like NMS are applied to suppress redundant detections and refine the final set of detected objects.
- **Thresholding:** Predictions with confidence scores below a certain threshold are discarded to improve detection precison.

➢ **Output Module:**

- **Visualization:** Detected objects, including human targets, are overlaid on the input images or video frames.
- **Bounding boxes:** Bounding boxes are drawn around detected objects to indicate their locations.
- **Class Labels:** Class labels (e.g., "person") are associated with each detected object to provide semantic information.
- **Output Format:** The final output may be visualized on a graphical user interface (GUI), saved to disk, or transmitted to other systems for further processing or action.

➢ **Integration and Deployment:**

- **System Integration:** The human target detection system may be integrated with other components or systems, such as surveillance systems, robotics platforms, or autonomous vehicles.
- **Real-Time Processing:** Considerations for real-time or near-real-time processing are addressed to ensure timely detection and response.

- **Training and Evaluation:**

- **Dataset Preparation:** A labelled dataset containing images or video data with annotations for human targets is used for training.
- **Training process:** The CNN model is trained using supervised learning techniques on the labelled dataset to detect human targets.
- **Performance Evaluation:** The trained model is evaluated on a separate validation dataset to assess its performance in terms of accuracy, precision, recall, and other metrics.

4.6 Technology and Description

Robust human target detection relies on various technologies and methodologies to accurately identify and localise human subjects in images or video streams. Below are some key technologies and their descriptions commonly used in robust human target detection systems:

- **Deep Learning:**
- **Description:** deep learning techniques, particularly convolutional neural networks (CNNs), have revolutionised object detection tasks, including human target detection. CNNs Are capable of automatically learning hierarchical features from raw data, enabling highly accurate and robust detection.
- **Role in Human Target Detection:** deep learning models, such as YOLO (You Only Look Once), Faster R-CNN, are commonly used for human target detection due to their ability to capture intricate patterns and variations in human appearances across diverse environments.
- **YOLO (You Only Look Once):**
  - **Description:** YOLO is a state-of-the-art real time object detection system that divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously. It is known for its speed and accuracy in detecting objects, including humans.
  - **Role in Human Target Detection:** YOLO models, such as YOLOv3 and YOLOv4, are Frequently employed in human target detection systems due to their real time performance and high accuracy, making them suitable for applications requiring fast and reliable detection.

- **OpenCV (Open-Source Computer Vision Library):**

- **Description:** OpenCV is an open-source computer vision library with a wide range of functionalities for image and video processing, including image manipulation, feature detection, object tracking and object detection.
- **Role in Human Target Detection:** OpenCV provides essential tools and functions for implementing human target detection systems, such as image loading, preprocessing, visualization an integration with deep learning frameworks for inference.

- **Convolutional Neural Networks (CNNs):**

- **Description:** CNNs are a class of deep neural networks designed to process structured grid-like data, such as images. They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers which learn hierarchical representations of the input data.
- **Role in Human Target Detection:** CNNs server as the backbone of many human target detection systems, where they extract discriminative features from input images or video frames to identify and localize human subjects with high accuracy.

- **Non-Maximum Suppression (NMS):**

- **Description:** NMS is post processing technique you should to remove attendant bounding box detections by suppressing boxes with lower confidence cores that significantly overlap with boxes having higher confidence scores.
- **Role in Human Target Detection:** NMS is Commonly applied after the detection step to refine the set of detected human targets, improving detection precision and reducing false positives.

CHAPTER 5

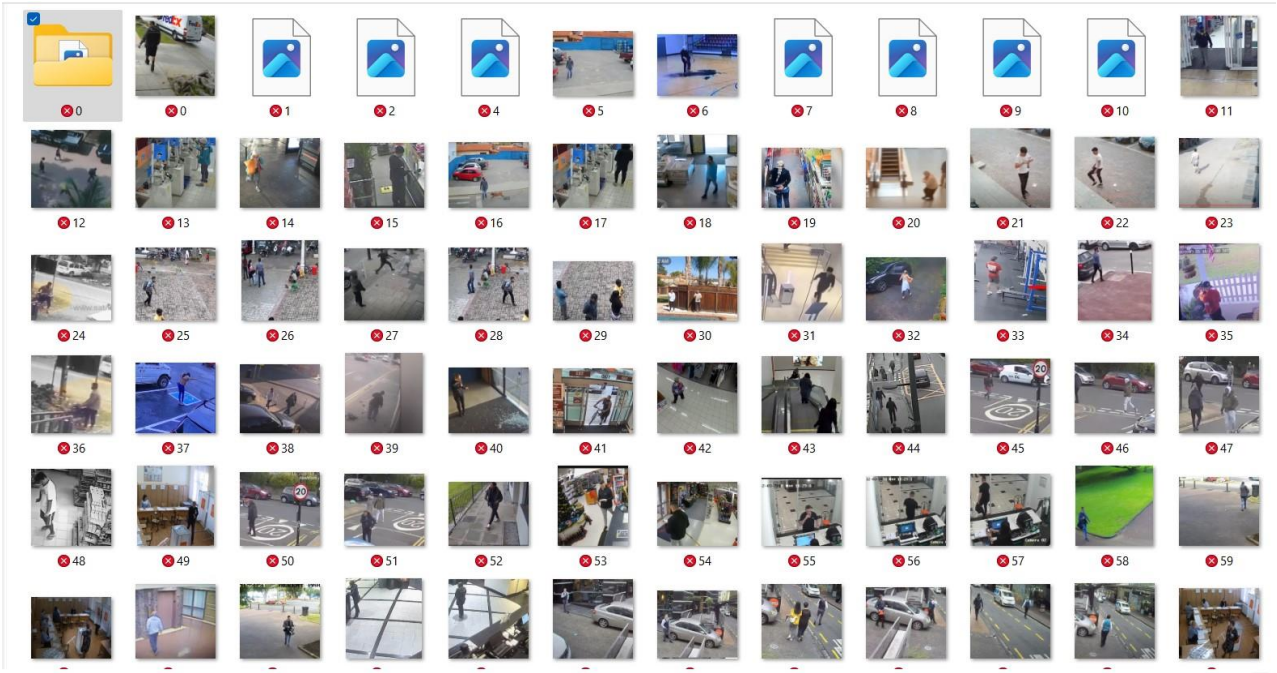
IMPLEMENTATION AND TESTING

5.1 Dataset Creation

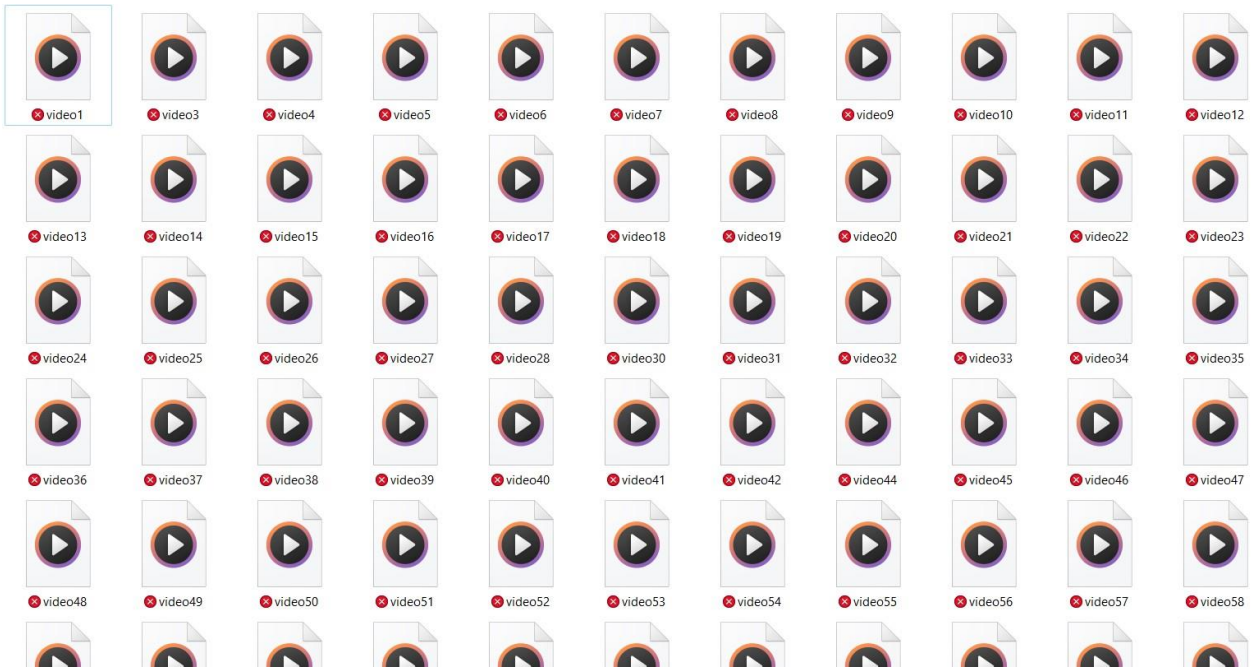
We have collected over 1000 human related images. These hand gesture images are collected from the following Kaggle datasets:

- Human Detection Dataset
- DCASCASS Dataset

The following screenshot consists of images that includes humans and various objects.



The dataset encompasses a wide range of scenes and environments where human targets may appear, including indoor and outdoor settings, urban and rural landscapes, and various lighting conditions (daytime, nighttime, low-light conditions).



The above screenshot contains videos related to robust human target detection and the video consists of humans, various objects and humans and their surroundings. Each video in the dataset should be meticulously annotated to indicate the presence and location of human targets within each frame or sequence. Annotations typically include bounding boxes around detected humans and corresponding class labels. Additionally, temporal annotations may be included to track human movements across frames. However, the dataset should also include videos of varying qualities to simulate real-world conditions, where low-resolution or noisy imagery may be encountered.

5.2 Implementation:

Here we divided the whole things into two parts. First will be covering setup of YOLOv3 for our problem next will be our applied yolov3.

YOLOv3 Setup: Joseph Redmon et al. introduced you look only once also known as YOLO in 2015. Later some improvements came into them and YOLOv2 and YOLOv3 introduced respectively in 2016, 2018. Now, YOLOv3 is the state of art object detection model followed by other versions of YOLO and YOLOv2. It's been given amazing results regarding object classification and detection. In previous version of Yolov2 Darknet-19 is used as a feature extractor. In yolov3 it changes with some improvements, and they called it as darknet-53. Darknet is a framework for training neural network that written in c language which performs better in these tasks. Before working with this architecture some steps we need to mention YOLOv3 configuration at first create a obj.names" files which contains the name of the classes which model wanted to detect. Then a obj.data file which contains number of classes in here it is 2, train data directory, validation data, "obj.names" and weights path which gonna save on backup folder. Last a cfg file contains 2 classes. Next, we change batch size as 64 and subdivisions as 16. For three yolo block set class as 2 and the previous convolution block set filter size at 21. Max batches for our case is 4000 which is calculated as number of class x 2000".

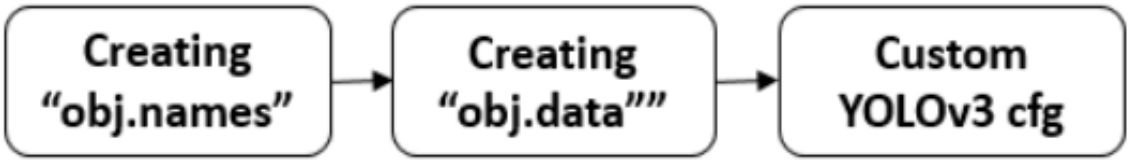


Figure 5.2 YOLO setup process

Applied YOLOv3: An input here is an image is passed into the YOLOv3 model. This object detector is going through the image and find the coordinates that present in an image. It's basically dividing the input into a grid and from that gird it'll analyses the target objects features. From the neighbouring cells that features were detected with high confidence rate are add at one place for produce model output. first image shows how actually model divided image as a grid. Next shows how it's find the features and last shows the detected object.

```
image = cv2.imread(args["image"])
(H, W) = image. shape[:2]
```

The above code is used to load our input image and grab its spatial dimensions.

```
blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
net. setInput(blob)
start = time. time()
layerOutputs = net.forward(ln)
end = time.time()
```

The above code is used to construct a blob from the input image and then perform a forward and pass of the YOLO object detector, giving us our bounding boxes and associated probabilities.

```
print("[INFO] YOLO took {:.6f} seconds".format(end - start))

boxes = []
confidences = []
classIDs = []

count_dict = {}

for output in layerOutputs:

    for detection in output:

        scores = detection [5:]
        classID = np.argmax(scores)
        confidence = scores[classID]

        if confidence > args["confidence"]:
            if LABELS [classID] == "person":

                box = detection [0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")
                x = int (centerX - (width / 2))
                y = int (centerY - (height / 2))
                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], args["threshold"])
```

- In the above code, the first line prints the time taken by the YOLO model to process the input frame. Here, end and start are variables that represent the time before and after YOLO processing, respectively. The format function is used to format the output with 6 decimal places.

- Initialization of lists are used to store information about the detected objects such as their bounding box coordinates, confidence scores, and class IDs.
- The loop iterates over the outputs of the YOLO model. Each output corresponds to the detections made by a specific layer of the YOLO network. Within each layer's output, this loop iterates over individual detections made by that layer. Each detection contains information about one detected object, such as its class probabilities and bounding box coordinates. Here, scores contains the class probabilities predicted by the YOLO model for each class.classID represents the index of the class with the highest probability, and confidence represents the probability/confidence score of that class. This condition filters out weak predictions by checking if the confidence score of the detected object exceeds a predefined threshold (args[“confidence”]). This condition checks if the detected object belongs to the "person" class. If so, it proceeds to process that detection.
- If the detected object meets the criteria (e.g., high enough confidence and belongs to the "person" class), its bounding box coordinates, confidence score, and class ID are added to the respective lists.
- After all detections are processed, non-maximum suppression (NMS) is applied to remove redundant bounding boxes and retain only the most confident ones. The resulting indexes of the selected bounding boxes are stored in the ‘idxs’ variable. These indexes can then be used to filter out the bounding boxes to be drawn on the image.

```
vs = cv2.VideoCapture(args["input"])
(W, H) = (None, None)
```

The above code initializes the video stream, pointer to output video file, and frame dimensions.

```
while True:
    (grabbed, frame) = vs.read()
    if not grabbed:
        break
    if W is None or H is None:
        (H, W) = frame.shape[:2]
```

The above code helps to loops frame from the video file stream and read the next frame from the file and if the frame was not grabbed, then we have reached the end of the stream. If the frame dimensions would be empty then it grabs them.

```
for output in layerOutputs:
        for detection in output:
            scores = detection [5:]
            classID = np.argmax(scores)
            confidence = scores[classID]
            if confidence > args["confidence"] and classID == person_class_index:

                box = detection [0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                x = int (centerX - (width / 2))
                y = int (centerY - (height / 2))


                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)


idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], args["threshold"])
```

The above loop iterates over the detections present in the output obtained from the YOLO model. Each detection contains information about one detected object.

- The scores represent the class probabilities for the detected object. detection[5:] is used to extract the class probabilities starting from index 5. YOLO typically outputs multiple scores for each object detected, representing the likelihood of each class.
- The class ID (classID) is determined by finding the index with the highest score among the scores obtained for the detected object. The confidence score (confidence) corresponds to the probability of the detected object belonging to the class represented by classID.

- The condition filters the detected objects based on two criteria:
- Confidence: It ensures that the confidence score of the detected object exceeds a certain threshold specified by args[“confidence”].
- Class ID: It checks that the confidence score of the detected object belongs to the “person” class, as specified by person_class_index.
- The bounding box coordinates (x, y, width, height) are calculated based on the detected object's position and size. These coordinates are scaled based on the dimensions of the input frame (W, H).
- If the detected object meets the filtering criteria, its bounding box coordinates, confidence score, and class ID are added to the respective lists (boxes, confidences, classIDs).
- After all detections are processed, non-maximum suppression (NMS) is applied to remove redundant bounding boxes and retain only the most confident ones. The resulting indexes of the selected bounding boxes are stored in the ’idxs’ variable. These indexes can then be used to filter out the bounding boxes to be drawn on the image.

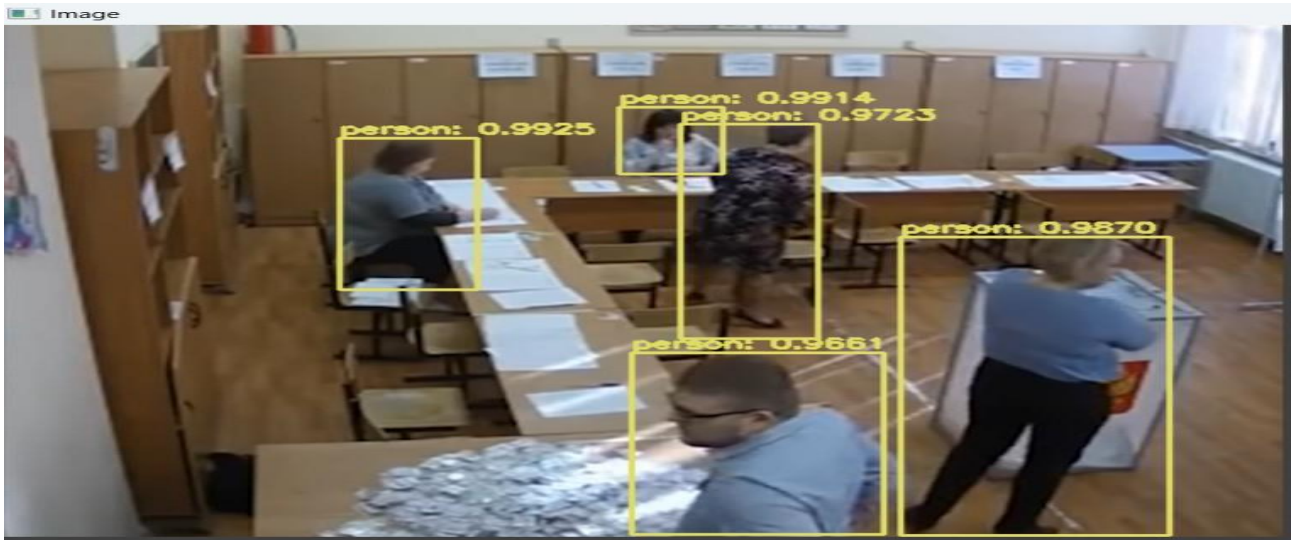
```
vs = cv2.VideoCapture(0)
```

The cv2.VideoCapture(0) function initializes a video capture object (vs) that can read video frames from the default webcam attached to the system. By specifying 0 as the argument to cv2.VideoCapture the code accesses the default webcam.

```
cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
```

The first line of the code displays the current frame on the screen with a window titled "Frame". The cv2.imshow() function is used for displaying images or frames in OpenCV. This line waits for a key press event for up to 1 millisecond. If a key is pressed during this time, cv2.waitKey() returns the ASCII value of the key. The & 0xFF operation is used to mask the key code to only the least significant 8 bits, ensuring compatibility across different platforms. Overall, these lines of code allow for real-time display of frames from the webcam feed with human detection and provide the capability for user interaction.

After implementation, we get output of detected humans in the bounding boxes with labelling as a person along with confidence score.



```
import numpy as np
import argparse
import time
import cv2
import os
import json
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-y", "--yolo", required=True,
    help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
    help="threshold when applying non-maxima suppression")
args = vars(ap.parse_args())
labelsPath = os.path.sep.join([args["yolo"], "obj.names"])
LABELS = open(labelsPath).read().strip().split("\n")
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3), dtype="uint8")
weightsPath = os.path.sep.join([args["yolo"], "yolo.weights"])
configPath = os.path.sep.join([args["yolo"], "yolo.cfg"])
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
image = cv2.imread(args["image"])
(H, W) = image.shape[:2]
ln = net.getLayerNames()
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
net.setInput(blob)
start = time.time()
layerOutputs = net.forward(ln)
end = time.time()
print("[INFO] YOLO took {:.6f} seconds".format(end - start))
boxes = []
confidences = []
classIDs = []
count_dict = {}
for output in layerOutputs:
    for detection in output:
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
        if confidence > args["confidence"]:
            if LABELS[classID] == "person":
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")
```

```
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))


                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], args["threshold"])

if len(idxs) > 0:

    for i in idxs.flatten():

        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        color = [int(c) for c in COLORS[classIDs[i]]]
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
        cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color,
2)

cv2.imshow("Image", image)
cv2.waitKey(0)
```

The above code performs object detection, specifically targeting human detection that is detecting humans in images, using the YOLO (You Only Look Once) deep learning model.

- Necessary libraries such as NumPy, argparse, OpenCV (cv2), and os are imported. These libraries are used for numerical operations, argument parsing, image processing, file operations, and JSON handling, respectively.
- The code defines command-line arguments such as the input image path (--image) the base path to the YOLO directory (--yolo), confidence threshold (--confidence) and non-maxima suppression threshold (--threshold). These arguments are parsed and stored in the args dictionary.
- The YOLO model architecture and pre-trained weights are loaded from disk. Additionally, the class labels (‘LABELS’) are loaded from a file.
- The input image is read from the provided path, and its dimensions(‘H’ for height and ‘W’ for width) are obtained.

- The input image is pre-processed into a blob suitable for YOLO input. Then, a forward pass is performed through the YOLO network, and the output is obtained.
- Detected objects are extracted from the YOLO output, and weak predictions are filtered based on the confidence threshold specified by the user.
- Non-maximum suppression is applied to remove overlapping bounding boxes. Bounding boxes with their corresponding class labels and confidence scores are drawn on the input image.
- The processed image with the bounding boxes is displayed. The program waits for a key press (cv2.waitKey(0)) before closing the image window.
- Overall, this code segment demonstrates how to perform human detection using YOLO on an input image with the flexibility to adjust confidence and threshold parameters via command-line arguments.

```
import numpy as np
import argparse
import time
import cv2
import os
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", required=True, help="path to input video")
ap.add_argument("-y", "--yolo", required=True, help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5, help="minimum
probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3, help="threshold when
applying non-maxima suppression")
args = vars(ap.parse_args())

labelsPath = os.path.sep.join([args["yolo"], "obj.names"])
LABELS = open(labelsPath).read().strip().split("\n")
if "person" not in LABELS:
    raise ValueError("The YOLO model does not include the 'person' class.")

person_class_index = LABELS.index("person")

np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3), dtype="uint8")

weightsPath = os.path.sep.join([args["yolo"], "yolo.weights"])
configPath = os.path.sep.join([args["yolo"], "yolo.cfg"])

print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames()
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]

vs = cv2.VideoCapture(args["input"])
(W, H) = (None, None)

while True:
    (grabbed, frame) = vs.read()

    if not grabbed:
        break
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)
```

```
    boxes = []
    confidences = []
    classIDs = []

    for output in layerOutputs:

        for detection in output:

            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            if confidence > args["confidence"] and classID == person_class_index:

                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)

    idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
args["threshold"])

    if len(idxs) > 0:
        for i in idxs.flatten():
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])

            color = [int(c) for c in COLORS[classIDs[i]]]
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
            text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
            cv2.putText(frame, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break

vs.release()

cv2.destroyAllWindows()
```

This code performs human detection in a video using the YOLO (You Only Look Once) object detection algorithm.

Importing libraries:

- numpy: Numerical operations library.
- argparse: Command-line argument parsing library.
- time: Time-related functions.
- cv2: OpenCV library for image processing
- os: Operating system related functions.

Argument Parsing:

The script parses command-line arguments using argparse. Four arguments are defined:

- -i or –input: Path to the input video.
- -y or –yolo: Base path to the YOLO directory.
- -c or –confidence: Minimum probability to filter weak detections.
- -t or –threshold: Threshold when applying non-maximum suppression.

Loading YOLO Model:

- The script loads the YOLO model configuration (yolo.cfg) and pre-traied weights (yolo.weights) using OpenCV's cv2.dnn.readNetFromDarknet() function.
- It also determines the output layer names that we need from YOLO.

Loading class labels:

The COCO class labels are loaded from the obj.names file inside the YOLO directory.

Initializing variables:

- Random colors are generated to represent different class labels.
- Video stream(vs) is initialized using cv2.VideoCapture(args["input"]).
- Width and height ('W' and 'H') of the frames are initialized.

Video Processing Loop:

- The script reads each frame from the video stream in a loop using vs.read().
- For each frame, a blob is constructed and passed through the YOLO network.
- Detected objects are filtered based on confidence scores and the "person" class.
- Bounding boxes are drawn around detected persons on the frame.

- The frame with bounding boxes and labels is displayed using cv2.imshow().
- The loop continues until the end of the video or until the user presses the 'q' key.

Displaying Output:

- After processing all frames, the video stream is released (vs.release()), and all OpenCV windows are closed (cv2.destroyAllWindows()).

This code demonstrates a complete pipeline for human detection in videos using YOLO, from loading the model and processing frames to displaying the output with bounding boxes and labels.

```
import numpy as np
import argparse
import time
import cv2
import os
ap = argparse.ArgumentParser()
ap.add_argument("-y", "--yolo", required=True, help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5, help="minimum
probability to filter weak detections")
args = vars(ap.parse_args())

labelsPath = os.path.sep.join([args["yolo"], "obj.names"])
LABELS = open(labelsPath).read().strip().split("\n")

person_class_index = LABELS.index("person")

weightsPath = os.path.sep.join([args["yolo"], "yolo.weights"])
configPath = os.path.sep.join([args["yolo"], "yolo.cfg"])

print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

vs = cv2.VideoCapture(0)
while True:
    (grabbed, frame) = vs.read()

    if not grabbed:
        break
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(net.getUnconnectedOutLayersNames())


    boxes = []
    confidences = []
    classIDs = []

    for output in layerOutputs:
        for detection in output:

            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            if classID == person_class_index and confidence > args["confidence"]:
```

```
                box = detection[0:4] * np.array([frame.shape[1], frame.shape[0],
frame.shape[1], frame.shape[0]])
                (centerX, centerY, width, height) = box.astype("int")

                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)

    idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], 0.3)
    if len(idxs) > 0:

        for i in idxs.flatten():

            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])

            color = (0, 255, 0)
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
            text = "Person: {:.2f}".format(confidences[i])
            cv2.putText(frame, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)

    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break

vs.release()
cv2.destroyAllWindows()
```

This Python script performs real-time human target detection using YOLO (You Only Look Once) deep learning model.

- The script starts by importing necessary libraries including NumPy (np), argparse, time, OpenCV (cv2) and os.
- Defines command-line arguments using argparse, allowing users to specify the path to the YOLO directory (--yolo) and the minimum confidence threshold for detections (--confidence).

- Loads the YOLO model architecture and pre-trained weights from the disk using cv2.dnn.readNetFromDarknet. The paths to the configuration file (yolo.cfg) and the pre-trained weights file (yolo.weights) are constructed from the command-line arguments.
- Initializes the video stream using OpenCV's cv2.VideoCapture(0) to capture frames from the default camera (index 0).
- Enters a loop to continuously read frames from the video stream.
- Each frame is processed for human target detection.
- Each frame is converted to a blob using cv2.dnn.blobFromImage, which preprocesses the image for input to the neural network. The blob is normalized, resized, and converted to the required format.
- It iterates through the output predictions to extract bounding box coordinates, confidence scores, and class IDs.
- Only detections with class ID corresponding to "person" and confidence above the specified threshold are retained.
- Non-maximum suppression (NMS) is applied using cv2.dnn.NMSBoxes to remove redundant bounding boxes and retain only the most confident detections.
- Bounding boxes are drawn around detected human targets on the frame using cv2.rectangle.
- Confidence scores are overlayed as text using cv2.putText.
- The processed frame with bounding boxes and confidence scores is displayed using ‘cv2.imshow’.
- The loop continues until the user presses the 'q' key, upon which the program releases the video stream and closes all OpenCV windows.
- This script demonstrates a simple yet effective implementation of real-time human target detection using YOLO and OpenCV, suitable for applications such as surveillance, security monitoring, and human-computer interaction.

5.3 Test cases:

Tested	Test name	Inputs	Expected output	Actual Output	Status
1	Load Dataset	Human detection Dataset	Read dataset	Load dataset	Success
2	Split dataset	Train80% and test20%	Divide the training set and Testing set	Split train and Test	Success
	Train Model	Train dataset, random value, predicted class	Train with best accuracy	Train with best accuracy	Success
4	Validate Model	No. of Epochs	Validate the Model with best fit	Model Generated	Success
5	Predict accuracy and Error Rate	Accuracy	Plot expected accuracy and predicted accuracy	Plot expected predicted accuracy	success
6	Test Data	Test column	Predicted accuracy	Predicted accuracy	success

Table 5.3 Test cases related data

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion:

Robust human target detection is a software that is designed to identifying and tracking humans accurately and reliably in various environments, conditions, and applications. This idea addresses the problem of detecting and tracking human targets in outdoor environments, even when the targets are partially visible. This approach of human detection can detect humans in illumination, and complex backgrounds. Advancements in the project incorporate deep learning models, which excel in learning discriminative features from an image or a video.

6.2 Future Scope:

The future scope of robust women target detection is vast and promising, especially considering advancement in technology, computer vision, and machine learning. Here are some potential future directions:

Improved accuracy: enhancing the accuracy of human target detection algorithms remains a priority. This involves refining models, training data, and techniques to reduce false positives and negatives.

Integration with robotics: integrating human detection capabilities into autonomous systems and Behaviour analysis robots to enable safe interaction, navigation, and collaboration in shared environments.

Behaviour analysis: moving beyond simple detection, future systems may analyse human behaviour, gestures, an interaction to infer intentions or identify suspicious activities, aiding in security and surveillance applications.

Context awareness: incorporating contextual information, such as scene understanding and object relationships, to improve the accuracy unreliability of human detection systems, especially in dynamic environments.

Adversarial robustness: developing robust detection algorithms resilient to adversarial attacks, ensuring reliable performance in the presence of deliberate perturbations or camouflage techniques.

REFERENCES:

1. Jiahui Sun, Huayong Ge and Zhehao Zhang, AS-YOLO: An Improved YOLOv4 based on Attention Mechanism and Squeeze Net for Person Detection, IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2021
2. Z.Zhao, Q. Zheng, P.Xu, S. T, and X. Wu, Object detection with deep learning: A review,IEEETransactions on neural networks and learning systems, 30(11), 3212-3232, 2019.
3. R.Bharti, K. Bhadane, P. Bhadane, and A. Gadhe, Object Detection and Recognition for BlindAssistance, International Research Journal of Engineering and Technology (IRJET) e-ISSN:2395-0056 Volume:06, 2019.
4. M. Ivašić and M. Pobar, “Human detection in thermal imaging using YOLO,” pp. 2–7.
5. Y. Wang, J. Wu, and H. Li, “Human Detection Based on Improved Mask R-CNN,” 2020.