PEOPLE COUNTING AND TRACKING SYSTEM

1. INTRODUCTION

1.1 OVERVIEW

The People Counting and Tracking System is a cutting-edge solution that leverages real-time computer vision techniques to estimate and analyze the flow of people in diverse environments. This system finds extensive applications in areas such as retail stores, educational institutions, corporate workplaces, restaurants, hospitality facilities, and washrooms. By accurately counting and tracking individuals, this system provides valuable insights for security, resource management, and marketing purposes.

1.2 PURPOSE

The purpose of this project is to develop an advanced people counting and tracking system capable of accurately quantifying the number of people traversing through specific areas, such as building entrances or corridors. By employing video or camera feeds along with sophisticated algorithms, the system can detect and track individuals in real-time. The generated data offers significant benefits, including enhanced security measures, optimized resource allocation, and informed decision-making.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

The accurate counting and tracking of people in real-time have been the focus of extensive research in the fields of computer vision and artificial intelligence. Most of the existing researches [iv] [v] [vi] uses YOLOv3 object detection model, which can struggle to detect smaller objects due to its anchor box design and large stride. YOLOv3 requires a significant amount of memory to run, which can be a challenge for devices with limited resources. Also, Training YOLOv3 can be time-consuming, requiring large amounts of data and computational resources. DEEPSORT tracking algorithm used in [iv] has many drawbacks like ID switches, bad occlusion handling, motion blur, and many more. The average accuracy for this algorithm is 28.6 which is very low.

2.2 PROPOSED SOLUTION

The provided solution implements an object-tracking system using a combination of object detection and correlation tracking. The proposed model aims to detect and track people in a video stream or a video file.

1. Importing necessary libraries and modules:

The code imports various modules and classes required for the implementation, such as CentroidTracker and TrackableObject from custom packages, VideoStream and FPS from the "imutils.video" module, numpy, time, cv2 (OpenCV), imutils, and dlib.

2. Initializing variables and constants:

a. The script initializes a list of class labels named CLASSES, which represents the objects that the MobileNet SSD (Single Shot MultiBox Detector) model was trained to detect.

b. The pretrained MobileNet SSD model is loaded from disk using cv2.dnn.readNetFromCaffe() function, providing the prototxt file and the pre-trained model weights.

3. Starting the video stream or opening a video file and Processing frames from the video stream:

4. Object detection and tracking:

The code performs object detection every skip_frames frame, while on other frames, it utilizes correlation tracking for higher processing throughput.

5. Centroid tracking and counting:

a. A horizontal line is drawn on the frame to represent the center line. Objects crossing this line will be used to determine their movement direction (up or down).

b. The CentroidTracker is used to associate old and new object centroids.

c. For each tracked object, the code checks if a TrackableObject already exists for that object ID. If not, a new TrackableObject is created.

6. Writing output and displaying the frame and Cleanup and reporting:

If an output file path is provided and the video writer is initialized, the current frame with overlays is written to the output file.

3. THEORETICAL ANALYSIS

3.1 BLOCK DIAGRAM

3.2 HARDWARE/SOFTWARE

Designing The successful implementation of the People Counting and Tracking System requires specific hardware and software components, including:

Hardware:

• Camera or video feed source

• Computer or embedded system with sufficient processing power

Software:

• Anaconda Navigator or Pycharm Community Edition

• Python programming language, known for its versatility and extensive libraries

• Deep learning frameworks such as TensorFlow or PyTorch for efficient training and inference

• OpenCV library, offering a wide range of computer vision tasks and algorithms

• Flask web framework to create a user-friendly web application for live streaming and visualization of visitor flow

• Additionally, the MobideSSD detection model is utilized for person detection due to its robust performance and accuracy.

4. EXPERIMENTAL INVESTIGATIONS

During the development of the People Counting and Tracking System, numerous experimental investigations are conducted to evaluate the system's performance, accuracy, and reliability. Key aspects analyzed include:

• Accuracy and robustness of person detection and tracking algorithms

• System's ability to handle challenging scenarios, such as occlusion and varying lighting conditions

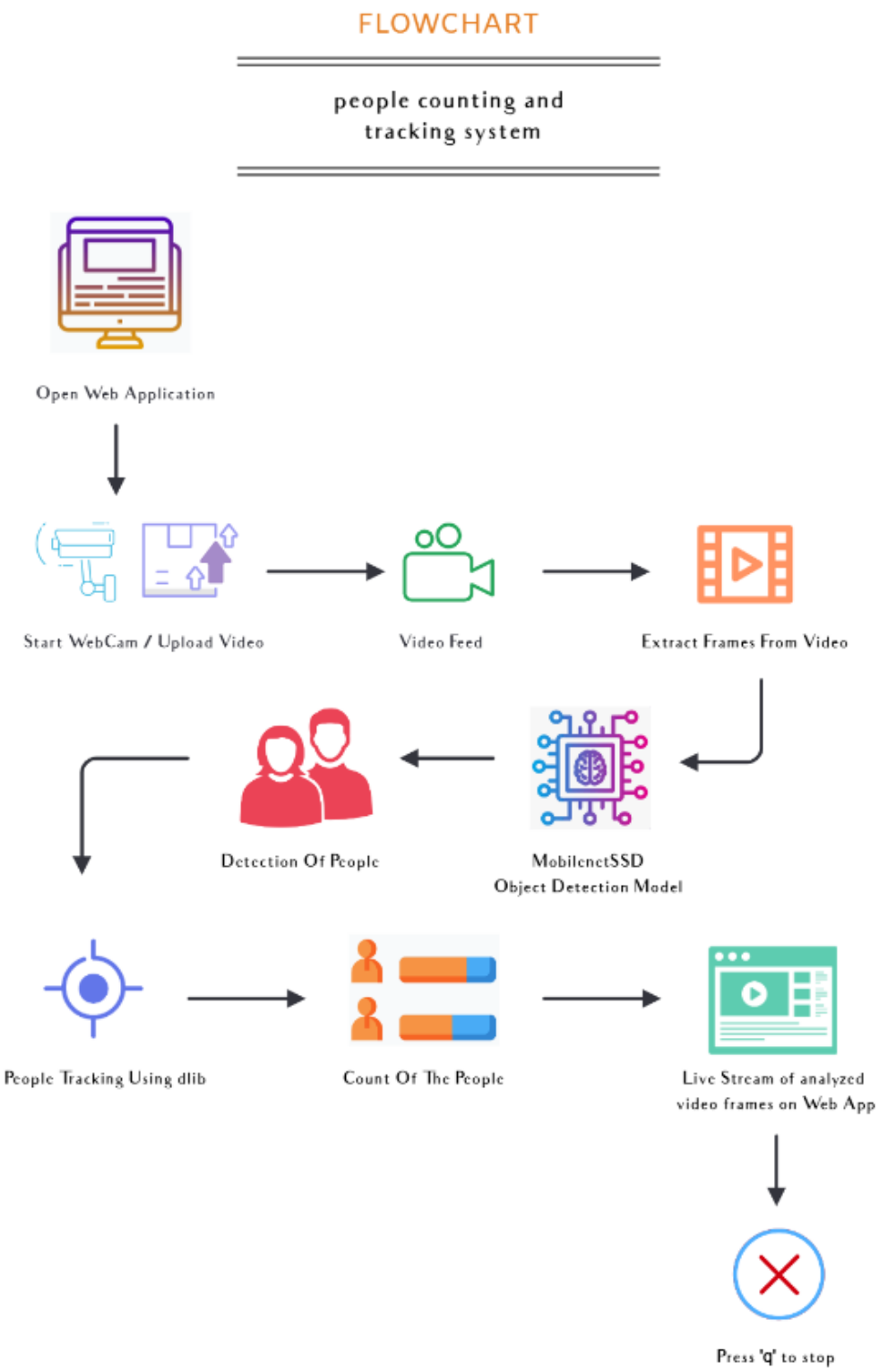
• Real-time performance and computational efficiency, ensuring the system can process video feeds seamlessly

• Integration of the system with a web application, allowing live streaming and visualization of visitor flow

• User interface design and usability testing to ensure a seamless and intuitive experience for users

These experimental investigations play a crucial role in refining and optimizing the system, making it highly effective and adaptable to various real-world scenarios.

5. FLOWCHART


FLOWCHART
people counting and tracking system
Open Web Application
Start WebCam / Upload Video
Video Feed
Extract Frames From Video
MobilenetSSD Object Detection Model
Detection Of People
People Tracking Using dlib
Count Of The People
Live Stream of analyzed video frames on Web App
Press 'q' to stop

6. RESULT

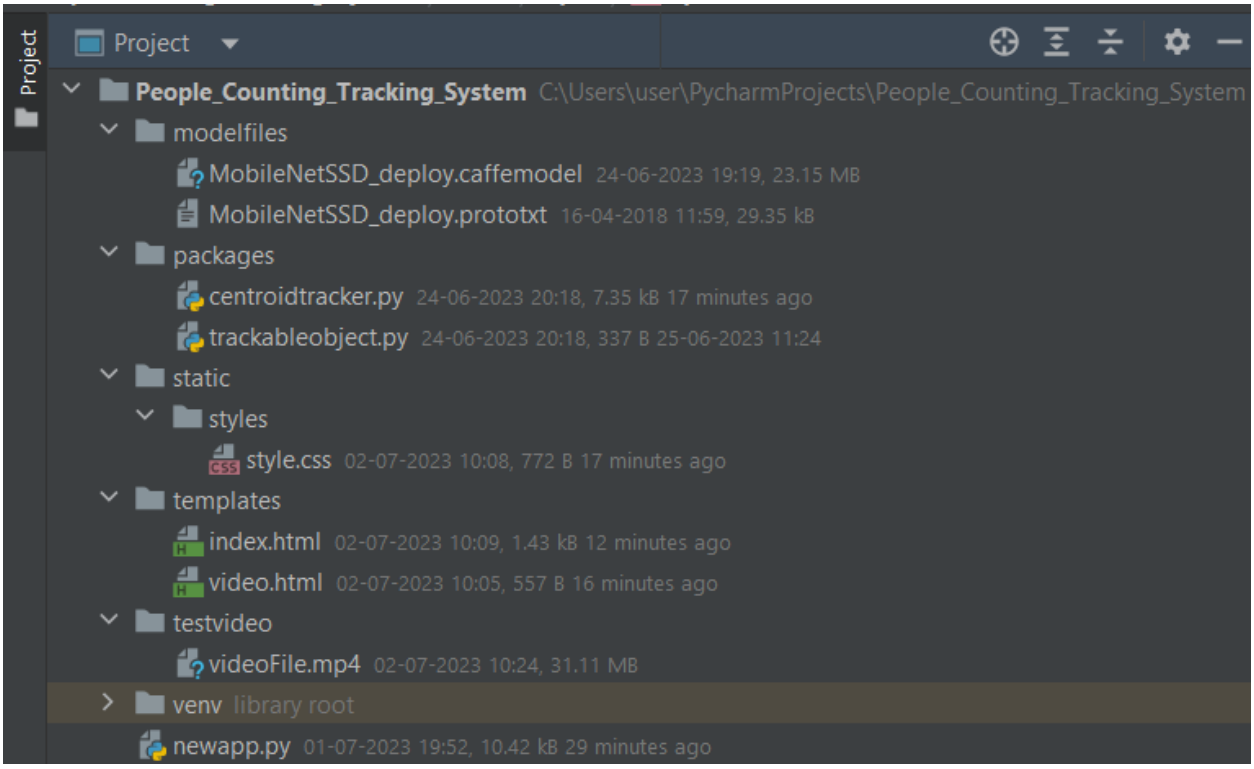
In this section, we present the results of the proposed model. We evaluate the performance of our system by uploading different videos containing CCTV footage from airport and pedestrian crossings. The goal of our evaluation is to assess the accuracy and robustness of our system in accurately counting and tracking people in real-world settings.

It is observed that our model can detect most of the people present and track their movement and direction of movement towards or away from the base line, into or out of the frame. It counts the total number of people moving towards or away from the base line almost correctly.

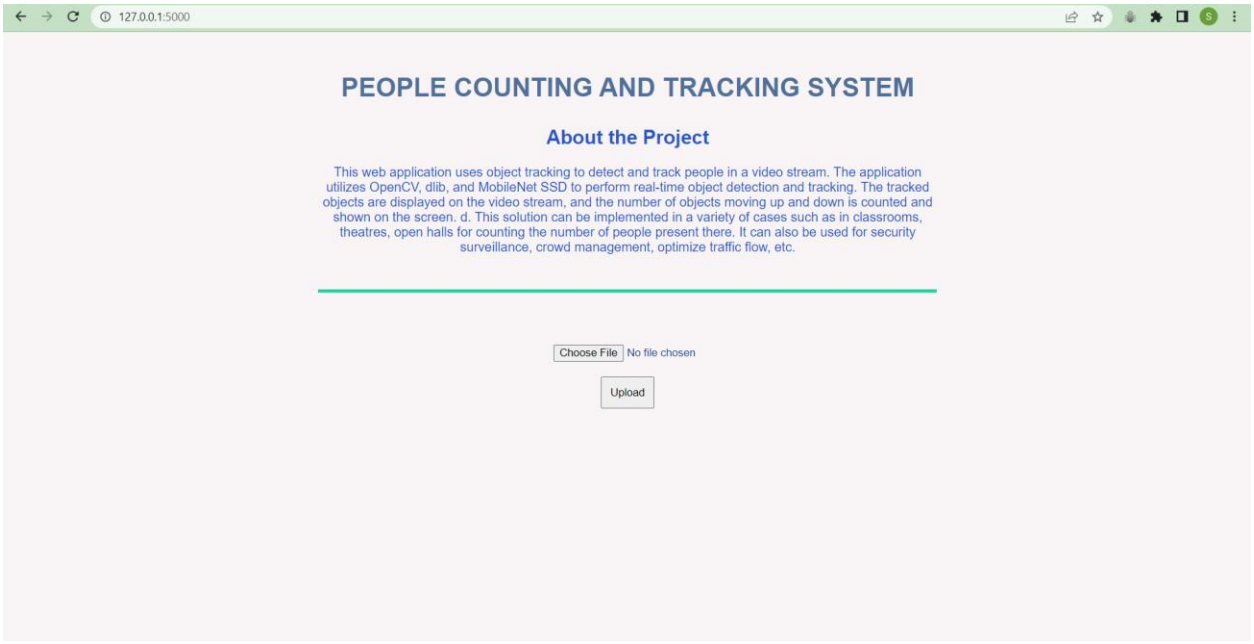
The proposed model is seamlessly integrated with a web application using Flask. In the web application we can upload the video to be analyzed. Then, the model analyzes the uploaded video and is able to render and run the output video in the web application instantly.

Screenshots:

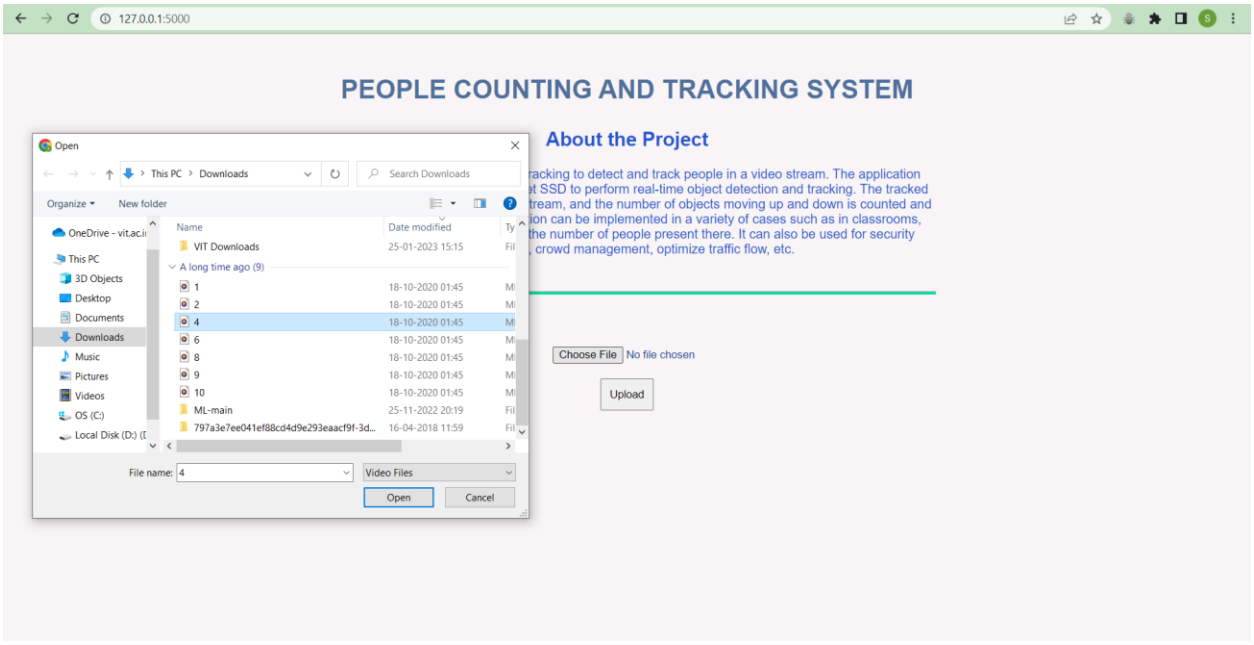
Project Structure →



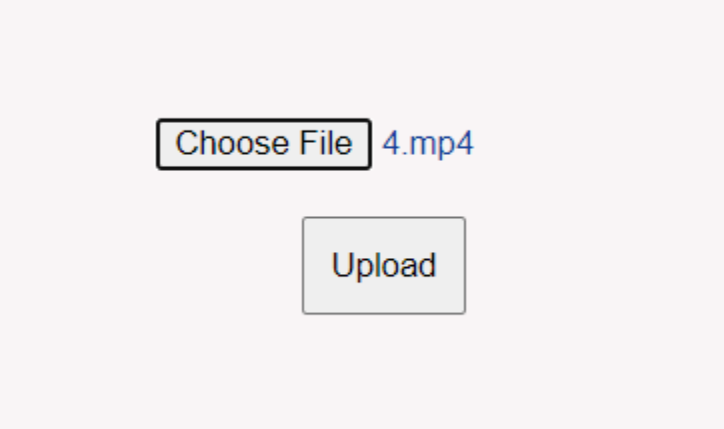
Homepage



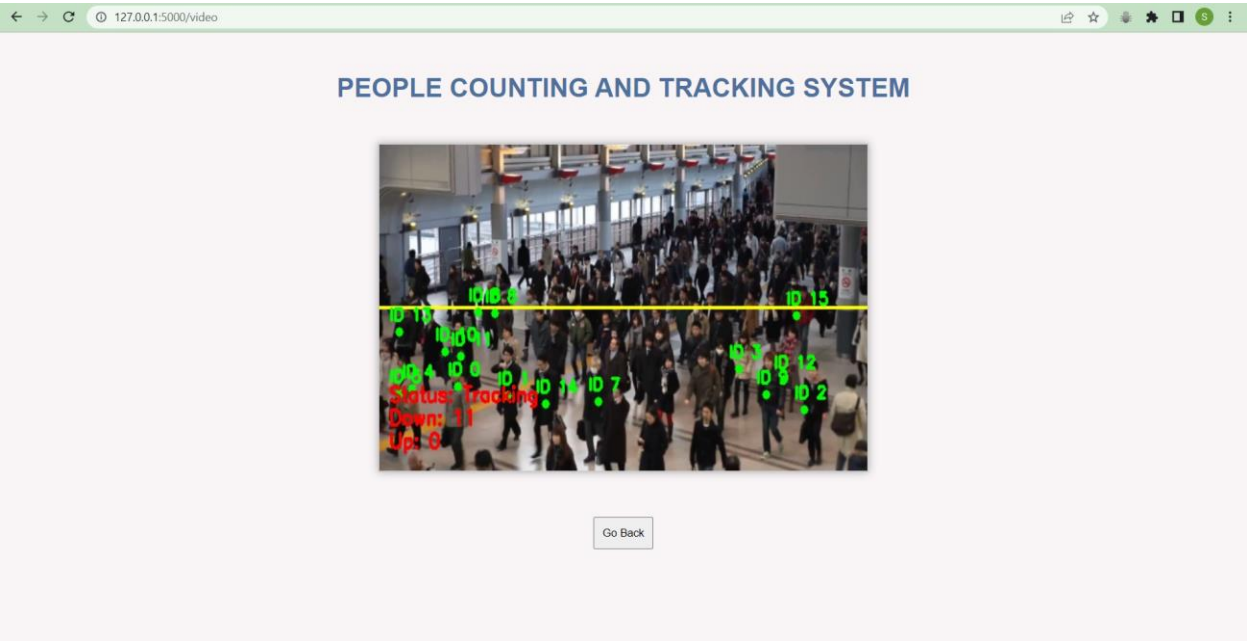
Choose File



File 4.mp4 chosen



Click ‘Upload’ button. Now the analyzed video is displayed.



7. ADVANTAGES AND DISADVANTAGES

Advantages of the proposed solution:

a. It uses MobileNetSSD for object detection, which is a lightweight and computationally efficient model. Its optimized architecture and efficient implementation allow for quick inference.
b. Proposed solution has high accuracy in tracking and counting people. It is able to detect people and track their movement with precision in most cases.
c. It does not require a large dataset to train the model as it uses pre-trained model for object detection and tracking.
d. This solution can be implemented in a variety of cases such as in classrooms, theatres, open halls for counting the number of people present there. It can also be used for security surveillance, crowd management, optimize traffic flow, etc.

Disadvantage:

a. People counting and tracking systems involve capturing and analyzing individuals' movement and behavior, which raises concerns about privacy.
b. Achieving 100% accuracy in people counting and tracking is challenging. These systems may encounter difficulties in accurately detecting and tracking individuals in crowded or complex scenes, leading to false positives (counting non-existent people) or false negatives (missing individuals).
c. Lighting conditions can significantly impact the performance of people counting and tracking systems. Poor lighting, shadows, glare, or sudden changes in illumination levels can affect the accuracy of detection and tracking algorithms.
d. It may struggle in complex environments with a high level of background clutter. Cluttered backgrounds, moving objects, or intricate scenes can introduce noise and confusion, making it difficult to distinguish individuals accurately.

8. APPLICATIONS

The solution can be applied in a variety of areas, such as:

a. In crowd management and ensuring public safety: They enable real-time monitoring of crowd density, movement, and flow in public spaces, transportation hubs, or event venues. This information helps authorities in crowd control, preventing overcrowding, identifying potential bottlenecks, and managing emergency situations more effectively.
b. Resource Planning and Optimization: By accurately counting and tracking people, organizations can optimize resource allocation. This includes staffing levels, facility utilization, and queue management. With real-time insights into crowd patterns and flow, businesses can efficiently allocate resources, reduce wait times, and enhance operational efficiency.

c. Security and Surveillance: People counting and tracking systems enhance security and surveillance efforts. They provide real-time information on the number and location of people within a monitored area, facilitating the detection of suspicious activities or identifying individuals of interest. Integrated with other security technologies, such as facial recognition or access control systems, these systems can strengthen security measures and improve situational awareness.
d. Traffic Flow Management: In transportation systems, people counting and tracking systems help manage traffic flow and optimize transportation services. By monitoring passenger volumes and movement in stations, terminals, or public transportation vehicles, authorities can adjust schedules, allocate resources efficiently, and identify areas that require infrastructure improvements to enhance the overall transportation experience.
e. Social Distancing and Occupancy Monitoring: In the context of public health and safety, people counting and tracking systems have gained significance during the COVID-19 pandemic. These systems can help enforce social distancing guidelines by monitoring occupancy levels in venues, stores, or public spaces. By providing real-time occupancy data, businesses and authorities can ensure compliance with regulations, prevent overcrowding, and maintain a safe environment for individuals.

9. CONCLUSIONS

In conclusion, proposed system has proven to be a more efficient, effective, and accurate solution for detecting, tacking and counting people from video feed. Despite the hardware limitations, this study has achieved the actual real-time performance with the help of pre-trained MobileNetSSD object detection model. The 'dlib' library is used for tracking purpose. The model is integrated with a web application with the help of flask. This solution has various applications in crowd management, surveillance, attendance, etc.

10. FUTURE SCOPE

Now, our proposed solution can only detect people and count their numbers. In future, we can add facial recognition feature, zooming into faces, detecting emotions of the person detected, detecting suspicious activities taking place, and many more, to make the solution more widely useful. Also, in future, we can try to improve the model so that it works well for videos in different lighting conditions as well as in very crowded areas.

11. BIBLIOGRAPHY

[i] https://www.analyticsvidhya.com/blog/2022/09/object-detection-using-yolo-and-mobilenet-ssd/

[ii] https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html

[iii] https://www.analyticsvidhya.com/blog/2021/06/crowd-counting-using-deep-learning/

[iv] Hamam Mokayed, Tee Zhen Quan, Lama Alkhaled and V. Sivakumar. (2022). “Real-Time Human Detection and Counting System Using Deep Learning Computer Vision Techniques”

[v] Suhane, Anuj & Vani, Aryan & Parihar, Harsh & Raghuwanshi, Udit & Nimbark, Arjun & Saxena, Lucky. (2023). “HUMAN DETECTION AND CROWD COUNTING USING YOLO”.

[vi] Gomes, Hugo, Nuno Redinha, Nuno Lavado, and Mateus Mendes. 2022. "Counting People and Bicycles in Real Time Using YOLO on Jetson Nano" *Energies* 15, no. 23

APPENDIX

A. SOURCE CODE

centroidtracker.py

```
from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np

class CentroidTracker:
    def __init__(self, maxDisappeared=50, maxDistance=50):
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()
        self.maxDisappeared = maxDisappeared
        self.maxDistance = maxDistance

    def register(self, centroid):
        self.objects[self.nextObjectID] = centroid
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1

    def deregister(self, objectID):
        del self.objects[objectID]
        del self.disappeared[objectID]

    def update(self, rects):
        if len(rects) == 0:
            for objectID in list(self.disappeared.keys()):
                self.disappeared[objectID] += 1
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)
```

```
            return self.objects

        inputCentroids = np.zeros((len(rects), 2), dtype="int")

        for (i, (startX, startY, endX, endY)) in enumerate(rects):
            cX = int((startX + endX) / 2.0)
            cY = int((startY + endY) / 2.0)
            inputCentroids[i] = (cX, cY)

        if len(self.objects) == 0:
            for i in range(0, len(inputCentroids)):
                self.register(inputCentroids[i])

        else:
            objectIDs = list(self.objects.keys())
            objectCentroids = list(self.objects.values())
            D = dist.cdist(np.array(objectCentroids), inputCentroids)
            rows = D.min(axis=1).argsort()
            cols = D.argmin(axis=1)[rows]
            usedRows = set()
            usedCols = set()


            for (row, col) in zip(rows, cols):
                if row in usedRows or col in usedCols:
                    continue

                if D[row, col] > self.maxDistance:
                    continue

                objectID = objectIDs[row]
                self.objects[objectID] = inputCentroids[col]
                self.disappeared[objectID] = 0
                usedRows.add(row)
                usedCols.add(col)

            unusedRows = set(range(0, D.shape[0])).difference(usedRows)
            unusedCols = set(range(0, D.shape[1])).difference(usedCols)

            if D.shape[0] >= D.shape[1]:
                for row in unusedRows:

                    objectID = objectIDs[row]
                    self.disappeared[objectID] += 1
                    if self.disappeared[objectID] > self.maxDisappeared:
                        self.deregister(objectID)
            else:
                for col in unusedCols:
                    self.register(inputCentroids[col])

        return self.objects
```

trackableobject.py

```
class TrackableObject:
   def __init__(self, objectID, centroid):
      self.objectID = objectID
      self.centroids = [centroid]
      self.counted = False
```

newapp.py

```
from flask import Flask, render_template, Response, request, redirect,
url_for
from packages.centroidtracker import CentroidTracker
from packages.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import time
import cv2
import imutils
import dlib

app = Flask(__name__)
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle",
"bus", "car",
           "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike",
"person",
           "pottedplant", "sheep", "sofa", "train", "tvmonitor"]
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(r"modelfiles/MobileNetSSD_deploy.prototxt",
                               r"modelfiles/MobileNetSSD_deploy.caffemodel")
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video', methods=["POST"])
def uploadVideo():
    videoFile = request.files['videoFile']
    video_path = "./testvideo/videoFile.mp4"
    videoFile.save(video_path)
    return render_template('video.html')

def generate():
    global vs, W, H, ct, trackers, trackableObjects, totalFrames, totalDown,
totalUp, fps
    file = "./testvideo/videoFile.mp4"
    if not (file, False):
        print("[INFO] starting video stream...")
        vs = VideoStream(src=0).start()
        time.sleep(2.0)

    else:
        print("[INFO] opening video file...")
        vs = cv2.VideoCapture(file)

    writer = None
```

```
    W = None
    H = None
    ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
    trackers = []
    trackableObjects = {}
    totalFrames = 0
    totalDown = 0
    totalUp = 0
    skip_frames = 3
    fps = FPS().start()

    while True:
        success, frame = vs.read()
        if frame is None:
            break;
        frame = imutils.resize(frame, width=500)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        if W is None or H is None:
            (H, W) = frame.shape[:2]
        status = "Waiting"
        rects = []
        if totalFrames % skip_frames == 0:
            status = "Detecting"
            trackers = []
            blob = cv2.dnn.blobFromImage(frame, 0.007843, (W,H), 127.5)
            net.setInput(blob)
            detections = net.forward()
            for i in np.arange(0, detections.shape[2]):
                confidence = detections[0, 0, i, 2]
                if confidence > 0.4:
                    idx = int(detections[0, 0, i, 1])
                    if CLASSES[idx] != "person":
                        continue
                    box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
                    (startX, startY, endX, endY) = box.astype("int")
                    tracker = dlib.correlation_tracker()
                    rect = dlib.rectangle(startX, startY, endX, endY)
                    tracker.start_track(rgb, rect)
                    trackers.append(tracker)
        else:
            for tracker in trackers:
                status = "Tracking"
                tracker.update(rgb)
                pos = tracker.get_position()
                startX = int(pos.left())
                startY = int(pos.top())
                endX = int(pos.right())
                endY = int(pos.bottom())
                rects.append((startX, startY, endX, endY))
        cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)
        objects = ct.update(rects)
        for (objectID, centroid) in objects.items():
            to = trackableObjects.get(objectID, None)
            if to is None:
                to = TrackableObject(objectID, centroid)
            else:
                y = [c[1] for c in to.centroids]
```

```
                direction = centroid[1] - np.mean(y)
                to.centroids.append(centroid)
                if not to.counted:
                    if direction < 0 and centroid[1] < H // 2:
                        totalUp += 1
                        to.counted = True
                    elif direction > 0 and centroid[1] > H // 2:
                        totalDown += 1
                        to.counted = True
            trackableObjects[objectID] = to
            text = "ID {}".format(objectID)
            cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)
        info = [
            ("Up", totalUp),
            ("Down", totalDown),
            ("Status", status),
        ]
        for (i, (k, v)) in enumerate(info):
            text = "{}: {}".format(k, v)
            cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
        (flag, encodedImage) = cv2.imencode(".jpg", frame)
        yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
bytearray(encodedImage) + b'\r\n')
        totalFrames += 1
        fps.update()

@app.route('/video_feed')
def video_feed():
    return Response(generate(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/',methods=['POST'])
def go_back():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
    vs.release()
    cv2.destroyAllWindows()
```

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>People Counting and Tracking</title>
    <link rel= "stylesheet" type= "text/css" href= "{{
url_for('static',filename='styles/style.css') }}">
</head>
<body>
    <h1>PEOPLE COUNTING AND TRACKING SYSTEM</h1>
    <div id="project-description">
        <h2>About the Project</h2>
```

```
        <div class="width">
            <p>
                This web application uses object tracking to detect and track
people in a video stream.
                The application utilizes OpenCV, dlib, and MobileNet SSD to
perform real-time object detection
                and tracking. The tracked objects are displayed on the video
stream, and the number of objects
                moving up and down is counted and shown on the screen. d.
This solution can be implemented in
                a variety of cases such as in classrooms, theatres, open
halls for counting the number of people
                present there. It can also be used for security surveillance,
crowd management, optimize traffic flow, etc.
            </p>
        </div>
    </div>
    <br><hr><br><br><br>
    <div>
        <form action="/video" method="post" enctype="multipart/form-data" >
            <input type="file" name="videoFile" id="videoFile"
accept="video/*" required>
            <br><br>
            <input type="submit" value="Upload">
        </form>
    </div>
</body>
</html>
```

video.html

```
<!DOCTYPE html>
<html>
<head>
    <title>People Counting and Tracking</title>
    <link rel= "stylesheet" type= "text/css" href= "{{
url_for('static',filename='styles/style.css') }}">
</head>
<body>
    <h1>PEOPLE COUNTING AND TRACKING SYSTEM</h1>
    <div id="video-container">
        <img id="video" src="{{ url_for('video_feed') }}" width="600"
height="400">
    </div>
    <br><br><br>
    <div>
        <form action="/" method="post">
            <input type="submit" value="Go Back">
        </form>
    </div>
</body>
</html>
```

style.css

```
body {
    font-family: Arial, sans-serif;
    text-align: center;
    background-color: #F9F5F6;
    color: #143e97;
}
h1 {
    margin-top: 50px;
    color: #4F709C;
}
#video-container {
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 50px;
}
#video {
    border: 2px solid #ccc;
    max-width: 800px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
}
#project-description {
    margin-top: 30px;
    color: #2555d0;
}
input[type=submit]{
    padding: 10px;
}
.width{
    width: 50%;
    margin: auto;
}
input[type=file]{
    margin: auto;
    padding-left: 70px;
}
input[type=submit]{
    padding: 10px;
}
hr{
    height: 3px;
    background-color: #00DFA2;
    width: 50%;
}
```

THE END