LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CT6057NI Computer Vision

60% Individual Coursework

Submission : Milestone 2

Academic Semester: Spring Semester 2025

Credit: 15 credit semester long module

Topic: Pedestrian Detection

Student Name: Rajita Maharjan

London Met ID: 22067335

College ID: np01ai4a220052

Assignment Due Date: Thursday, April 24, 2025

Assignment Submission Date: Thursday, April 24, 2025

Submitted To: Shreejan Shrestha

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Document Details

Submission ID
trn:oid:::3618:92660449

Submission Date
Apr 24, 2025, 12:20 PM GMT+5:45

Download Date
Apr 24, 2025, 12:21 PM GMT+5:45

File Name
13.docx

File Size
17.4 KB

14 Pages

2,645 Words

14,871 Characters

4% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **9** Not Cited or Quoted 3%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 1%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

2% Internet sources
0% Publications
3% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **9** Not Cited or Quoted 3%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 1%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 2% Internet sources
- 0% Publications
- 3% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	Institute of Technology, Nirma University on 2024-05-22 <1%
2	Internet	opus4.kobv.de <1%
3	Internet	www.ai-hive.net <1%
4	Internet	www.jatit.org <1%
5	Submitted works	University of Salford on 2023-05-05 <1%
6	Submitted works	Liverpool John Moores University on 2024-04-14 <1%
7	Internet	ins.sjtu.edu.cn <1%
8	Submitted works	King's College on 2025-04-17 <1%
9	Submitted works	Universidad de las Islas Baleares on 2023-09-30 <1%

Table of Contents

Table of Figures

1. Introduction

Pedestrian detection is a important task in computer vision which is widely used in applications like surveillance. The development of this system relies on deep learning through the implementation of YOLO (You Only Look Once) for building a fast and precise real-time detection system targeting pedestrians. The pedestrian detection procedure requires training the system using labeled pedestrian data.

The detection of pedestrians in real-time occurs through YOLO by dividing images into grids and applying box prediction for the pedestrians. The system optimization involves solutions which improve performance during situations with blocked subjects and inconsistent lighting conditions.

The system introduces rectangular boxes to outline detected pedestrians along with confidence score estimates that make understanding results more simple. The project works towards establishing a solid real-time pedestrian detection system that operates across different actual usage environments.

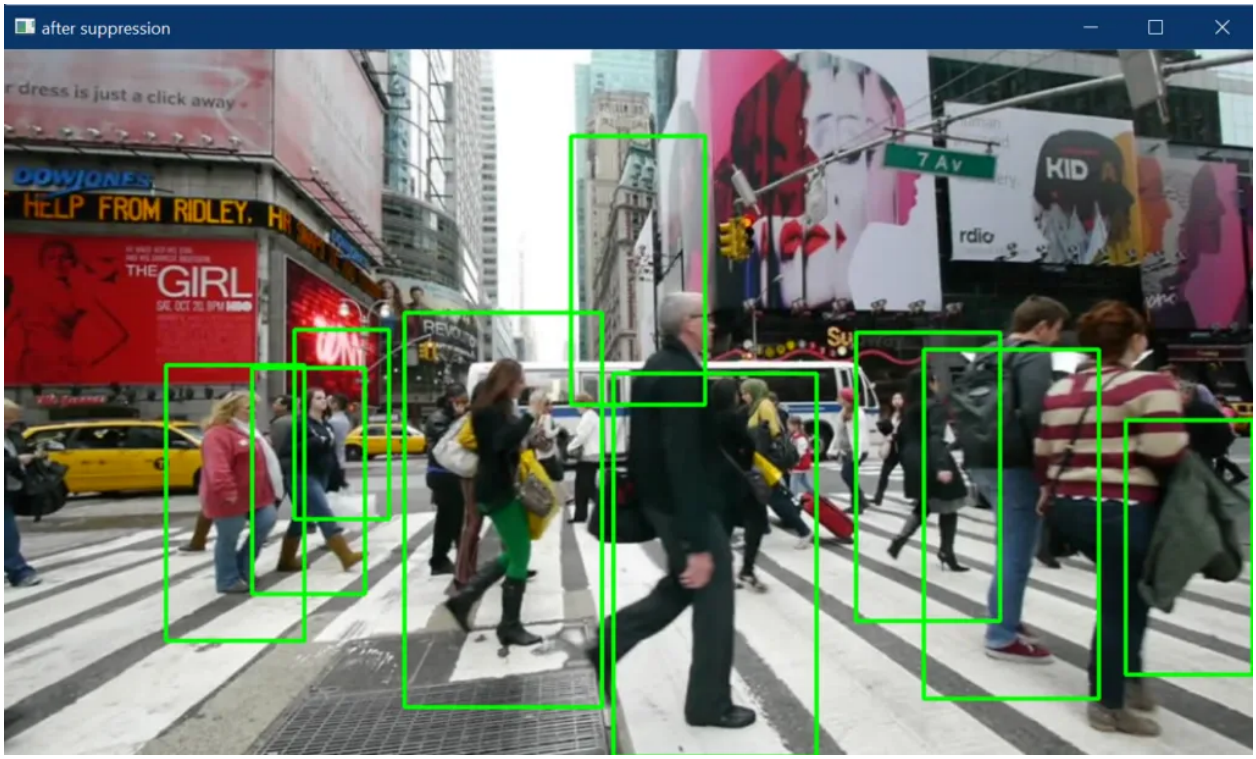

Figure 1: (Sharma, 2021)

1.1. Problem Statement

- **Real-Time Pedestrian Detection:**

The system needs to read input images or video frames at high speed and keep an accurate watch for pedestrians in multiple environmental situations.

- **Variability of Pedestrian Appearances:**

Different conditions result in pedestrians displaying diverse appearances. The system handles differences that source from the way pedestrians stand or walk or sit while wearing certain outfits and holding particular items. Detecting pedestrians becomes difficult because their physical dimensions differ according to how far they stand from the camera.

- **Lighting and Environmental Factors:**

To achieve reliable pedestrian detection the system needs protection against various lighting conditions including night and day lighting together with shadowed and glared or low-lighting environments. Common environmental conditions including weather alongside fog and rain create major obstacles for the model's functional ability.

- **Blockages:**

During actual pedestrian situations people walking in front of cameras get partially hidden by various objects or humans. The detection process becomes considerably more complicated when the system needs to detect partially blocked pedestrians who are hidden by trees and vehicles or other walking pedestrians.

- **Multiple Pedestrians in Complex Scenes:**

Installation of surveillance systems requires pedestrian detection capabilities to track several subjects within busy and complicated outdoor areas. Detecting many pedestrians within a single picture becomes harder to detect because their identification and distinction becomes harder when they stand close together.

- **Scalability and Speed:**

The detection system should process large datasets at high speed since it operates with video feeds or real-time data streams. Extremely complex processing takes too long to execute that makes systems not properly fit for immediate usage in live monitoring or active applications.

1.2. Objectives

- The main goal focuses on creating an efficient system for detecting pedestrians that incorporates deep learning approaches.

- Real-time pedestrian object detection will be performed with the use of the YOLO (You Only Look Once) algorithm.
- To preprocess resizing as well as normalization while converting annotations in order to improve model performance on datasets containing labeled pedestrian information.
- The model needs training until it reaches its maximum performance level in pedestrian detection.
- The detection system needs to operate in real time when processing video feeds in order to fulfill live monitoring requirements.
- Evaluation occurs to address challenging situations involving obstacles as well as changing positions and lighting patterns for better detection precision.
- The system will generate two types of visual display which shows detected pedestrians with bounding boxes and their corresponding confidence scores.

1.3. Dataset Overview

This project uses the Pedestrian Detection Dataset from Kaggle as its database. The Pedestrian Detection Dataset presents images with pedestrians situated in different settings to enable testing and training of pedestrian detection models. The dataset contains labeled images with bounding boxes drawing a frame around human bodies. (Chandran, 2020)

Key Details of the Dataset:

- The database consists of images which consists of different conditions that include different pedestrian backgrounds and lighting situations and body positions. Each image contains annotations which show bounding boxes to specify where pedestrians exist.
- Manually labeled bounding boxes exist for every pedestrian within the dataset images. The model learns the image locations of pedestrians as a result which boosts its detection accuracy.
- The dataset contains images of pedestrians recorded across different environments under mixed lighting situations while showing partial pedestrian view which allows the development of a proper pedestrian detection model.
- The dataset functions best for teaching object detection solutions particularly designed to recognize pedestrians within the environment of real traffic or public dense areas.
- This dataset is used for training the pedestrian detection system it enables the system to recognize pedestrians within a lot of different poses and clothing options and also background settings which increases the overall understanding capacity for real-time detection.

2. Methodology

2.1. Algorithm Used

The project uses **YOLOv5,** one of the most popular object-detection models currently available. YOLOv5 divides the picture into a grid and then predicts bounding boxes and class labels simultaneously for each grid cell. This makes it very fast and effective for real-time applications like video processing. Rather than inspecting the image multiple times, YOLO looks at the entire image in a single pass and detects multiple objects at a time. (YOLO: Algorithm for Object Detection Explained [+Examples], 2023)

This project was developed using the 'nano' version of YOLOv5 which is smaller and lighter, the YOLOv5n was designed with speed and ease of running in mind, even on systems that don't have a top-of-the-line graphics card. YOLOv5n is smaller however, it still provides good accuracy and is very much capable of pedestrian detection across various environments. (Solawetz, 2020)

The custom dataset was used to train the model for two classes, 'person' and 'person-like'. Therefore, the algorithm allows the system to recognize and keep pedestrians through bounding boxes inside images and videos given to it.
More

2.2. Frameworks and Libraries Used:

2.2.1. PyTorch:

PyTorch is an open-source deep learning framework that was developed by Meta AI. It gives a flexible, intuitive platform for working on neural network building and training since it has a dynamic computation graph and support for a GPU acceleration. PyTorch is popularly used in applications related to computer vision, natural language processing, and many more such machine learning tasks. (Yasar, n.d.)

Figure 2: Pytorch

2.2.2. OpenCV:

OpenCV is an open-source programming library applicable in fields like computer vision and machine learning. It contains some very important useful tools and functions applied for real-time computer vision applications. (About, 2025)

Figure 3: OpenCV

2.2.3. NumPy:

NumPy is an open-source extension library for python. It gives python the ability to handle large multidimensional arrays and matrices along with a collection of mathematical functions operating on these arrays. It is very useful for data manipulation and mathematical operations in ML. (NumPy: the absolute basics for beginners, 2025)

Figure 4: Numpy

2.2.4. Pandas:

Pandas is an open-source tool that is widely popular for data manipulation and analysis. It provides great help in most of the functionality performed in structured data with structures like DataFrames or Series. It is an excellent tool for cleaning, transforming, and analyzing data using machine learning and data science workflows. (pandas documentation, 2024)

Figure 5: pandas

2.2.5. Google Colab:

Google Colab is a free cloud-based service by Google. It enables you to code in Python and execute it for the same within an interactive web environment. It has access for a variety of computing sources, including GPUs and TPUs, without any setup. Colab is integrated with Google Drive for easy sharing and collaborative work on Notebooks. (How to use Google Colab, 2025)

Figure 6: Google colab

2.3. Techniques Used

2.3.1. Transfer Learning

Model training started with pre-trained weights of a model which was trained already on a large dataset, that is COCO. This is called transfer learning, which helps in improving accuracy and reducing training time.

2.3.2. Label Conversion

The dataset had XML annotations but YOLOv5 needs a specific text format. So the labels were converted to YOLO format using a Python script, with normalized coordinates for each bounding box.

2.3.3. Data Preprocessing

Images were resized to 416x416, which is the required size for YOLOv5. This makes things uniform during training and increases detection speed.

2.3.4. Video Frame Inference

The video was processed frame by frame. The model drew bounding boxes around the detected pedestrians' frames for each frame. These frames were streamed live and also saved in an output video file.

3. Implementation and Results

3.1. Data Collection and Annotation

- The data set was collected from internet (Kaggle). This dataset had images and videos showing persons or pedestrians walking in different places.
- These images were used to train the model to recognize pedestrians.
- Annotations were saved to. XML format that basically indicated where the person was in the image.

3.2. Converting Labels to YOLO Format

- YOLOv5 doesn't support .xml files, so needed to convert them into a .txt format.
- At first, a Python script was written that would convert the .xml files into YOLO format.
- The .txt file describes the class (e.g. "person") and box position within the image.
- The positions are then saved with numbers between 0 and 1.
- Each image was given a .txt file with the exact name of the image.

convert_voc_to_yolo.py × data.yaml detect_pedestrians.py detect

```python
# Importing necessary libraries
import os
import xml.etree.ElementTree as ET
# For parsing XML files containing annotation data
#setting the pedestrian classes
classes = ["person", "person-like"]

splits = ["train", "val", "test"]
base_path = "dataset"

# converting bounding box from XML format to YOLO format
8 usages (7 dynamic)
def convert_bbox(size, box):
    dw = 1. / size[0]
    dh = 1. / size[1]

    x_center = (box[0] + box[1]) / 2.0
    y_center = (box[2] + box[3]) / 2.0
    w = box[1] - box[0]
    h = box[3] - box[2]

    return (x_center * dw, y_center * dh, w * dw, h * dh)

# Loop through each dataset split (train, val, test)
for split in splits:
    # Path to the XML label files for this split
    xml_folder = os.path.join(base_path, "labels", split)

    # Loop through each file in the XML folder
    for file in os.listdir(xml_folder):
        # Skip non-XML files
```

Figure 7: Python script used to convert XML annotations to YOLO format by calculating normalized bounding box coordinates.

Description:

This displays a Python script that converts the XML annotation files in VOC format into.txt files compatible with YOLO. The script reads the bounding box values, normalizes them, and saves the output in YOLO format for training. It also does a loop through the train, val, and test label folders.

3.3. Data Pre-Processing

- All images had to be adjusted to 416x416 pixels so it would be readable by YOLOv5.
- Data was divided into three parts: training data, validation data, and testing data.
- It was checked whether each image had a corresponding label file ending with .txt.
- The class names were supposed to be added in a file named data.yaml. This file was supposed to inform YOLOv5 how many classes are there and where the images are located.

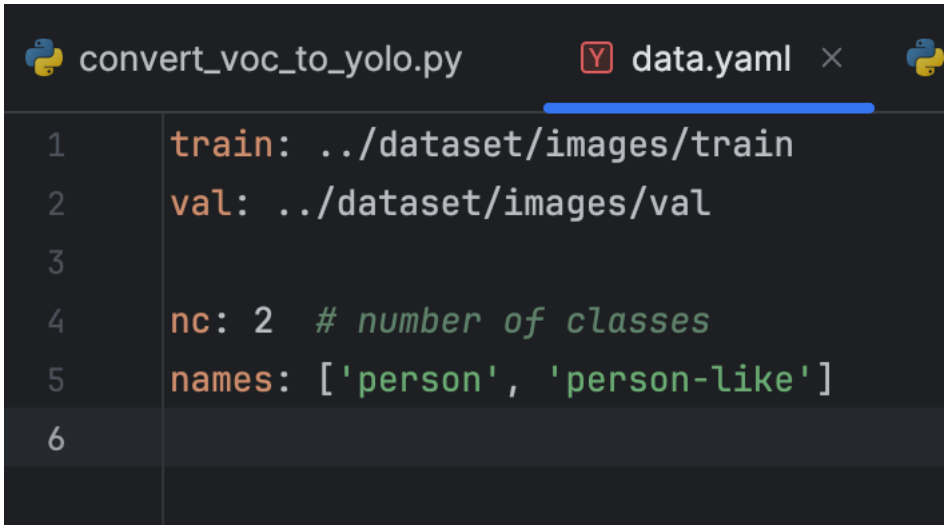

Figure 8: data.yaml file used to configure dataset paths and class names for training the YOLOv5 model.

Description:

Thhe contents of the data.yaml file which is a important file needed for training YOLOv5. It has the paths of the training and validation images, the number of object classes 'nc', along with their names. The classes here are 'person' and 'person-like'.

3.4. Feature Extraction Using YOLOv5

- Slightly smaller version of YOLOv5 was used here referred to as yolov5n. The reason is that it is very fast and works well on a common laptop.
- The special layers that YOLOv5 uses to detect the patterns in the images are known as convolutional layers.
- These patterns assist in understanding where the pedestrians are within an image.
- It looks at one full image once and gives a result in one step.

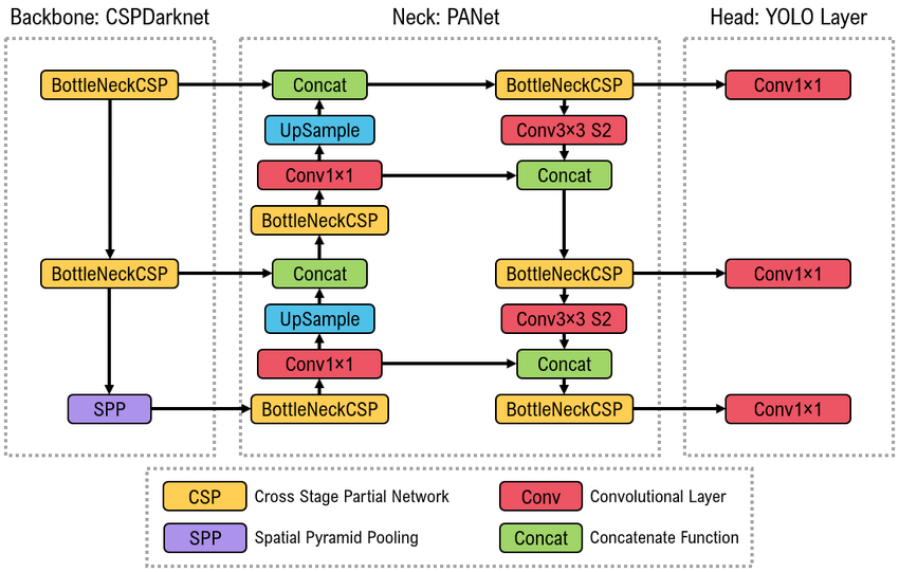

Figure 9: YOLOv5 model architecture showing three main parts

Description:

The architecture of the YOLOv5 model is explained through this diagram. The representation is in three main parts: the Backbone (to extract features), Neck (to combine these features), and Head (to make final predictions). Each block in the diagram represents specified layers used for processing image information.

3.5. Model Training

- Model training was done by using Google Colab, which provides free GPU to speed up the training.
- The Command used had img (640), batch (16), epochs (20)
- The model also used pre-trained knowledge, coming from yolov5n.pt, to detect pedestrians.
- During training, the model was taught how near its predictions were to the real boxes.
- The best result after training was saved in a file named best.pt.

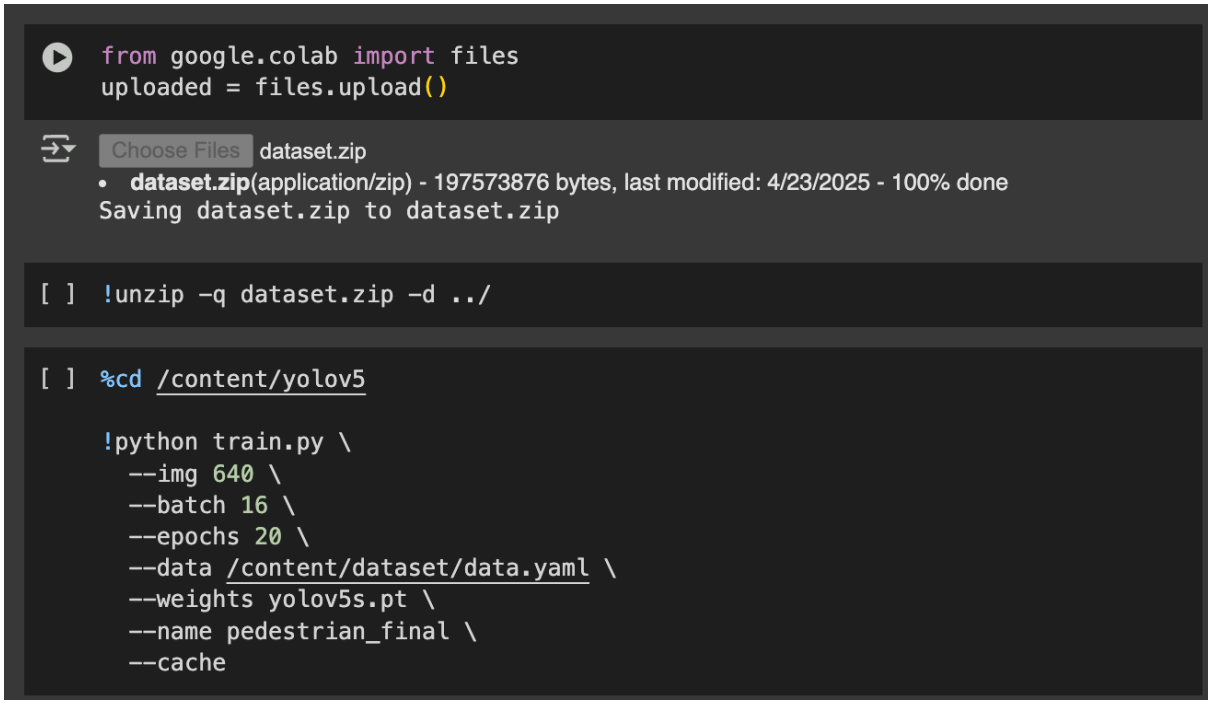

Figure 10: Training the YOLOv5 model in Google Colab using the parameters and dataset.

Description:

This shows the training command, which was executed in Google Colab. The size of the image, batch size, epochs, dataset path, and model weights are the relevant settings here.

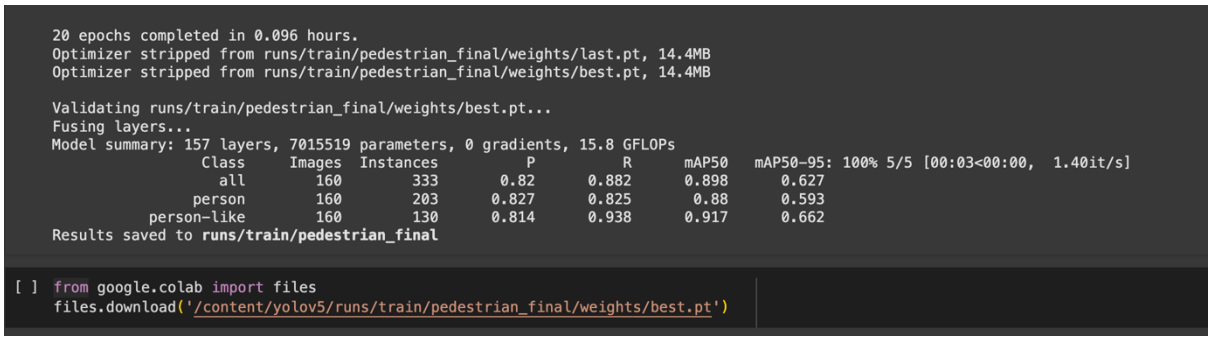

Figure 11: Validation results after training.

Description:

Summary of the output during training, where it gives back model performance on validation, precision, recall, and mAP for each class.

3.6. Running The Detection Video

- A Python script was developed to test the model with a real video, pedestrians.mp4(Sample Video).

- The script opened the video, read it frame by frame, and used the model for pedestrians detection.
- It drew bounding boxes around the people in the video.
- The resulting video was saved as output_pedestrians.mp4.
- Detection worked smoothly, and the pedestrians detections were shown correctly.

Figure 12: Pedestrian detection results from a sample video showing bounding boxes and confidence scores for each detected person.

Description:

Spotlights the live output video for pedestrian detection. The bounding boxes and respective confidence scores for various pedestrians were displayed across the screenshot. The model was 100% correct in detecting and labeling each person during post-training to efficiency.

4. Challenges and Future Improvements

4.1. Challenges Faced

- **Framework Confusion**

Even though the project was planned to be done with TensorFlow, the model that was selected (YOLOv5) is officially built on PyTorch. This led to the need to change the framework to a PyTorch environment.

- **Annotation Format Conversion**

The dataset annotations were originally in XML format, which made some of the data corrupted and not easily compatible with YOLOv5. For that reason, a custom script had to be developed that converted the annotations correctly to .txt files, which YOLOv5 could work with.

- **Strict Dataset Structure**

Really strict in such a way that the folders and files themselves should have a certain structure for training with YOLOv5 to work.

If there is a minor error in the paths of the folders or in the names of the files, or they are not correctly applied, the training procedure will fail or else return errors.

- **Slow Training on CPU**

Model training is very, very slow on local machines without a GPU. It was sorted out by using Google Colab because it offers us free GPU resources to train the models at speed.

- **Initial Detection Errors**

During first tests, the model does not detect a few pedestrians or gives them inaccurate bounding boxes. The main reason for this is the wrong label format or less training data.

- **Delayed Video Output**

On video detections, I realized that the output playback became very slow and less responsive in nature. This was caused by the fact that every frame took a longer time to process through the model in real time.

4.2. Future Improvements

- **Increasing the Dataset**

Improving the performance of the model might be obtained by more pictures regarding pedestrians, taken under different conditions of light and angles.

- **Upgrading to a Larger YOLOv5 Model**

Training on a more powerful YOLOv5 (YOLOv5s, YOLOv5m) might improve the accuracy of the detections.

- **Adding Real-time People Counting**

Probably a very simple counting system might be added so that could show the number of pedestrians being detected in every video frame.

- **Increase the Processing Speed**

Could skip some frames for the video to run almost close to real-time, along with a lighter preprocessing.

- **Web or Mobile Integration**

The system could be extended to work as a web or mobile application to make it more usable.

- **Object Tracking Feature**

Could combine a tracking algorithm that would follow pedestrians identified in a single frame over multiple frames of the video.

5. Conclusion

The project showcased a demonstration through computer vision and deep learning techniques applied to real-time pedestrian detection by using the YOLOv5 model. The system functions as a pedestrian recognition system which marks pedestrian subjects with bounding shapes and accompanies them with confidence percentages. The dataset collection followed a labeling process for training purposes which benefited from the Google Colab for improved speed.

The complete project implementation involved each step from dataset development through label transformation until model training and video execution. The YOLOv5n model achieved selection because of its efficient processing speed running on various system configurations. The detection model delivered precise pedestrian recognition in all different poses accompanied by different lighting conditions and various background environments.

The project can be expanded through the utilization of bigger datasets together with stronger models and additional features for people counting and tracking.

6. Bibliography

Sharma, A. (2021, Dec 4). *Pedestrian Detection using HOGs in Python — easy project— with source code*. Retrieved from Medium: https://sharmaji27.medium.com/pedestrian-detection-using-hogs-in-python-5ef014c3f741

Ultralytics YOLO Docs . (2025). Retrieved from Ultralytics: https://docs.ultralytics.com/

Why TensorFlow. (2025). Retrieved from TesnorFlow: https://www.tensorflow.org/about

Chandran, K. N. (2020). *Pedestrian Detection Data set*. Retrieved from kaggle: https://www.kaggle.com/datasets/karthika95/pedestrian-detection

About. (2025). Retrieved from OpenCV: https://opencv.org/about/

NumPy: the absolute basics for beginners. (2025). Retrieved from Numpy: https://numpy.org/doc/2.2/user/absolute_beginners.html

Matplotlib. (2025). Retrieved from Quick start guide: https://matplotlib.org/stable/users/explain/quick_start.html#a-simple-example

pandas documentation. (2024, sep 20). Retrieved from pandas: https://pandas.pydata.org/pandas-docs/stable/

Yasar, K. (n.d.). *Definition PyTorch*. Retrieved from TechTarget : https://www.techtarget.com/searchenterpriseai/definition/PyTorch

YOLO: Algorithm for Object Detection Explained [+Examples]. (2023, jan 17). Retrieved from v7labs.: https://www.v7labs.com/blog/yolo-object-detection

Solawetz, J. (2020, Jan 29). *What is YOLOv5? A Guide for Beginners.* Retrieved from roboflow: https://blog.roboflow.com/yolov5-improvements-and-evaluation/

How to use Google Colab. (2025, jan 2). Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/how-to-use-google-colab/

7. Appendix

7.1. Algorithms Used

The system uses YOLO (You Only Look Once) which represents a real-time object detection algorithm. The detection system based on YOLO provides effective and quick performance to identify pedestrians in picture-based or video-based applications. The model produces results along with bounding boxes while executing the single-forward network pass making it optimal for real-time applications due to its fast and accurate performance.

Specific training of the model is done using data from the Pedestrian Detection Dataset on Kaggle and focuses on pedestrian identification. (Ultralytics YOLO Docs , 2025)

Advantages of using YOLO

- Real-time object detection capability.
- Doing classification and finding location at the same time
- Availability of pre-trained weights for faster learning or quicker results
- The detection system provides high precision for detecting small objects that may be partly hidden from view.

To go up Click Here