



islington college
(इरिलहटन 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

14 Pages

2,645 Words

14,871 Characters

Download Date

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

File Name

13.docx

File Size

17.4 KB



Page 1 of 17 - Cover Page

Submission ID trn:oid::3618:92660449



Page 2 of 17 - Integrity Overview

Submission ID trn:oid::3618:92660449

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

1.	Introduction	1
1.1.	Problem Statement.....	2
1.2.	Objectives	2
1.3.	Dataset Overview.....	3
2.	Methodology	4
2.1.	Algorithm Used.....	4
2.2.	Frameworks and Libraries Used:	4
2.2.1.	PyTorch:	4
2.2.2.	OpenCV:.....	5
2.2.3.	NumPy:.....	5
2.2.4.	Pandas:.....	5
2.2.5.	Google Colab:	6
2.3.	Techniques Used.....	6
2.3.1.	Transfer Learning.....	6
2.3.2.	Label Conversion.....	6
2.3.3.	Data Preprocessing.....	6
2.3.4.	Video Frame Inference	6
3.	Implementation and Results	7
3.1.	Data Collection and Annotation.....	7
3.2.	Converting Labels to YOLO Format	7
3.3.	Data Pre-Processing.....	9
3.4.	Feature Extraction Using YOLOv5.....	9
3.5.	Model Training	10
3.6.	Running The Detection Video	11
4.	Challenges and Future Improvements	12
4.1.	Challenges Faced	12
4.2.	Future Improvements	13
5.	Conclusion.....	14
6.	Bibliography.....	15
7.	Appendix	16
7.1.	Algorithms Used	16

Table of Figures

Figure 1: (Sharma, 2021)	1
Figure 2: Pytorch	4
Figure 3: OpenCV	5
Figure 4: Numpy	5
Figure 5: pandas	5
Figure 6: Google colab	6
Figure 7: Python script used to convert XML annotations to YOLO format by calculating normalized bounding box coordinates.	8
Figure 8: data.yaml file used to configure dataset paths and class names for training the YOLOv5 model.	9
Figure 9: YOLOv5 model architecture showing three main parts	10
Figure 10: Training the YOLOv5 model in Google Colab using the parameters and dataset.	11
Figure 11: Validation results after training.	11
Figure 12: Pedestrian detection results from a sample video showing bounding boxes and confidence scores for each detected person.	12

1. Introduction

Pedestrian detection is an 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.

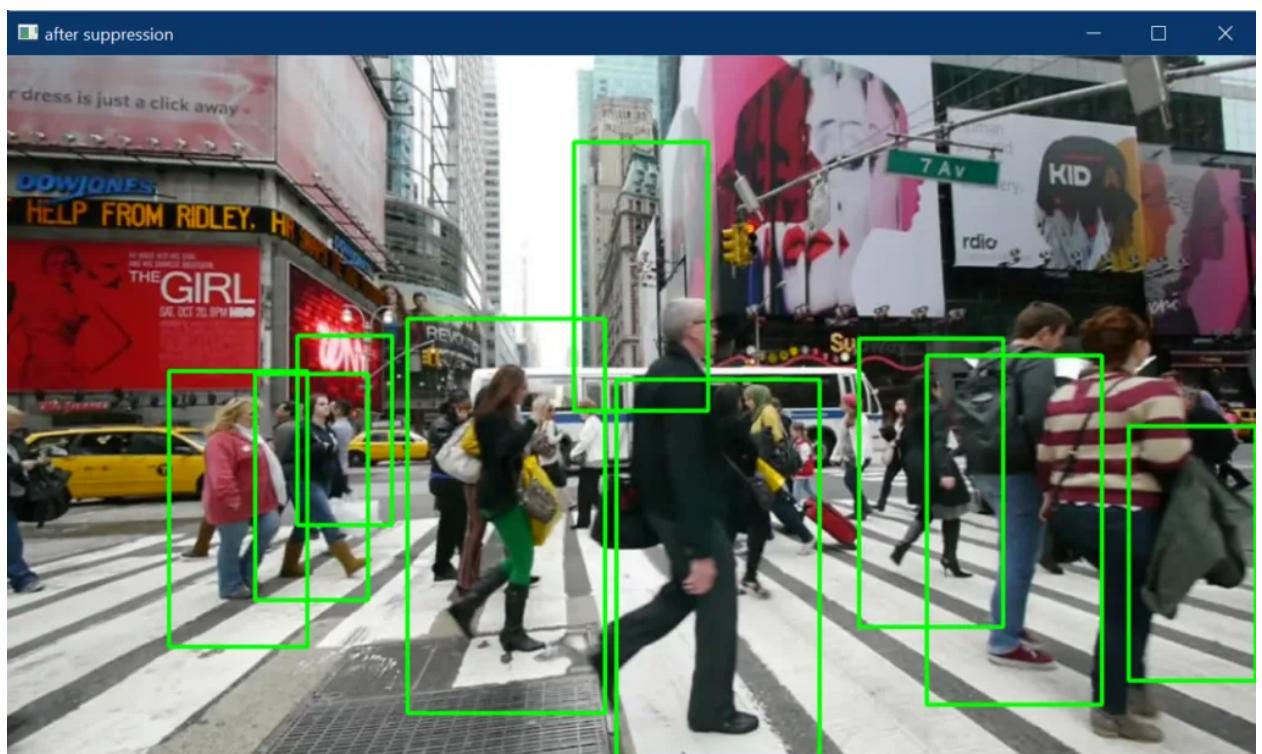


Figure 1: (Sharma, 2021)

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.

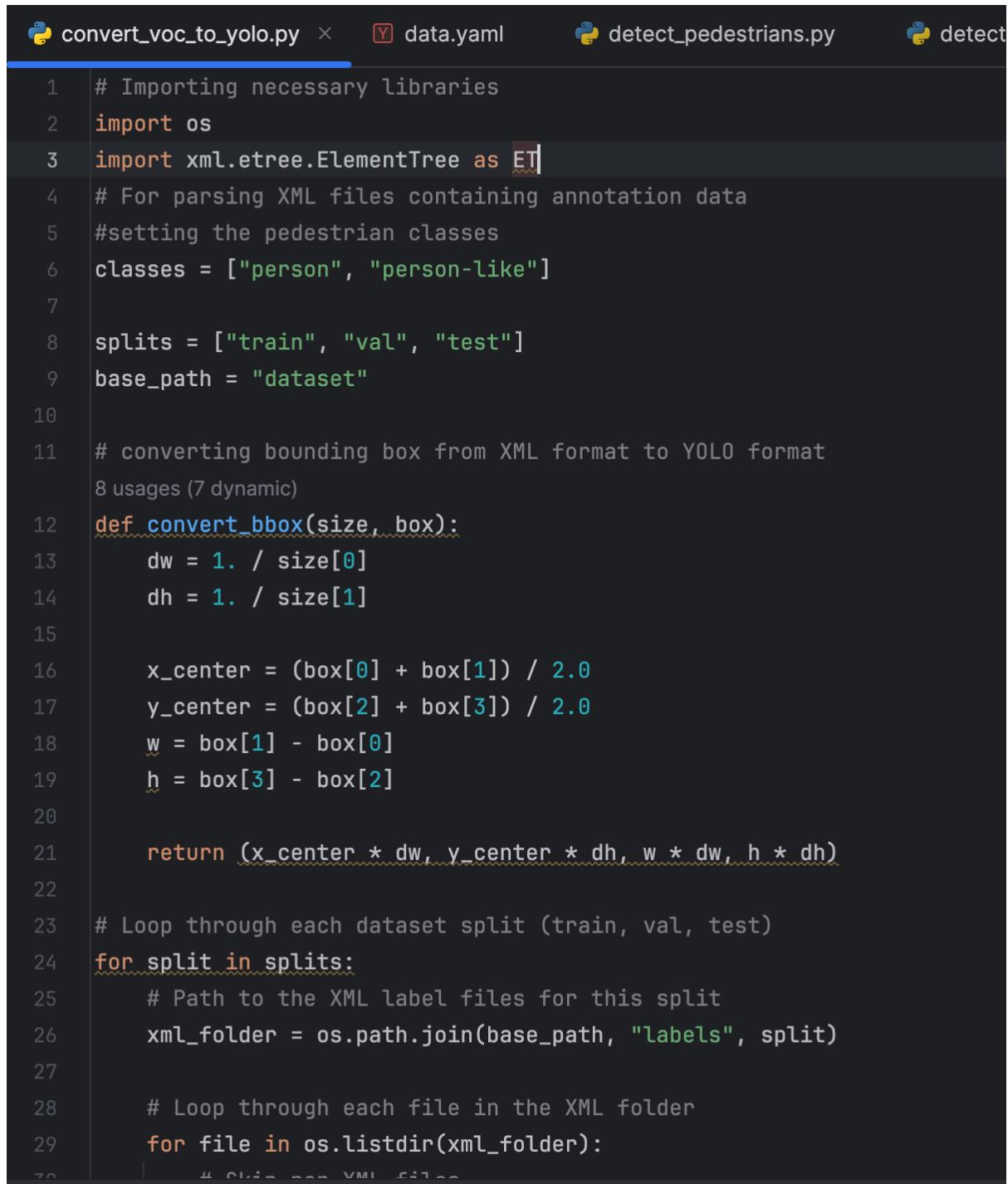
3. Implementation and Results

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
  1 # Importing necessary libraries
  2 import os
  3 import xml.etree.ElementTree as ET
  4 # For parsing XML files containing annotation data
  5 #setting the pedestrian classes
  6 classes = ["person", "person-like"]
  7
  8 splits = ["train", "val", "test"]
  9 base_path = "dataset"
 10
 11 # converting bounding box from XML format to YOLO format
 12 # 8 usages (7 dynamic)
 13 def convert_bbox(size, box):
 14     dw = 1. / size[0]
 15     dh = 1. / size[1]
 16
 17     x_center = (box[0] + box[1]) / 2.0
 18     y_center = (box[2] + box[3]) / 2.0
 19     w = box[1] - box[0]
 20     h = box[3] - box[2]
 21
 22     return (x_center * dw, y_center * dh, w * dw, h * dh)
 23
 24 # Loop through each dataset split (train, val, test)
 25 for split in splits:
 26     # Path to the XML label files for this split
 27     xml_folder = os.path.join(base_path, "labels", split)
 28
 29     # Loop through each file in the XML folder
 30     for file in os.listdir(xml_folder):
 31         if file.endswith("xml"):
 32             print(f"Processing {file} in {split} split")
 33
 34             # Parse XML file
 35             tree = ET.parse(os.path.join(xml_folder, file))
 36             root = tree.getroot()
 37
 38             # Extract bounding box information
 39             for obj in root.findall("object"):
 40                 name = obj.find("name").text
 41                 if name in classes:
 42                     bndbox = obj.find("bndbox")
 43                     x_min = int(bndbox.find("xmin").text)
 44                     y_min = int(bndbox.find("ymin").text)
 45                     x_max = int(bndbox.find("xmax").text)
 46                     y_max = int(bndbox.find("ymax").text)
 47
 48                     # Convert to YOLO format
 49                     normalized_box = convert_bbox(size, [x_min, x_max, y_min, y_max])
 50
 51                     # Save to output file
 52                     output_file = os.path.join(base_path, "labels", split, file.replace("xml", "txt"))
 53                     with open(output_file, "a") as f:
 54                         f.write(f"{normalized_box}\n")
 55
 56             print(f"Completed processing {file} in {split} split")
 57
 58             # Break out of inner loop
 59             break
 60
 61     print(f"Completed processing {split} split")
 62
 63     # Break out of outer loop
 64     break
 65
 66
 67             # Break out of inner loop
 68             break
 69
 70     print(f"Completed processing {split} split")
 71
 72     # Break out of outer loop
 73     break
 74
 75
 76             # Break out of inner loop
 77             break
 78
 79     print(f"Completed processing {split} split")
 80
 81     # Break out of outer loop
 82     break
 83
 84
 85             # Break out of inner loop
 86             break
 87
 88     print(f"Completed processing {split} split")
 89
 90     # Break out of outer loop
 91     break
 92
 93
 94             # Break out of inner loop
 95             break
 96
 97     print(f"Completed processing {split} split")
 98
 99     # Break out of outer loop
 100    break
 101
 102
 103             # Break out of inner loop
 104             break
 105
 106     print(f"Completed processing {split} split")
 107
 108     # Break out of outer loop
 109     break
 110
 111
 112             # Break out of inner loop
 113             break
 114
 115     print(f"Completed processing {split} split")
 116
 117     # Break out of outer loop
 118     break
 119
 120
 121             # Break out of inner loop
 122             break
 123
 124     print(f"Completed processing {split} split")
 125
 126     # Break out of outer loop
 127     break
 128
 129
 130             # Break out of inner loop
 131             break
 132
 133     print(f"Completed processing {split} split")
 134
 135     # Break out of outer loop
 136     break
 137
 138
 139             # Break out of inner loop
 140             break
 141
 142     print(f"Completed processing {split} split")
 143
 144     # Break out of outer loop
 145     break
 146
 147
 148             # Break out of inner loop
 149             break
 150
 151     print(f"Completed processing {split} split")
 152
 153     # Break out of outer loop
 154     break
 155
 156
 157             # Break out of inner loop
 158             break
 159
 160     print(f"Completed processing {split} split")
 161
 162     # Break out of outer loop
 163     break
 164
 165
 166             # Break out of inner loop
 167             break
 168
 169     print(f"Completed processing {split} split")
 170
 171     # Break out of outer loop
 172     break
 173
 174
 175             # Break out of inner loop
 176             break
 177
 178     print(f"Completed processing {split} split")
 179
 180     # Break out of outer loop
 181     break
 182
 183
 184             # Break out of inner loop
 185             break
 186
 187     print(f"Completed processing {split} split")
 188
 189     # Break out of outer loop
 190     break
 191
 192
 193             # Break out of inner loop
 194             break
 195
 196     print(f"Completed processing {split} split")
 197
 198     # Break out of outer loop
 199     break
 200
 201
 202             # Break out of inner loop
 203             break
 204
 205     print(f"Completed processing {split} split")
 206
 207     # Break out of outer loop
 208     break
 209
 210
 211             # Break out of inner loop
 212             break
 213
 214     print(f"Completed processing {split} split")
 215
 216     # Break out of outer loop
 217     break
 218
 219
 220             # Break out of inner loop
 221             break
 222
 223     print(f"Completed processing {split} split")
 224
 225     # Break out of outer loop
 226     break
 227
 228
 229             # Break out of inner loop
 230             break
 231
 232     print(f"Completed processing {split} split")
 233
 234     # Break out of outer loop
 235     break
 236
 237
 238             # Break out of inner loop
 239             break
 240
 241     print(f"Completed processing {split} split")
 242
 243     # Break out of outer loop
 244     break
 245
 246
 247             # Break out of inner loop
 248             break
 249
 250     print(f"Completed processing {split} split")
 251
 252     # Break out of outer loop
 253     break
 254
 255
 256             # Break out of inner loop
 257             break
 258
 259     print(f"Completed processing {split} split")
 260
 261     # Break out of outer loop
 262     break
 263
 264
 265             # Break out of inner loop
 266             break
 267
 268     print(f"Completed processing {split} split")
 269
 270     # Break out of outer loop
 271     break
 272
 273
 274             # Break out of inner loop
 275             break
 276
 277     print(f"Completed processing {split} split")
 278
 279     # Break out of outer loop
 280     break
 281
 282
 283             # Break out of inner loop
 284             break
 285
 286     print(f"Completed processing {split} split")
 287
 288     # Break out of outer loop
 289     break
 290
 291
 292             # Break out of inner loop
 293             break
 294
 295     print(f"Completed processing {split} split")
 296
 297     # Break out of outer loop
 298     break
 299
 300
 301             # Break out of inner loop
 302             break
 303
 304     print(f"Completed processing {split} split")
 305
 306     # Break out of outer loop
 307     break
 308
 309
 310             # Break out of inner loop
 311             break
 312
 313     print(f"Completed processing {split} split")
 314
 315     # Break out of outer loop
 316     break
 317
 318
 319             # Break out of inner loop
 320             break
 321
 322     print(f"Completed processing {split} split")
 323
 324     # Break out of outer loop
 325     break
 326
 327
 328             # Break out of inner loop
 329             break
 330
 331     print(f"Completed processing {split} split")
 332
 333     # Break out of outer loop
 334     break
 335
 336
 337             # Break out of inner loop
 338             break
 339
 340     print(f"Completed processing {split} split")
 341
 342     # Break out of outer loop
 343     break
 344
 345
 346             # Break out of inner loop
 347             break
 348
 349     print(f"Completed processing {split} split")
 350
 351     # Break out of outer loop
 352     break
 353
 354
 355             # Break out of inner loop
 356             break
 357
 358     print(f"Completed processing {split} split")
 359
 360     # Break out of outer loop
 361     break
 362
 363
 364             # Break out of inner loop
 365             break
 366
 367     print(f"Completed processing {split} split")
 368
 369     # Break out of outer loop
 370     break
 371
 372
 373             # Break out of inner loop
 374             break
 375
 376     print(f"Completed processing {split} split")
 377
 378     # Break out of outer loop
 379     break
 380
 381
 382             # Break out of inner loop
 383             break
 384
 385     print(f"Completed processing {split} split")
 386
 387     # Break out of outer loop
 388     break
 389
 390
 391             # Break out of inner loop
 392             break
 393
 394     print(f"Completed processing {split} split")
 395
 396     # Break out of outer loop
 397     break
 398
 399
 400             # Break out of inner loop
 401             break
 402
 403     print(f"Completed processing {split} split")
 404
 405     # Break out of outer loop
 406     break
 407
 408
 409             # Break out of inner loop
 410             break
 411
 412     print(f"Completed processing {split} split")
 413
 414     # Break out of outer loop
 415     break
 416
 417
 418             # Break out of inner loop
 419             break
 420
 421     print(f"Completed processing {split} split")
 422
 423     # Break out of outer loop
 424     break
 425
 426
 427             # Break out of inner loop
 428             break
 429
 430     print(f"Completed processing {split} split")
 431
 432     # Break out of outer loop
 433     break
 434
 435
 436             # Break out of inner loop
 437             break
 438
 439     print(f"Completed processing {split} split")
 440
 441     # Break out of outer loop
 442     break
 443
 444
 445             # Break out of inner loop
 446             break
 447
 448     print(f"Completed processing {split} split")
 449
 450     # Break out of outer loop
 451     break
 452
 453
 454             # Break out of inner loop
 455             break
 456
 457     print(f"Completed processing {split} split")
 458
 459     # Break out of outer loop
 460     break
 461
 462
 463             # Break out of inner loop
 464             break
 465
 466     print(f"Completed processing {split} split")
 467
 468     # Break out of outer loop
 469     break
 470
 471
 472             # Break out of inner loop
 473             break
 474
 475     print(f"Completed processing {split} split")
 476
 477     # Break out of outer loop
 478     break
 479
 480
 481             # Break out of inner loop
 482             break
 483
 484     print(f"Completed processing {split} split")
 485
 486     # Break out of outer loop
 487     break
 488
 489
 490             # Break out of inner loop
 491             break
 492
 493     print(f"Completed processing {split} split")
 494
 495     # Break out of outer loop
 496     break
 497
 498
 499             # Break out of inner loop
 500             break
 501
 502     print(f"Completed processing {split} split")
 503
 504     # Break out of outer loop
 505     break
 506
 507
 508             # Break out of inner loop
 509             break
 510
 511     print(f"Completed processing {split} split")
 512
 513     # Break out of outer loop
 514     break
 515
 516
 517             # Break out of inner loop
 518             break
 519
 520     print(f"Completed processing {split} split")
 521
 522     # Break out of outer loop
 523     break
 524
 525
 526             # Break out of inner loop
 527             break
 528
 529     print(f"Completed processing {split} split")
 530
 531     # Break out of outer loop
 532     break
 533
 534
 535             # Break out of inner loop
 536             break
 537
 538     print(f"Completed processing {split} split")
 539
 540     # Break out of outer loop
 541     break
 542
 543
 544             # Break out of inner loop
 545             break
 546
 547     print(f"Completed processing {split} split")
 548
 549     # Break out of outer loop
 550     break
 551
 552
 553             # Break out of inner loop
 554             break
 555
 556     print(f"Completed processing {split} split")
 557
 558     # Break out of outer loop
 559     break
 560
 561
 562             # Break out of inner loop
 563             break
 564
 565     print(f"Completed processing {split} split")
 566
 567     # Break out of outer loop
 568     break
 569
 570
 571             # Break out of inner loop
 572             break
 573
 574     print(f"Completed processing {split} split")
 575
 576     # Break out of outer loop
 577     break
 578
 579
 580             # Break out of inner loop
 581             break
 582
 583     print(f"Completed processing {split} split")
 584
 585     # Break out of outer loop
 586     break
 587
 588
 589             # Break out of inner loop
 590             break
 591
 592     print(f"Completed processing {split} split")
 593
 594     # Break out of outer loop
 595     break
 596
 597
 598             # Break out of inner loop
 599             break
 600
 601     print(f"Completed processing {split} split")
 602
 603     # Break out of outer loop
 604     break
 605
 606
 607             # Break out of inner loop
 608             break
 609
 610     print(f"Completed processing {split} split")
 611
 612     # Break out of outer loop
 613     break
 614
 615
 616             # Break out of inner loop
 617             break
 618
 619     print(f"Completed processing {split} split")
 620
 621     # Break out of outer loop
 622     break
 623
 624
 625             # Break out of inner loop
 626             break
 627
 628     print(f"Completed processing {split} split")
 629
 630     # Break out of outer loop
 631     break
 632
 633
 634             # Break out of inner loop
 635             break
 636
 637     print(f"Completed processing {split} split")
 638
 639     # Break out of outer loop
 640     break
 641
 642
 643             # Break out of inner loop
 644             break
 645
 646     print(f"Completed processing {split} split")
 647
 648     # Break out of outer loop
 649     break
 650
 651
 652             # Break out of inner loop
 653             break
 654
 655     print(f"Completed processing {split} split")
 656
 657     # Break out of outer loop
 658     break
 659
 660
 661             # Break out of inner loop
 662             break
 663
 664     print(f"Completed processing {split} split")
 665
 666     # Break out of outer loop
 667     break
 668
 669
 670             # Break out of inner loop
 671             break
 672
 673     print(f"Completed processing {split} split")
 674
 675     # Break out of outer loop
 676     break
 677
 678
 679             # Break out of inner loop
 680             break
 681
 682     print(f"Completed processing {split} split")
 683
 684     # Break out of outer loop
 685     break
 686
 687
 688             # Break out of inner loop
 689             break
 690
 691     print(f"Completed processing {split} split")
 692
 693     # Break out of outer loop
 694     break
 695
 696
 697             # Break out of inner loop
 698             break
 699
 700     print(f"Completed processing {split} split")
 701
 702     # Break out of outer loop
 703     break
 704
 705
 706             # Break out of inner loop
 707             break
 708
 709     print(f"Completed processing {split} split")
 710
 711     # Break out of outer loop
 712     break
 713
 714
 715             # Break out of inner loop
 716             break
 717
 718     print(f"Completed processing {split} split")
 719
 720     # Break out of outer loop
 721     break
 722
 723
 724             # Break out of inner loop
 725             break
 726
 727     print(f"Completed processing {split} split")
 728
 729     # Break out of outer loop
 730     break
 731
 732
 733             # Break out of inner loop
 734             break
 735
 736     print(f"Completed processing {split} split")
 737
 738     # Break out of outer loop
 739     break
 740
 741
 742             # Break out of inner loop
 743             break
 744
 745     print(f"Completed processing {split} split")
 746
 747     # Break out of outer loop
 748     break
 749
 750
 751             # Break out of inner loop
 752             break
 753
 754     print(f"Completed processing {split} split")
 755
 756     # Break out of outer loop
 757     break
 758
 759
 760             # Break out of inner loop
 761             break
 762
 763     print(f"Completed processing {split} split")
 764
 765     # Break out of outer loop
 766     break
 767
 768
 769             # Break out of inner loop
 770             break
 771
 772     print(f"Completed processing {split} split")
 773
 774     # Break out of outer loop
 775     break
 776
 777
 778             # Break out of inner loop
 779             break
 780
 781     print(f"Completed processing {split} split")
 782
 783     # Break out of outer loop
 784     break
 785
 786
 787             # Break out of inner loop
 788             break
 789
 790     print(f"Completed processing {split} split")
 791
 792     # Break out of outer loop
 793     break
 794
 795
 796             # Break out of inner loop
 797             break
 798
 799     print(f"Completed processing {split} split")
 800
 801     # Break out of outer loop
 802     break
 803
 804
 805             # Break out of inner loop
 806             break
 807
 808     print(f"Completed processing {split} split")
 809
 810     # Break out of outer loop
 811     break
 812
 813
 814             # Break out of inner loop
 815             break
 816
 817     print(f"Completed processing {split} split")
 818
 819     # Break out of outer loop
 820     break
 821
 822
 823             # Break out of inner loop
 824             break
 825
 826     print(f"Completed processing {split} split")
 827
 828     # Break out of outer loop
 829     break
 830
 831
 832             # Break out of inner loop
 833             break
 834
 835     print(f"Completed processing {split} split")
 836
 837     # Break out of outer loop
 838     break
 839
 840
 841             # Break out of inner loop
 842             break
 843
 844     print(f"Completed processing {split} split")
 845
 846     # Break out of outer loop
 847     break
 848
 849
 850             # Break out of inner loop
 851             break
 852
 853     print(f"Completed processing {split} split")
 854
 855     # Break out of outer loop
 856     break
 857
 858
 859             # Break out of inner loop
 860             break
 861
 862     print(f"Completed processing {split} split")
 863
 864     # Break out of outer loop
 865     break
 866
 867
 868             # Break out of inner loop
 869             break
 870
 871     print(f"Completed processing {split} split")
 872
 873     # Break out of outer loop
 874     break
 875
 876
 877             # Break out of inner loop
 878             break
 879
 880     print(f"Completed processing {split} split")
 881
 882     # Break out of outer loop
 883     break
 884
 885
 886             # Break out of inner loop
 887             break
 888
 889     print(f"Completed processing {split} split")
 890
 891     # Break out of outer loop
 892     break
 893
 894
 895             # Break out of inner loop
 896             break
 897
 898     print(f"Completed processing {split} split")
 899
 900     # Break out of outer loop
 901     break
 902
 903
 904             # Break out of inner loop
 905             break
 906
 907     print(f"Completed processing {split} split")
 908
 909     # Break out of outer loop
 910     break
 911
 912
 913             # Break out of inner loop
 914             break
 915
 916     print(f"Completed processing {split} split")
 917
 918     # Break out of outer loop
 919     break
 920
 921
 922             # Break out of inner loop
 923             break
 924
 925     print(f"Completed processing {split} split")
 926
 927     # Break out of outer loop
 928     break
 929
 930
 931             # Break out of inner loop
 932             break
 933
 934     print(f"Completed processing {split} split")
 935
 936     # Break out of outer loop
 937     break
 938
 939
 940             # Break out of inner loop
 941             break
 942
 943     print(f"Completed processing {split} split")
 944
 945     # Break out of outer loop
 946     break
 947
 948
 949             # Break out of inner loop
 950             break
 951
 952     print(f"Completed processing {split} split")
 953
 954     # Break out of outer loop
 955     break
 956
 957
 958             # Break out of inner loop
 959             break
 960
 961     print(f"Completed processing {split} split")
 962
 963     # Break out of outer loop
 964     break
 965
 966
 967             # Break out of inner loop
 968             break
 969
 970     print(f"Completed processing {split} split")
 971
 972     # Break out of outer loop
 973     break
 974
 975
 976             # Break out of inner loop
 977             break
 978
 979     print(f"Completed processing {split} split")
 980
 981     # Break out of outer loop
 982     break
 983
 984
 985             # Break out of inner loop
 986             break
 987
 988     print(f"Completed processing {split} split")
 989
 990     # Break out of outer loop
 991     break
 992
 993
 994             # Break out of inner loop
 995             break
 996
 997     print(f"Completed processing {split} split")
 998
 999     # Break out of outer loop
 1000    break
 1001
 1002
 1003             # Break out of inner loop
 1004             break
 1005
 1006     print(f"Completed processing {split} split")
 1007
 1008     # Break out of outer loop
 1009     break
 1010
 1011
 1012             # Break out of inner loop
 1013             break
 1014
 1015     print(f"Completed processing {split} split")
 1016
 1017     # Break out of outer loop
 1018     break
 1019
 1020
 1021             # Break out of inner loop
 1022             break
 1023
 1024     print(f"Completed processing {split} split")
 1025
 1026     # Break out of outer loop
 1027     break
 1028
 1029
 1030             # Break out of inner loop
 1031             break
 1032
 1033     print(f"Completed processing {split} split")
 1034
 1035     # Break out of outer loop
 1036     break
 1037
 1038
 1039             # Break out of inner loop
 1040             break
 1041
 1042     print(f"Completed processing {split} split")
 1043
 1044     # Break out of outer loop
 1045     break
 1046
 1047
 1048             # Break out of inner loop
 1049             break
 1050
 1051     print(f"Completed processing {split} split")
 1052
 1053     # Break out of outer loop
 1054     break
 1055
 1056
 1057             # Break out of inner loop
 1058             break
 1059
 1060     print(f"Completed processing {split} split")
 1061
 1062     # Break out of outer loop
 1063     break
 1064
 1065
 1066             # Break out of inner loop
 1067             break
 1068
 1069     print(f"Completed processing {split} split")
 1070
 1071     # Break out of outer loop
 1072     break
 1073
 1074
 1075             # Break out of inner loop
 1076             break
 1077
 1078     print(f"Completed processing {split} split")
 1079
 1080     # Break out of outer loop
 1081     break
 1082
 1083
 1084             # Break out of inner loop
 1085             break
 1086
 1087     print(f"Completed processing {split} split")
 1088
 1089     # Break out of outer loop
 1090     break
 1091
 1092
 1093             # Break out of inner loop
 1094             break
 1095
 1096     print(f"Completed processing {split} split")
 1097
 1098     # Break out of outer loop
 1099     break
 1100
 1101
 1102             # Break out of inner loop
 1103             break
 1104
 1105     print(f"Completed processing {split} split")
 1106
 1107     # Break out of outer loop
 1108     break
 1109
 1110
 1111             # Break out of inner loop
 1112             break
 1113
 1114     print(f"Completed processing {split} split")
 1115
 1116     # Break out of outer loop
 1117     break
 1118
 1119
 1120             # Break out of inner loop
 1121             break
 1122
 1123     print(f"Completed processing {split} split")
 1124
 1125     # Break out of outer loop
 1126     break
 1127
 1128
 1129             # Break out of inner loop
 1130             break
 1131
 1132     print(f"Completed processing {split} split")
 1133
 1134     # Break out of outer loop
 1135     break
 1136
 1137
 1138             # Break out of inner loop
 1139             break
 1140
 1141     print(f"Completed processing {split} split")
 1142
 1143     # Break out of outer loop
 1144     break
 1145
 1146
 1147             # Break out of inner loop
 1148             break
 1149
 1150     print(f"Completed processing {split} split")
 1151
 1152     # Break out of outer loop
 1153     break
 1154
 1155
 1156             # Break out of inner loop
 1157             break
 1158
 1159     print(f"Completed processing {split} split")
 1160
 1161     # Break out of outer loop
 1162     break
 1163
 1164
 1165             # Break out of inner loop
 1166             break
 1167
 1168     print(f"Completed processing {split} split")
 1169
 1170     # Break out of outer loop
 1171     break
 1172
 1173
 1174             # Break out of inner loop
 1175             break
 1176
 1177     print(f"Completed processing {split} split")
 1178
 1179     # Break out of outer loop
 1180     break
 1181
 1182
 1183             # Break out of inner loop
 1184             break
 1185
 1186     print(f"Completed processing {split} split")
 1187
 1188     # Break out of outer loop
 1189     break
 1190
 1191
 1192             # Break out of inner loop
 1193             break
 1194
 1195     print(f"Completed processing {split} split")
 1196
 1197     # Break out of outer loop
 1198     break
 1199
 1200
 1201             # Break out of inner loop
 1202             break
 1203
 1204     print(f"Completed processing {split} split")
 1205
 1206     # Break out of outer loop
 1207     break
 1208
 1209
 1210             # Break out of inner loop
 1211             break
 1212
 1213     print(f"Completed processing {split} split")
 1214
 1215     # Break out of outer loop
 1216     break
 1217
 1218
 1219             # Break out of inner loop
 1220             break
 1221
 1222     print(f"Completed processing {split} split")
 1223
 1224     # Break out of outer loop
 1225     break
 1226
 1227
 1228             # Break out of inner loop
 1229             break
 1230
 1231     print(f"Completed processing {split} split")
 1232
 1233     # Break out of outer loop
 1234     break
 1235
 1236
 1237             # Break out of inner loop
 1238             break
 1239
 1240     print(f"Completed processing {split} split")
 1241
 1242     # Break out of outer loop
 1243     break
 1244
 1245
 1246             # Break out of inner loop
 1247             break
 1248
 1249     print(f"Completed processing {split} split")
 1250
 1251     # Break out of outer loop
 1252     break
 1253
 1254
 1255             # Break out of inner loop
 1256             break
 1257
 1258     print(f"Completed processing {split} split")
 1259
 1260     # Break out of outer loop
 1261     break
 1262
 1263
 1264             # Break out of inner loop
 1265             break
 1266
 1267     print(f"Completed processing {split} split")
 1268
 1269     # Break out of outer loop
 1270     break
 1271
 1272
 1273             # Break out of inner loop
 1274             break
 1275
 1276     print(f"Completed processing {split} split")
 1277
 1278     # Break out of outer loop
 1279     break
 1280
 1281
 1282             # Break out of inner loop
 1283             break
 1284
 1285     print(f"Completed processing {split} split")
 1286
 1287     # Break out of outer loop
 1288     break
 1289
 1290
 1291             # Break out of inner loop
 1292             break
 1293
 1294     print(f"Completed processing {split} split")
 1295
```

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.

```
convert_voc_to_yolo.py      data.yaml ×
```

1	<code>train: ../dataset/images/train</code>
2	<code>val: ../dataset/images/val</code>
3	<code>nc: 2 # number of classes</code>
4	<code>names: ['person', 'person-like']</code>
5	
6	

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

Description:

The contents of the data.yaml file which is an 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.

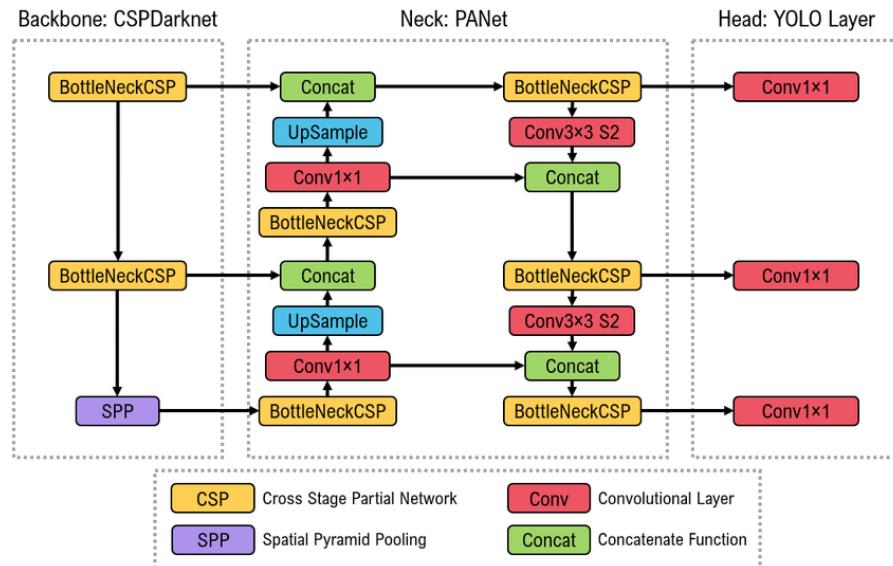


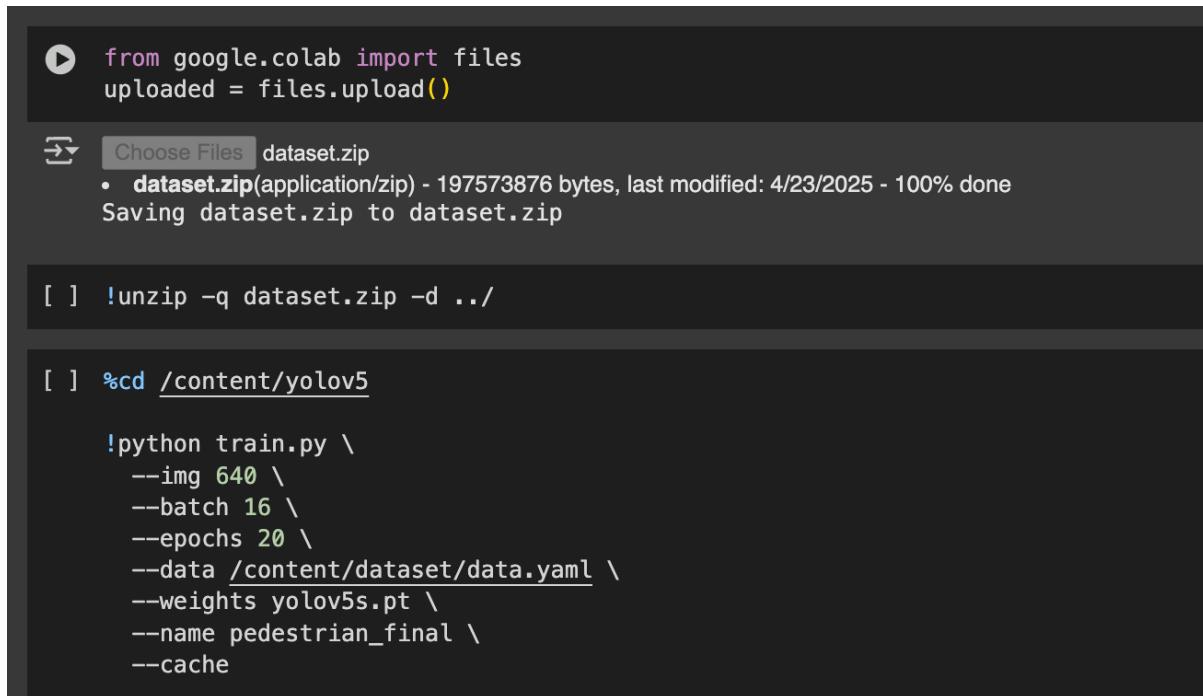
Figure 9: YOLOv5 model architecture showing three main parts

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.



```

▶ from google.colab import files
uploaded = files.upload()

Choose Files dataset.zip
• dataset.zip(application/zip) - 197573876 bytes, last modified: 4/23/2025 - 100% done
Saving dataset.zip to dataset.zip

[ ] !unzip -q dataset.zip -d ../

[ ] %cd /content/yolov5

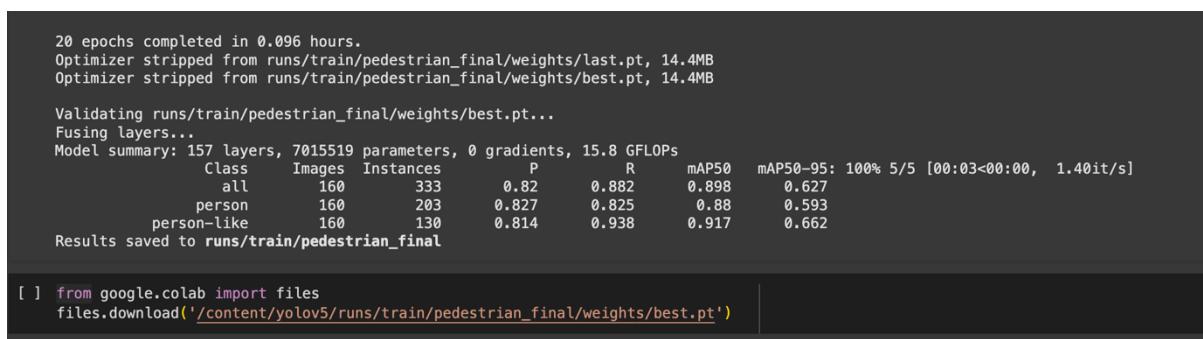
!python train.py \
--img 640 \
--batch 16 \
--epochs 20 \
--data /content/dataset/data.yaml \
--weights yolov5s.pt \
--name pedestrian_final \
--cache

```

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.



```

20 epochs completed in 0.096 hours.
Optimizer stripped from runs/train/pedestrian_final/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/pedestrian_final/weights/best.pt, 14.4MB

Validating runs/train/pedestrian_final/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
      Class   Images Instances     P      R    mAP50    mAP50-95: 100% 5/5 [00:03<00:00,  1.40it/s]
        all      160      333    0.82    0.882    0.898    0.627
      person     160      203    0.827    0.825    0.88    0.593
  person-like     160      130    0.814    0.938    0.917    0.662
Results saved to runs/train/pedestrian_final

[ ] from google.colab import files
files.download('/content/yolov5/runs/train/pedestrian_final/weights/best.pt')

```

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.

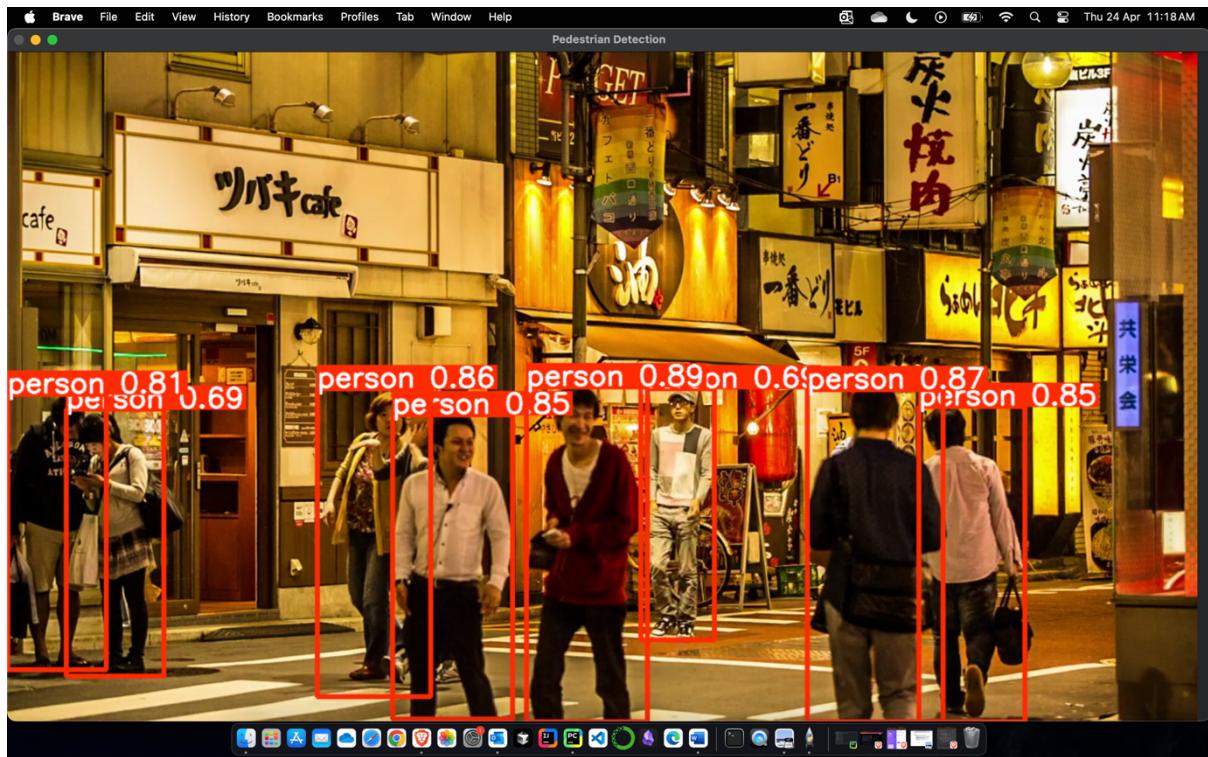


Figure 12: Pedestrian detection results from a sample video showing bounding boxes and confidence scores for each detected person.

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](#)