



# Safety Helmet Detection Using Python

## PRESENTED BY

<b>AMIT MANDAL</b>	<b>10200321068</b>
<b>PAPAI SARKAR</b>	<b>10200321070</b>
<b>SUBHADEEP SAHA</b>	<b>10200320036</b>
<b>PRONAY CHANDRA MRIDHA</b>	<b>10200321055</b>

**UNDER THE GUIDANCE OF DR. BANDANA BARMAN**



## Aim Of The Project

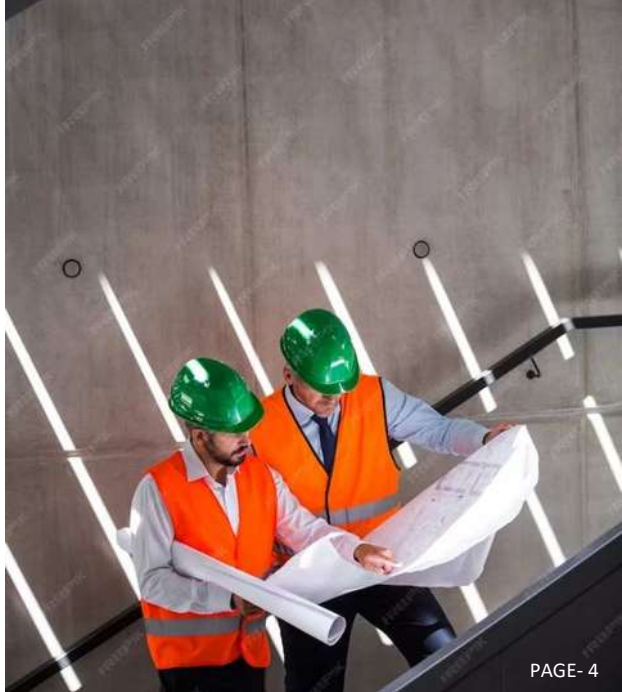
The primary aim of the project is to enhance construction site safety through the implementation of a robust Safety Helmet Detection system using Python (YOLOv5). The overarching goal is to leverage computer vision technology to automate the monitoring of safety helmet usage among construction site personnel.

# CONTENT

- INTRODUCTION
- PREVIOUS RESEARCHES ON SAFETY  
HELMET DETECTION
- SOFTWARE AND SIMULATION TOOLS
- FLOWCHART
- ALGORITHM
- INPUT SOURCE
- RESULT
- OUTPUT
- REFERENCES

# INTRODUCTION

The construction industry, inherently fraught with hazards, necessitates stringent safety measures. Among these, the use of safety helmets stands as a foundational practice to safeguard workers from head injuries. This project delves into the realm of automated safety helmet detection on construction sites, utilizing a synergy of Python programming, YOLOv5, and the Thonny Python IDE. Python, a dynamic and versatile language, is employed both for coding purposes (Python 3.10) and for model training (Python 3.9). The Thonny Python IDE, chosen for its simplicity and suitability for educational contexts, provides an accessible environment for coding, ensuring that the project remains pedagogically valuable. YOLOv5, renowned for its real-time object detection capabilities, becomes the fulcrum of our safety helmet detection system.



# Previous Researches On Safety Helmet Detection

- 1) Safety Helmet Detection Based on YOLOv5 by Fangbo Zhou, Huailin Zhao, Zhen (2021) ,this research work proposes a safety helmet detection method based on YOLOv5 and annotates the 6045 collected data sets to establish a digital safety helmet monitoring system and shows the effectiveness of helmet detection based YOLOv5.
- 2) Safety Helmet Wearing Detection Based on Jetson Nano and Improved YOLOv5 by Zaihui Deng,Chong Yao,and Qiyu Yin(2023), This study introduces an improved safety helmet-wearing detection model named YOLOv5-SN, aiming to address the shortcomings of the existing YOLOv5 models, including a large number of model parameters, slow reasoning speed, and redundant network structure.



While the mentioned research papers contribute significantly to the field of safety helmet detection using YOLOv5, it's common for studies to encounter gaps, difficulties, or areas that warrant further exploration. Here are some potential gaps or challenges that might be found in these research papers:

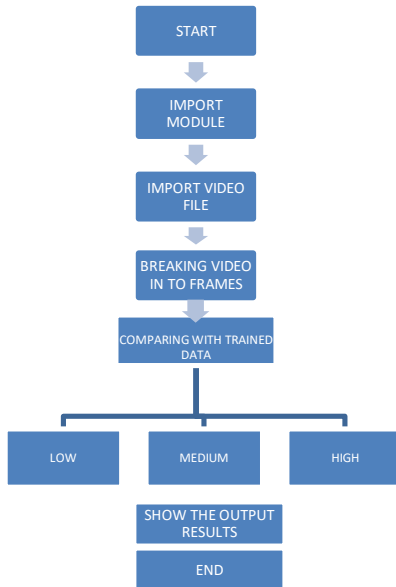
1. Handling Highly Occluded Helmets : Occlusions, where safety helmets are partially or fully hidden, pose a considerable challenge in real-world scenarios.
2. Generalization Across Diverse Construction Environments : Construction sites vary widely in terms of lighting conditions, backgrounds, and types of helmets used. Ensuring the model's generalization across diverse environments is crucial.
3. Ethical and Privacy Concerns : safety helmet detection systems become more prevalent, there is a growing need to address ethical considerations and privacy implications
4. Explainability and Transparency : The interpretability and transparency of YOLOv5-based safety helmet detection systems may not be well-explored in some studies.

# Software And Simulation Tools

The selection of tools and technologies is pivotal in determining the project's success. Python, with its 3.9 and 3.10 versions, serves a dual purpose. Python 3.10 becomes the coding backbone, providing an environment conducive to scripting and development. Python 3.9 takes center stage for model training, where the nuances of machine learning and deep learning algorithms are harnessed. Thonny Python IDE emerges as the coding environment of choice due to its simplicity, particularly beneficial in educational settings. Its lightweight nature and ease of use make it an ideal platform for coding tasks related to the safety helmet detection project. YOLOv5, characterized by its You Only Look Once architecture, exemplifies efficiency in real-time object detection, making it the cornerstone of our system.



# Flowchart



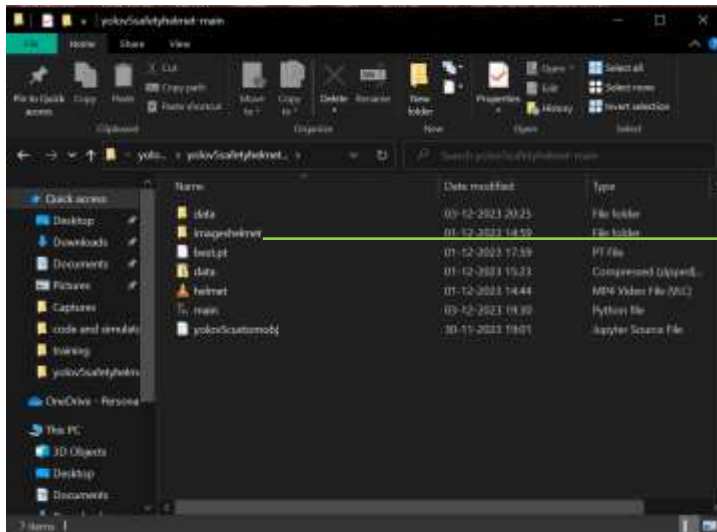
# ALGORITHM

## **Our Approach for Safety Helmet Detection**

- 1) step 1: Install Thonny Python IDE, yolov5 ,open CV module, NumPy module
- 2) step 2: Download the Images and move our main file.
- 3) step 3: Created the project video and train all images. we create the 'Rectangle' box to measure the size of our helmets.
- 4) step 4: Then create a new folder for storing our data and move all pictures here and make a zip file of it and upload in the g-drive.
- 5) step 5: Opens google collab for collaboration our code with our image file.
- 6) step 6: Download our custom model and store in the main folder.
- 7) step 7: Open Thonny Python IDE & give the code for the model, add the path, capture the video file.
- 8) step 8: Now our basic code is ready , we call the custom model and framing our models.
- 9) step 9: Finally detecting Safety Helmets in the Images and the project is done.

# RESULTS

## Main File Location



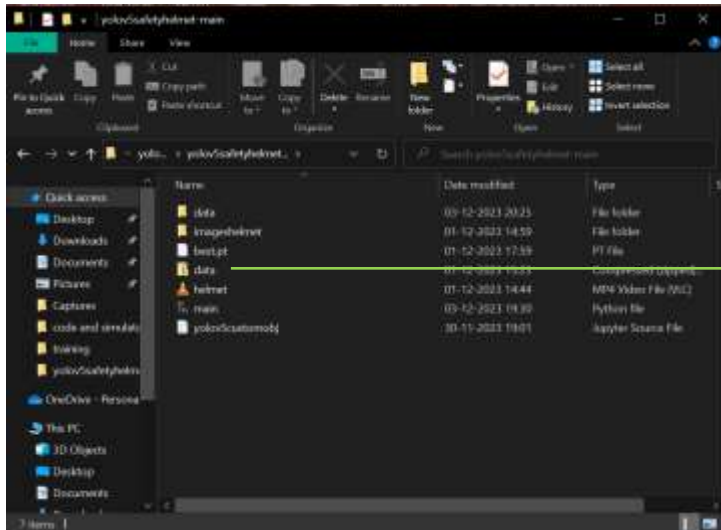
DOWNLOAD IMAGES

[illegible]



TRAINING AND LABELLING THE IMAGES. HERE CREATING 'RECTANGLE' BOX ON HELMETS.





DATA ZIP FILE

The screenshot shows a Kali Linux terminal window with the following content:

```

root@kali:~# ssh root@10.10.10.10
Warning: Permanently added the RSA host key to the list of known hosts.
root@kali:~# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
cron:x:4:4:cron:/var/spool/cron/root:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin

```

[illegible]

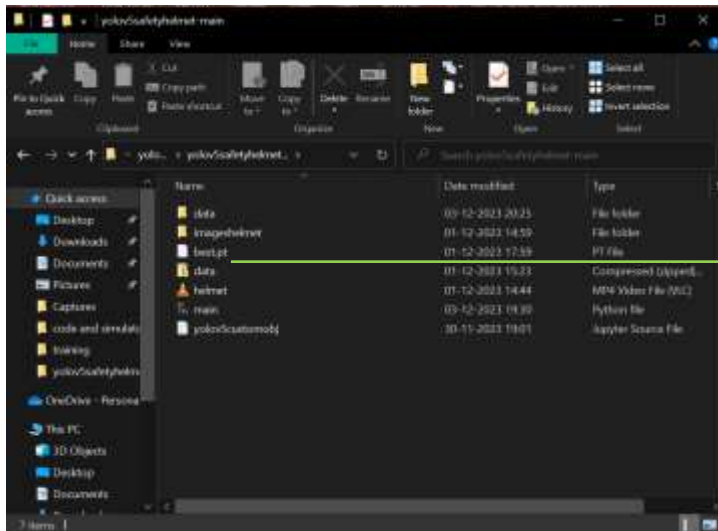


## HERE YOLOV5 MODEL GENERATE

The screenshot shows a Jupyter Notebook titled 'yolov5customobj.ipynb' with a terminal window open. The terminal is running a YOLOv5 training command. The output shows the progress of the training across five epochs (0/99 to 4/99). Each epoch's output includes GPU memory usage, box loss, obj loss, cls loss, instance count, and size. The training is progressing as expected, with metrics being updated for each epoch.

```
python3 /content/yolov5train/yolov5/train.py --img 416 --batch 16 --epochs 100 --data /content/yolov5train/yolov5/defaults.yaml --weights yolov5s.pt
```

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
0/99	1.426	0.127	0.91793	0.00137	26	416: 100% 1/3 [01:47:00:00, 99.87s/it]
Class	Images	Instances	P	0	0	0.00000
all	48	58	0.00000	0.000	0.00000	7.00e-05
1/99	1.476	0.1240	0.91821	0.00068	18	416: 100% 1/3 [01:50:00:00, 11.30s/it]
Class	Images	Instances	P	0	0	0.00000
all	48	58	0.00000	0.000	0.00000	1.70e-04
2/99	1.476	0.122	0.91814	0.00097	14	416: 100% 1/3 [01:51:00:00, 10.47s/it]
Class	Images	Instances	P	0	0	0.00000
all	48	58	0.00000	0.0017	0.00000	1.20e-05
3/99	1.476	0.1208	0.91991	0.00076	22	416: 100% 1/3 [01:52:00:00, 14.17s/it]
Class	Images	Instances	P	0	0	0.00000
all	48	58	0.00000	0.0045	0.00000	1.01e-04
4/99	1.476	0.1183	0.92149	0.00015	25	416: 100% 1/3 [01:53:00:00, 10.03s/it]
Class	Images	Instances	P	0	0	0.00000
all	48	58	0.00000	0.000	0.00000	0.00000



CUSTOM MODEL FILE

## WRITING OUR MAIN CODE USING PYTHON VERSION 3.10 IN THONNY IDE .

A screenshot of the Thonny Python IDE interface. The main editor window displays a Python script for object detection using YOLOv5. The code imports cv2, torch, and numpy as np. It defines a path to a YOLOv5 model file. The model is loaded using torch.hub.load. A video capture object is created for 'helmet.mp4'. A while loop reads frames from the video, processes them with the model, and displays the results using imshow. The code is as follows:

```
1 import cv2
2 import torch
3 import numpy as np
4
5 path='C:/Users/pepal/OneDrive/Desktop/p/yolov5safetymet-main/yolov5safetymet-main/best.pt'
6
7 model = torch.hub.load('ultralytics/yolo5', 'custom', path, force_reload=True)
8
9 cap=cv2.VideoCapture('helmet.mp4')
10 count=0
11 while True:
12     ret,frame=cap.read()
13     if not ret:
14         break
15     count += 1
16     if count % 3 != 0:
17         continue
18     frame=cv2.resize(frame,(3820,600))
19     results=model(frame)
20     frame=np.squeeze(results.render())
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

The IDE interface includes a menu bar at the top with options like File, Edit, View, Run, Tools, Help. Below the menu is a toolbar with icons for file operations and running the code. The left sidebar shows the file explorer with 'helmetA.py' selected. The right sidebar is empty. The status bar at the bottom indicates 'Local Python 3.10.5 - Thonny Python 3.10.5'.

```
File Edit View Run Tools Help
Python 3.8.11 (C:\Users\Thomaz\AppData\Local\Programs\Python\Python38-64\python.exe)

# Import libraries
import cv2
import torch
import numpy as np

# Load YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'custom', path, force_reload=True)

# Open video file
cap = cv2.VideoCapture('helmet.mp4')

# Initialize variables
count = 0

# Loop through frames
while True:
    # Read frame
    ret, frame = cap.read()
    # If not read, break
    if not ret:
        break
    # Increment count
    count += 1
    # If count is not 0, continue
    if count % 1 != 0:
        continue
    # Resize frame
    frame = cv2.resize(frame, (1024, 1024))
    # Run model
    results = model(frame)
    # Squeeze results
    frame = np.squeeze(results.render())
    # Show frame
    cv2.imshow("Frame", frame)
    # If key pressed, break
    if cv2.waitKey(1) && chr(key) != 'q':
        break
# Release cap and destroy windows
cap.release()
cv2.destroyAllWindows()

# Run
Python 3.8.11 (C:\Users\Thomaz\AppData\Local\Programs\Python\Python38-64\python.exe)
>>>
```

# Input Source



# OUTPUT



OUTPUT IN THE FORM OF VIDEO



OUTPUT IN THE FORM OF IMAGE

# Conclusion

In conclusion, this project has successfully realized the development and implementation of a safety helmet detection system, contributing to the broader goal of improving construction site safety. The automated monitoring system, empowered by YOLOv5 and Python, stands as a testament to the intersection of technology and safety. By leveraging these tools, we have created a reliable solution that has the potential to mitigate risks and enhance the well-being of construction site workers.



# REFERENCES

1. "Single-Image Crowd Counting via Multi-Column Convolutional Neural Network" by S. Zhang et al.
2. "Cross-Scene Crowd Counting via Deep Convolutional Neural Networks" by Z. Cao et al.
3. "Switching Convolutional Neural Network for Crowd Counting" by Y. Yuan et al.
4. "CNN-Based Cascaded Multi-task Learning of High-Level Prior and Density Estimation for Crowd Counting" by Y. Li et al.
5. "Adaptive Density Map Generation for Crowd Counting" by V. Lempitsky and A. Zisserman. 6.
6. "Convolutional Neural Networks for Crowd Counting" by S. Sindagi and V. M. Patel. 7.
7. "Leveraging Unlabeled Data for Crowd Counting by Learning to Rank" by Y. Zhang et al. 8.
8. "Crowd Counting Using Deep Recurrent Spatial-Aware Network" by X. Shi et al. 9.
9. "Composition Loss for Counting, Density Map Estimation and Localization in Dense Crowds" by R. S. Zanotto et al. 10.
10. "Cross-Scene Crowd Counting via Deeply Learned Descriptors" by Z. Cao et al. 11.
11. "Generating Semantically Segmented Density Maps for Crowd Counting" by K. Chen et al.
12. "Crowd Counting via Adversarial Cross-Scale Consistency Pursuit" by Y. Bai et al. 13.
13. "Hydra-CNN: Recursive Learning for Single-Image Crowd Counting" by J. Lian et al

14. "Weakly Supervised Crowd Counting via Adaptive Density Estimation" by Y. Zhao et al.
15. "Context-Aware Crowd Counting" by C. Zhang et al.
16. Wu, Y.; Meng, Z.; Palaiahnakote, S.; Lu, T. Compressing YOLO network by compressive sensing. Proceedings on the 4th Asian Conference on Pattern Recognition, ACPR2017, Nanjing, China; pp. 19–24. [CrossRef]
17. Pan, H.; Jiang, J.; Chen, G. TDFSSD: Top-Down Feature Fusion Single Shot MultiBox Detector. Signal Process. Image Commun. 2020, 89, 115987. [CrossRef]
18. Li, Y.; Wei, H.; Han, Z.; Huang, J.; Wang, W. Deep Learning-Based Safety Helmet Detection in Engineering Management Based on Convolutional Neural Networks. Adv. Civ. Eng. 2020, 2020, 1–10. [CrossRef]
19. Barro-Torres, S.; Fernández-Caramés, T.M.; Pérez-Iglesias, H.J.; Escudero, C.J. Real-time personal protective equipment monitoring system. Comput. Commun. 2012, 36, 42–50. [CrossRef]
20. Wu, J.; Cai, N.; Chen, W.; Wang, H.; Wang, G. Automatic detection of hardhats worn by construction personnel: A deep learning approach and benchmark dataset. Autom. Constr. 2019, 106, 102894. [CrossRef]
21. Waranusast, R.; Bundon, N.; Timtong, V.; Tangnoi, C.; Pattanathaburt, P. Machine vision techniques for motorcycle safety helmet detection. In Proceedings of the in International Conference Image and Vision Computing New Zealand, Wellington, New Zealand, 27–29 November 2013; pp. 35–40. [CrossRef]

**Thanks!**