

B.Tech. CSE (IoT & IS)

III Year (5th Sem)

Project Based Learning – III

IS3170

Project Report

On[[1]](#footnote-1)

Helmet And Vest Detection System Using RaspberryPi

Group Members:

1. Aayush Bindal (229311202)

2. Siddharth (229311265)

3. Atharva Deshmukh (229311049)

|  |  |
| --- | --- |
| Submitted To  Dr. Hemlata Goyal | Submitted By:  Name:  Registration No.: |

**Index**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Name** | **Page No.** |
| 1 | Objective | 3 |
| 2 | Problem Statement | 3 |
| 3 | Microcontroller Structure | 4 |
| 4 | Components Used | 5 |
| 5 | Circuit Diagram | 5 |
| 6 | Code | 6 |
| 7 | Result | 15 |

**Objective**

The primary objective of this project is to design and implement a real-time system capable of detecting the presence of safety helmets and vests on workers using Raspberry Pi and advanced image processing techniques. This system leverages machine learning algorithms to identify personal protective equipment (PPE) in live video streams. The detection mechanism aims to ensure adherence to safety regulations in industrial and construction environments, where non-compliance could result in severe injuries or fatalities. Additionally, the system seeks to provide:

* Automated monitoring for workplace safety.
* Real-time alerts in case of non-compliance to reduce the burden on human supervisors.
* Data logging and analytics for safety compliance trends over time.

By integrating a compact and cost-effective microcontroller like Raspberry Pi, this project offers a scalable solution suitable for deployment in various industrial and occupational settings.

**Problem Statement**

Workplace safety remains a critical concern across industries, particularly in construction, manufacturing, and mining, where workers are exposed to hazardous conditions. Personal protective equipment (PPE), such as safety helmets and reflective vests, plays a vital role in reducing the risk of injury. However, the following challenges hinder effective PPE compliance:

1. **Manual Monitoring**: Current compliance checks rely heavily on human supervisors, which can be inconsistent and prone to oversight, especially in large-scale operations.
2. **Time and Resource Intensive**: Continuous manual monitoring requires significant resources, making it inefficient and expensive.
3. **Human Error**: Fatigue or distractions may result in non-compliance going unnoticed, increasing the likelihood of accidents.
4. **Scalability Issues**: Large industrial sites with numerous workers are difficult to monitor effectively using traditional methods.

The absence of a reliable automated system to ensure PPE compliance exacerbates these challenges. This project aims to address this gap by providing a cost-effective, automated detection system that enhances safety, reduces risks, and ensures adherence to safety regulations without constant human intervention.

**Microcontroller Structure**

The Raspberry Pi serves as the central processing unit for this project due to its versatility, cost-effectiveness, and ability to handle computational tasks. The microcontroller structure includes the following key components:

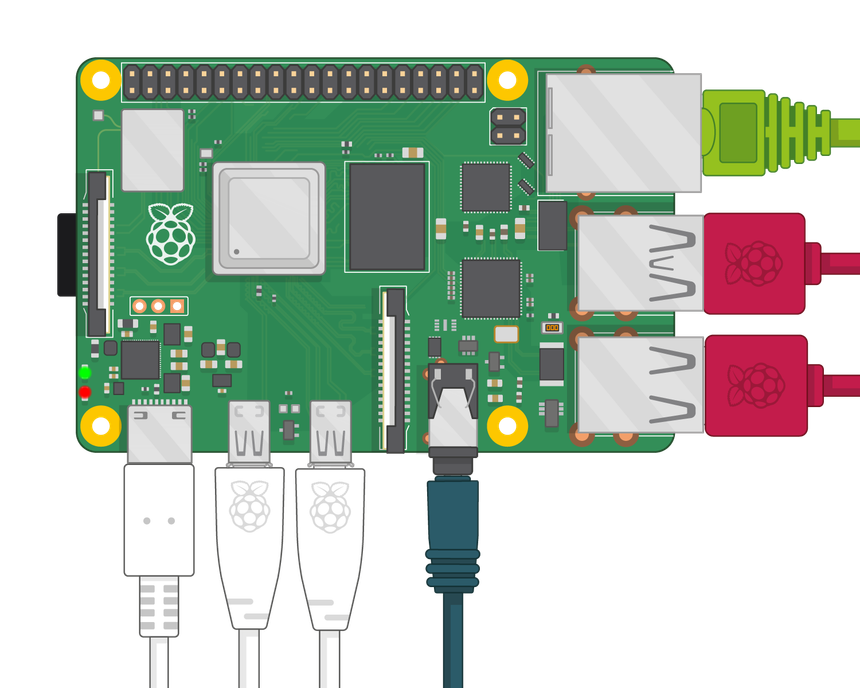
1. **Raspberry Pi 4 Model B**:
   * **Processor:** Equipped with a quad-core Cortex-A72 CPU, the Raspberry Pi 4 provides sufficient computational power to run real-time object detection models.
   * **Memory:** Available in multiple configurations (2GB, 4GB, or 8GB RAM), ensuring smooth operation for image processing tasks.
   * **Operating System:** Runs on Raspberry Pi OS, a Debian-based Linux distribution that supports Python and various libraries for machine learning and GPIO control.
2. **Camera Module**:
   * **Integration:** The Raspberry Pi Camera Module V2 connects directly to the CSI port on the Raspberry Pi.
   * **Resolution:** Supports 8MP resolution, allowing for high-quality image capture to improve detection accuracy.
   * **Compatibility:** Alternatively, USB cameras can be used if the native module is unavailable.
3. **Network Connectivity**:
   * **Wi-Fi/Ethernet:** Enables the system to upload compliance logs to a remote server or integrate with cloud-based monitoring solutions.
   * **Bluetooth (optional):** Facilitates wireless communication with other devices if necessary.
4. **Storage**:
   * **SD Card:** A high-speed microSD card is used to store the operating system, detection model, and log files.
   * **Cloud Storage (optional):** Can be configured to upload data for remote access and analysis.

This modular structure ensures the flexibility and scalability of the system, making it suitable for a variety of environments and applications. Each component is chosen to balance performance, reliability, and cost-effectiveness.

**Components Used**

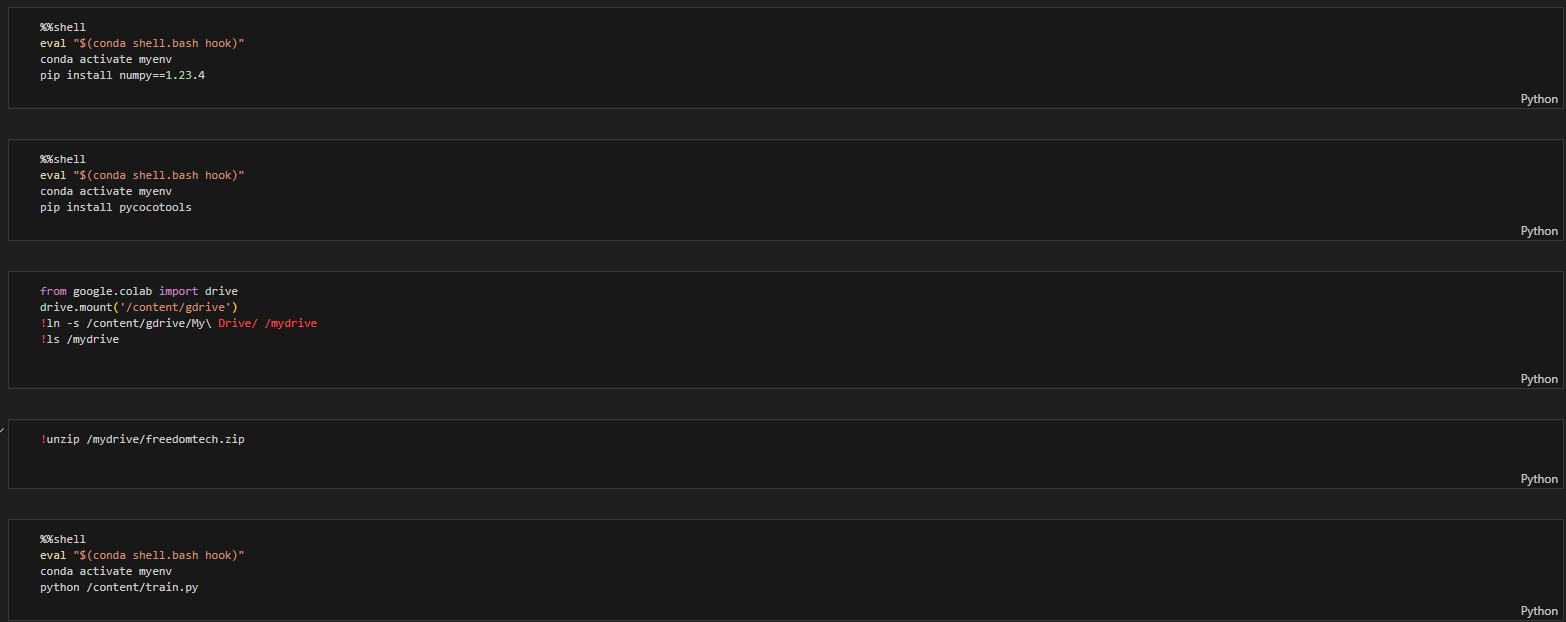
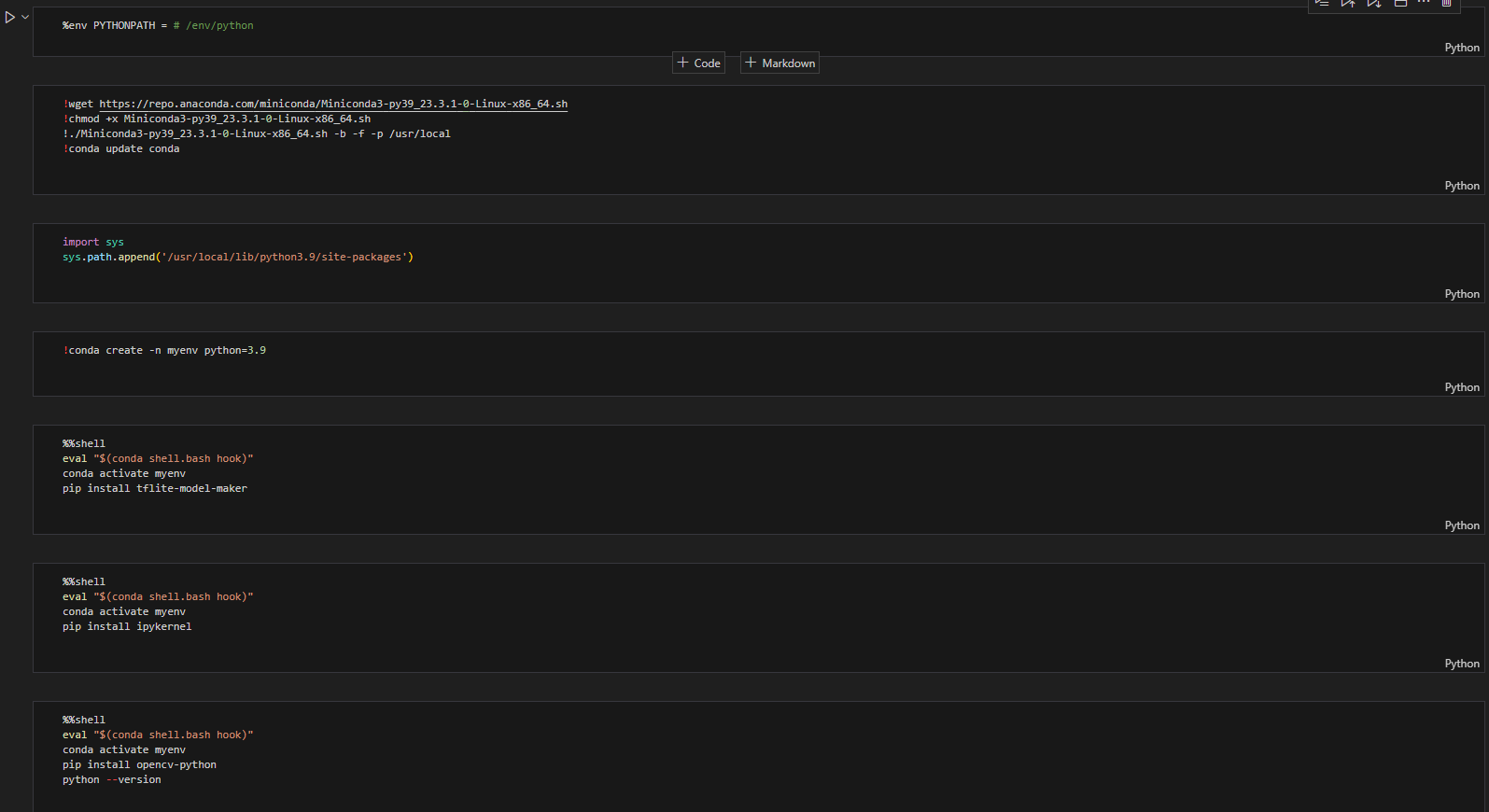
1. **Raspberry Pi 4 Model B**:
   * Serves as the primary computational unit.
2. **RJ45 Cable**:
   * Used for stable network connectivity.
3. **32GB MicroSD Card**:
   * Provides storage for the operating system and detection model.
4. **Inbuilt Camera of Laptop**:
   * Captures real-time video for object detection.

**Circuit Diagram**

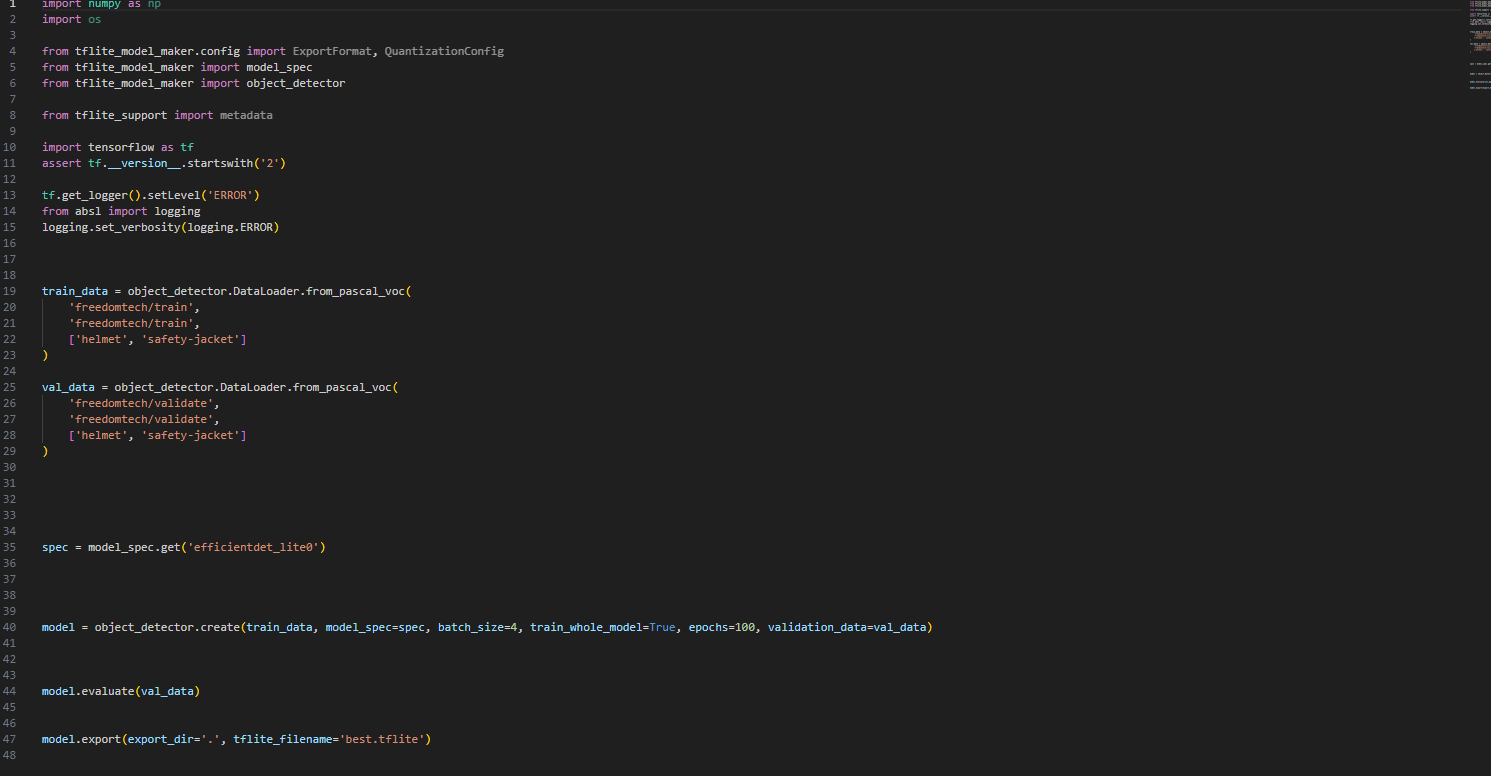


**Code**

**1. Code for Training ML Model :**



* **train.py :**



**2. Code to execute the trained model:**

* **Detect.py:**

import argparse

import sys

import time

import cv2

import mediapipe as mp

from mediapipe.tasks import python

from mediapipe.tasks.python import vision

from utils import visualize

# Global variables to calculate FPS

COUNTER, FPS = 0, 0

START\_TIME = time.time()

def run(model: str, max\_results: int, score\_threshold: float,

camera\_id: int, width: int, height: int) -> None:

"""Continuously run inference on images acquired from the camera.

Args:

model: Name of the TFLite object detection model.

max\_results: Max number of detection results.

score\_threshold: The score threshold of detection results.

camera\_id: The camera id to be passed to OpenCV.

width: The width of the frame captured from the camera.

height: The height of the frame captured from the camera.

"""

# Start capturing video input from the camera

cap = cv2.VideoCapture(0)

cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, width)

cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, height)

# Visualization parameters

row\_size = 50 # pixels

left\_margin = 24 # pixels

text\_color = (0, 0, 0) # black

font\_size = 1

font\_thickness = 1

fps\_avg\_frame\_count = 10

detection\_frame = None

detection\_result\_list = []

def save\_result(result: vision.ObjectDetectorResult, unused\_output\_image: mp.Image, timestamp\_ms: int):

global FPS, COUNTER, START\_TIME

# Calculate the FPS

if COUNTER % fps\_avg\_frame\_count == 0:

FPS = fps\_avg\_frame\_count / (time.time() - START\_TIME)

START\_TIME = time.time()

detection\_result\_list.append(result)

COUNTER += 1

# Initialize the object detection model

base\_options = python.BaseOptions(model\_asset\_path=model)

options = vision.ObjectDetectorOptions(base\_options=base\_options,

running\_mode=vision.RunningMode.LIVE\_STREAM,

max\_results=max\_results, score\_threshold=score\_threshold,

result\_callback=save\_result)

detector = vision.ObjectDetector.create\_from\_options(options)

# Continuously capture images from the camera and run inference

while cap.isOpened():

success, image = cap.read()

image=cv2.resize(image,(640,480))

if not success:

sys.exit(

'ERROR: Unable to read from webcam. Please verify your webcam settings.'

)

image = cv2.flip(image, 1)

# Convert the image from BGR to RGB as required by the TFLite model.

rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

mp\_image = mp.Image(image\_format=mp.ImageFormat.SRGB, data=rgb\_image)

# Run object detection using the model.

detector.detect\_async(mp\_image, time.time\_ns() // 1\_000\_000)

# Show the FPS

fps\_text = 'FPS = {:.1f}'.format(FPS)

text\_location = (left\_margin, row\_size)

current\_frame = image

cv2.putText(current\_frame, fps\_text, text\_location, cv2.FONT\_HERSHEY\_DUPLEX,

font\_size, text\_color, font\_thickness, cv2.LINE\_AA)

if detection\_result\_list:

# print(detection\_result\_list)

current\_frame = visualize(current\_frame, detection\_result\_list[0])

detection\_frame = current\_frame

detection\_result\_list.clear()

if detection\_frame is not None:

cv2.imshow('object\_detection', detection\_frame)

# Stop the program if the ESC key is pressed.

if cv2.waitKey(1) == 27:

break

detector.close()

cap.release()

cv2.destroyAllWindows()

def main():

parser = argparse.ArgumentParser(

formatter\_class=argparse.ArgumentDefaultsHelpFormatter)

parser.add\_argument(

'--model',

help='Path of the object detection model.',

required=False,

# default='efficientdet\_lite0.tflite')

default='best.tflite')

parser.add\_argument(

'--maxResults',

help='Max number of detection results.',

required=False,

default=5)

parser.add\_argument(

'--scoreThreshold',

help='The score threshold of detection results.',

required=False,

type=float,

default=0.25)

# Finding the camera ID can be very reliant on platform-dependent methods.

# One common approach is to use the fact that camera IDs are usually indexed sequentially by the OS, starting from 0.

# Here, we use OpenCV and create a VideoCapture object for each potential ID with 'cap = cv2.VideoCapture(i)'.

# If 'cap' is None or not 'cap.isOpened()', it indicates the camera ID is not available.

parser.add\_argument(

'--cameraId', help='Id of camera.', required=False, type=int, default=0)

parser.add\_argument(

'--frameWidth',

help='Width of frame to capture from camera.',

required=False,

type=int,

default=640)

parser.add\_argument(

'--frameHeight',

help='Height of frame to capture from camera.',

required=False,

type=int,

default=480)

args = parser.parse\_args()

run(args.model, int(args.maxResults),

args.scoreThreshold, int(args.cameraId), args.frameWidth, args.frameHeight)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**3. Code for Capturing images for model training :**

* **Img.py:**

import cv2

import time

cpt = 0

maxFrames = 70 # if you want 5 frames only.

cap=cv2.VideoCapture(0)

while cpt < maxFrames:

ret, frame = cap.read()

frame=cv2.resize(frame,(640,480))

cv2.imshow("test window", frame) # show image in window

cv2.imwrite("/home/pi/rpibookworm-yolov4tiny-main/images/safety-jacket\_%d.jpg" %cpt, frame)

time.sleep(0.5)

cpt += 1

if cv2.waitKey(1)&0xFF==27:

break

cap.release()

cv2.destroyAllWindows()

train.py

import numpy as np

import os

from tflite\_model\_maker.config import ExportFormat, QuantizationConfig

from tflite\_model\_maker import model\_spec

from tflite\_model\_maker import object\_detector

from tflite\_support import metadata

import tensorflow as tf

assert tf.\_\_version\_\_.startswith('2')

tf.get\_logger().setLevel('ERROR')

from absl import logging

logging.set\_verbosity(logging.ERROR)

train\_data = object\_detector.DataLoader.from\_pascal\_voc(

'freedomtech/train',

'freedomtech/train',

['helmet', 'safety-jacket']

)

val\_data = object\_detector.DataLoader.from\_pascal\_voc(

'freedomtech/validate',

'freedomtech/validate',

['helmet', 'safety-jacket']

)

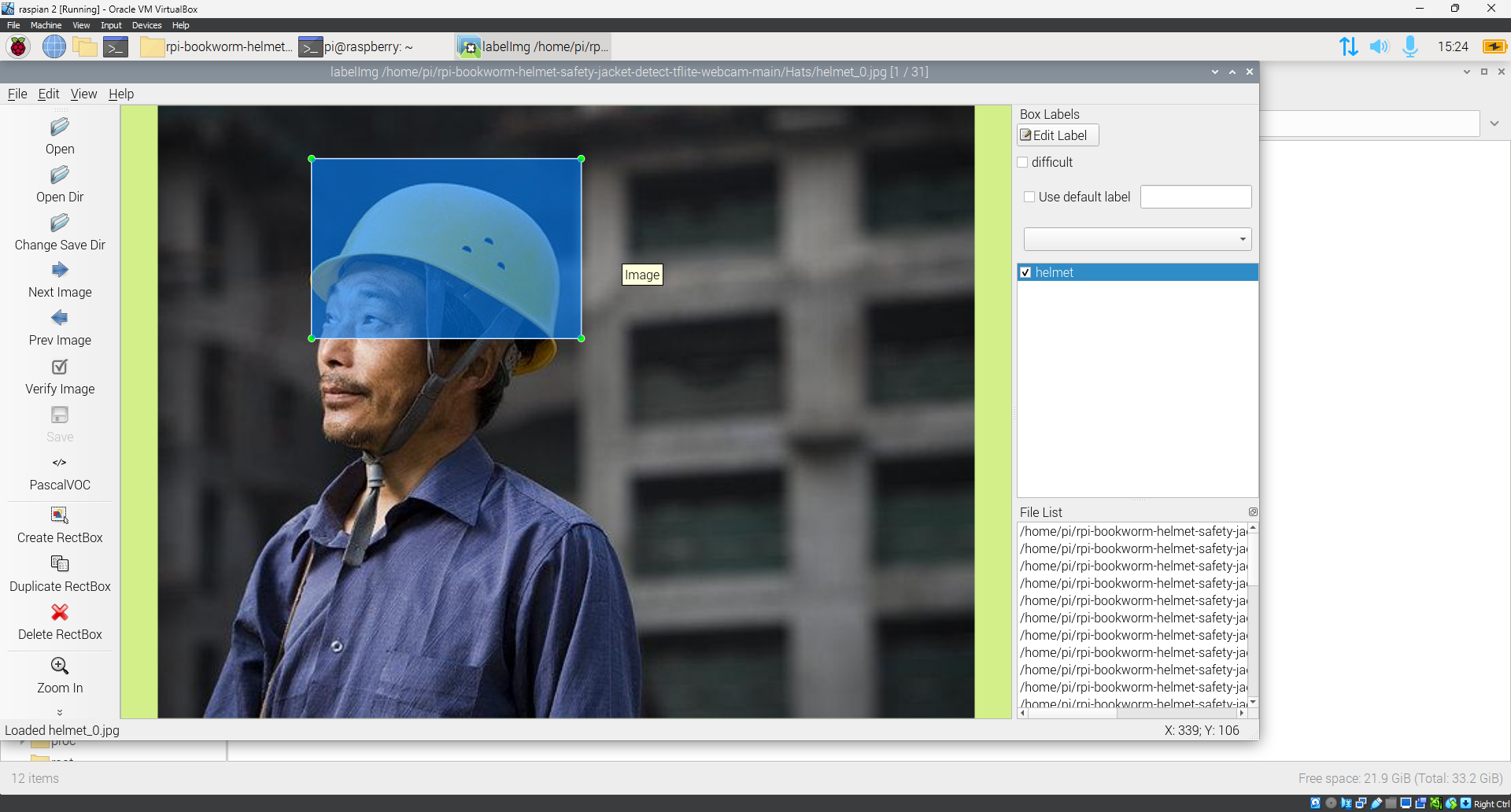
spec = model\_spec.get('efficientdet\_lite0')

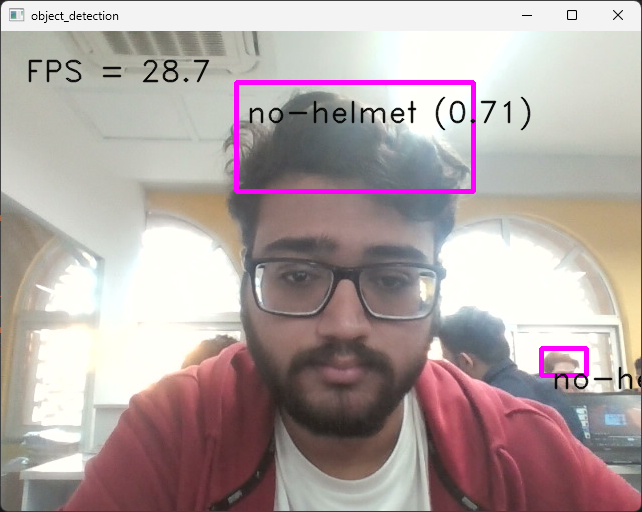
model = object\_detector.create(train\_data, model\_spec=spec, batch\_size=4, train\_whole\_model=True, epochs=100, validation\_data=val\_data)

model.evaluate(val\_data)

model.export(export\_dir='.', tflite\_filename='best.tflite')

**Result**





1. [↑](#footnote-ref-1)