**Helmet detection in ATM**

!pip install pygame

!pip install pydub

import cv2

import numpy as np

from IPython.display import display, Image, Audio

import time

# Define paths to YOLOv3 model and configuration files

weights\_path = '/content/drive/MyDrive/helmet/yolov3-helmet.weights'

config\_path = '/content/drive/MyDrive/helmet/yolov3-helmet.cfg'

names\_path = '/content/drive/MyDrive/helmet/helmet.names' # File containing class names

# Load YOLOv3 model

net = cv2.dnn.readNet(weights\_path, config\_path)

# Load class names for helmet detection

with open(names\_path, 'r') as f:

classes = f.read().strip().split('\n')

# Define the path to your video file

video\_path = '/content/drive/MyDrive/helmet.mp4'

# Initialize video capture from the specified file

cap = cv2.VideoCapture(video\_path)

# Define the buzzer sound

buzzer\_sound\_path = '/content/drive/MyDrive/16945006549878f5ach8-voicemaker.in-speech.mp3'

# Define the interval for capturing frames (10 seconds)

frame\_capture\_interval = 1 # in seconds

# Initialize the timer

start\_time = time.time()

while True:

# Read a frame from the video

ret, frame = cap.read()

if not ret:

break

# Get frame dimensions

height, width, \_ = frame.shape

# Calculate the elapsed time

elapsed\_time = time.time() - start\_time

if elapsed\_time >= frame\_capture\_interval:

# Reset the timer

start\_time = time.time()

# Create a blob from the input frame

blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)

net.setInput(blob)

# Get output layer names

layer\_names = net.getLayerNames()

output\_layer\_names = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Perform forward pass through the network

outs = net.forward(output\_layer\_names)

# Initialize lists for bounding boxes, confidences, and class IDs

boxes = []

confidences = []

class\_ids = []

# Set confidence threshold for detection

conf\_threshold = 0.5

# Set non-maximum suppression threshold

nms\_threshold = 0.4

# Iterate over each detection

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > conf\_threshold and classes[class\_id] == "Helmet":

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

# Apply non-maximum suppression

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)

# Draw bounding boxes and labels on the frame

if len(indices) > 0:

for i in indices.flatten():

x, y, w, h = boxes[i]

label = classes[class\_ids[i]]

confidence = confidences[i]

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

text = f"{label}: {confidence:.2f}"

cv2.putText(frame, text, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

# Play the buzzer sound when a helmet is detected

display(Audio(buzzer\_sound\_path, autoplay=True))

# Convert the frame to RGB format for display

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Display the frame with detections

display(Image(data=cv2.imencode('.jpg', frame\_rgb)[1]))

# Release the video file

cap.release()