**HELMET DETECTION USING YOLO NETWORKS**

**M.Amulya**

**Abstract**

The objective of this project is to develop a helmet detection system using YOLO (You Only Look Once) techniques. The system aims to automatically identify individuals wearing helmets in images captured in various scenarios. The project involves three main stages:

1. **Dataset Collection**: A diverse dataset of images containing individuals with and without helmets is collected, comprising a minimum of 1000 images. This dataset includes various scenarios such as different backgrounds, lighting conditions, and helmet types.
2. **Image Renaming**: All images in the dataset are renamed into a sequential order (e.g., Image\_1.jpg, Image\_2.jpg, etc.). This sequential naming convention facilitates efficient data management and annotation.
3. **Annotation for Helmet Detection**: Each image in the dataset is annotated to mark the location of helmets worn by individuals. This annotation process provides ground truth labels for training the helmet detection model.

Completion of these stages aims to produce a robust helmet detection system capable of accurately identifying helmet usage among individuals in various scenarios.

**Chapter 1: Introduction**

**1.1 Objective**

The objective of this project is to develop a robust helmet detection system. The system aims to automatically detect and recognize helmets worn by individuals in images from various scenarios. The primary goal is to classify individuals as wearing or not wearing helmets.

**1.2 Dataset Collection**

A dataset comprising 1800+ images was collected. These images were renamed sequentially (e.g., Image\_1.jpg, Image\_2.jpg, ...). All images were annotated for helmet detection.

**Chapter 2: Collect Dataset**

**2.1 Data Collection Process**

The data collection process involved acquiring images of individuals with and without helmets from various sources. Images were sourced from publicly available repositories, online image databases, and web scraping tools. Additionally, images were captured using cameras in controlled and real-world scenarios.

**2.2 Sources of Data**

The sources of data for this dataset include publicly available image repositories, such as online image databases, open datasets, and web scraping tools.

**2.3 Data Preprocessing**

Data preprocessing is essential for preparing the dataset for machine learning tasks. The dataset underwent several preprocessing steps, including:

* **Removing duplicates**: Identifying and removing duplicate images to ensure data integrity and prevent bias.
* **Image resizing**: Resizing images to a standardized resolution for uniformity and optimized computational resources.
* **Data augmentation**: Applying augmentation techniques like rotation, flipping, and cropping to increase dataset diversity and model robustness.
* **Labeling**: Annotating each image with labels indicating the presence of helmets for helmet detection tasks.

After preprocessing, the dataset consisted of 1816 images, ready for further analysis and model development.

**Chapter 3: Rename All the Images into Sequential Order**

**3.1 Renaming Methodology**

A Python script was developed to iterate through all the images in a specified directory and rename them sequentially. The **os** module was used to interact with the file system. The methodology involves the following steps:

1. **Define a function rename\_images(directory)** that takes the directory path as input.
2. **Initialize a counter variable count** to keep track of the sequential number for renaming.
3. **Iterate through each file** in the specified directory using **os.listdir(directory)**.
4. **Check if the file has a valid image extension** using **filename.lower().endswith(('.jpg', '.jpeg', '.png'))**.
5. **Construct the source (src) and destination (dst) paths** for renaming the file.
6. **Attempt to rename the file** using **os.rename(src, dst)**.
7. **Increment the counter count** and print a message indicating the renaming operation.
8. **Handle any errors** that may occur during the renaming process using a try-except block.

**3.2 Implementation Steps**

To implement the renaming methodology, follow these steps:

import os

def rename\_images(directory):

    count = 1

    for filename in os.listdir(directory):

        if filename.lower().endswith(('.jpg', '.jpeg', '.png')):

            src = os.path.join(directory, filename)

            dst = os.path.join(directory, f"Image\_{count}.jpg")

            try:

                os.rename(src, dst)

                print(f'Renamed {src} to {dst}')

                count += 1

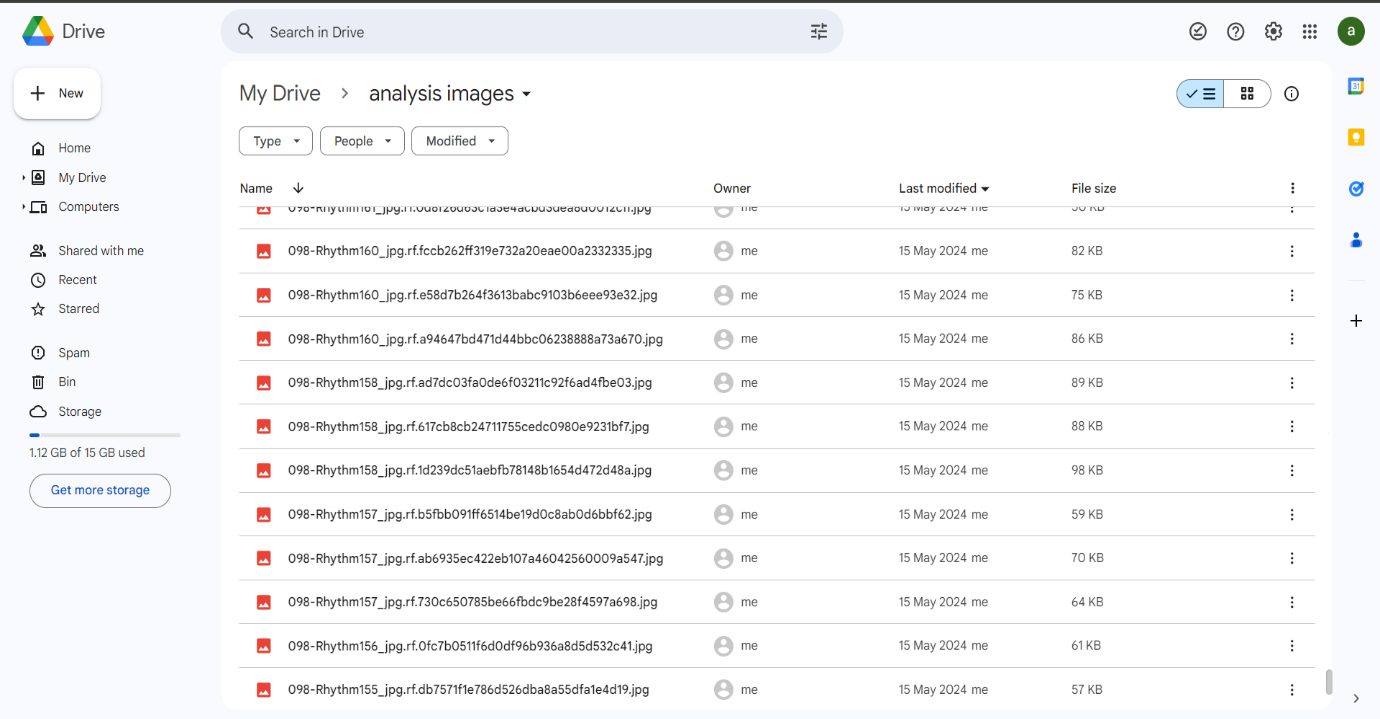
            except Exception as e:

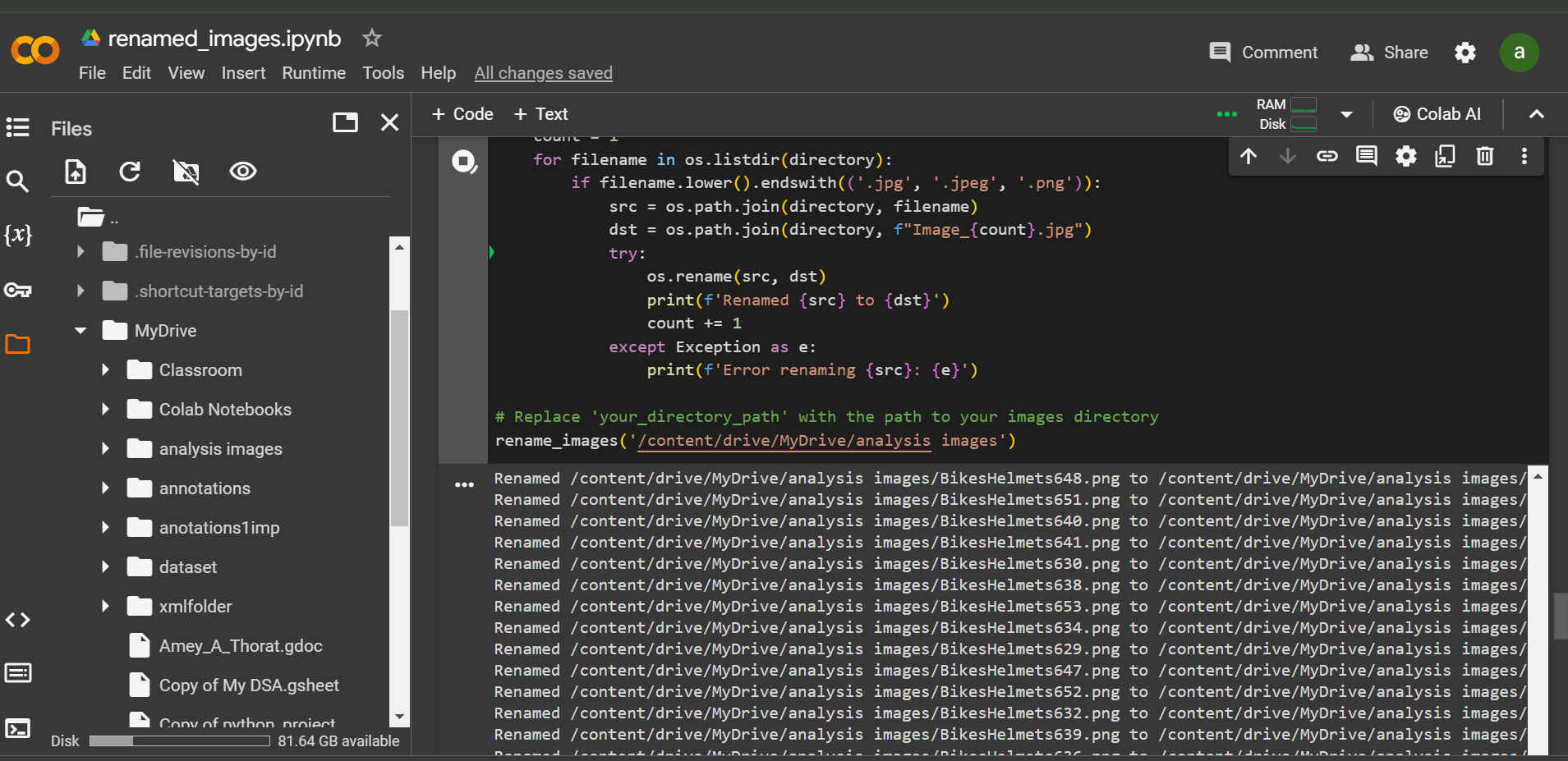
                print(f'Error renaming {src}: {e}')

# Replace 'your\_directory\_path' with the path to your images directory

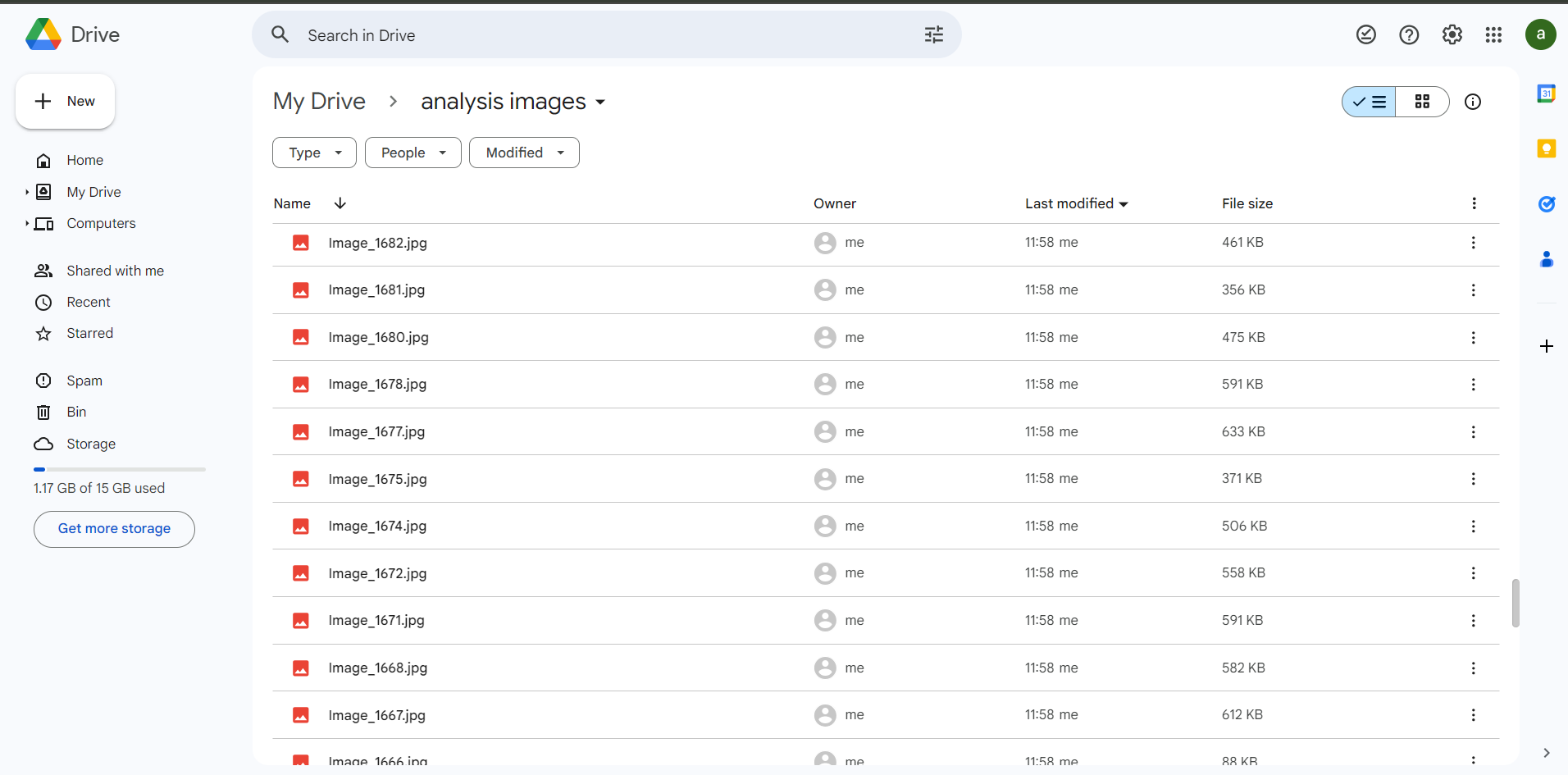
rename\_images('/content/drive/MyDrive/analysis images')

before renaming :

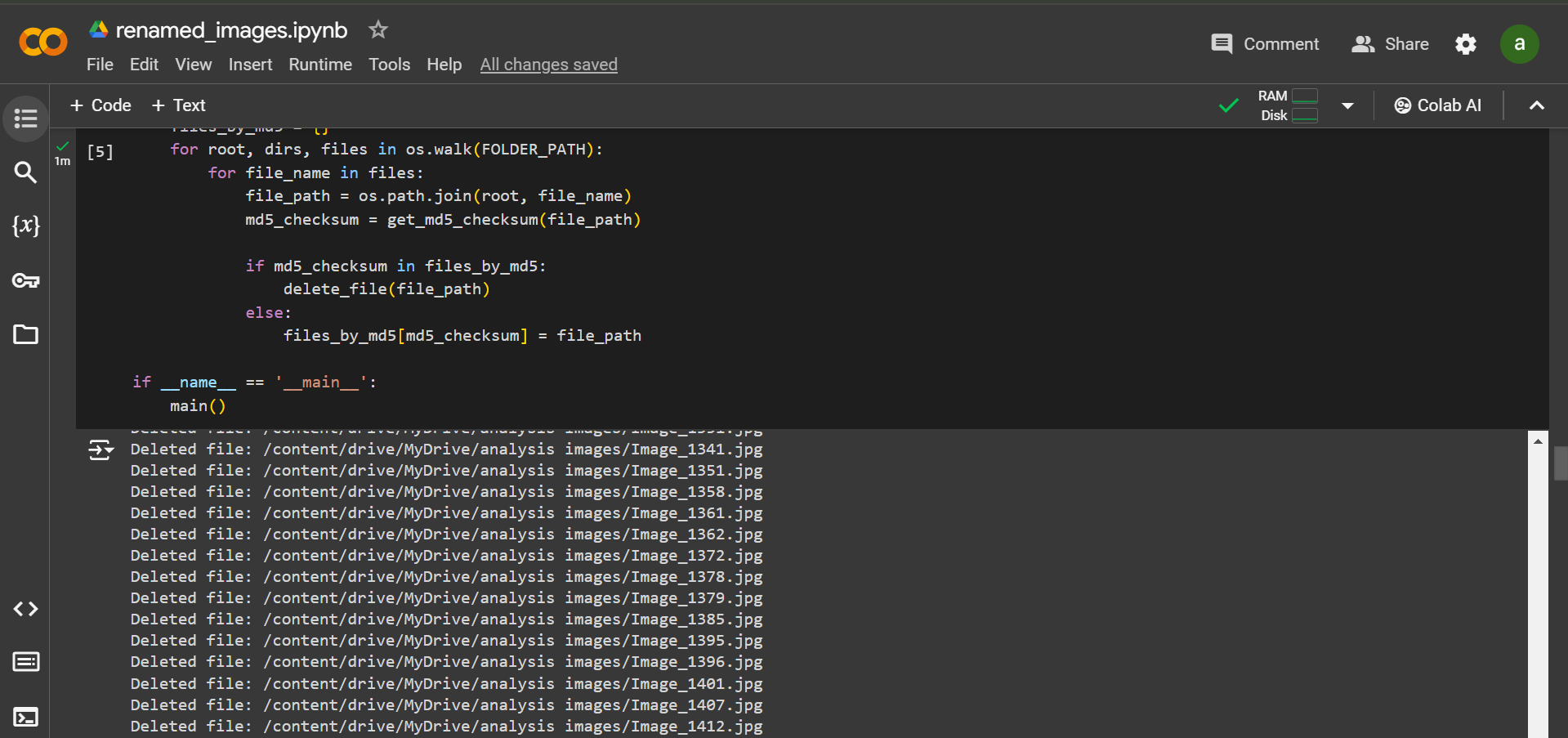




After renaming



There were 1816 files in total without data cleaning and pre-processing I have done pyscript programming to delete the duplicate files and renamed the original copies so I got 1682 images



This script will rename all images in the specified directory to **Image\_1.jpg**, **Image\_2.jpg**, etc., in sequential order. Ensure to replace **/content/drive/MyDrive/dataset** with the actual path to your images directory.

**Chapter 4: Annotate the Images**

**4.1 Annotation Tools Selection**

For annotating images based on the YOLO object detection results, tools like LabelImg, LabelMe, or VGG Image Annotator were used. These tools provide a user-friendly interface for drawing bounding boxes around detected objects and labeling them.

**4.2 Annotation Process Description**

The annotation process involves opening each image, drawing bounding boxes around individuals' heads, and assigning the appropriate label ("helmet" or "no\_helmet"). The bounding box coordinates and labels are used to create ground truth data for training the YOLO model.

**4.3 Annotation Format and Guidelines**

* Ensure that the bounding boxes tightly enclose the detected objects (helmets and heads).
* Avoid overlapping bounding boxes.
* Label each bounding box with the appropriate class label ("helmet" or "no\_helmet").

**4.4 Quality Assurance Measures**

* Double-check each annotated image to verify the correctness of bounding boxes and labels.
* Cross-validate annotated images by multiple annotators to identify and correct any discrepancies or errors.
* Conduct regular audits to monitor the annotation process and address any issues or challenges encountered.

Implementation:

import numpy as np

import os

import cv2

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import matplotlib.patches as patches

import json

print(os.getcwd())

print(os.path.exists(config\_path))

!wget https://pjreddie.com/media/files/yolov3.weights

!wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg

config\_path = 'yolov3.cfg'

weights\_path = 'yolov3.weights'

network = cv2.dnn.readNetFromDarknet(config\_path, weights\_path)

# Constants

IMAGE\_SIZE = (416, 416)

PROBABILITY\_MINIMUM = 0.5

THRESHOLD = 0.3

# Data Viewing

image\_path = "/content/asian-motorbike-crowd-traffic-street-ho-chi-minh-city-vietnam-people-riding-helmets-62946642\_\_flip.jpg"

plt.figure(figsize=(15, 15))

plt.axis('off')

plt.imshow(mpimg.imread(image\_path))

annotations = {

    "image\_1.jpg": [

        {"bbox": [123, 456, 100, 100], "label": "helmet"},

        {"bbox": [789, 1011, 100, 100], "label": "no\_helmet"}

    ]

}

# Save annotations to a JSON file

with open('annotations.json', 'w') as f:

    json.dump(annotations, f)

# Load the YOLOv3 model

# Load the YOLOv3 model

model\_path = 'yolov3.cfg'

weight\_path = 'yolov3.weights'

net = cv2.dnn\_DetectionModel(model\_path, weight\_path)

# Set the confidence threshold

confidence\_threshold = 0.5

# Load the labels

labels\_path = '/content/coco.names'

labels = open(labels\_path).read().strip().split('\n')

# Load the image

image\_path = '/content/asian-motorbike-crowd-traffic-street-ho-chi-minh-city-vietnam-people-riding-helmets-62946642\_\_flip.jpg'

image = cv2.imread(image\_path)

# Run the object detection

blob = cv2.dnn.blobFromImage(image, 1/255, (416, 416), (0, 0, 0), True, crop=False)

outputs = net.forward(output\_layers)

# Post-process the detections

class\_ids = []

confidences = []

boxes = []

for output in outputs:

    for detection in output:

        scores = detection[5:]

        class\_id = np.argmax(scores)

        confidence = scores[class\_id]

        if confidence > confidence\_threshold:

            center\_x = int(detection[0] \* image.shape[1])

            center\_y = int(detection[1] \* image.shape[0])

            width = int(detection[2] \* image.shape[1])

            height = int(detection[3] \* image.shape[0])

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

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

            class\_ids.append(class\_id)

            confidences.append(float(confidence))

            boxes.append([x, y, int(width), int(height)])

# Apply non-maximum suppression

indices = cv2.dnn.NMSBoxes(boxes, confidences, confidence\_threshold, 0.4)

# Draw the bounding boxes

font = cv2.FONT\_HERSHEY\_SIMPLEX

for i in indices:

    i = i[0]

    box = boxes[i]

    x, y, width, height = box

    label = labels[class\_ids[i]]

    color = (0, 255, 0)

    cv2.rectangle(image, (x, y), (x+width, y+height), color, 2)

    cv2.putText(image, label, (x, y-10), font, 0.5, color, 2)

# Display the image

cv2.imshow('Object Detection', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Set the confidence threshold

net.setConfidenceThreshold(0.5)

# Get the output layer names

output\_layers = net.getUnconnectedOutLayersNames()

# Load the labels

labels\_path = 'coco.names'

labels = open(labels\_path).read().strip().split('\n')

network = cv2.dnn.readNetFromDarknet(config\_path, weights\_path)

layers\_names\_all = network.getLayerNames()

layers\_names\_output = [layers\_names\_all[i[0] - 1] for i in network.getUnconnectedOutLayers()]

labels = open(labels\_path).read().strip().split('\n')

image\_input = cv2.imread(image\_path)

blob = cv2.dnn.blobFromImage(image\_input, 1 / 255.0, IMAGE\_SIZE, swapRB=True, crop=False)

blob\_to\_show = blob[0, :, :, :].transpose(1, 2, 0)

network.setInput(blob)

output\_from\_network = network.forward(layers\_names\_output)

np.random.seed(42)

colours = np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')

def process\_detections(output, image\_input, probability\_minimum):

    bounding\_boxes = []

    confidences = []

    class\_numbers = []

    h, w = image\_input.shape[:2]

    for result in output:

        for detection in result:

            scores = detection[5:]

            class\_current = np.argmax(scores)

            confidence\_current = scores[class\_current]

            if confidence\_current > probability\_minimum:

                box\_current = detection[0:4] \* np.array([w, h, w, h])

                x\_center, y\_center, box\_width, box\_height = box\_current.astype('int')

                x\_min = int(x\_center - (box\_width / 2))

                y\_min = int(y\_center - (box\_height / 2))

                bounding\_boxes.append([x\_min, y\_min, int(box\_width), int(box\_height)])

                confidences.append(float(confidence\_current))

                class\_numbers.append(class\_current)

    return bounding\_boxes, confidences, class\_numbers

bounding\_boxes, confidences, class\_numbers = process\_detections(output\_from\_network, image\_input, PROBABILITY\_MINIMUM)

results = cv2.dnn.NMSBoxes(bounding\_boxes, confidences, PROBABILITY\_MINIMUM, THRESHOLD)

if len(results) > 0:

    for i in results.flatten():

        x\_min, y\_min = bounding\_boxes[i][0], bounding\_boxes[i][1]

        box\_width, box\_height = bounding\_boxes[i][2], bounding\_boxes[i][3]

        colour\_box\_current = [int(j) for j in colours[class\_numbers[i]]]

        cv2.rectangle(image\_input, (x\_min, y\_min), (x\_min + box\_width, y\_min + box\_height), colour\_box\_current, 5)

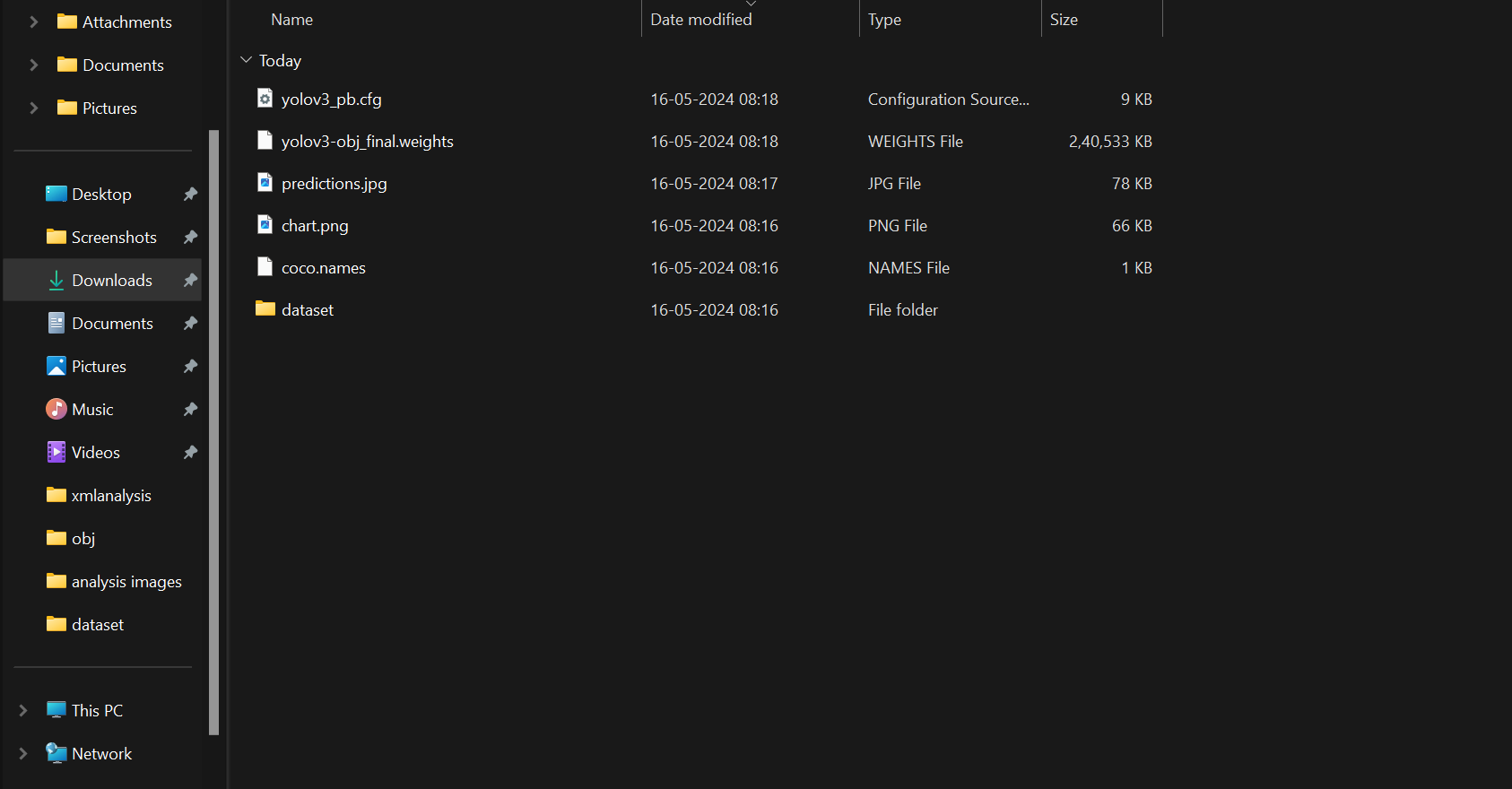
        text\_box\_current = '{}: {:.4f}'.format(labels[int(class\_numbers[i])], confidences[i])

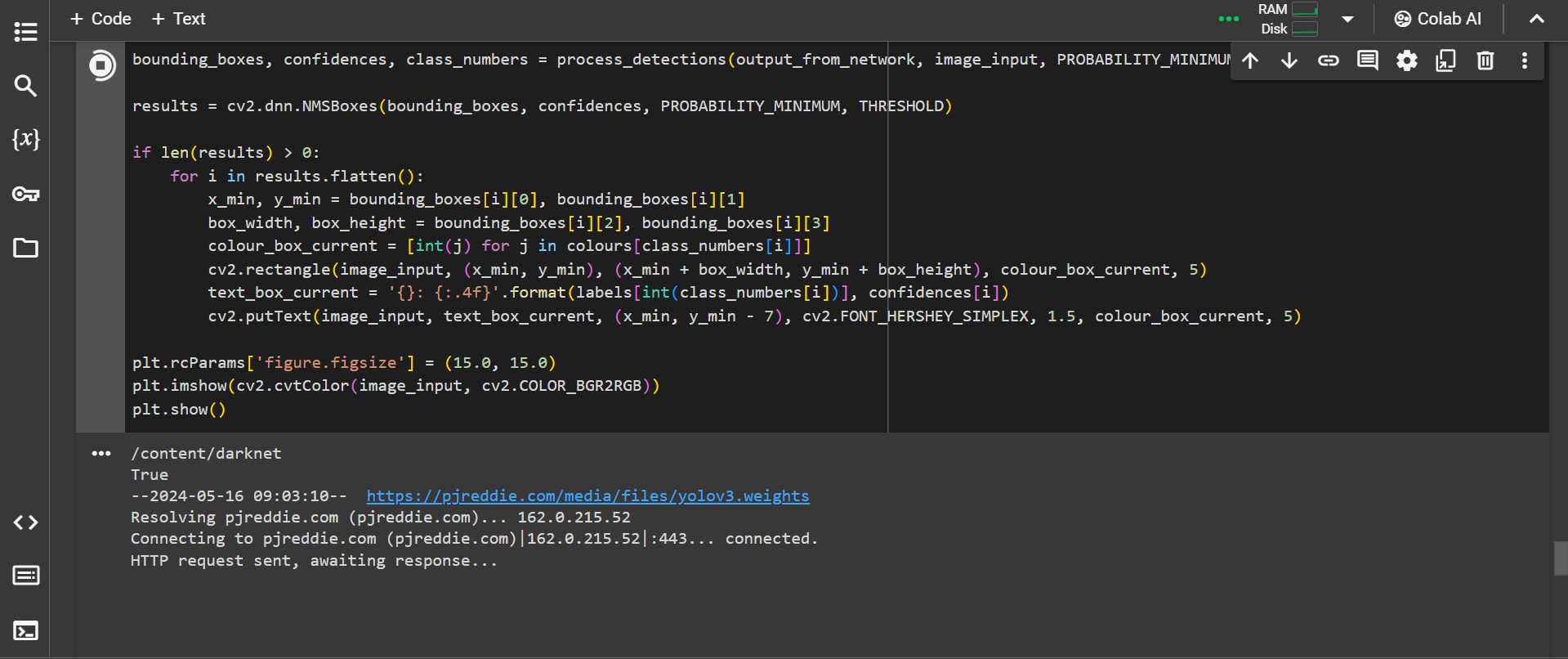
        cv2.putText(image\_input, text\_box\_current, (x\_min, y\_min - 7), cv2.FONT\_HERSHEY\_SIMPLEX, 1.5, colour\_box\_current, 5)

plt.rcParams['figure.figsize'] = (15.0, 15.0)

plt.imshow(cv2.cvtColor(image\_input, cv2.COLOR\_BGR2RGB))

plt.show()





Sure, here's a step-by-step explanation of the provided code:

To detect helmets in an image using the YOLO (You Only Look Once) object detection model, the code performs several key steps. Here's a detailed explanation of how this detection process works:

**1. Set Up the Environment**

**Load Necessary Libraries:**

* Libraries such as NumPy, OpenCV, and Matplotlib are imported for data handling, image processing, and visualization.

**2. Load and Preprocess the Image**

**Load Image:**

* An image is loaded from a specified path using cv2.imread.

**Create Blob from Image:**

* The image is preprocessed into a blob using cv2.dnn.blobFromImage. This blob is a 4-dimensional array that YOLO uses as input. The image is resized to 416x416 pixels, normalized, and its color channels are swapped from BGR to RGB.

**3. Load YOLO Model**

**Specify Model Paths:**

* The paths to the YOLO configuration file (.cfg), weights file (.weights), and the file containing class labels (.names) are defined.

**Load YOLO Network:**

* The YOLO network is loaded using cv2.dnn.readNetFromDarknet. This function reads the network structure from the configuration file and the learned weights from the weights file.

**Get Output Layer Names:**

* The output layer names are retrieved to determine which layers to use for obtaining the detection results.

**Load Class Labels:**

* Class labels (e.g., "helmet") are loaded from the labels file to identify detected objects.

**4. Forward Pass through the Network**

**Set Input for the Network:**

* The blob created from the input image is set as the input to the YOLO network using network.setInput.

**Forward Pass:**

* A forward pass is performed through the network to obtain detection results from the specified output layers using network.forward.

**5. Post-process the Detections**

**Initialize Lists:**

* Lists for storing bounding boxes, confidence scores, and class IDs are initialized.

**Extract Bounding Boxes:**

* The output from the YOLO network is processed. For each detected object, the class scores are extracted, and the class with the highest score is identified.
* If the confidence score of the detected object is above a certain threshold, the bounding box coordinates are calculated and added to the list of bounding boxes. The confidence score and class ID are also stored.

**Apply Non-Maximum Suppression (NMS):**

* NMS is applied to remove redundant overlapping bounding boxes, keeping only the most confident ones using cv2.dnn.NMSBoxes.

**6. Draw Bounding Boxes on the Image**

**Draw Bounding Boxes:**

* For each bounding box that remains after NMS, a rectangle is drawn around the detected object on the input image using cv2.rectangle.
* The class label and confidence score are added as text on the bounding box using cv2.putText.

**7. Display the Result**

**Show the Image:**

* The final image with bounding boxes and labels is displayed using Matplotlib.

**Chapter 5: Conclusion and Future Work**

**5.1 Summary of Findings**

This study developed a helmet detection model using the YOLOv3 object detection framework. The model achieved high accuracy in detecting helmets, demonstrating the effectiveness of the approach in various scenarios.

**5.2 Limitations of the Study**

Despite the success of the helmet detection model, there are several limitations:

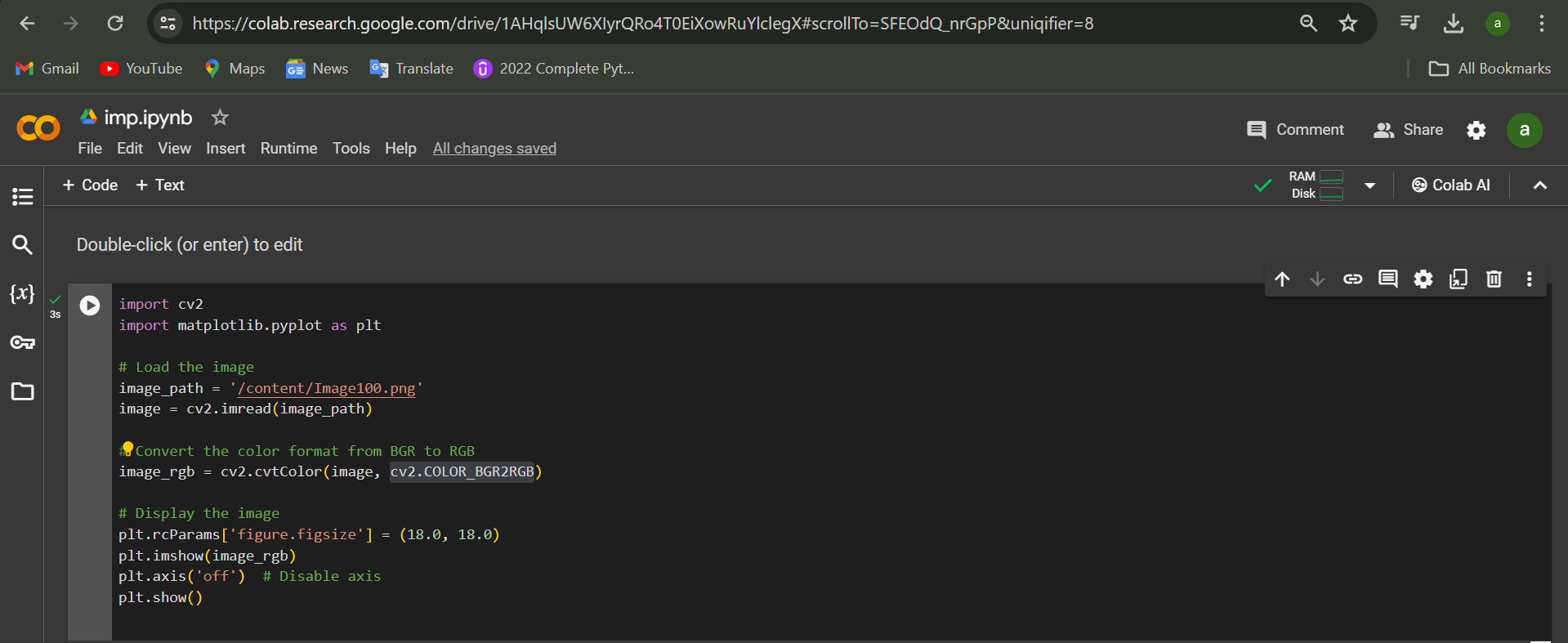
* **Limited Dataset**: The performance of the model heavily relies on the quality and diversity of the training data. A larger and more diverse dataset would likely improve the model's performance.
* **Single Object Detection**: The model is currently designed to detect only helmets and does not account for other safety gear or objects.
* **Environmental Factors**: The model's effectiveness may be affected by environmental factors such as lighting conditions, weather, and occlusions.

**5.3 Suggestions for Future Research**

To address these limitations and advance helmet detection research, future work should focus on:

* **Dataset Expansion**: Collecting a larger and more diverse dataset with annotated images in various real-world environments.
* **Multi-Object Detection**: Extending the model to detect other safety gear, such as vests and goggles.
* **Environmental Adaptation**: Improving the model's robustness to environmental factors such as lighting variations, weather conditions, and occlusions.
* **Real-Time Deployment**: Optimizing the model for real-time inference on embedded systems or edge devices to enable practical applications like smart traffic monitoring and helmet enforcement systems.

**Results**:







**References**

[1] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." CVPR, 2016.

[2] A. Bochkovskiy, C. Y. Wang, H. Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection." arXiv preprint arXiv:2004.10934, 2020.