**Abstract**

Bike rider helmet detection is a crucial computer vision task aimed at improving road safety by identifying bike rider who are not wearing helmets. This system leverages deep learning models, such as YOLOv8, to accurately detect helmets in real-time or from images or video streams. It can be integrated into traffic

monitoring systems to automatically flag violations and enhance law enforcement efficiency. The implementation involves preprocessing images, training the model, and saving the model. I have also created a streamlit and django web application and deployed the web application in streamlit cloud. The proposed system helps reduce head injuries and fatalities in bike rider accidents by promoting helmet use.

**Index**

|  |  |  |
| --- | --- | --- |
| Sr. No | Topic | Pg. No |
| 1 | 1. Introduction    1. Computer Vison : Introduction    2. Bike Rider Helmet Detection : Introduction | 1 |
| 2 | 1. Objective    1. Problem    2. Goal | 2 |
| 3 | 1. Tools and Technology    1. Hardware Requirements    2. Sofware Requirements | 3 |
| 4 | 1. Literature Review    1. Image Acquisition    2. Image Preprocessing    3. Feature Extraction    4. Object Detection and Recognition    5. Image Segmentation    6. Motion Analysis    7. Scene Understanding | 4 |
| 5 | 1. Methodology    1. Data Collection    2. Data pre-processing    3. Model Training    4. Model Evaluation    5. Model Testing    6. Deployment | 5 |

|  |  |  |
| --- | --- | --- |
| 6 | 1. Implementation details    1. Dataset Used    2. YOLOv8 Metrics    3. What is YOLOv8?    4. Why YOLOv8?    5. What is Ultralytics?    6. What are base models?    7. What are target models?    8. How pre-processing of the dataset is done?    9. How images are pre-processed?    10. How videos are pre-processed?    11. What is Ontology?    12. Sample Code    13. Data and YOLOv8 configuration file | 6-16 |
| 7 | 1. Analysis of result    1. Confusion Matrix    2. Precision -Recall Curve and Train, Validation and Loss metrics    3. F1-Confidence and Recall-Confidence Curve    4. Precision-Recall Curve    5. Labels-Class Chart    6. Inference on Images    7. Inference on Videos    8. Django Web application Demo    9. Streamlit Web application Demo    10. Mkdocs Documentation Demo | 17-31 |
| 8 | Conclusion | 32 |
| 9 | Future Scope | 33 |
| 10 | Program code | 34 |
| 11 | Reference | 35 |

**Chapter 1**

1. **Introduction**

# Computer Vison: Introduction

Computer Vision is an interdisciplinary field of study that enables computers to interpret and understand visual information from the world, much like the human visual system. It encompasses the development of algorithms, models, and systems that can process, analyze, and extract meaningful insights from visual data, typically in the form of images and videos. Computer Vision has wide-ranging applications across various industries, including healthcare, automotive, entertainment, surveillance, robotics, and more. This overview provides a comprehensive understanding of Computer Vision, its key components, applications, challenges, and future prospects.

# Bike Rider Helmet Detection: Introduction

The importance of road safety has been increasingly recognized, leading to the implementation of various measures to protect vulnerable road users such as cyclists and motorcyclists or bike riders. One critical safety measure is the use of helmets, which significantly reduces the risk of head injuries in the event of an accident. Despite regulations mandating helmet use, compliance remains a challenge. To address this, the Bike Rider Helmet Detection project aims to leverage advanced computer vision techniques to automatically detect helmet usage among bike riders and passangers.

The Bike Rider Helmet Detection project utilizes one of the state-of-the-art deep learning models to accurately identify whether a bike rider is wearing a helmet. The core of the project is built on the YOLOv8 (You Only Look Once version 8) model, a highly efficient and powerful object detection algorithm developed and maintained by Ultralytics. YOLOv8 is renowned for its speed and accuracy, making it an ideal choice for real-time applications like helmet detection.

There are a significant number of the accidents on the roads today. The accident can have multiple scenarios whether due to pot holes or any car accidentally bumbs bike driver while driving etc. By wearing a helmet not only we are saving our head from any fatal accidents but also saving any passanger life too.

This explains why it is important to do more work in this field with an aim to reduce the occurrence of accidents related to any driving related injury and motivate overselves the importance of a helmet while riding a 2 wheeler vehicle.

**Chapter 2**

1. **Objective**

# Problem:

Bike riders drivers who do not wear helmet which may result in fatal accidents and death in some cases.

# Goal:

Create a deep learning model that an detect if a person is wearing helmet or not.

The main objective of this project is to develop a system that would detect if the bike rider along with the passengers are wearing a safety helmet or not. The dataset is collected through kaggle dataset. As manually annotation of the labels or semi label annotation would take significant amount of time. Only images are needed and and no explicit annotations are needed for this project as I am using ultralytics foundation models for auto labelling the images and video images to target model of YOLOv8. After training the model I have made a streamlit and django web application for this project along with documentation.

**Chapter 3**

1. **Tools and Technology**

As this is a deep learning project a significant amount of computation and memory allocation does matter a lot for this project. For inference of model on images and videos too required a significant amount of computation power.

# Hardware Requirements :-

1. Desktop / Laptop / Server
2. 8 GB RAM at least
3. 150 GB Disk space or higher
4. Any processor Intel i5 / AMD
5. Google GPU - Tesla T4

# Sofware Requirements :-

1. Windows / Ubuntu os (64 or 32 bit)
2. Google colab / Kaggle jupyter notebook
3. Python 3.10.12 or higher
4. Visual studio code editor, jupyter notebook
5. Sqlite version 0.5.6 or higher

For more in depth requirements of the major, minor packages and respective project dependencies. Please take a look at the respective github repository.

**Chapter 4**

1. **Literature Review**

# Key Components of Computer Vision:

* 1. **Image Acquisition:**

Computer Vision begins with the acquisition of visual data, which can come from various sources, including cameras, sensors, or image databases.

# Image Preprocessing:

Raw visual data often requires preprocessing to enhance quality, remove noise, and prepare it for analysis. This includes tasks like image resizing, filtering, and color correction.

# Feature Extraction:

Feature extraction involves identifying and isolating relevant visual patterns or features within an image, such as edges, corners, or texture.

# Object Detection and Recognition:

This component focuses on identifying and classifying objects within images or videos. Object detection and recognition algorithms enable computers to recognize and label objects, faces, or specific patterns.

# Image Segmentation:

Image segmentation divides an image into meaningful regions or segments. It's crucial for tasks like medical image analysis, where different parts of an image may represent different anatomical structures.

# Motion Analysis:

Motion analysis techniques track moving objects and can be used in applications like surveillance, sports analysis, and robotics.

# Scene Understanding:

This involves higher-level interpretation of images or videos to understand the context and relationships between objects within a scene.

**4**

**Chapter 5**

1. **Methodology**

# Data Collection:

Collect publicly available images / videos of bike rider, helmet and no helmet for the model to train upon. The images are collected from kaggle dataset

# Data pre-processing:

Pre-processing and auto labeling images and videos are done by ultralytics framework by using a Grounding Segment Anything Model based on Ontology. For videos every 3 frame is considered.

# Model Training:

Train YOLOv8 Model on custom dataset with 3 classes, Socastic Gradient Decent (SGD) optimizer, learning rate 0.01, batch size 16, epochs 50, image size 640, momentum 0.937, validation split around 20 %.

# Model Evaluation:

Evaluate trained YOLOv8 Model on validation dataset with the metrics such as box loss, class loss, precision and recall. Confusion matrix.

# Model Testing:

Run Inference on images and videos. To check how well the model perform.

# Deployment:

The final model is deployed as a streamlit web application and django for local deployment along with application programming interface (API).

**5**

**Chapter 6**

1. **Implementation details**

# Dataset Used:

The dataset that is used for this project can be find below:- <https://www.kaggle.com/datasets/andrewmvd/helmet-detection>

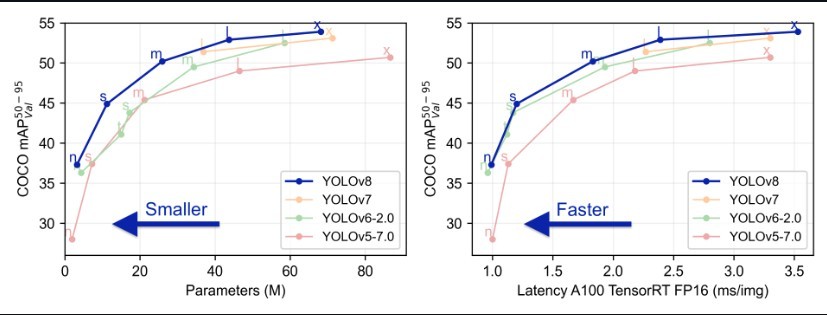
This dataset contains 764 images of 2 distinct classes for the objective of helmet detection.

Bounding box annotations are provided in the PASCAL VOC format.

The classes are:

* With helmet
* Without helmet

# YOLOv8 Metrics

****

* 1. **What is YOLOv8?**

YOLOv8 is from the YOLO family of models and was released on January 10, 2023. YOLO stands for You Only Look Once, and this series of models are thus named because of their ability to predict every object present in an image with one forward pass.

# Why YOLOv8?

**YOLOv8** by ultalytics is a state-of-the-art deep learning model designed for real- time object detection in computer vision applications. With its advanced architecture and cutting-edge algorithms, YOLOv8 has revolutionized the field of object detection, enabling accurate and efficient detection of objects in real-time scenarios.

YOLOv8 is quite stable as compare to latest YOLOv9 and recent YOLOv10.

# What is Ultralytics?

**Ultralytics** is a company that specializes in developing advanced computer vision technologies, particularly the YOLO (You Only Look Once) series of models. Their framework, Ultralytics YOLO, provides state-of-the-art object detection, image segmentation, and classification capabilities. The base model, often a pre-trained YOLOv8 model or it can be any model like YOLOv5, can be fine-tuned or used directly to create custom target models for various applications for any custom dataset. This framework is widely used for real-time object detection tasks due to its speed and accuracy.

# What are base models?

**Base Model** - A Base Model is a large foundation model that knows a lot about a lot. Base models are often multimodal and can perform many tasks. They're large, slow, and expensive. Examples of Base Models are GroundedSAM and GPT-4's upcoming multimodal variant. We use a Base Model (along with unlabeled input data and an Ontology) to create a Dataset.

# What are target models?

**Target Model** - a Target Model is a supervised model that consumes a dataset and outputs a distilled model that is ready for deployment. Target Models are usually small, fast, and fine-tuned to perform a specific task very well (but they don't generalize well beyond the information described in their Dataset). Examples of Target Models are YOLOv8 and YOLOv5.

# How pre-processing of the dataset is done?

Only images and videos are needed for this project and no annotations are needed from the data as annotation are generated by ultralytics framework called as **Autodistill** which uses ***Grounding SAM which is combination of Grounding DiNO (Zero short object detection model) and SAM (Segment Anything Model) (zero short object detection with prompting) from Meta for autolabeling dataset and preprocess and train on YOLOv8 large model***.

**Autodistill** uses big, slower foundation models to train small, faster supervised models. Using autodistill, you can go from unlabeled images to inference on a custom model running at the edge with no human intervention in between.

As foundation models get better and better they will increasingly be able to augment or replace humans in the labeling process.

# How images are pre-processed?

The dataset image consist of 764 images of various Bike rider, Rider wearing helmet, Rider not wearing helmet. Out of 764 images only 611 images are used. I have removed some images because my google colab kernel is crashing significantly.

# How videos are pre-processed?

Video is processed is such a way in which every 3 frame (can be changed through code) are considered as image data for the model to train upon.

For autolabelling the dataset of images and video frame autodistill uses something which is called as ontology

# What is Ontology?

**Ontology** - an Ontology defines how your Base Model like Grounding SAM is prompted, what your Dataset will describe, and what your Target Model like YOLOv8 will predict. A simple Ontology is the CaptionOntology which prompts a Base Model with text captions and maps them to class names. Other Ontologies may, for instance, use a CLIP vector or example images instead of a text caption.

```python

from autodistill.detection import CaptionOntology

# "<description of label>": "<label\_name>" # "bike rider": "Bike\_Rider", --> label 0

# "bike rider and passanger with helmet": "Helmet", --> label 1

# "bike rider and passanger with no helmet": "No\_Helmet" --> label 2

ontology=CaptionOntology({ "bike rider": "Bike\_Rider", "helmet": "Helmet",

"no helmet": "No\_Helmet"

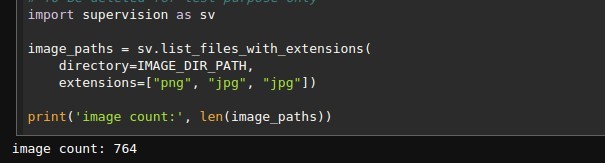
})

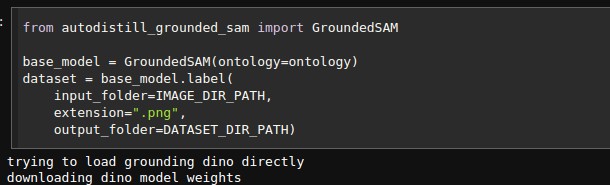
```

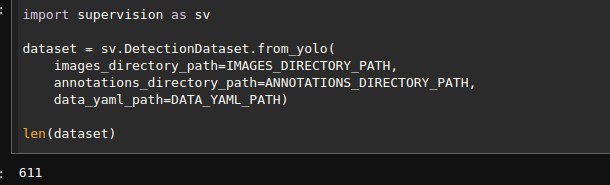
The preprocessed dataset is then divided into training and validation dataset to check the performance of model.

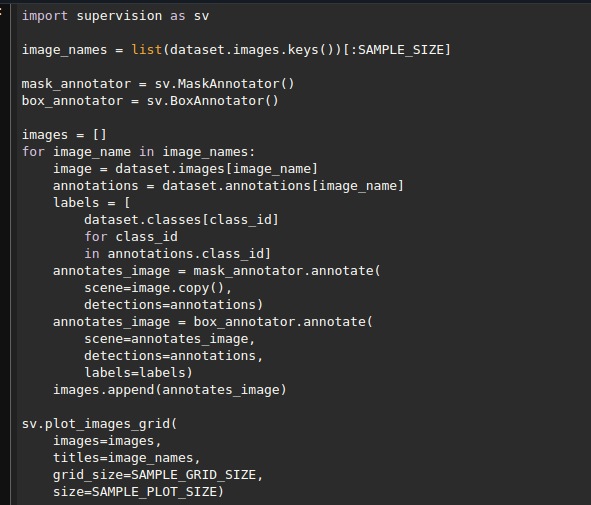
This YOLOv8 model is fine-tuned for custom dataset target values of **bike rider**, **helmet** and **no helme**t. Based on which best.pt and last.pt pytorch training weights are generated.

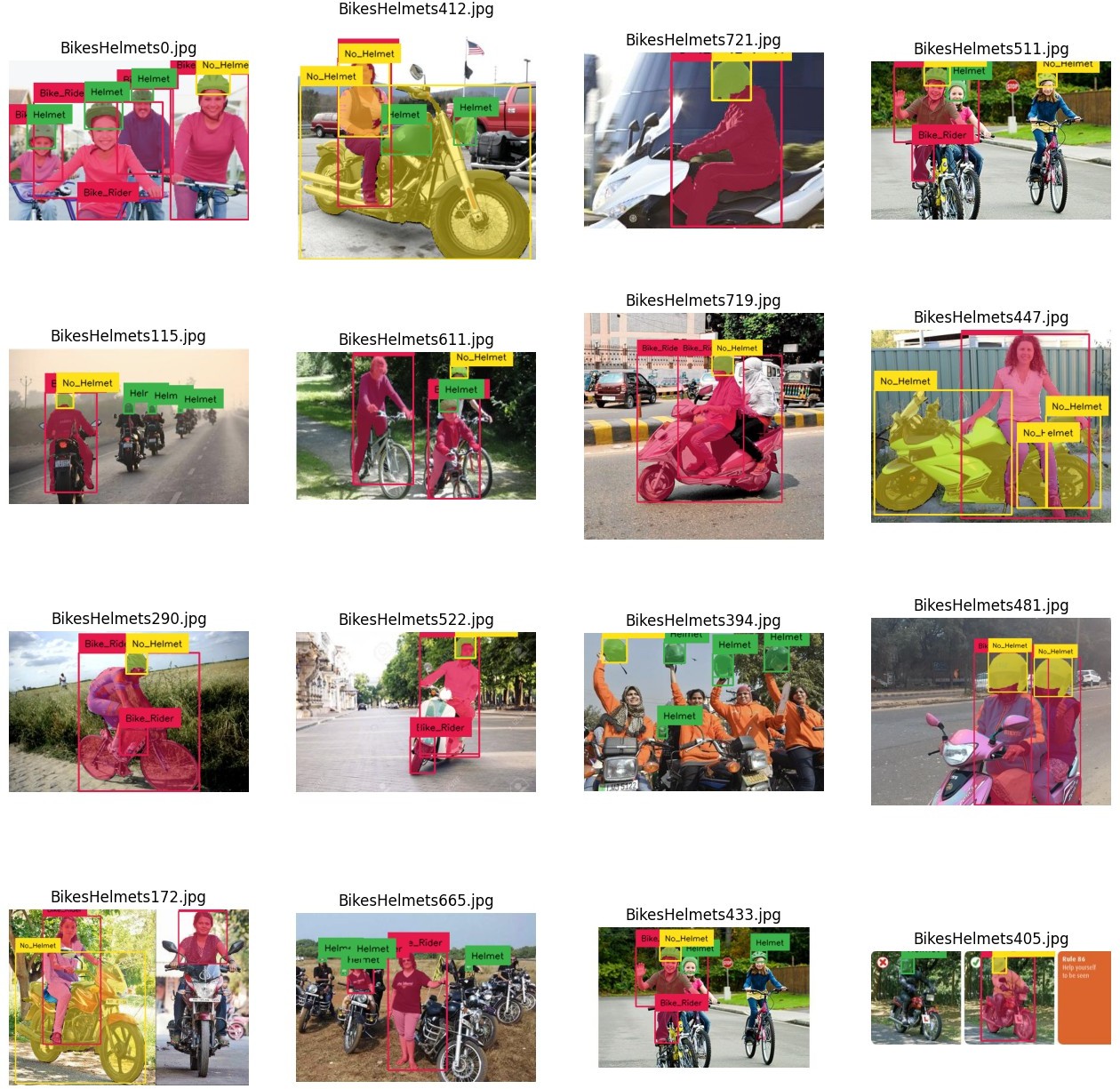
* 1. **Sample Code**

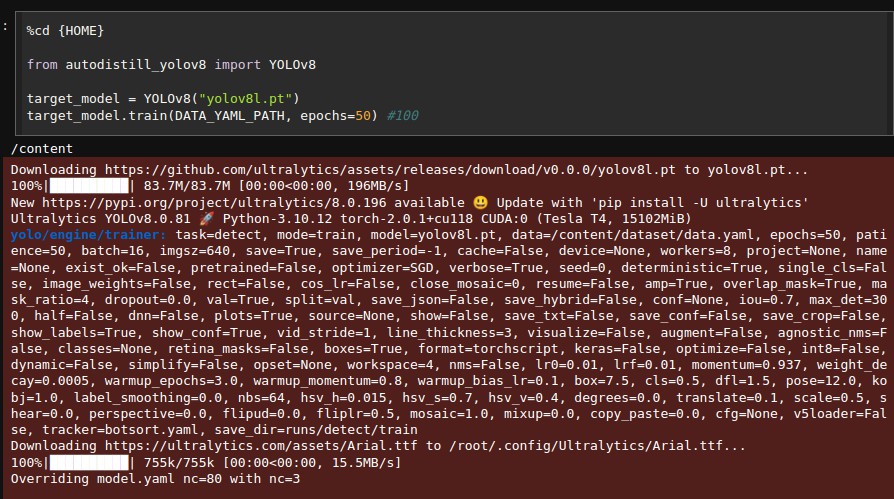
****

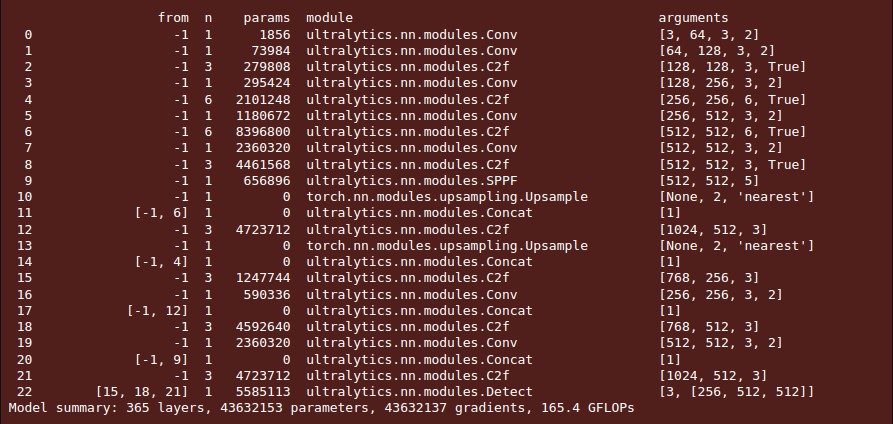
****

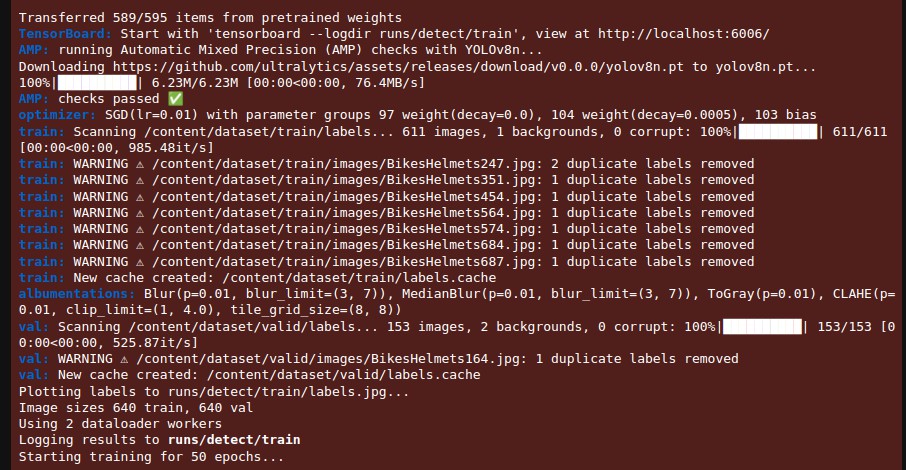
****

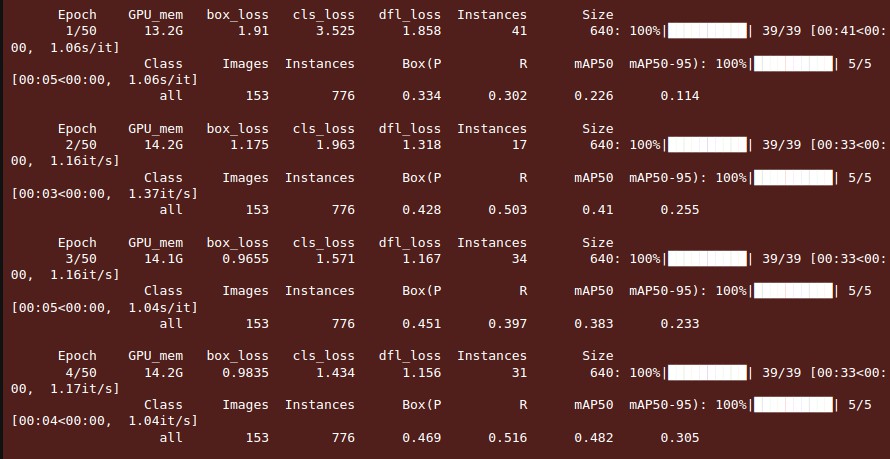






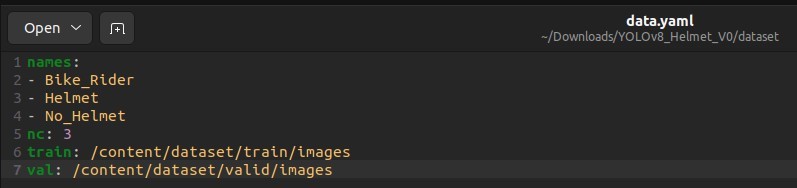
****

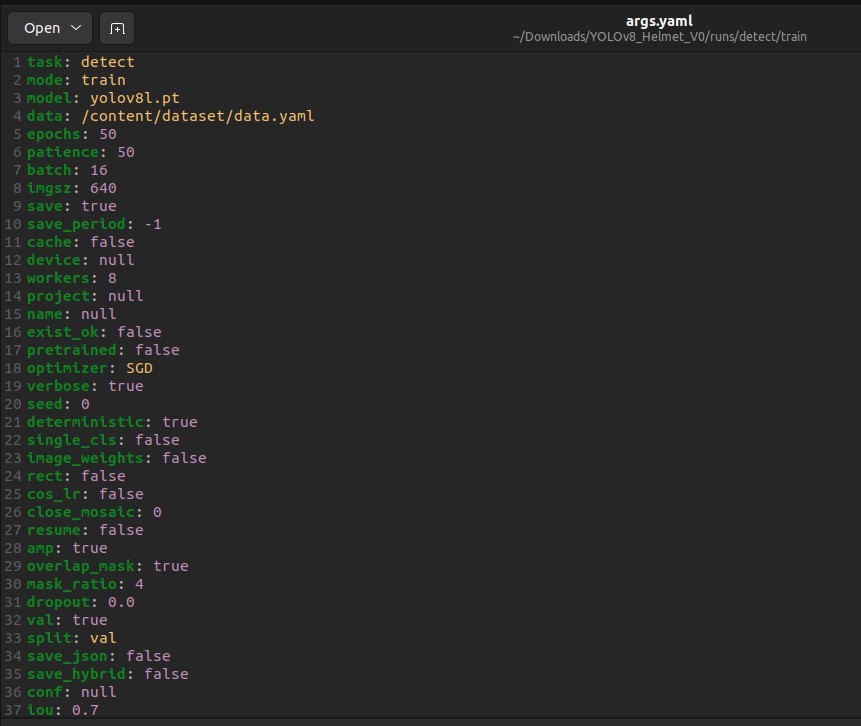


****



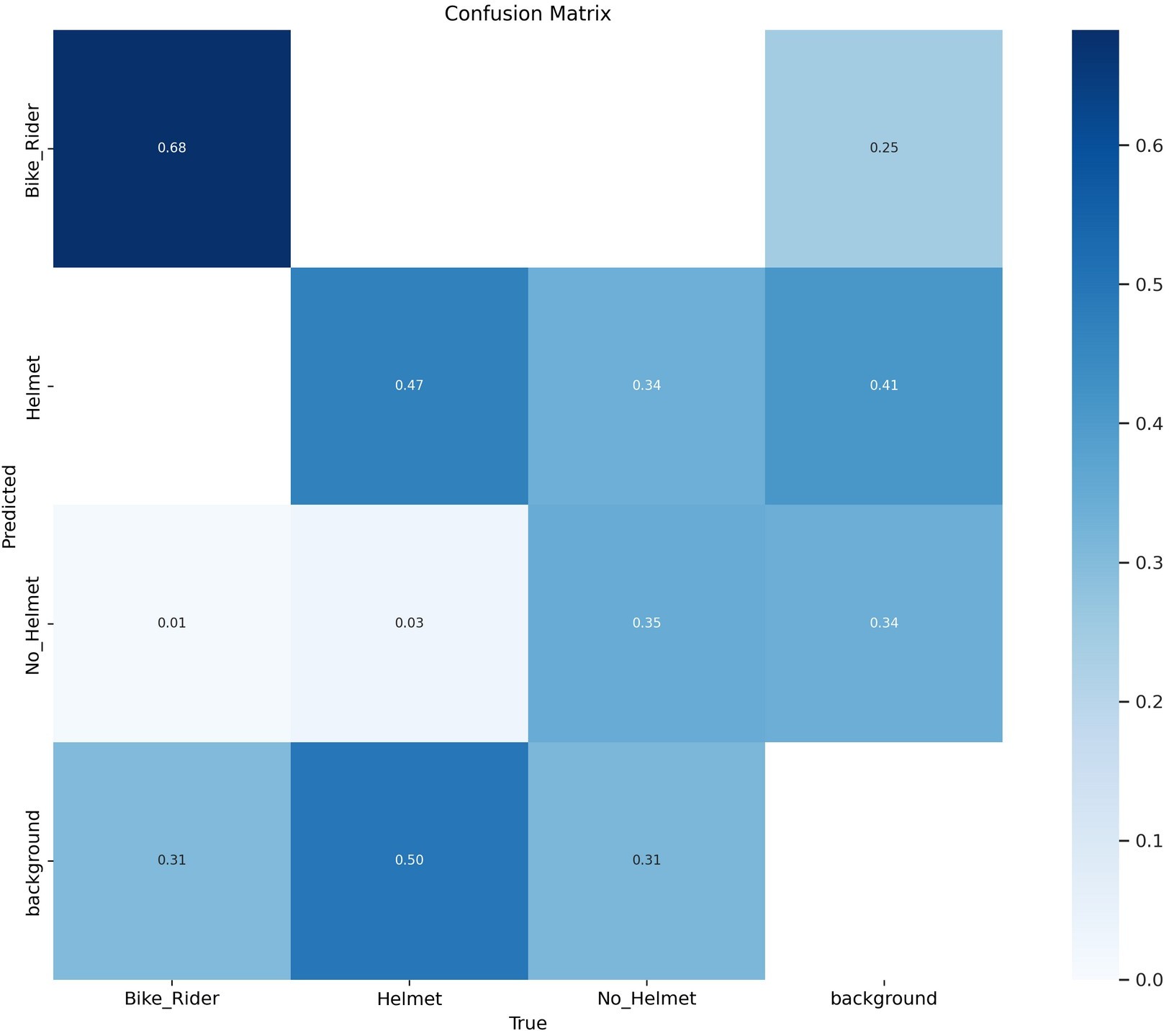
* 1. **Data and YOLOv8 configuration file**

****

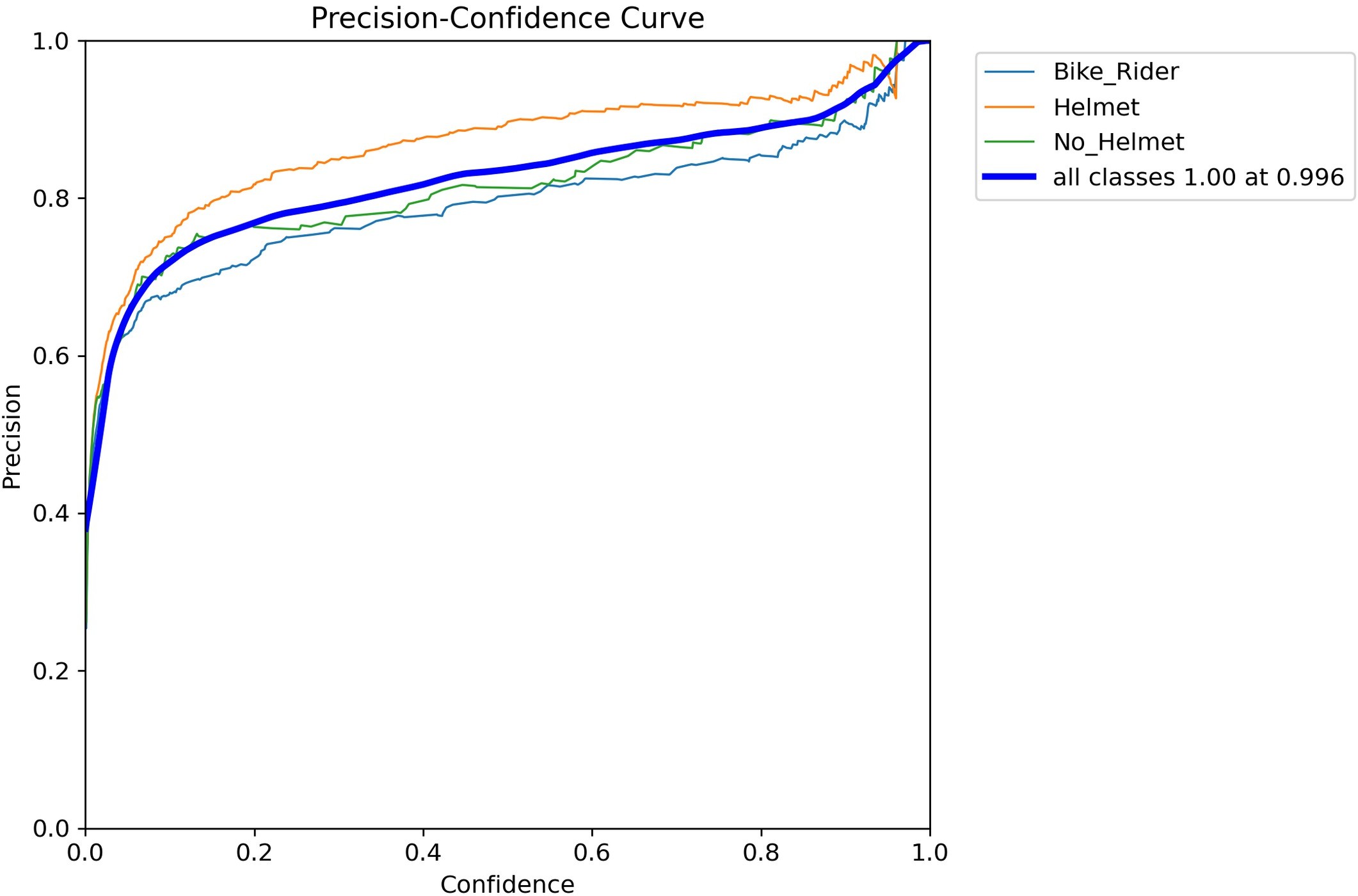
****

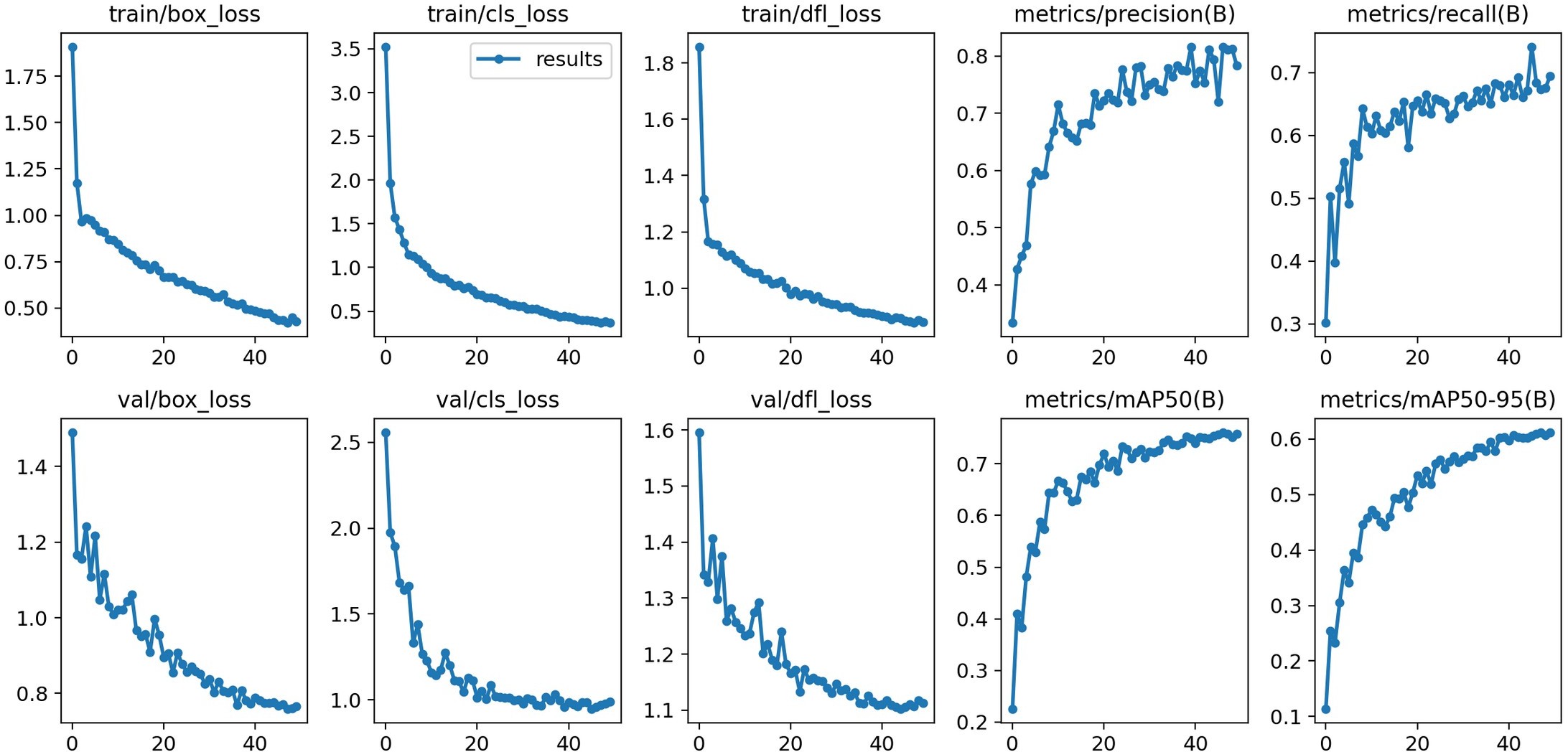
**Chapter 7**

1. **Analysis of result**
   1. **Confusion Matrix**

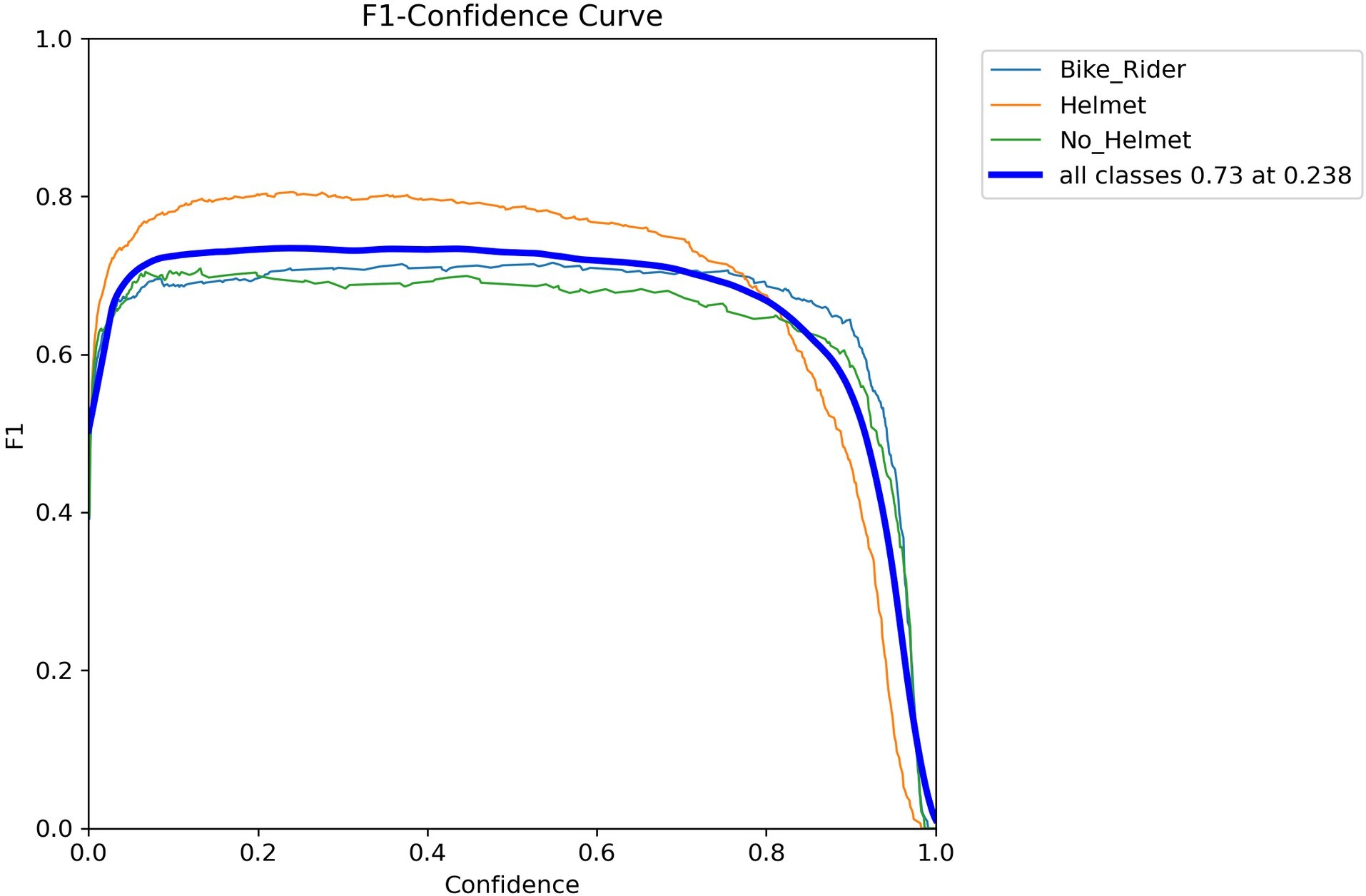
****

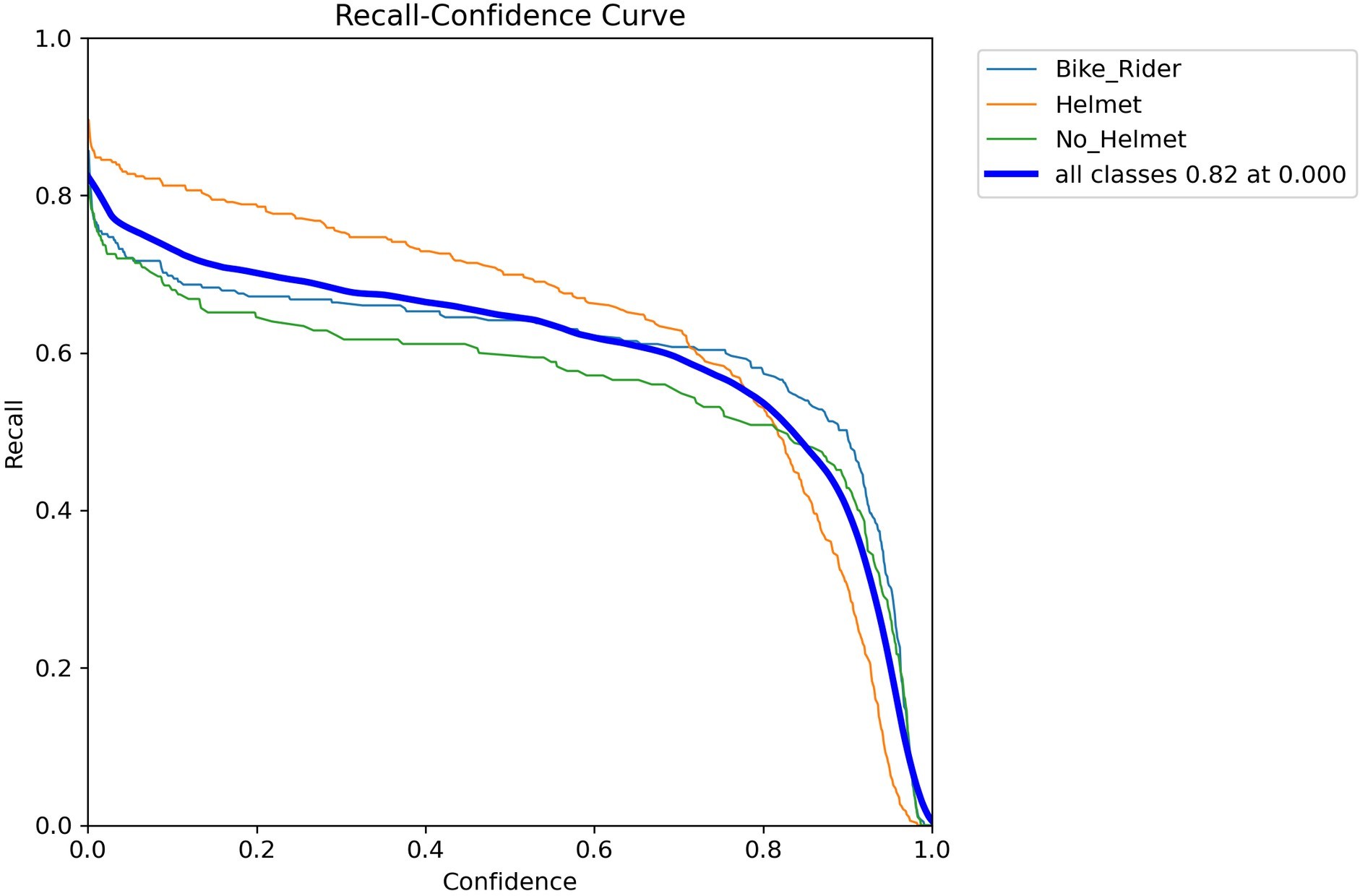
* 1. **Precision-Confidence Curve and Train, Validation and Loss metrics**

****

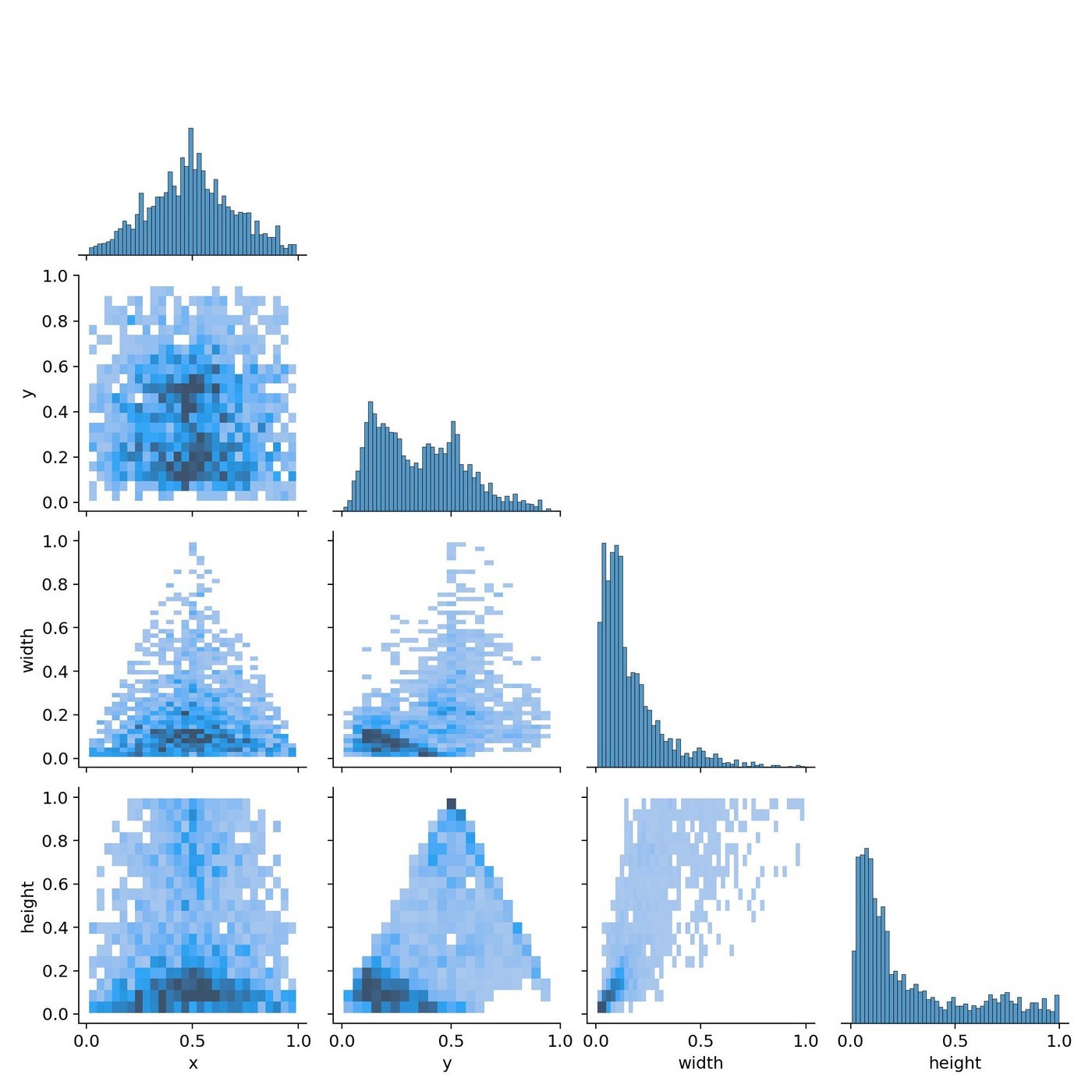
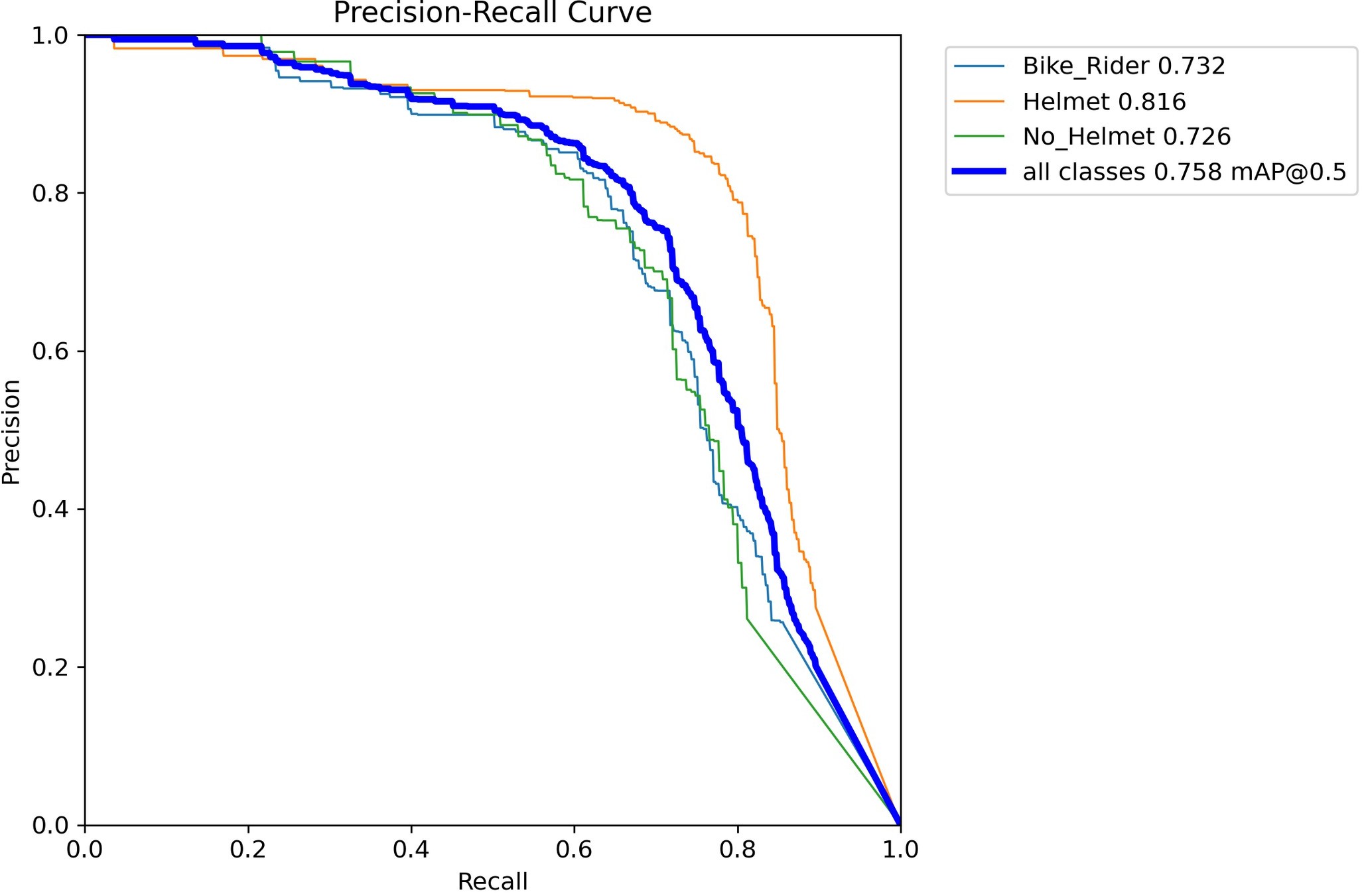
****

* 1. **F1-Confidence and Recall-Confidence Curve**

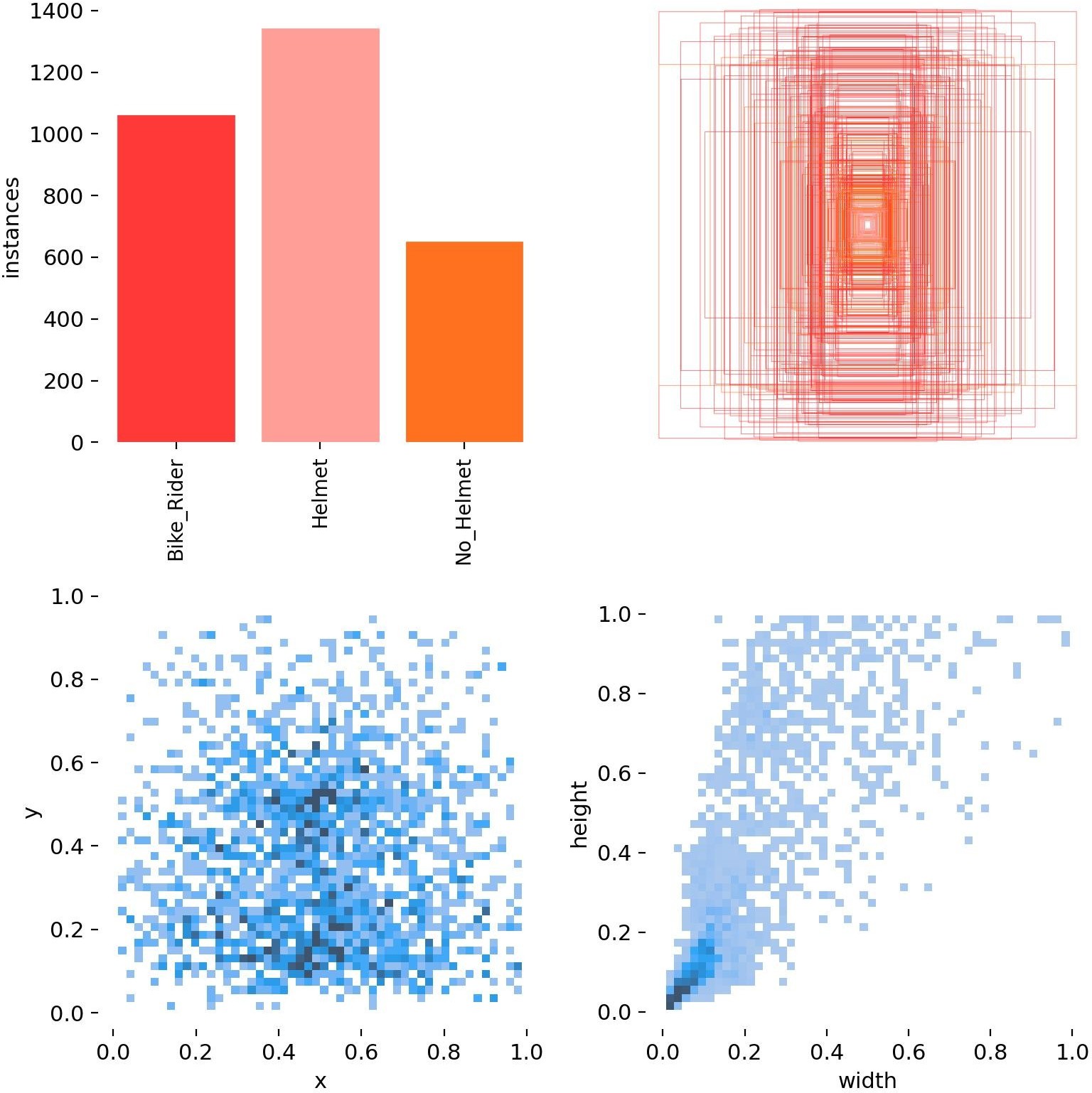
****

****

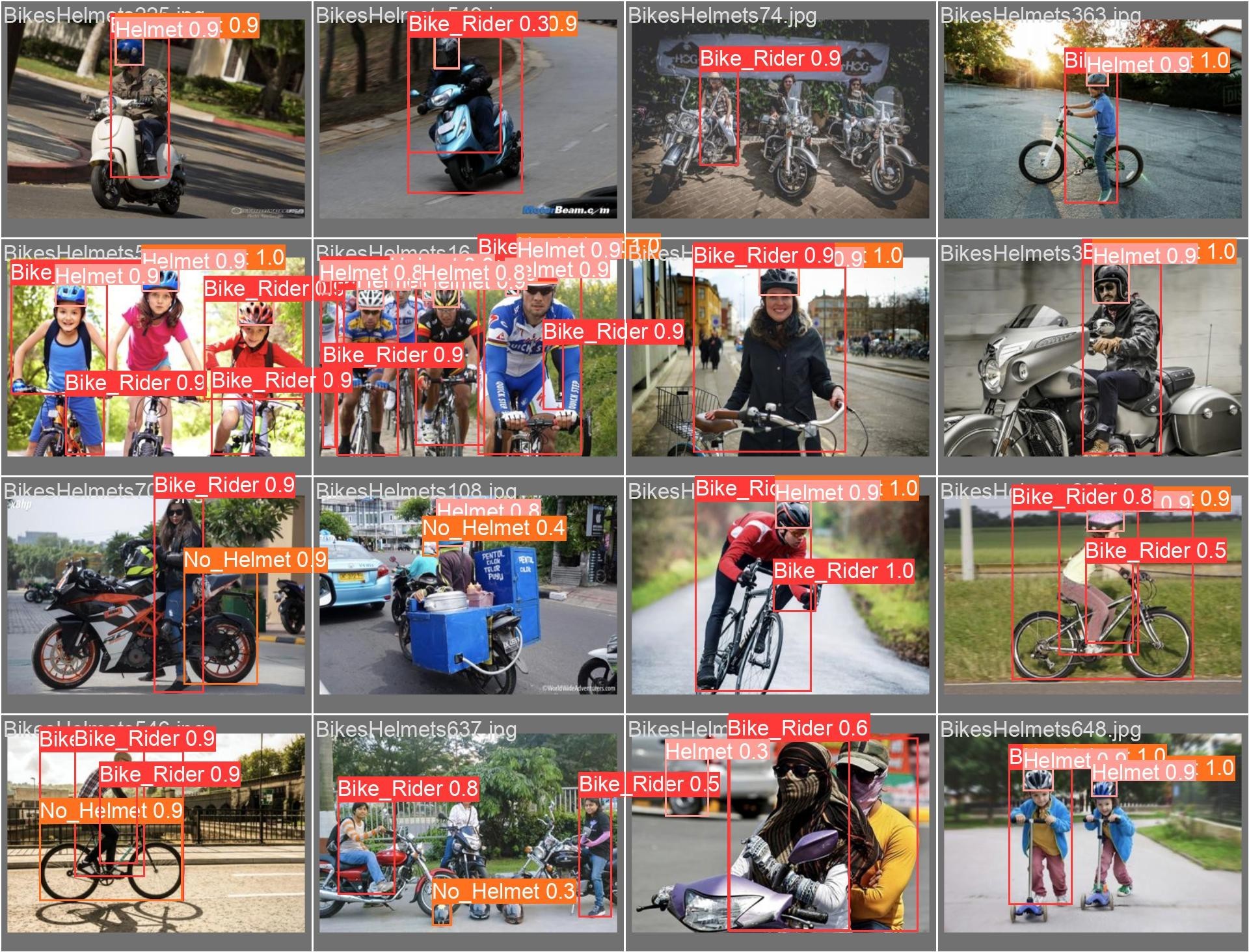
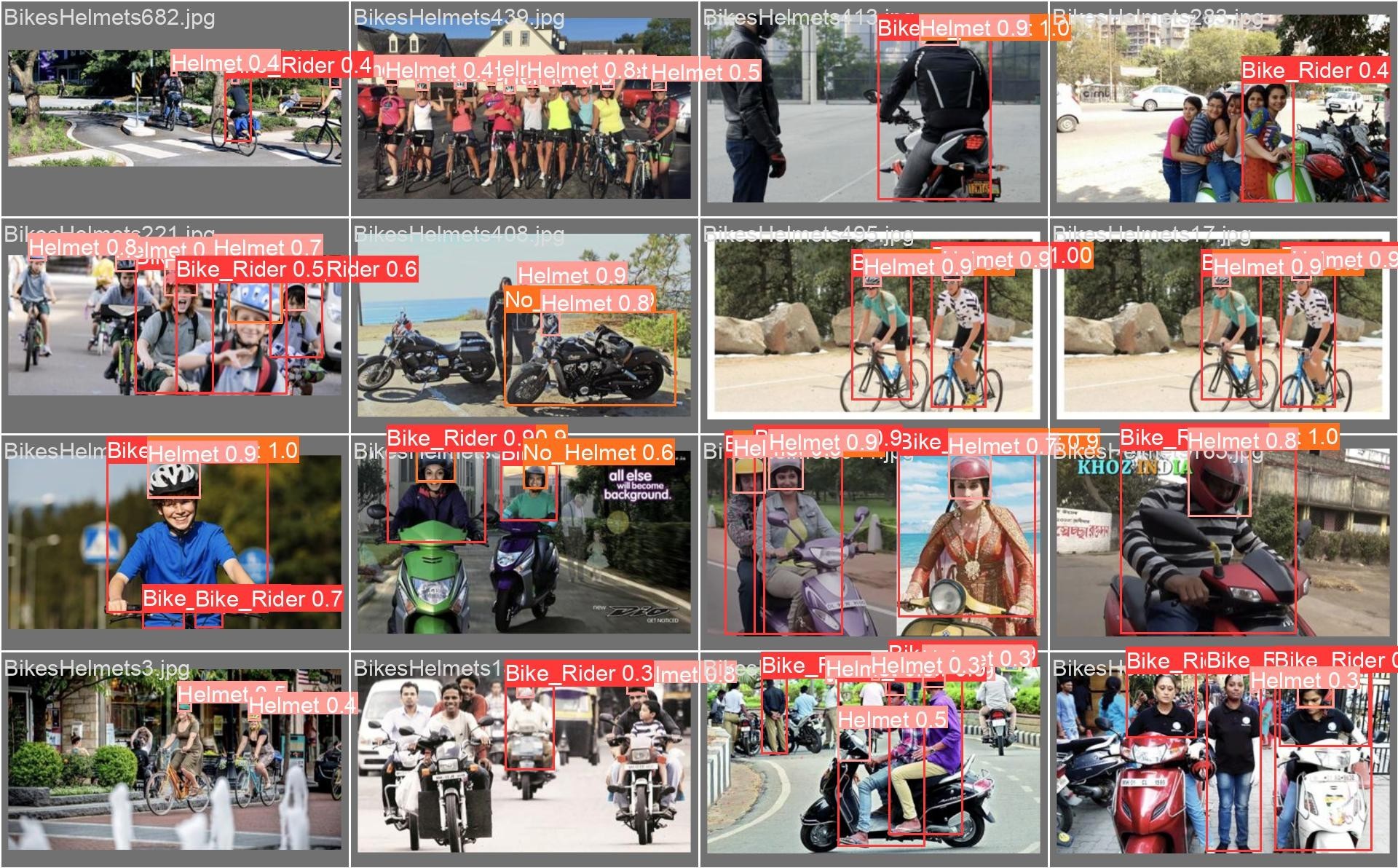
* 1. **Precision-Recall Curve**



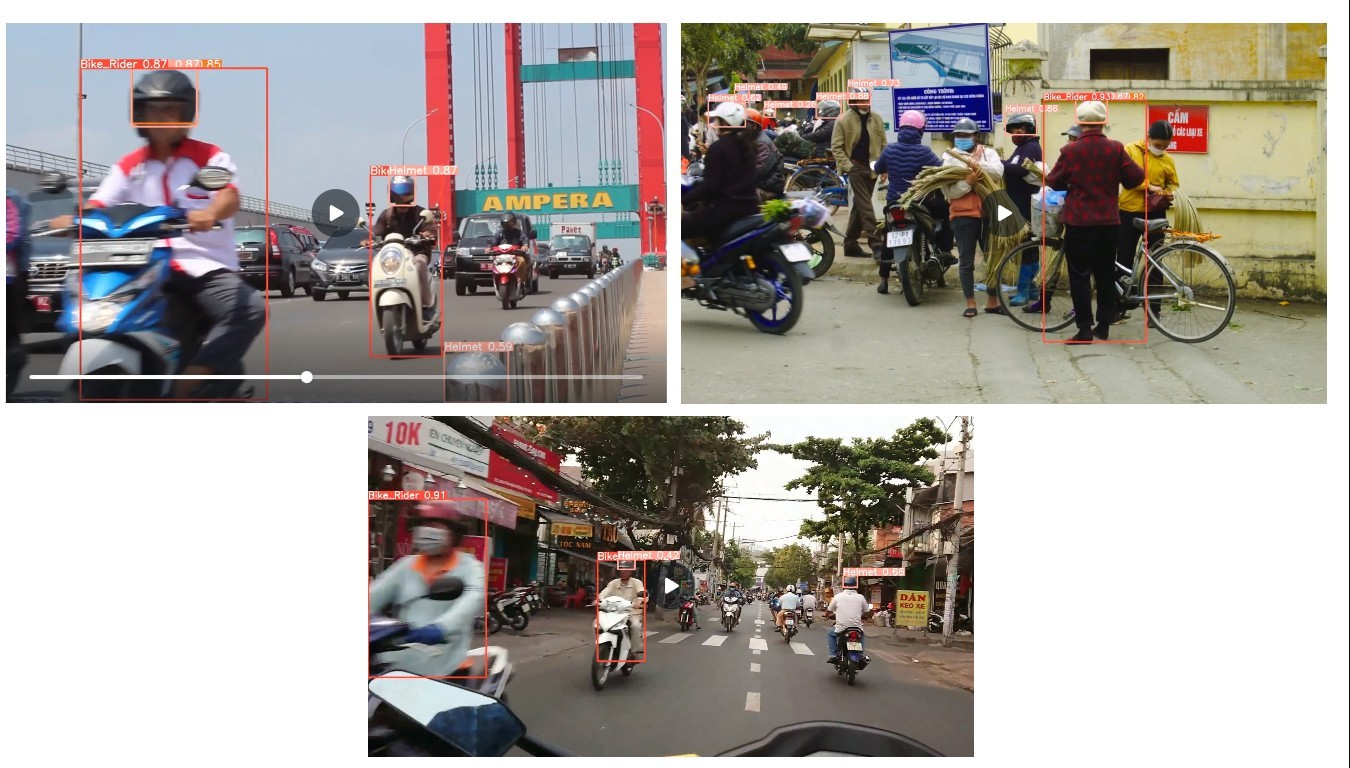
* 1. **Labels-Class Chart**

****

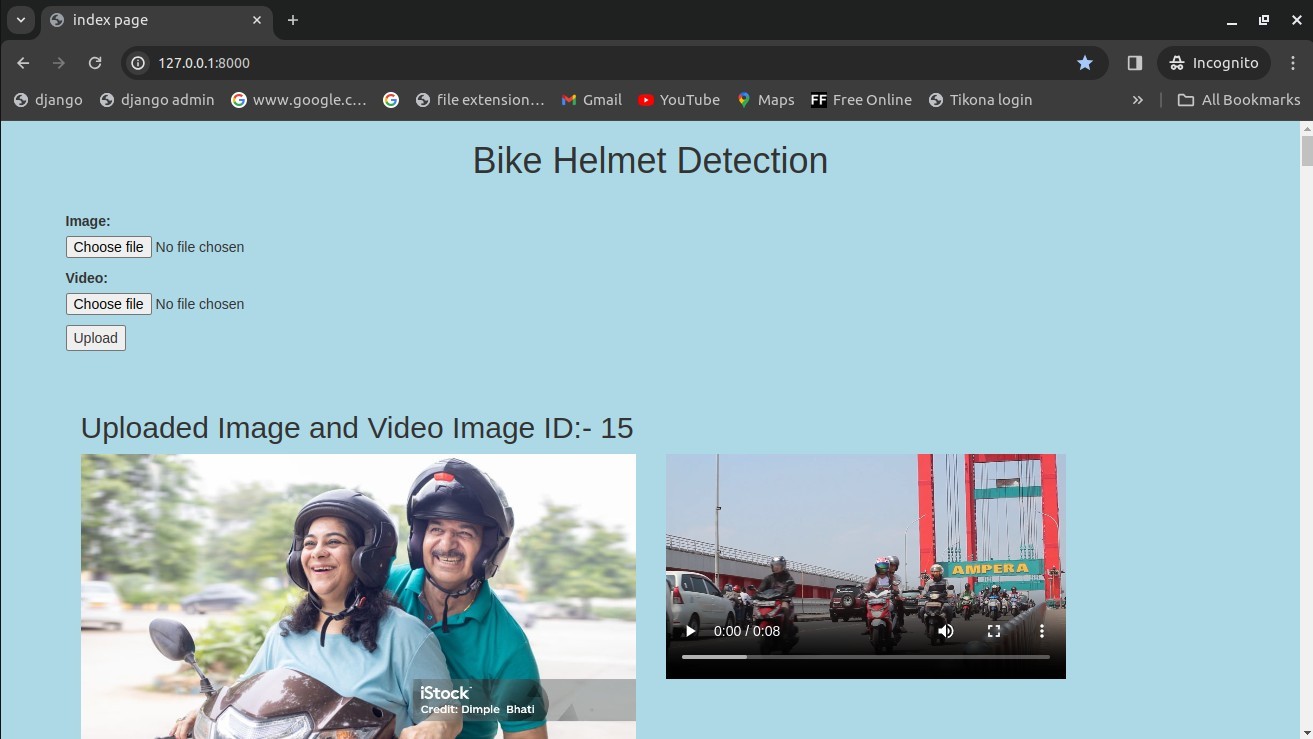
* 1. **Inference on Images**

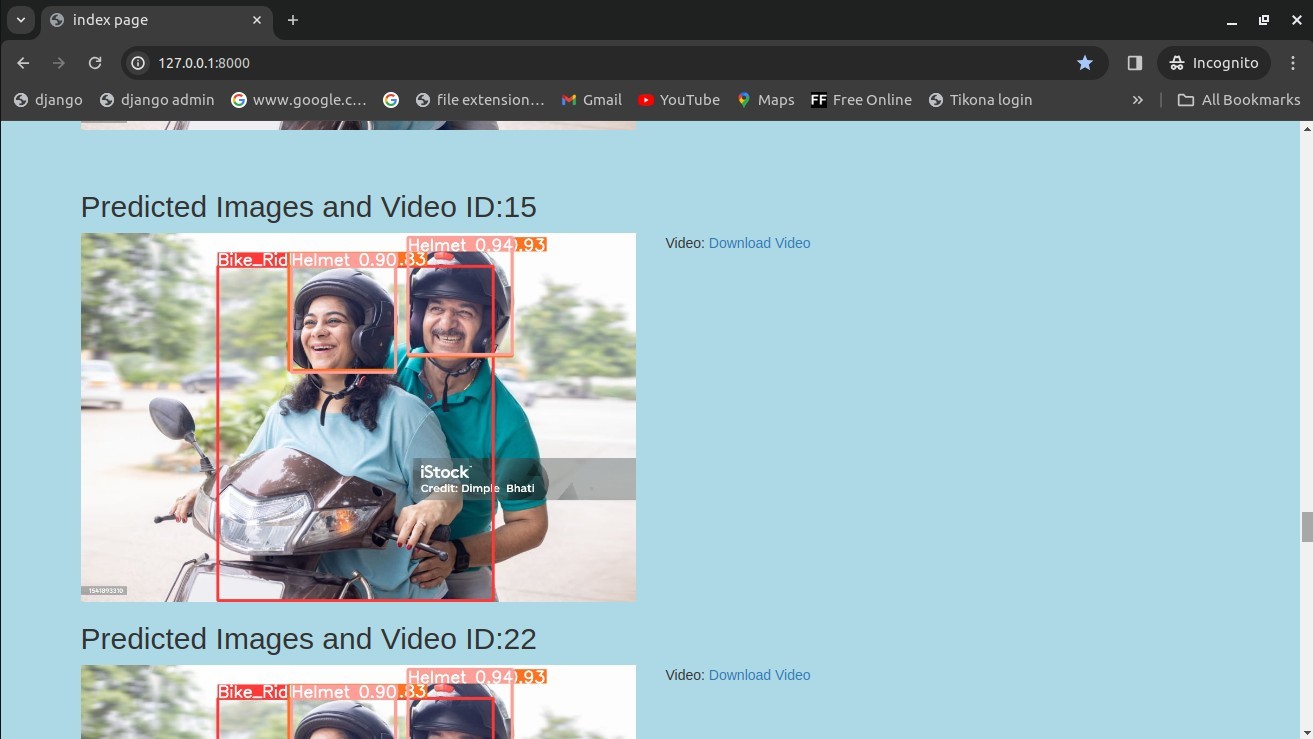


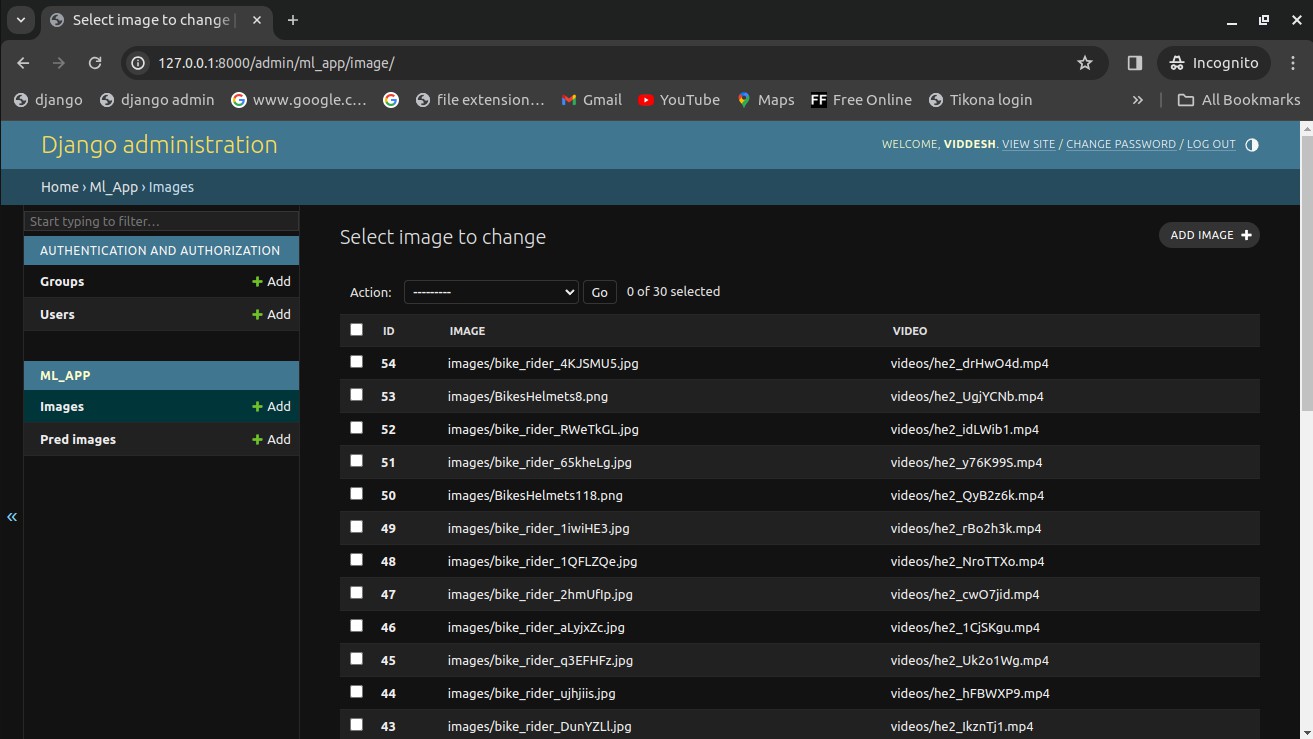
* 1. **Inference on Videos**

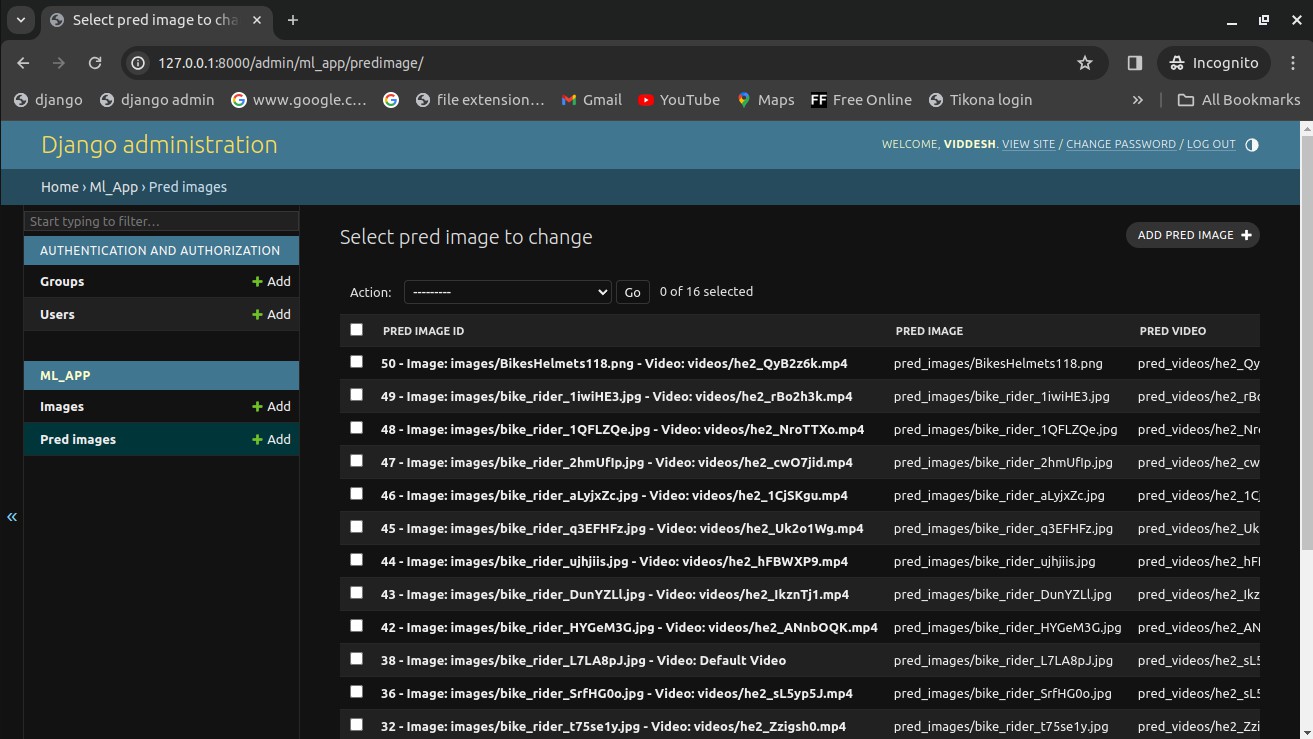
****

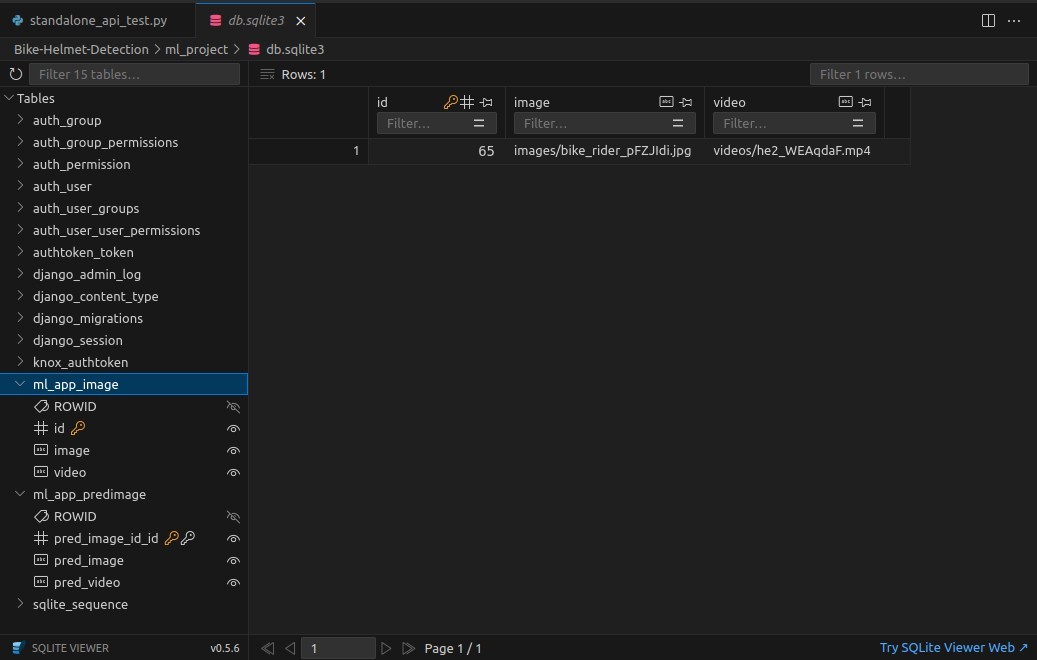
* 1. **Django Web application Demo**

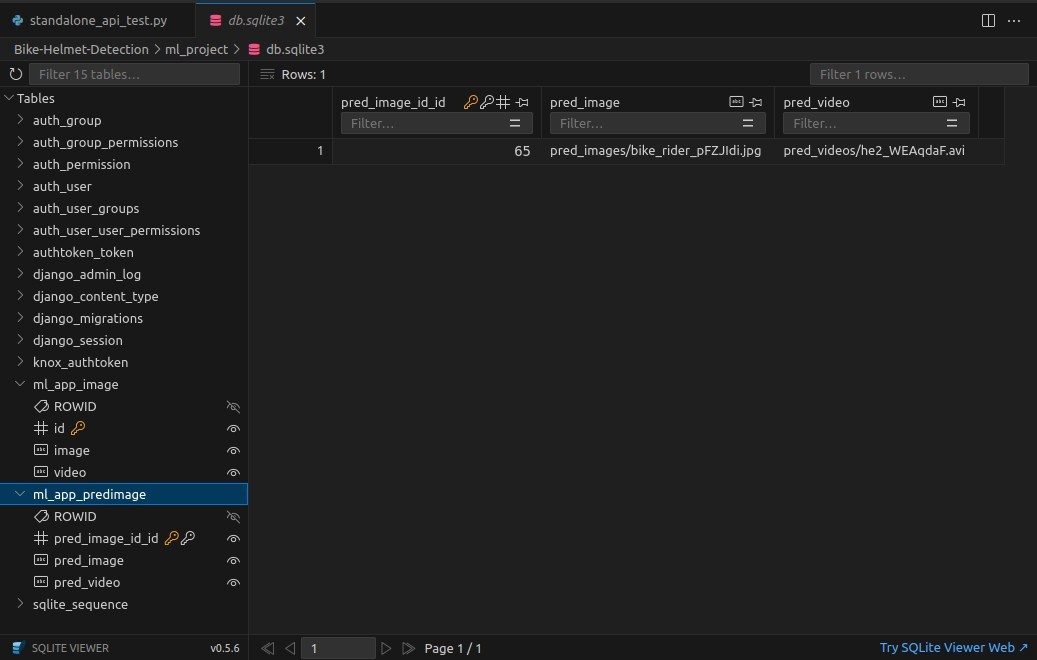
****

****

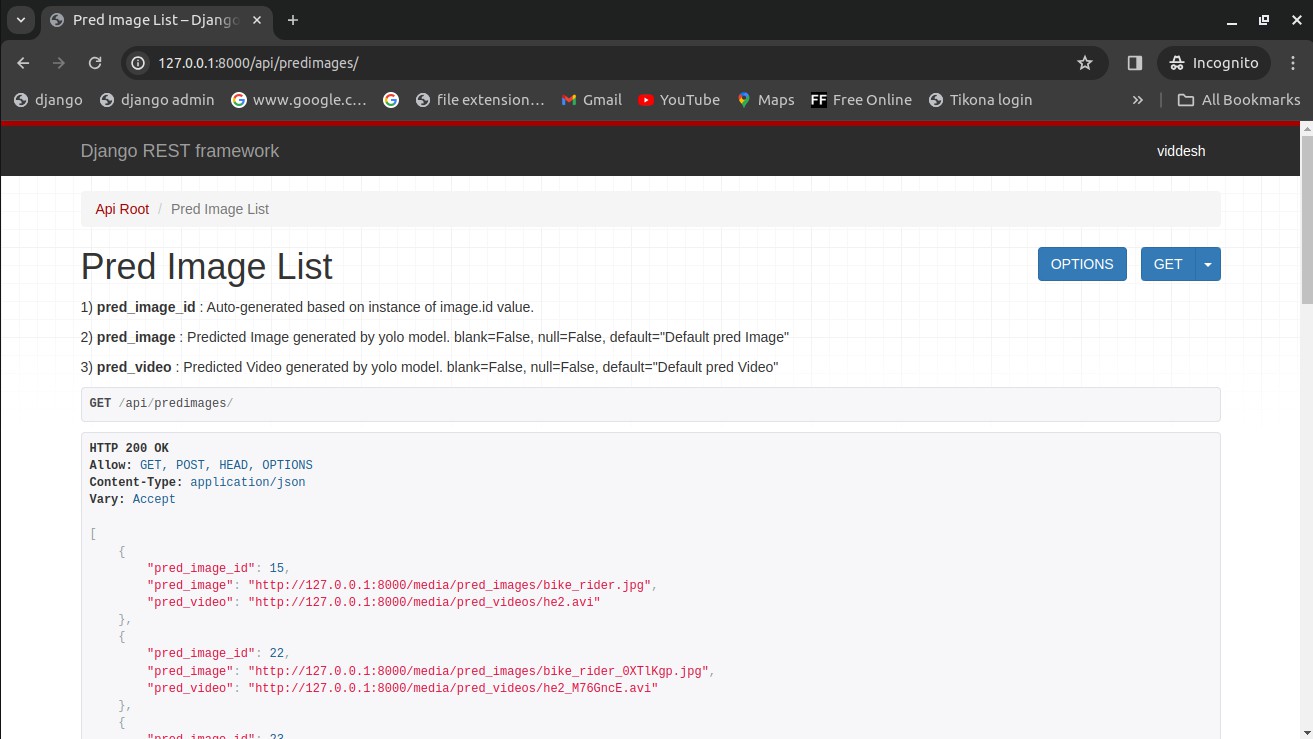


****

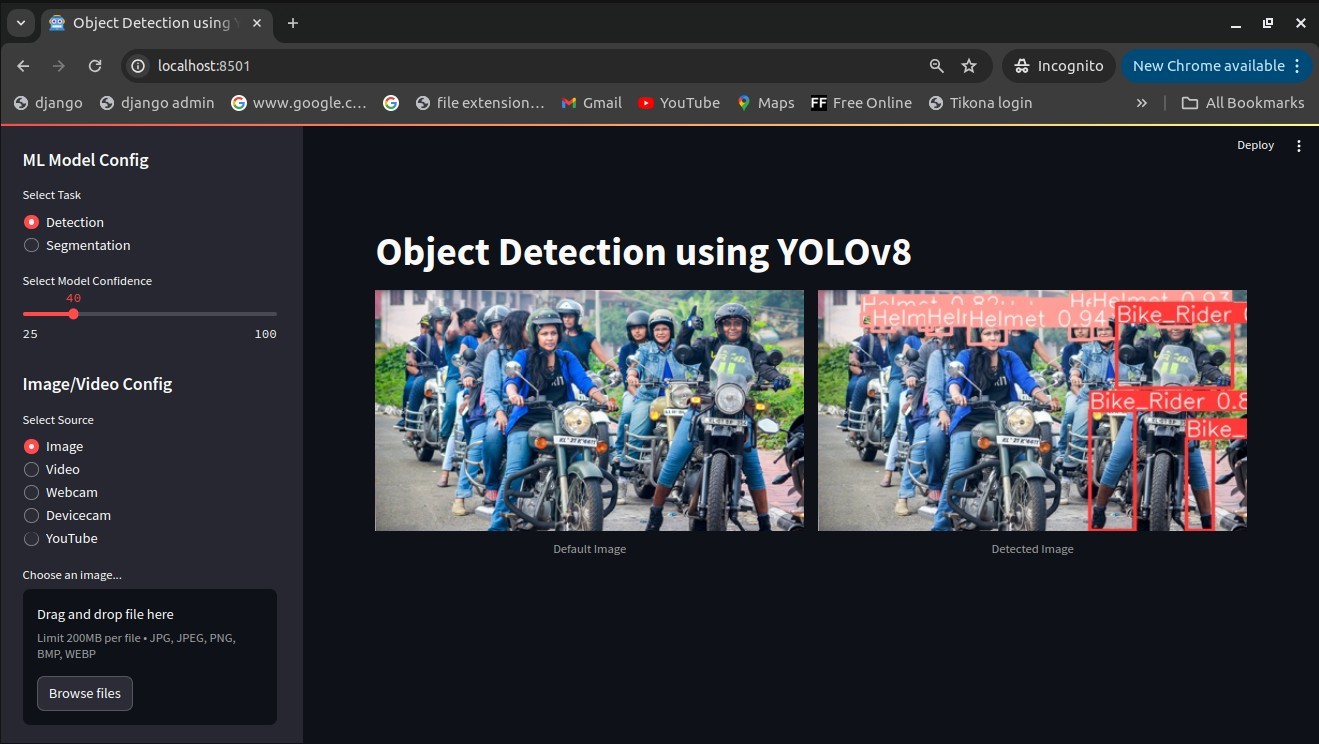


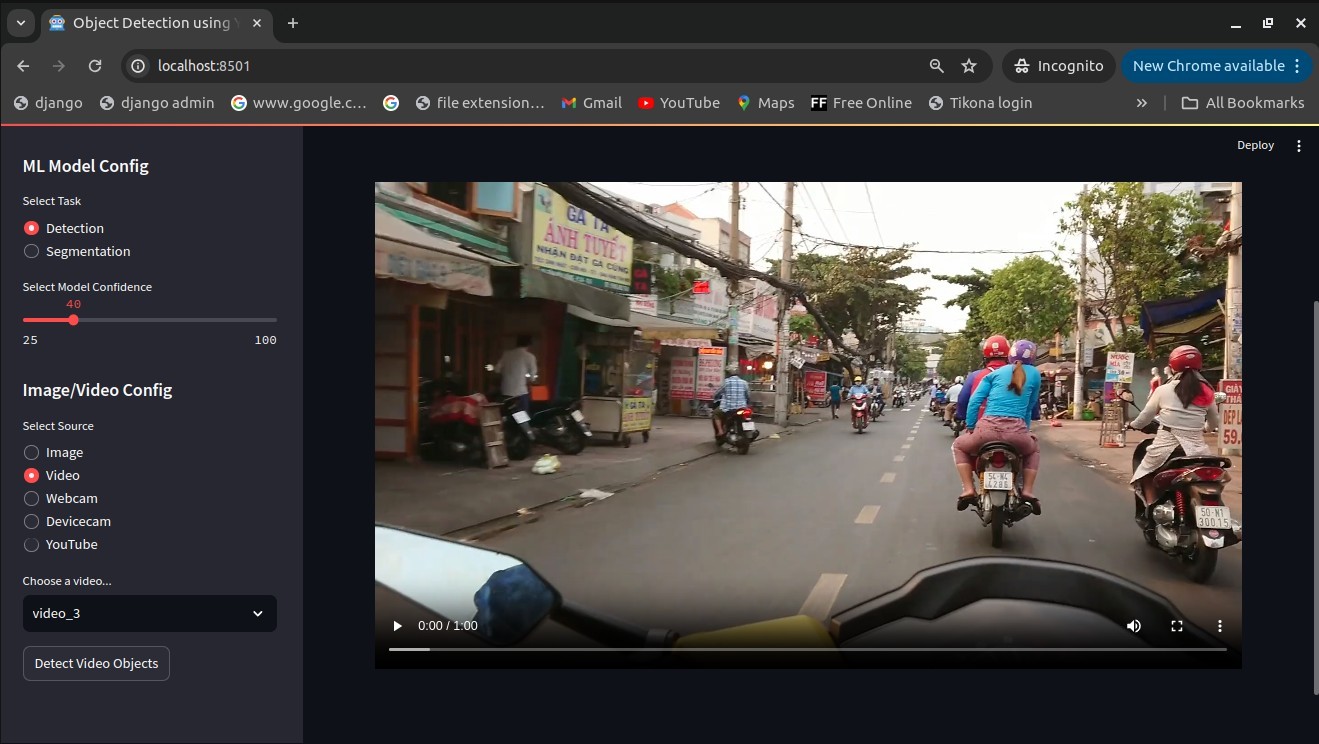
****

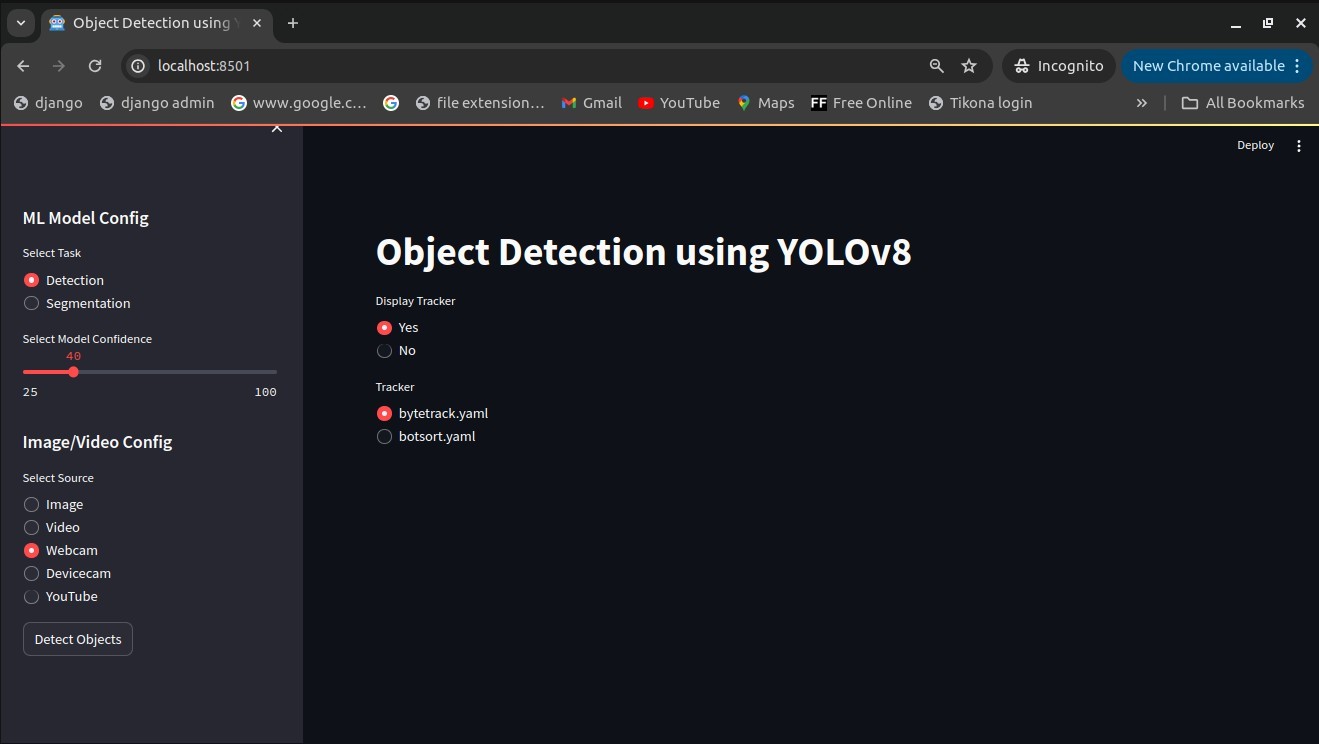


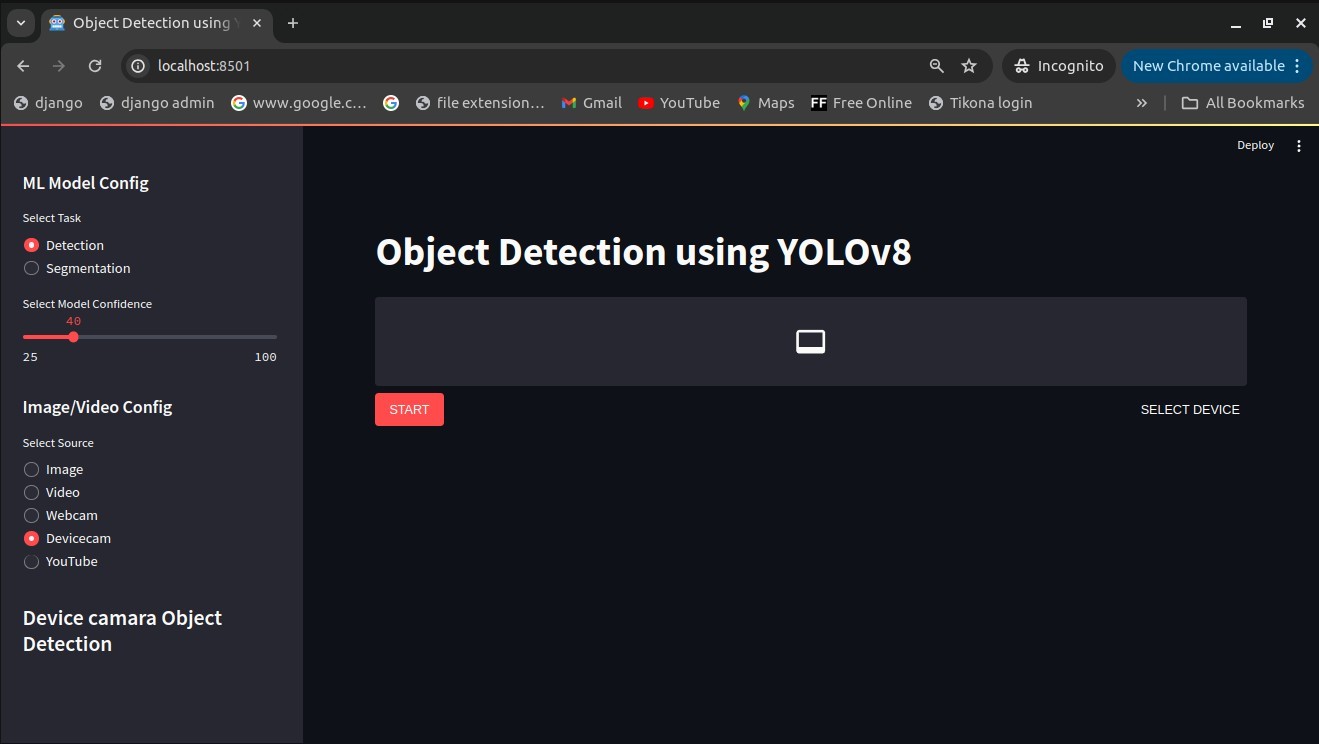
****

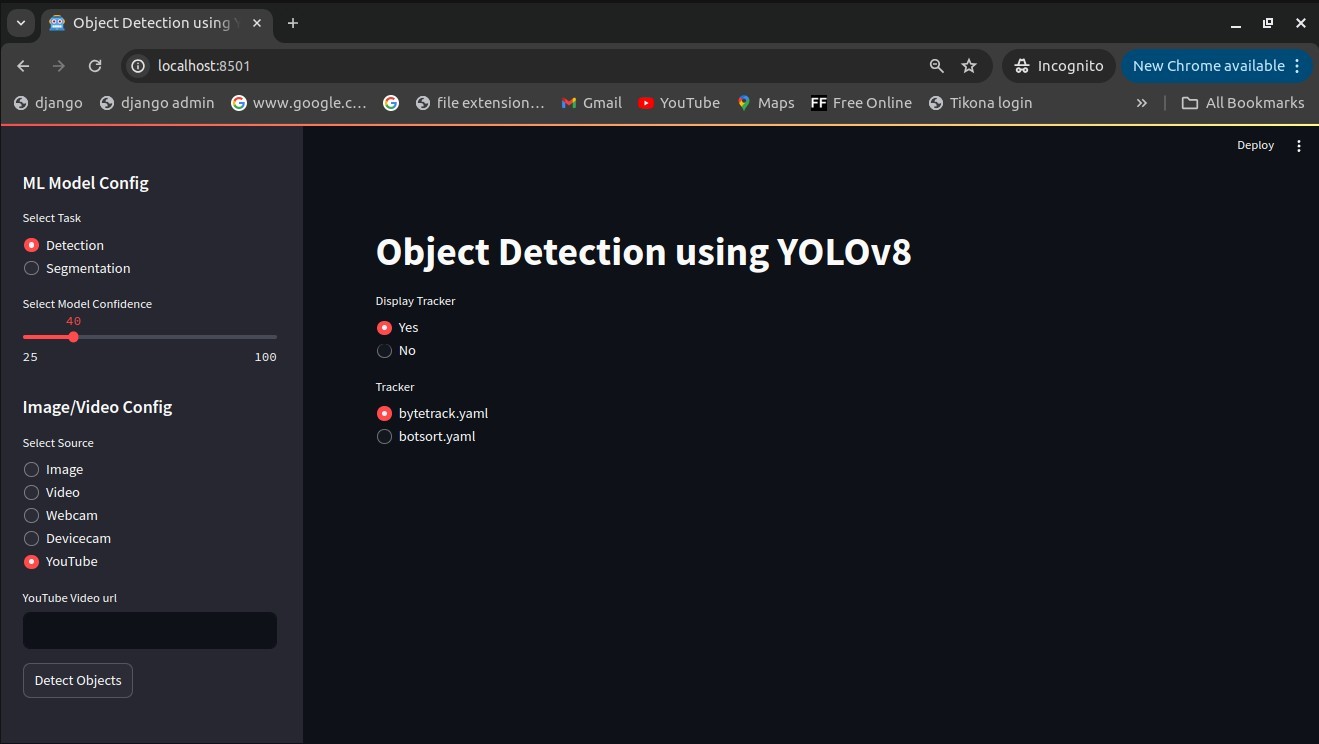
* 1. **Streamlit Web application Demo**

****

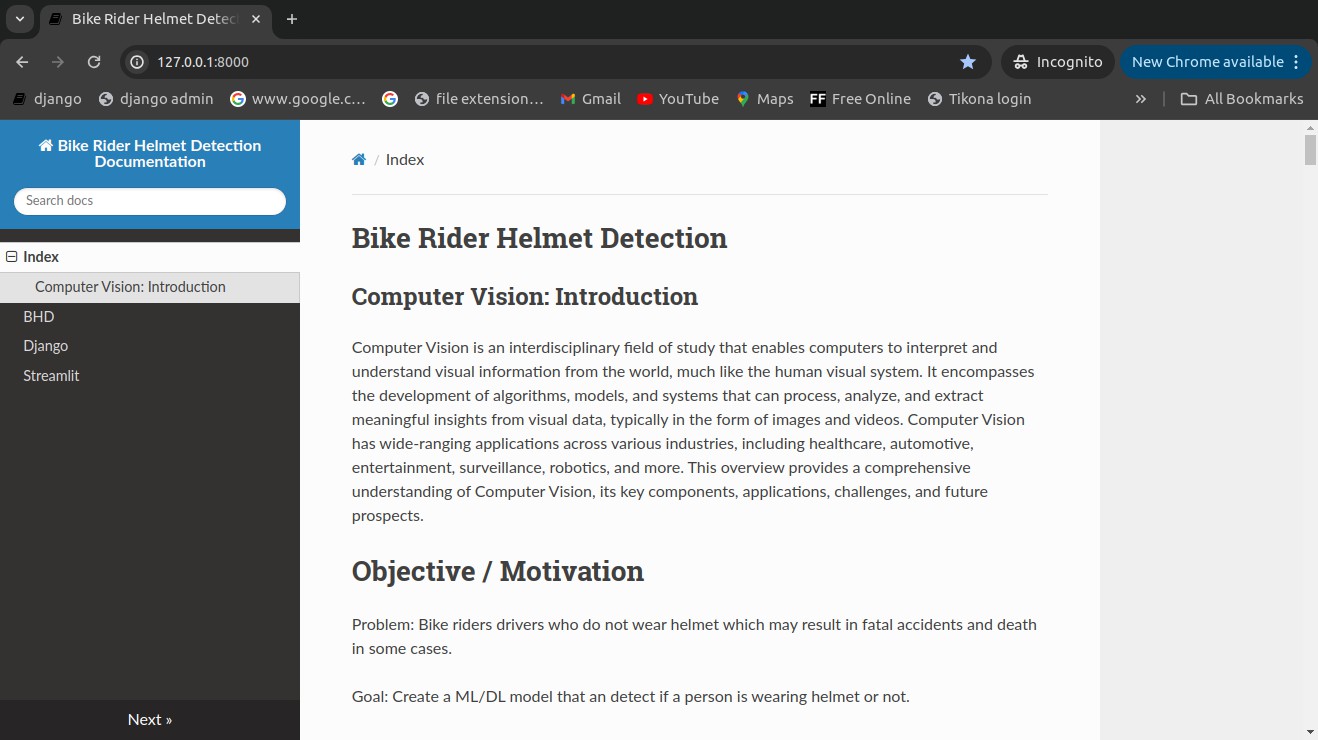
****

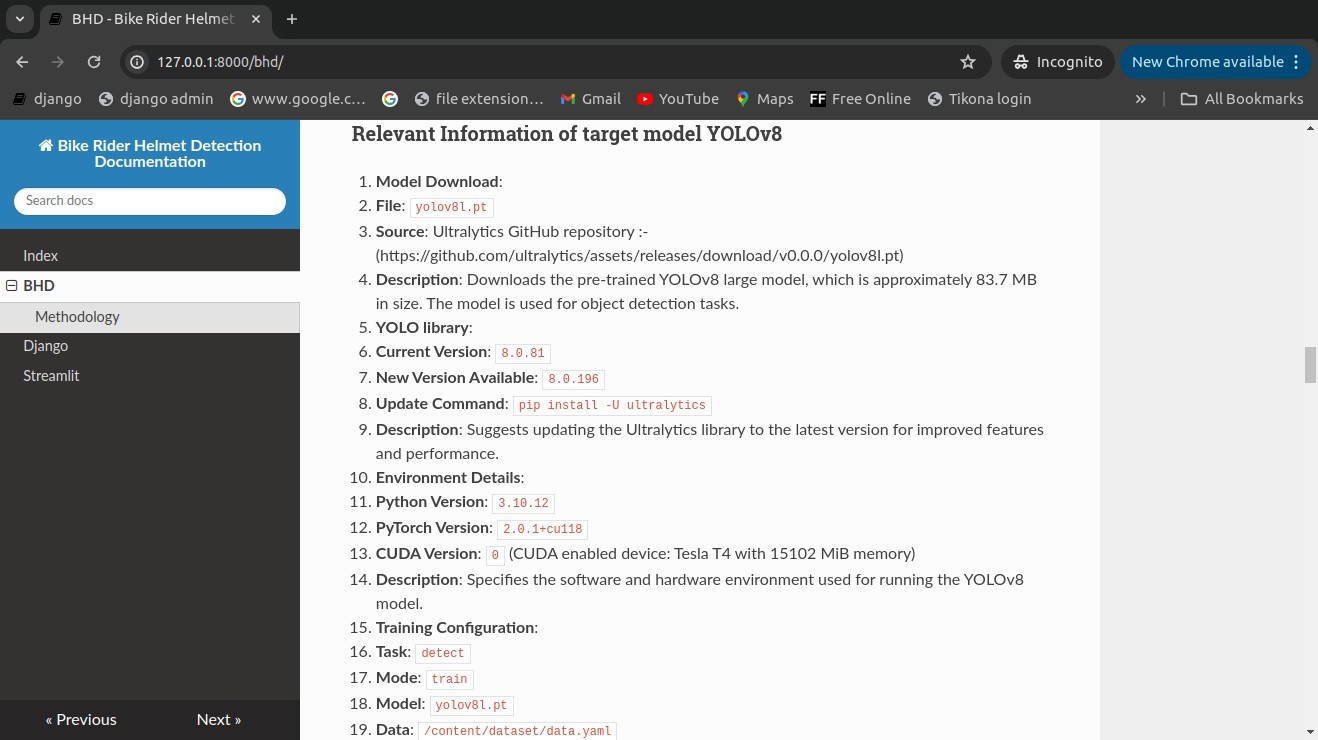


****



* 1. **Mkdocs Documentation Demo**

****

****

**Chapter 8**

1. **Conclusion**

In conclusion, the development of a computer vision model using YOLOv8 for bike rider helmet detection represents a significant advancement in enhancing safety measures for riders along with passengers. The model demonstrates impressive accuracy and efficiency in identifying helmet-wearing individuals, which is crucial for ensuring compliance with safety regulations and reducing the risk of head injuries in bike-related accidents.

This project highlights the potential of computer vision technology to address real- world safety challenges effectively.

**Chapter 9**

1. **Future Scope**

The Future scope of this project should be focused on latest implementation of **YOLOv9** and recent **YOLOv10**. There should be seperate repository for implementation of YOLOv9 and YOLOv10 along with any demo related to it with any technology for creating web application like django, flask, mesop, streamlit, gradio etc. or any desktop application using various techology as reflex, nicegui, tkinter, kivy etc.

**Chapter 10**

1. **Program code**
2. Implementation of Bike Helmet Detection Jupyter Notebook

[**https://github.com/Viddesh1/Helmet\_test\_1**](https://github.com/Viddesh1/Helmet_test_1)

1. Output generated by YOLOv8

[**https://drive.google.com/drive/folders/**](https://drive.google.com/drive/folders/1M4FckJJeyPQTTWqgo6xWhW8L4tf0EJ4l?usp=sharing)[**1M4FckJJeyPQTTWqgo6xWhW8L4tf0EJ4l?usp=sharing**](https://drive.google.com/drive/folders/1M4FckJJeyPQTTWqgo6xWhW8L4tf0EJ4l?usp=sharing)

1. Bike Helmet Detection Django Web application

[**https://github.com/Viddesh1/Bike-Helmet-Detection**](https://github.com/Viddesh1/Bike-Helmet-Detection)

1. Bike Helmet Detection Stremlit Web application

[**https://github.com/Viddesh1/Bike-Helmet-Detectionv2**](https://github.com/Viddesh1/Bike-Helmet-Detectionv2)

1. Hosted on Streamlit

[**https://bike-helmet-detectionv2-dmehozp3lkef4wnssaepjf.streamlit.app/**](https://bike-helmet-detectionv2-dmehozp3lkef4wnssaepjf.streamlit.app/)

1. Overall Documentation

[**https://github.com/Viddesh1/Bike-Helmet-Detection-Docs**](https://github.com/Viddesh1/Bike-Helmet-Detection-Docs)

**Chapter 11**

1. **References**
2. <https://github.com/ultralytics/ultralytics>
3. [https://towardsdatascience.com/dino-emerging-properties-in-self-supervised-vision-](https://towardsdatascience.com/dino-emerging-properties-in-self-supervised-vision-transformers-summary-ab91df82cc3c) [transformers-summary-ab91df82cc3c](https://towardsdatascience.com/dino-emerging-properties-in-self-supervised-vision-transformers-summary-ab91df82cc3c)
4. <https://github.com/facebookresearch/dino/>
5. Emerging Properties in Self-Supervised Vision Transformers : <https://arxiv.org/abs/2104.14294/>
6. <https://dinov2.metademolab.com/>
7. <https://segment-anything.com/>
8. <https://blog.roboflow.com/whats-new-in-yolov8/>