# STUDY ON COMPUTER VISION

### AN INTERNSHIP REPORT

*Submitted by*

**Nitin Udmale**

**[EA2232251010390]**

**Under the Guidance of**

## Dr.G.Babu

(Assistant Professor, Directorate of Online Education)

*in partial fulfillment for the award of the degree of*

**MASTER OF COMPUTER APPLICATIONS**



DIRECTORATE OF ONLINE EDUCATION

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR- 603 203

DEC 2023

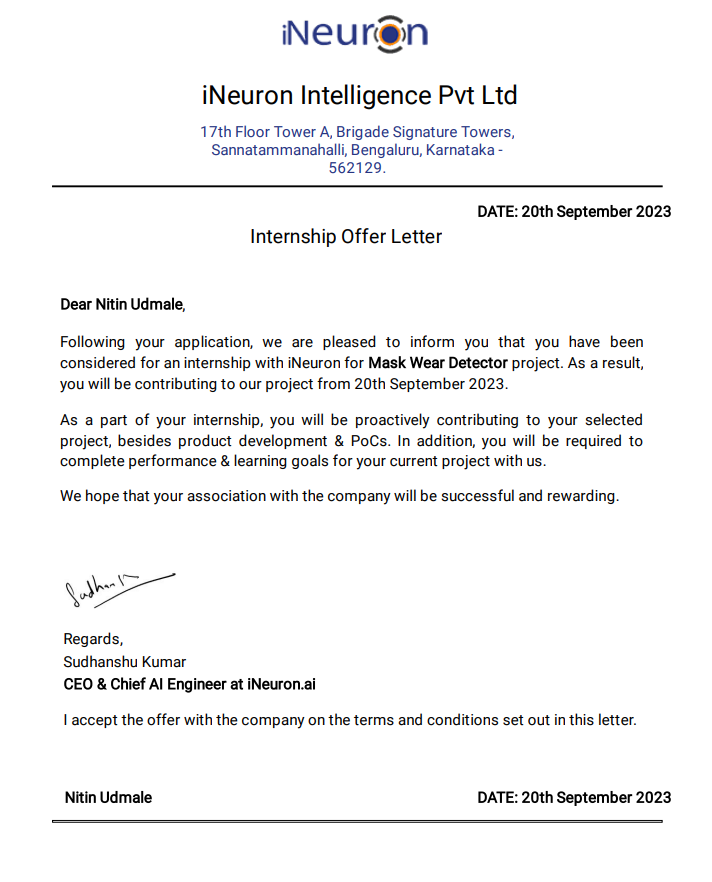
DIRECTORATE OF ONLINE EDUCATION

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

This internship report titled **“Study on Computer Vision”** is the Bonafede work of **“Nitin Udmale [EA2232251010390]”**, who carried out the internship work under my supervision along with the company mentor. Certified further, that to the best of my knowledge the work reported herein does not form any other internship report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate

# INTERNSHIP OFFER LETTER



# ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Director DOE, SRM Institute of Science and Technology, Prof. **Dr Manoranjan Pon Ram,** for his invaluable support. SRM Institute of Science and Technology, for her support throughout the project work. We want to convey our thanks to Programme Coordinator **Dr. G. Babu**, Directorate of online Education, SRM Institute of Science and Technology, for their inputs during the project reviews and support. Our inexpressible respect and thanks to my guide, **Dr G.Babu.**, Assistant Professor & Programme Coordinator Directorate of online Education,, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under her mentorship. She provided me with the freedom and support to explore the research topics of my interest. Her passion for solving problems and making a difference in the world has always been inspiring. We sincerely thank the Directorate of online Education, staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

**Nitin Udmale**

# TABLE OF CONTENTS

**CHAPTER NO. TITLE PAGE NO.**

* 1. **Abstract 5**
  2. **System Analysis 6**
  3. [**System Requirements**](#_TOC_250006) **Specifications 9**
  4. [**Technology 10**](#_TOC_250005)

**4.1** [**Python**](#_TOC_250004) **10**

**4.2 Streamlit 13**

**4.3** [**Yolo**](#_TOC_250004) **15**

**4.4 Overview of web application using Python, Streamlit, Yolo 18**

**5. Coding 20**

**6. Result 30**

**7. Conclusion 32**

**8. References 33**

# ABSTRACT

The objective of this Mini Project is to develop a Python-based web application for real-time face mask detection using YOLOv8, a state-of-the-art object detection model, integrated with Streamlit, a user-friendly web app framework. The application is designed to efficiently manage and display data related to face mask compliance. The development incorporates the following key technologies:

I. **YOLOv8**: A cutting-edge object detection model that excels in accurately identifying objects in images and videos.

II. **Streamlit**: A powerful framework for building web applications with a straightforward and intuitive user interface.

III. **Python**: A versatile and widely-used programming language that facilitates seamless integration of various components.

The project's primary objective is to create a progressive web application (PWA) dedicated to monitoring face mask adherence. Users will be able to view a list of individuals along with their corresponding mask compliance status. The application offers hover functionality for each person, providing additional information about their mask status.

Additionally, the application enables users to search for specific individuals, facilitating easy access to relevant data. The search functionality supports partial matching, ensuring efficient retrieval of information.

The Mask Wear Detector project not only serves as a technological milestone in computer vision applications but also stands as a practical contribution towards public health initiatives. By providing a reliable tool for monitoring mask compliance, it offers a tangible solution for mitigating the spread of contagious diseases within public spaces. This project underscores the potential of computer vision to directly impact and safeguard the well-being of communities, marking a significant stride towards a safer and healthier society.

# 2. SYSTEM ANALYSIS

## Requirement Analysis

## Existing System:

In the existing system, we address the challenges associated with real-time face mask compliance monitoring. We take into consideration the complexities of efficiently detecting and managing face mask adherence in various settings. Our approach involves iterative processes to incrementally consolidate requirements. This method allows for the seamless integration of new information requirements into an existing design. We demonstrate how to effectively accommodate new compliance needs and update the system design in response to evolving requirements. Ultimately, a final design that meets all requirements is achieved.

## Proposed System

Based on the defined requirements, we propose the development of a web application dedicated to real-time face mask detection using YOLOv8 and Streamlit. The proposed system aims to provide a user-friendly interface for monitoring face mask compliance. The blueprint of the application showcases a dynamic display of individuals and their corresponding mask status. Through the utilization of YOLOv8's object detection capabilities, the system will accurately identify individuals wearing masks in real-time. The team has recognized the potential impact of this application and has assigned the task of transforming the blueprint into a functional outcome. The development process will involve the integration of Python, YOLOv8, and Streamlit to create a robust face mask compliance management system. Upon completion, the team is expecting a highly efficient and reliable solution for ensuring face mask adherence.

**Benefits of Face Mask Detector Web Application :**

1. **Public Safety**: The primary benefit of a face mask detector web application is promoting public safety during health crises like pandemics. It helps enforce mask-wearing protocols in public places, reducing the spread of contagious diseases.

2. **Automated Monitoring**: The application can automatically monitor a large number of individuals in real-time, making it more efficient and less labor-intensive compared to manual monitoring.

3. **Alerting and Compliance**: It can provide alerts or notifications to individuals not wearing masks, encouraging compliance with safety measures.

4. **Data Logging and Reporting**: The application can log data on mask compliance, providing valuable information for analysis, reporting, and decision-making by authorities or businesses.

5. **Integration with Surveillance Systems**: It can be integrated with existing surveillance systems to enhance security measures and public safety in high-traffic areas.

6. **Cost-Effective**: It can potentially reduce the need for additional staff dedicated to mask monitoring, making it a cost-effective solution for businesses and public institutions.

**Limitations of Face Mask Detector Web Application :**

1**. Accuracy and False Positives/Negatives**: The accuracy of face mask detection algorithms may vary, and there can be instances of false positives (incorrectly detecting a mask) or false negatives (missing a mask).

2. **Privacy Concerns**: The use of facial recognition technology raises privacy concerns, and users may be apprehensive about being monitored, even for safety purposes.

3**. Dependence on Camera Quality**: The accuracy of face mask detection can be affected by the quality of the camera and lighting conditions. Low-quality cameras or poor lighting may result in less accurate detections.

4. **Ethical Considerations**: The deployment of such applications requires careful consideration of ethical implications, including consent, transparency, and user rights.

5. **Limited to Mask Detection**: The application is designed specifically for mask detection and may not be suitable for other security or surveillance purposes.

6. **Hardware and Bandwidth Requirements**: Depending on the scale of deployment, the application may require specific hardware and adequate bandwidth to handle the processing of video feeds.

7. **Adaptation to Changing Regulations**: The application may require updates or adjustments to adapt to changing mask-wearing regulations, which could be time-consuming and resource-intensive.

8. **Cultural Sensitivity**: Different cultures may have varying perspectives on mask-wearing, and the application may need to be sensitive to cultural norms and practices.

# SOFTWARE REQUIREMENTS SPECIFICATIONS

**3.1 Hardware Requirement Specification**

* Processor - Intel Core i5 or more
* RAM - 8 GB Or more
* Hard Disc - 256 GB SSD or more
* Monitor - Any Monitor
* Keyboard - Any Keyboard
* Mouse - Any mouse
* GPU - Optional but recommended (NVIDIA GPU)
* Cameras - One or More as per project scope
* Internet Connectivity - For Cloud Based processing/Monitoring

**3.2 Software Requirement Specification**

* Operating System - Windows 10/Linux/macOS
* Language - Python(3.10)
* Browser - Preferably Chrome, Safari
* Framework - [StreamLit](https://streamlit.io/) , [Ultralytics(YOLOv8)](https://github.com/ultralytics/ultralytics)
* Application - Visual Studio Code, Anaconda

# 4. TECHNOLOGY

## Python :

Python is a high-level, dynamically typed, and versatile programming language known for its simplicity and readability. Guido van Rossum created Python and released its first version in 1991. Since then, it has gained immense popularity and has become one of the most widely used programming languages in various domains, including web development, data analysis, artificial intelligence, scientific computing, automation, and more.

Here are some key characteristics and features of Python:

* **Readability and Simplicity**: Python's syntax is designed to be clear and readable, which makes it an excellent language for beginners. The use of indentation for code blocks (instead of braces or keywords) enforces clean and organized code.
* **Interpreted Language**: Python is an interpreted language, which means that it doesn't need to be compiled before running. This makes development faster and more flexible.
* **Multi-Purpose Language**: Python can be used for a wide range of tasks, from web development (using frameworks like Django and Flask) to data analysis (using libraries like Pandas and NumPy) to scientific computing and machine learning.
* **Platform Independence** : Python is a cross-platform language, meaning that code written in Python can run on different operating systems without modification. This is possible due to the Python interpreter that is available for various platforms.
* **Large Standard Library**: Python comes with a vast standard library that provides modules and packages for a wide range of tasks, such as file I/O, networking, database access, and more. This eliminates the need for developers to write code for common functionalities from scratch.
* **Dynamically Typed**: Python is dynamically typed, which means that you don't need to declare the type of a variable explicitly. The type is inferred at runtime. This leads to more flexible and concise code.
* **Object-Oriented**: Python supports both procedural and object-oriented programming paradigms. It allows you to define classes, create objects, and implement inheritance and polymorphism.
* **High-Level Data Types**: Python provides built-in high-level data types like lists, dictionaries, sets, and tuples, which simplify data manipulation and storage.
* **Community and Ecosystem**: Python has a large and active community of developers. This leads to a rich ecosystem of libraries and frameworks that enhance Python's capabilities in various domains.
* **Open Source**: Python is an open-source language, which means that its source code is freely available and can be modified and redistributed. This has contributed to its widespread adoption and continuous improvement.
* **Web Frameworks and Libraries** : Python has powerful web frameworks like Django, Flask, and Pyramid, which simplify web development. Additionally, libraries like requests, BeautifulSoup, and Selenium facilitate web scraping and automation.

**Limitations of Python**

* **Slower Execution Speed**: Compared to languages like C++ or Java, Python can be slower in terms of execution speed. This is due to its dynamic nature and interpreted nature. For performance-critical applications, Python may not be the best choice.
* **Not Ideal for Mobile Development**: While there are frameworks like Kivy and BeeWare that allow for mobile development in Python, it is not as widely used or supported as languages like Java or Swift for mobile app development.
* **Less Suitable for Memory-Intensive Tasks**: Python's memory consumption can be higher compared to languages like C or C++. This can be a limitation for applications that need to handle very large datasets or perform memory-intensive computations.
* **Dependency on External Libraries**: While Python's standard library is extensive, for specialized tasks, you may need to rely on external libraries. Managing dependencies can sometimes be a challenge, especially when versions are not compatible.

**Python Development Tools**

Python development is facilitated by a wide range of tools that assist programmers in writing, testing, debugging, and managing Python code. Here are some popular Python development tools:

* **IDLE (Integrated Development and Learning Environment)**: This is Python's built-in IDE. It provides features like syntax highlighting, code completion, and a basic debugger.
* **PyCharm**: Developed by JetBrains, PyCharm is a powerful and feature-rich IDE for Python. It offers code analysis, debugging, version control, and integration with various web frameworks.
* **Visual Studio Code**: A lightweight and versatile code editor, Visual Studio Code supports Python through extensions. It provides a rich set of features including debugging, IntelliSense, and version control.
* **Sublime Text**: Sublime Text is a highly customizable and lightweight text editor that is popular among developers for Python development. It offers a range of plugins and themes to enhance productivity.
* **Jupyter Notebooks**: Jupyter Notebooks provide an interactive environment for Python programming. It's widely used for data science and data analysis tasks as it allows for the combination of code, visualizations, and narrative text.

## Streamlit :

## Streamlit is an open-source Python library that is used for creating web applications for data science and machine learning projects. It allows developers to turn data scripts into shareable web apps with minimal effort. Streamlit is designed to be simple, intuitive, and highly interactive, making it an excellent tool for rapid prototyping and showcasing data-driven projects.

## Advantages of Streamlit:

## Simplicity and Ease of Use: Streamlit is designed to be easy for both beginners and experienced developers. It requires minimal boilerplate code, making it straightforward to create interactive web applications.

## Rapid Prototyping: With Streamlit, you can quickly turn data scripts into shareable web apps. It allows for fast iteration and prototyping of data-driven applications.

## Pythonic Syntax: Since Streamlit is primarily designed for Python, developers can leverage their existing Python knowledge and skills without the need to learn additional languages or frameworks.

## Intuitive Widgets and Components: Streamlit provides a variety of widgets and components (like sliders, buttons, and charts) that can be easily integrated into the application. These components can be added with simple function calls.

## Interactive Data Visualization: Streamlit integrates well with popular data visualization libraries like Matplotlib, Plotly, and Altair, allowing for the creation of dynamic and interactive charts.

## Real-Time Updates: Streamlit automatically updates the web app in real-time as code changes are made, making it easy to see the effects of modifications instantly.

## Wide Ecosystem Integration: Streamlit can be easily integrated with popular data science and machine learning libraries like Pandas, NumPy, Scikit-learn, and TensorFlow.

## Customizable Themes and Layouts: Streamlit allows for customization of the appearance of the web application, including themes, colors, and layouts.

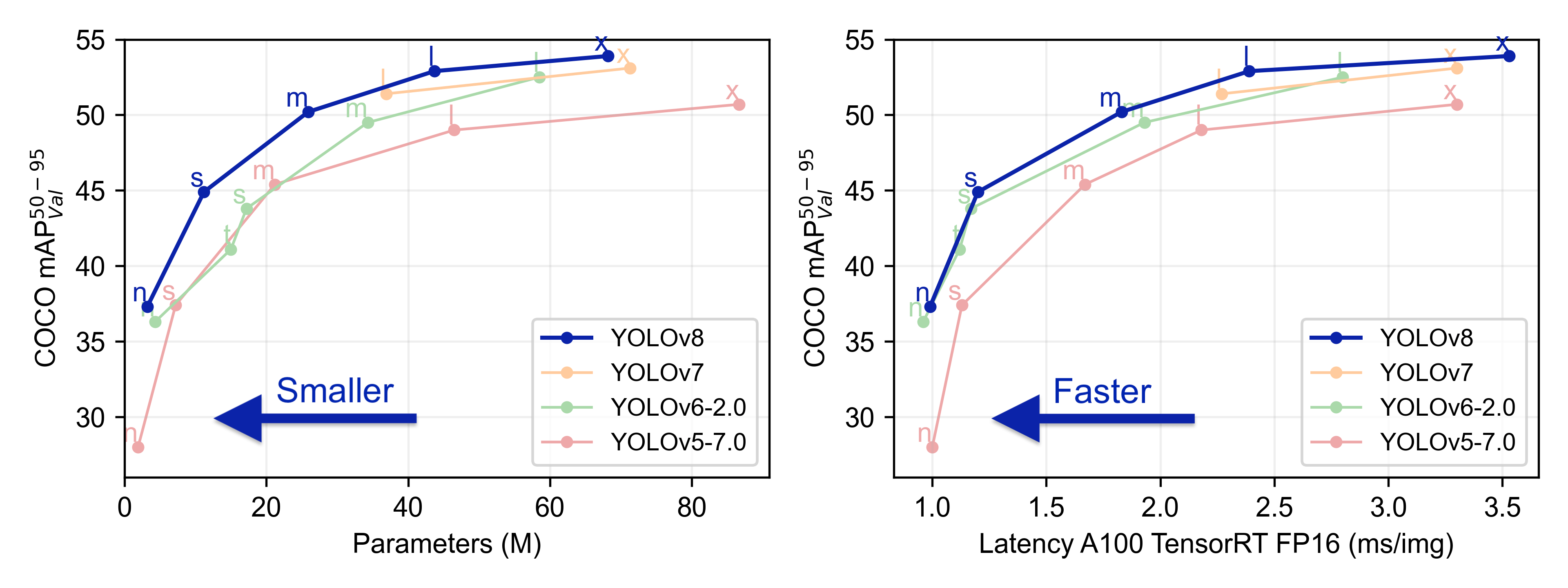
## Deployment Options: Streamlit apps can be easily deployed on platforms like Heroku, AWS, or even as standalone applications. It also provides options for sharing applications via Streamlit Sharing.

**Limitations of Streamlit:**

* **Limited for Complex Frontend UI**: While Streamlit excels at creating simple and interactive web applications, it may not be the best choice for complex, highly customized user interfaces that require extensive JavaScript or CSS.
* **Limited to Python**: Streamlit is designed to work primarily with Python. If a project requires the integration of other programming languages, it may not be the most suitable tool.
* **Limited Widget Variety**: While Streamlit provides a good set of basic widgets, it may not offer the same level of variety and customization as more specialized frontend frameworks like React/Angular.
* **Less Control Over Web Components**: For developers who require fine-grained control over every aspect of the web application, Streamlit's high-level abstractions may limit their ability to implement specific features or behaviors.
* **Less Suitable for Large-Scale Applications**: While Streamlit is excellent for rapid prototyping and building data-centric applications, it may face performance challenges when used for very large-scale or enterprise-level applications.

## YOLO :

* Reference : <https://www.youtube.com/watch?v=9s_FpMpdYW8> (How yolo Works)
* YOLO (You Only Look Once), a popular object detection and image segmentation model, was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, YOLO quickly gained popularity for its high speed and accuracy.
* YOLOv2, released in 2016, improved the original model by incorporating batch normalization, anchor boxes, and dimension clusters.
* YOLOv3, launched in 2018, further enhanced the model's performance using a more efficient backbone network, multiple anchors and spatial pyramid pooling.
* YOLOv4 was released in 2020, introducing innovations like Mosaic data augmentation, a new anchor-free detection head, and a new loss function.
* YOLOv5 further improved the model's performance and added new features such as hyperparameter optimization, integrated experiment tracking and automatic export to popular export formats.
* YOLOv6 was open-sourced by Meituan in 2022 and is in use in many of the company's autonomous delivery robots.
* YOLOv7 added additional tasks such as pose estimation on the COCO keypoints dataset.
* YOLOv8 is the latest version of YOLO by Ultralytics. As a cutting-edge, state-of-the-art (SOTA) model, YOLOv8 builds on the success of previous versions, introducing new features and improvements for enhanced performance, flexibility, and efficiency. YOLOv8 supports a full range of vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification. This versatility allows users to leverage YOLOv8's capabilities across diverse applications and domains.



We will be using YOLOv8 model for our project as it has fastest inference speed as well as accuracy.

YOLOv8 Github Link : <https://github.com/ultralytics/ultralytics>

**Benefits of YOLOv8:**

1. **Real-Time Detection**: YOLOv8 is designed for real-time object detection. It's capable of processing images and videos in near real-time, making it suitable for applications like surveillance, autonomous driving, and more.

2. **High Accuracy**: YOLOv8 achieves high accuracy in object detection tasks. It's known for its ability to accurately detect and localize objects within images.

3**. Efficiency**: YOLOv8 is efficient in terms of speed and computational resources. It achieves high performance even on resource-constrained devices, making it suitable for edge computing applications.

4. **Multi-Class Detection**: YOLOv8 can simultaneously detect multiple classes of objects within an image. This is crucial for tasks where there are multiple object categories of interest.

5. **Flexibility in Model Architecture**: YOLOv8 allows for flexibility in model architecture, allowing users to choose from different versions (such as YOLOv3, YOLOv4, etc.) and backbones (like Darknet, CSPDarknet, etc.) depending on their specific requirements.

6. **Easy Integration**: YOLOv8 can be integrated with various programming languages and platforms, including Python, making it accessible to a wide range of developers.

**Limitations of YOLOv8:**

1. **Training Data and Time**: Achieving high accuracy with YOLOv8 often requires a substantial amount of annotated training data and significant training time. This can be a resource-intensive process.

2**. Resource Intensive during Inference**: While YOLOv8 is efficient, it can still be resource-intensive during inference, especially on lower-end hardware. This can be a limitation for deployment on edge devices.

3**. Limited for Small Object Detection**: YOLOv8 may struggle with detecting very small objects within an image. It's more suited for detecting medium to large objects.

4. **Model Complexity**: Depending on the specific configuration and backbone used, YOLOv8 can be a relatively complex model. This may require careful optimization for deployment on certain devices.

5. **Domain Specific Tuning**: Achieving optimal performance with YOLOv8 often requires fine-tuning on a specific domain or dataset. This can be a challenge, especially for users without access to large and diverse datasets.

6. **Interpretability**: Like many deep learning models, YOLOv8 can be considered a "black box" in terms of understanding how it arrives at its predictions. This can be a limitation in scenarios where interpretability is crucial.

**4.4 Overview of Web application using Python, Streamlit and YOLOv8**

**Python:**

Python is a versatile and widely used programming language known for its simplicity, readability, and extensive ecosystem of libraries and frameworks. It is used in various domains including web development, data science, artificial intelligence, scientific computing, automation, and more.

**Streamlit:**

Streamlit is an open-source Python library that simplifies the process of creating web applications for data science and machine learning projects. It allows developers to turn data scripts into interactive and shareable web apps with minimal effort. Streamlit is known for its simplicity, ease of use, and rapid prototyping capabilities**.**

**YOLOv8 (You Only Look Once version 8):**

YOLOv8 is a state-of-the-art object detection model known for its speed and accuracy. It's designed for real-time object detection tasks and is capable of processing images and videos in near real-time. YOLOv8 is efficient and suitable for applications like surveillance, autonomous driving, and more.

**Overview of the Web Application:**

**1. Integration of Python for backend :**

- Python serves as the primary programming language for the application. It is used to handle server-side logic, data processing, and interactions with the YOLOv8 model.

- Python's extensive libraries and frameworks make it ideal for managing data, performing computations, and handling various tasks in the backend.

**2. Streamlit for Frontend:**

- Streamlit is utilized for creating the frontend of the web application. It provides a simple and intuitive interface for users to interact with the application.

- Streamlit's widgets and components (like buttons, sliders, charts) are used to create an interactive user experience.

**3. YOLOv8 for Object Detection :**

- YOLOv8 is integrated into the backend to perform object detection tasks. It processes images or video frames to identify and locate objects within them.

- The model's accuracy and efficiency in real-time detection contribute to the application's effectiveness in scenarios like surveillance or image analysis.

**4. Real-time Feedback:**

- The application likely provides real-time feedback to users as they interact with it. This is made possible by the combination of Python's processing capabilities and YOLOv8's speed.

**5. Deployment and Accessibility:**

- Once developed, the web application can be deployed on various platforms or servers to make it accessible to users. This could include cloud services, web servers, or even deployment platforms specific to Streamlit applications.

**The web application integrates Python for backend processing, Streamlit for frontend development, and YOLOv8 for real-time object detection. This combination leverages the strengths of each tool to create an effective and user-friendly application, particularly in scenarios where object detection is a crucial component.**

1. **Coding**

**Project Github Link :** [**https://github.com/nitin7478/YOLOv8-StreamLit-FaceMask-Detection**](https://github.com/nitin7478/YOLOv8-StreamLit-FaceMask-Detection)

**5.1 Project Structure :**

**.**

**├── src/**

**│ ├── components/**

**│ │ ├── init.py**

**│ │ ├── image\_detector.py**

**│ │ ├── video\_detector.py**

**│ ├── constants/**

**│ │ ├── init.py**

**│ │ ├── constant.py**

**├── sample\_dataset/**

**│ ├── demo.jpeg**

**│ ├── demo.mp4**

**├── sample\_output/**

**│ ├── [Sample output files]**

**├── models/**

**│ ├── best\_3.pt**

**├── app.py**

**├── requirements.txt**

**├── packages.txt**

**├── README.md**

**├── .gitignore**

**├── YOLOv8\_Tutorial.ipynb**

**├──model\_trainer.ipynb**

**Description**

Explanation of purpose of each major component or directory in the project structure.

- `src/`: Contains source code for the project.

- `components/`: Holds various components used in the project.

- `\_\_init\_\_.py`: An empty file that makes the "components" directory a Python package.

- `image\_detector.py`: Contains code for detecting objects in images.

- `video\_detector.py`: Contains code for detecting objects in videos.

- `constants/`: Houses constant values used throughout the project.

- `\_\_init\_\_.py`: An empty file that makes the "constants" directory a Python package.

- `[Constant files]`: Actual constant files go here.

- `sample\_dataset/`: Contains a sample dataset for testing and demonstration purposes.

- `sample\_output/`: Contains sample output files generated by the project.

- `models/`: Contains pre-trained machine learning models used in the project.

- `app.py`: Main web application file.

- `requirements.txt`: List of Python packages required to run the project.

- `packages.txt`: Additional packages or libraries used in the project, if any.

**5.2 Setup python conda environment**

Make sure you have anaconda or miniconda installed in your system.

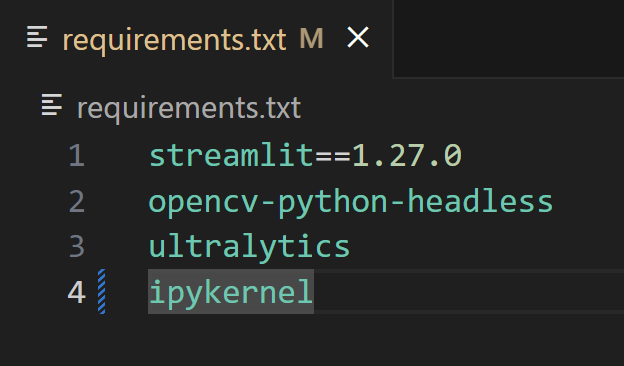
Create conda environment

    conda create -p venv python=3.10 -y

Install pytorch and torchvision libraries as per your system(CPU or GPU)

<https://pytorch.org/get-started/locally/>

Install required packages from requirements.txt file containing following libraries streamlit , opencv-python-headless, ultralytics, ipykernel



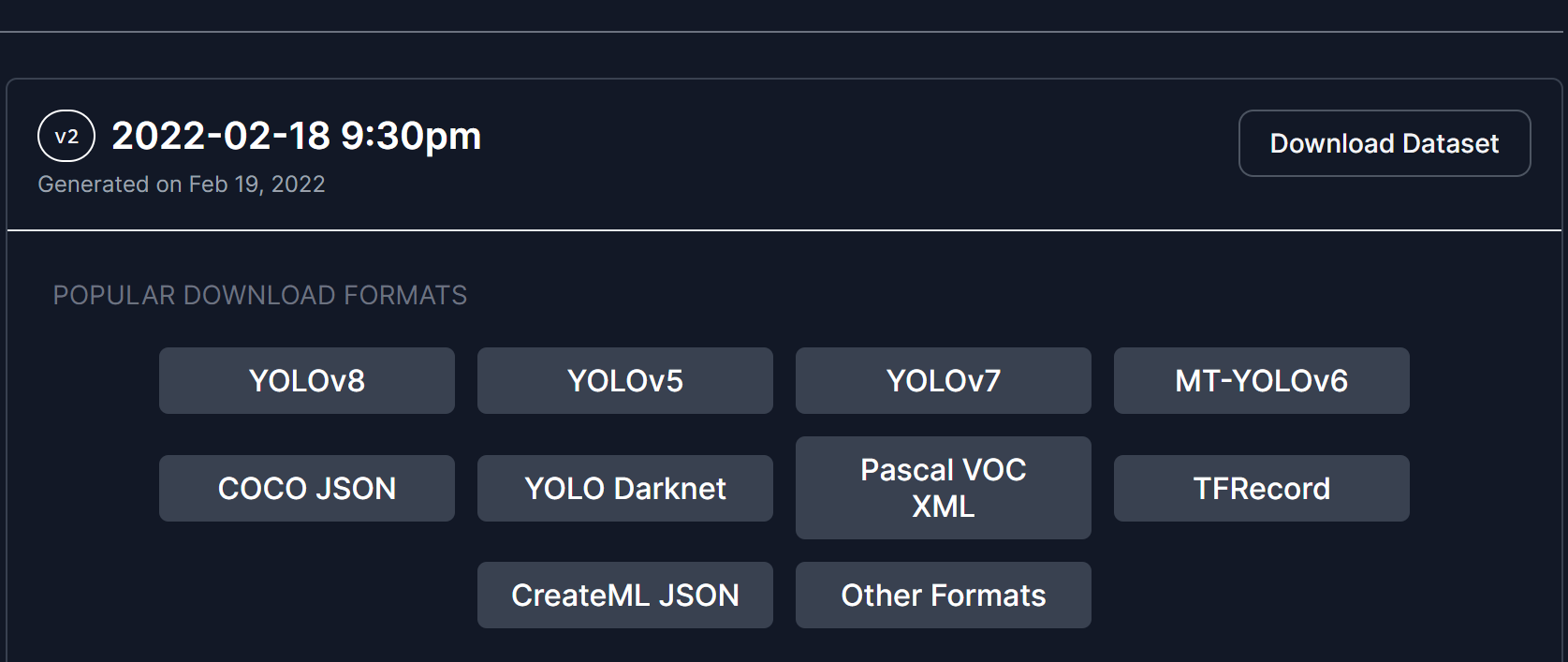
pip install -r requirements.txt

**5.3 Model Training (Refer to** [**https://docs.ultralytics.com/modes/**](https://docs.ultralytics.com/modes/)**)**

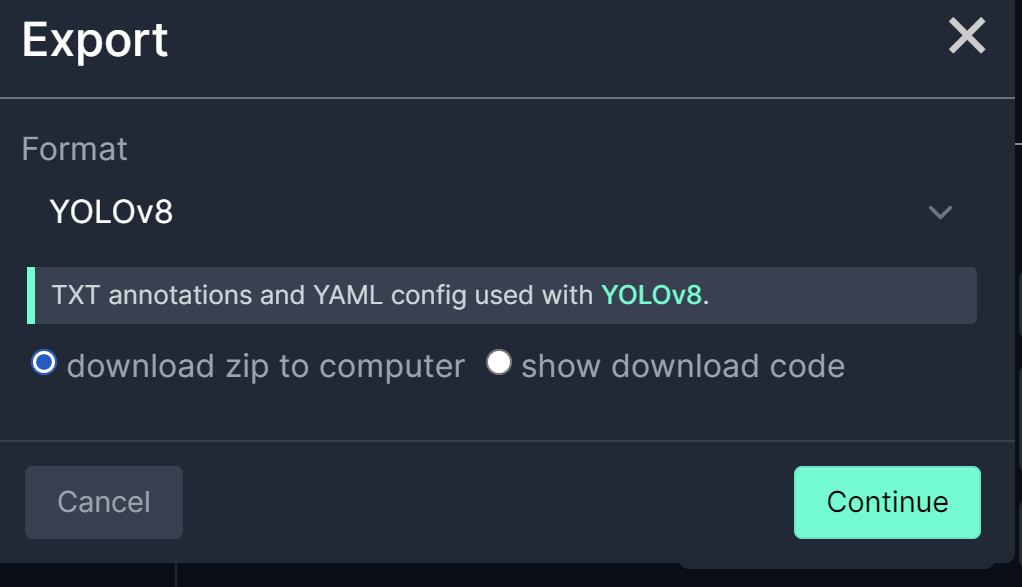
* Downloading Dataset :

Dataset Link : <https://universe.roboflow.com/new-workspace-2cnfr/mask-ecop7/dataset/2>

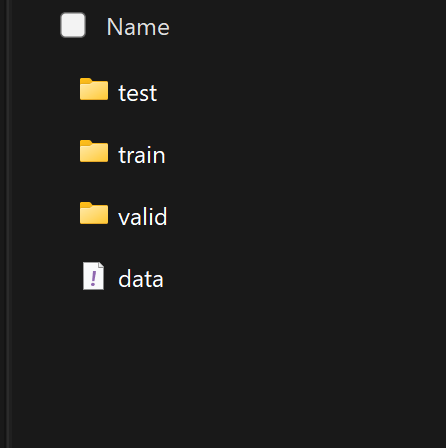
* Clink on button Download Dataset button



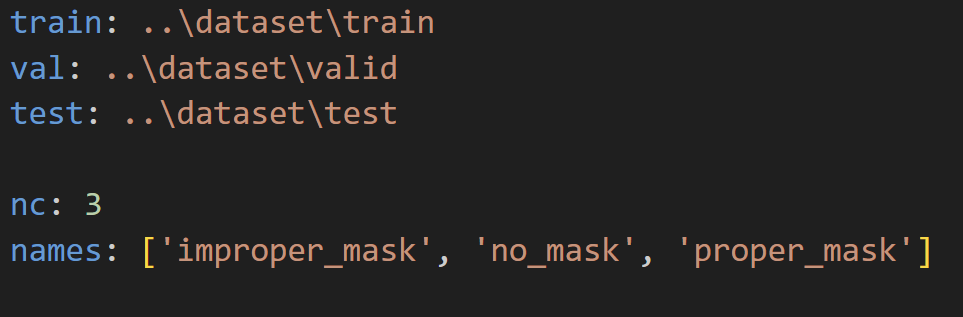
* Select export format YOLOv8 and select download ZIP to computer



* Create dataset folder in your project folder and extract zipfile inside dataset folder. Dataset folder will look like this:



* Configure data.yaml file inside dataset folder. Enter correct paths for train, valid and test sets. data.yaml file will look like this:



* We will create model\_trainer.ipynb jupyter notebook file for training model, and write following code for training model

import ultralytics

ultralytics.checks()

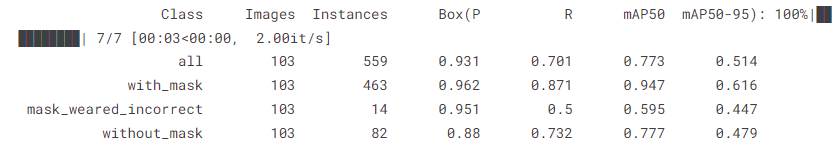


from ultralytics import YOLO

model = YOLO('yolov8n.pt')  # load a pretrained YOLOv8n detection model

model.train(data='dataset/data.yaml', epochs=10)

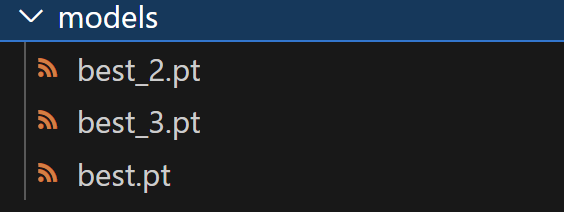
* Above code will start model training on out custom dataset on pretrained YOLOv8n.pt model. After completion metrics will generated like below:



* We will check accuracy on test set by below code to check whther model generalizes will or not.

metrics = model.val(data='dataset/data.yaml')

* After training runs/train/weights folders will be created. Select best.pt model file and copy it in outer project folder models



Note : I have trained multiple models. best.pt / best\_2.pt / best\_3.pt

* For prediction use code (replace image path with your image or video path)

model.predict(image\_path)

* We have completed model training now we will write code for web application and project files

**5.4 src/components/image\_detector.py file code**

import cv2

from ultralytics import YOLO

import math

def load\_yolonas\_process\_each\_image(image , confidence , st):

    model = YOLO('models/best\_3.pt')

    classNames = ['proper\_mask','no\_mask','improper\_mask']

    detections = model.predict(image , conf=confidence)[0]

    for detection in detections.boxes.data.tolist():

        x1, y1, x2, y2, conf, class\_id = detection

        x1, y1, x2, y2 = int(x1) , int(y1), int(x2) , int(y2)

        classname = classNames[int(class\_id)]

        conf = math.ceil(conf\*100)/100

        label = f'{classname}{conf}'

        t\_size = cv2.getTextSize(label , 0 , fontScale=1, thickness=2)[0]

        c2 = x1 + t\_size[0], y1 - t\_size[1] -3

        cv2.rectangle(image , (x1,y1) , (x2, y2), [0,255,255] , 3)

        cv2.rectangle(image, (x1 , y1), c2, [255,144,30], -1 , cv2.LINE\_AA)

        cv2.putText(image , label , (x1, y1-2), 0 , 1, [255, 255, 255], thickness=2, lineType=cv2.LINE\_AA)

    # resize\_image = cv2.resize(image , (0,0), fx=0.4 , fy=0.4 , interpolation=cv2.INTER\_AREA)

    st.subheader('Output Image')

    st.image(image, channels ='BGR', use\_column\_width = True)

    # cv2.imshow('Frame', resize\_image)

    # cv2.waitKey(0)

    # cv2.destroyAllWindows()

**5.5 src/components/video\_detector.py file code**

import cv2

from ultralytics import YOLO

import time

def load\_yolonas\_process\_each\_frame(video\_name, kpi\_text  ,kpi2\_text,kpi3\_text, stframe, conf):

    cap = cv2.VideoCapture(video\_name)

    width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

    height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

    model = YOLO('models/best\_3.pt')

    previous\_time = 0

    # Loop through the video frames

    while cap.isOpened():

        # Read a frame from the video

        success, frame = cap.read()

        if success:

            # Run YOLOv8 inference on the frame

            results = model(frame , conf=conf)

            # Visualize the results on the frame

            annotated\_frame = results[0].plot()

            stframe.image(annotated\_frame , channels='BGR', use\_column\_width = True)

            current\_time = time.time()

            fps = 1/(current\_time - previous\_time)

            previous\_time = current\_time

            kpi\_text.write(f'<h1 style= "text-align:center"; color:red> {"{:.1f}".format(fps)}</h1>', unsafe\_allow\_html=True)

            kpi2\_text.write(f'<h1 style= "text-align:center"; color:red> {"{:.1f}".format(width)}</h1>', unsafe\_allow\_html=True)

            kpi3\_text.write(f'<h1 style= "text-align:center"; color:red> {"{:.1f}".format(height)}</h1>', unsafe\_allow\_html=True)

        else:

            break

    cap.release()

**5.6 app.py file code**

import streamlit as st

import cv2 , tempfile

import numpy as np

from PIL import Image

from src.components.image\_detector import load\_yolonas\_process\_each\_image

from src.components.video\_detector import load\_yolonas\_process\_each\_frame

def main():

    st.title('Object Detection with YOLOv8')

    st.sidebar.title('Settings')

    st.sidebar.markdown('---')

    st.sidebar.subheader('')

    st.markdown(

        """

        <style>

        [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {

            width: 300px;

        }

        [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {

            width: 300px;

            margin-left:-300px;

        }

        </style>

        """,

        unsafe\_allow\_html=True

        )

    app\_mode = st.sidebar.selectbox(' Choose the App Mode ',['About App', 'Run on Image','Run on Video']) #,'Output/Processed Video'

    if app\_mode =='About App':

        st.markdown('In this project I am using \*\*YOLO-V8\*\* model to do Object Detection on Images and Videos and we are using \*\*\*StreamLit\*\*\* to create web application and GUI.')

        st.markdown(

        """

        <style>

        [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {

            width: 300px;

        }

        [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {

            width: 300px;

            margin-left:-300px;

        }

        </style>

        """,

        unsafe\_allow\_html=True

        )

        # st.video('')

        st.markdown('''

                    ## About Me \n

                    ## Its Nitin Udmale , Data Scientist Enthusiast. \n

                    ## [Linkedn Profile] (https://www.linkedin.com/in/nitinudmale/) \n

                    ## [Github] (https://github.com/nitin7478/) \n

                    ## ''')

    elif app\_mode=='Run on Image':

        # logging.info(f"Run on image app mode started")

        confidence = st.sidebar.slider('Confidence', min\_value=0.15, max\_value=1.0)

        st.markdown(

        """

        <style>

        [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {

            width: 300px;

        }

        [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {

            width: 300px;

            margin-left:-300px;

        }

        </style>

        """,

        unsafe\_allow\_html=True

        )

        img\_file\_buffer = st.sidebar.file\_uploader('Upload an Iamage', type=['jpg', 'jpeg', 'png'])

        Demo\_image = 'sample\_dataset/demo.jpg'

        if img\_file\_buffer is not None:

            img = cv2.imdecode(np.fromstring(img\_file\_buffer.read(), np.uint8),1)

            image = np.array(Image.open(img\_file\_buffer))

        else:

            img = cv2.imread(Demo\_image)

            image = np.array(Image.open(Demo\_image))

        st.sidebar.text('Original Image')

        st.sidebar.image(image)

        load\_yolonas\_process\_each\_image(img, confidence, st)

        # logging.info(f"Run on image mode completed successfully")

    elif app\_mode=='Run on Video':

        conf = st.sidebar.slider('Confidence', min\_value=0.25, max\_value=1.0)

        st.markdown(

        """

        <style>

        [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {

            width: 300px;

        }

        [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {

            width: 300px;

            margin-left:-300px;

        }

        </style>

        """,

        unsafe\_allow\_html=True

        )

        use\_webcam = st.sidebar.checkbox('Use Webcam')

        st.sidebar.markdown('---')

        video\_file\_buffer = st.sidebar.file\_uploader('Upload a Video', type=["mp4","avi","mov","asf"])

        Demo\_video = 'sample\_dataset/demo.mp4'

        tffile = tempfile.NamedTemporaryFile(suffix='.mp4', delete=False)

        st.markdown(

            """ Detection performance may vary as per your system configuration

            """)

        if not video\_file\_buffer:

            if use\_webcam:

                tffile.name = 0

            else:

                tffile.name = Demo\_video

                demo\_vid = open(tffile.name , 'rb')

                demo\_bytes = demo\_vid.read()

                st.sidebar.text('Input Video')

                st.sidebar.video(demo\_bytes)

        else:

            tffile.write(video\_file\_buffer.read())

            demo\_vid = open(tffile.name , 'rb')

            demo\_bytes = demo\_vid.read()

            st.sidebar.text('Input Video')

            st.sidebar.video(demo\_bytes)

        stframe = st.empty()

        st.markdown("<hr/>", unsafe\_allow\_html=True)

        kpi1, kpi2, kpi3 = st.columns(3)

        with kpi1:

            st.markdown("\*\*Frame Rate\*\*")

            kpi1\_text = st.markdown("0")

        with kpi2:

            st.markdown("\*\*Width\*\*")

            kpi2\_text = st.markdown("0")

        with kpi3:

            st.markdown("\*\*Height\*\*")

            kpi3\_text = st.markdown("0")

        st.markdown("<hr/>", unsafe\_allow\_html=True)

        load\_yolonas\_process\_each\_frame(tffile.name, kpi1\_text, kpi2\_text, kpi3\_text, stframe, conf)

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

    try:

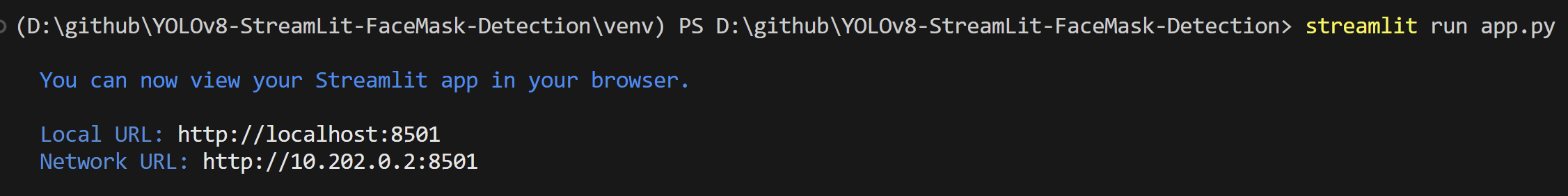
        main()

    except Exception as e:

        raise Exception(e)

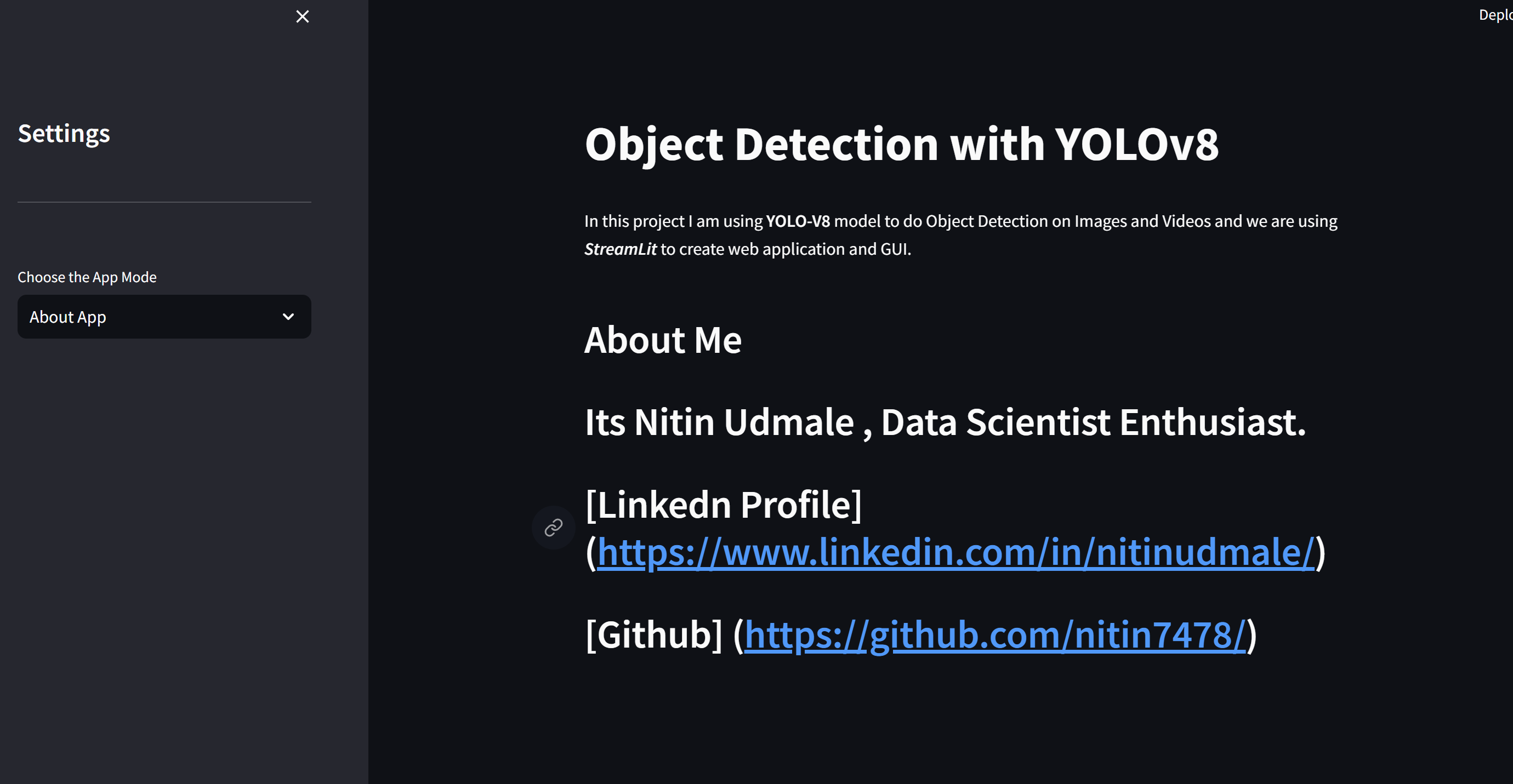
1. **Result**

**To run web application type following command in vscode terminal :**

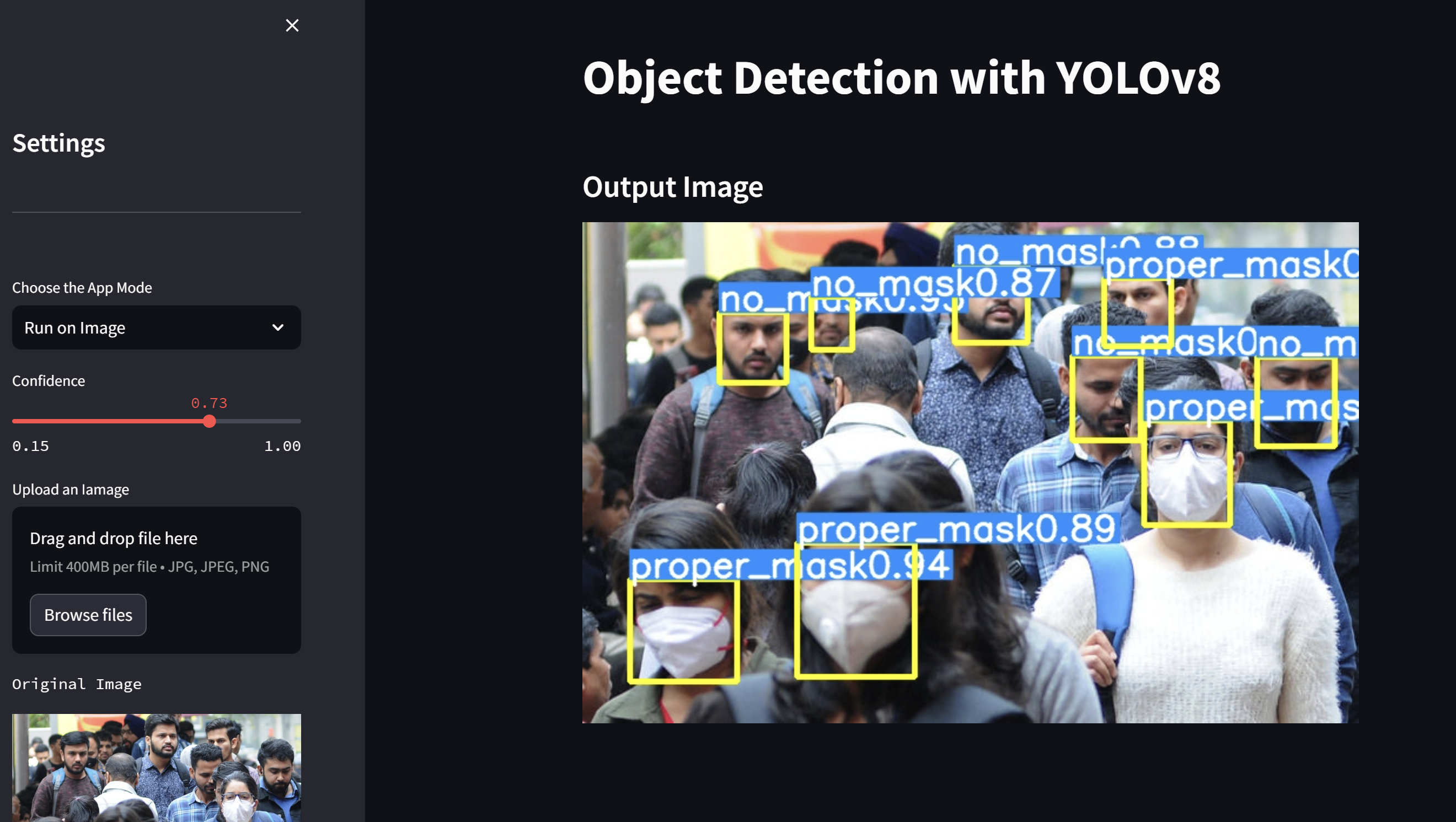
****

It will start local sever on vscode and open web application on you browser.

**About App section :**



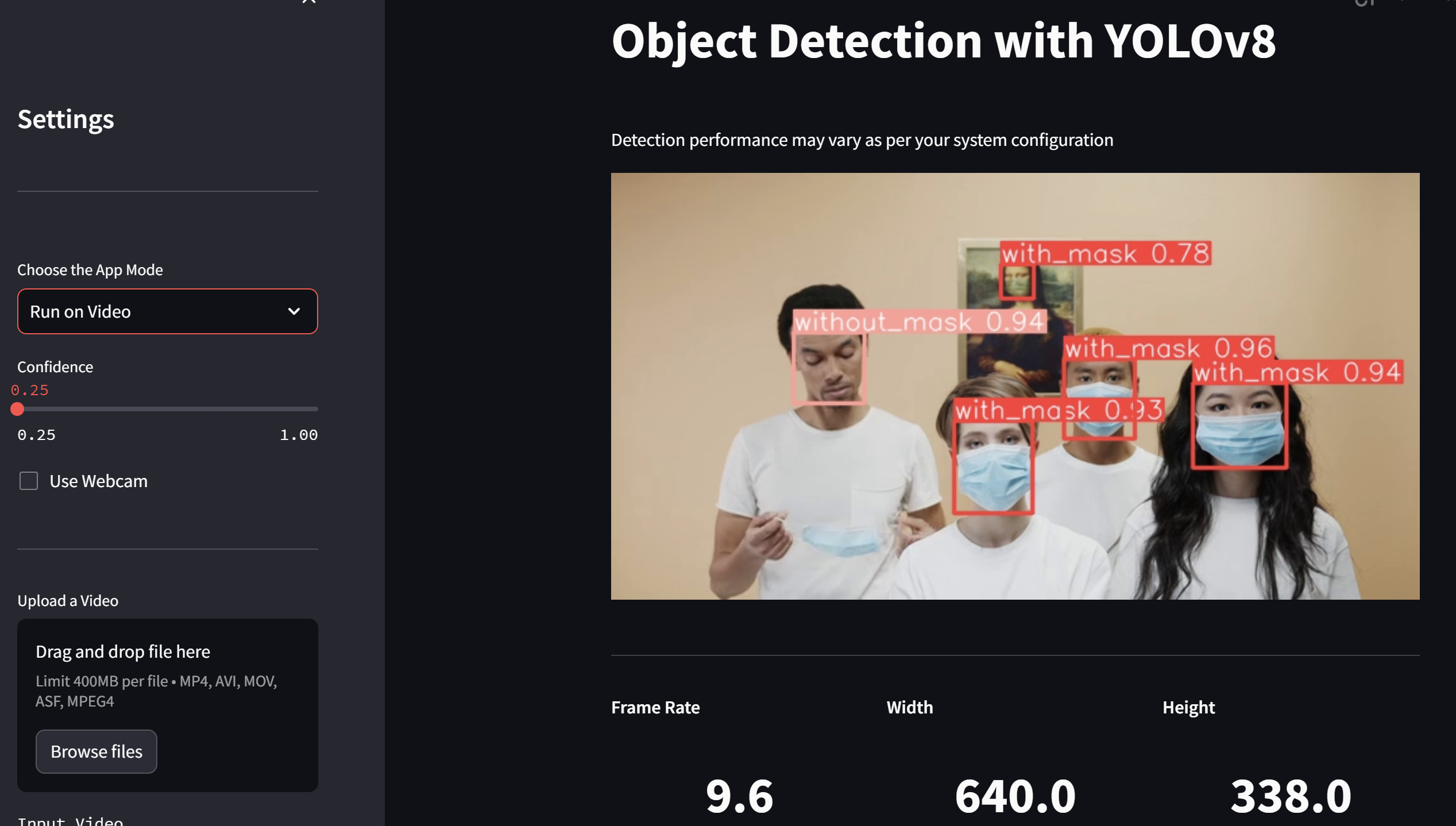
**Run on Image section : We can see model detects mask , no mask classes.**



User can upload any image for mask detection task in this section.

User can set confidence value(threshold) of the predicted outputs.

**Run on Video section : We can see model detects mask , no mask classes.**



Use can upload any video or start webcam by clicking on the checkbox. As we can see system is working as expected.We can see frame rate of processing task, width x height.

1. **Conclusion**

In this mini-project, we successfully developed a Face Mask Detector web application using Streamlit, a popular Python web app framework. The application leverages computer vision techniques to detect whether individuals are wearing face masks in real-time video streams.

The key components of our project include:

Data Collection and Annotation : We gathered a diverse dataset of images containing individuals with and without face masks. This dataset was annotated to train our machine learning model.

Machine Learning Model: We employed a pre-trained Convolutional Neural Network (CNN) architecture, fine-tuned for mask detection. This model was integrated into the Streamlit application.

Real-time Video Stream: The application is capable of processing live video streams from a webcam, making it practical for real-world scenarios.

User Interface: Streamlit provided an intuitive and user-friendly interface for interacting with the application. Users can easily upload images or access their webcam feed for mask detection.

Accuracy and Performance: The model demonstrated high accuracy in detecting face masks, with a Mean Average Precision(50%) of 77.3 and Mean Average Precision(50-95%) of 51.4. Which are pretty good results for real world application.

Deployment: The application is ready for deployment and can be hosted on various platforms for public access.

This project has practical implications in scenarios where monitoring compliance with face mask mandates is crucial for public health and safety. It can be further enhanced with features such as notifications, crowd density estimation, and mask utilization statistics.

In conclusion, this mini-project has not only provided hands-on experience in developing a computer vision-based web application but also contributed to the broader efforts in mitigating the spread of contagious diseases. With further refinement and integration of advanced features, this application has the potential to be a valuable tool in various public spaces.

# REFERENCES

* <https://www.coursera.org/specializations/deep-learning> (Deep Learning Course by Andrew NG)
* <https://github.com/ultralytics/ultralytics> (YOLOv8 Docs and papers)
* <https://docs.roboflow.com/> (Roboflow documents)
* <https://www.youtube.com/watch?v=m9fH9OWn8YM> (Training Yolov8 custom dataset)
* <https://www.youtube.com/watch?v=FPH58P89p1E> (Object detection on custom dataset)
* <https://www.youtube.com/watch?v=_Um12_OlGgw> (Streamlit web app tutorial)
* <https://www.youtube.com/watch?v=9s_FpMpdYW8> (How yolo Works)

Project Source Code Github Link : [**https://github.com/nitin7478/YOLOv8-StreamLit-FaceMask-Detection**](https://github.com/nitin7478/YOLOv8-StreamLit-FaceMask-Detection)