# Mask Wear Detector with Computer Vision

### AN MINI PROJECT 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

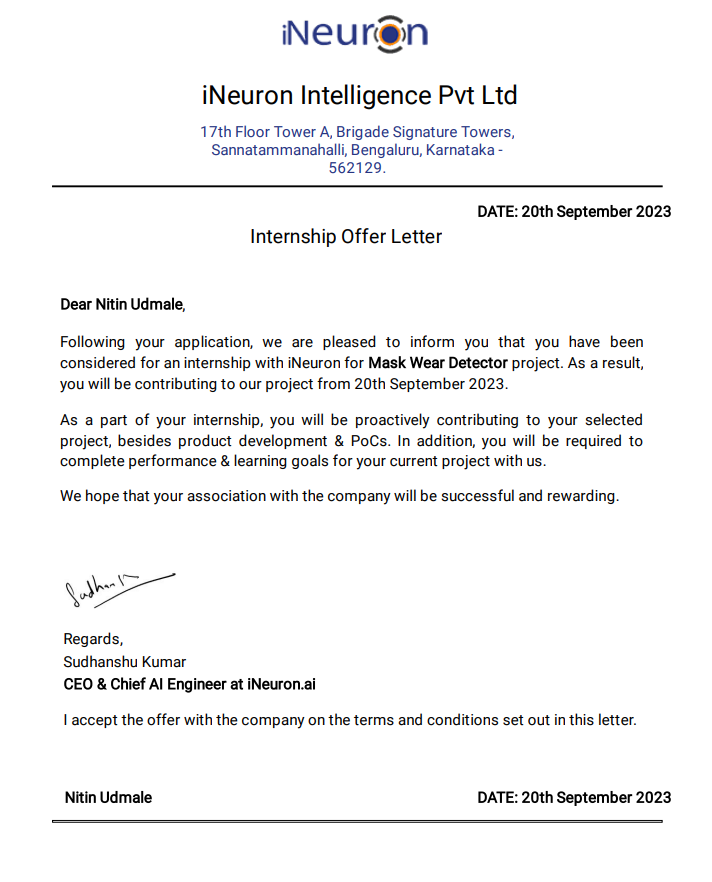
DIRECTORATE OF ONLINE EDUCATION

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

This Mini Project Work Report titled **“Mask Wear Detector with Computer Vision”** of **“Nitin Udmale [EA2232251010390]”**, who carried out the Mini Project 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

# MINI PROJECT OFFER LETTER



**Project Source Code Github :** **<https://github.com/nitin7478/YOLOv8-StreamLit-FaceMask-Detection>**

# 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. **Computer Vision Introduction 6**
  3. [**System Requirements**](#_TOC_250006) **9**
  4. [**Topics 9**](#_TOC_250005)

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

**4.2 Tools for Data Annotation 12**

**4.3** [**Data Exploration & Preprocessing**](#_TOC_250004) **13**

**4.4 Deep Learning 15**

**4.5** [**Convolutional Neural Networks**](#_TOC_250004) **17**

**4.6 YOLOv8 19**

**5. Coding 21**

**6. Result 31**

**7. Conclusion 33**

**8. References 34**

# ABSTRACT

Computer vision, Subset of Machine Learning, is like giving computers "eyes" so they can understand pictures and videos, just like we do. It's a mix of computer science and clever math that helps computers recognize objects, shapes, and even people in images or videos. This helps in things like self-driving cars, medical scans, and even fun filters on social media.

Imagine a computer being able to tell if someone is wearing a mask or not, just by looking at a picture. That's what computer vision can do! It's really important for things like health and safety.

We also need to make sure the computer is really good at this job. We use special tests to check how accurate it is, like when a teacher gives you a test to see how well you've learned something. This way, we can trust the computer to do its job well.

The conceptual foundation encompasses the utilization of convolutional neural networks (CNNs) and image recognition methodologies to discern facial features and distinguish between individuals with and without masks. Through supervised learning techniques, a robust model is developed and trained on a diverse dataset, enabling real-time detection of mask adherence.

A pivotal aspect of the project involves the rigorous evaluation of model performance, encompassing metrics such as accuracy, precision, recall, and F1-score. The system's efficacy is further assessed through extensive testing across various environments and scenarios, including diverse lighting conditions and camera perspectives.

In a world full of pictures and videos, computer vision is like a superpower that helps computers see and understand the world just like we do. It's exciting and has a lot of potential to make our lives better.

The Mask Wear Detector 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.

# COMPUTER VISION INTRODUCTION

**2.1 Computer Vision and CNNs (Convolutional Neural Networks)**

Computer Vision: Computer vision is a field of artificial intelligence focused on teaching computers to interpret and understand visual information from images and videos, just like humans do. It's used in applications like image recognition, object detection, and facial recognition.

CNNs (Convolutional Neural Networks): CNNs are a class of deep learning models designed for computer vision tasks. They are inspired by the human visual system and are exceptionally effective at tasks like image classification and object detection**.**

* **Layers of Artificial Intelligence:**

**Artificial Intelligence**

**Machine Learning**

**Deep Learning**

**ANN**

**Natural Language Processing**

**Computer Vision**

RNN LSTM

**CNN**

**Generative AI**

**2.2 Supervised Learning in Computer Vision:**

In supervised learning for computer vision, we have a dataset where each example consists of both input images and their corresponding labels or categories. The algorithm learns to map the visual features in the input images to their respective labels during the training process. This mapping is then used to make predictions on new, unseen images.

* **Image Classification:**

Description: Given a dataset of images, each image is assigned a label or category (e.g., "cat", "dog", "car").

Example : Training a model to recognize different types of fruits in images.

* **Object Detection :**

Description : This task involves identifying and localizing specific objects within an image by drawing bounding boxes around them.

Example : Building a system that can detect and locate cars in traffic camera images.

* **Semantic Segmentation :**

Description : In this task, each pixel in an image is assigned a label, effectively dividing the image into meaningful segments.

Example : Identifying and segmenting different types of road elements in autonomous driving scenarios.

* **Face Recognition :**

Description : The goal here is to recognize individuals based on facial features, typically mapping faces to specific identities.

Example : Creating a system for access control using facial recognition.

**2.2 Unsupervised Learning in Computer Vision :**

In unsupervised learning for computer vision, we work with datasets that lack explicit labels or categories. The algorithm must discover patterns, structures, or relationships within the data without guidance.

* **Clustering :**

Description : This involves grouping similar images together based on their visual features, without any prior knowledge of specific categories.

Example : Organizing a large dataset of images into groups of visually similar content.

* **Dimensionality Reduction :**

Description : Techniques like Principal Component Analysis (PCA) or t-SNE can be used to reduce the complexity of high-dimensional image data while preserving its meaningful structure.

Example : Simplifying the representation of images for faster processing or visualization.

* **Generative Models :**

Description : Generative models aim to learn the underlying probability distribution of the data in order to generate new, similar samples.

Example : Creating realistic images of human faces that were not in the original dataset.

* **Anomaly Detection :**

Description : This task involves identifying unusual or anomalous patterns in image data, which may indicate errors or outliers.

Example : Detecting defects in manufactured products on an assembly line.

NOTE :Supervised learning in computer vision relies on labeled data to train models for specific tasks, while unsupervised learning seeks to uncover patterns and structures within the data itself, without explicit guidance. Both approaches play crucial roles in advancing the capabilities of computer vision systems.

### SYSTME REQUIREMENTS

**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

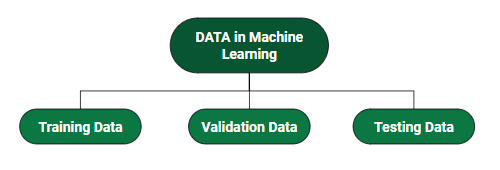
**3.2 Software Requirement Specification**

* 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. TOPICS

**4.1 Data :**

It can be any unprocessed fact, value, text, sound, or picture that is not being interpreted and analyzed. Data is the most important part of all Data Analytics, Machine Learning, Artificial Intelligence. Without data, we can’t train any model and all modern research and automation will go in vain. Big Enterprises are spending lots of money just to gather as much certain data as possible.



**Data in Computer Vision:**

In computer vision, data refers to the collection of visual information, typically in the form of images or videos, used to train, validate, and test machine learning models. This data is crucial for teaching algorithms to recognize patterns, objects, and features within images. The quality, diversity, and quantity of data play a significant role in the performance of computer vision models.

**Types of Data in Computer Vision:**

* Images: The most common type of data in computer vision is static images. These can be photographs, medical scans, satellite imagery, or any other form of visual representation.
* Videos: Video data consists of a sequence of images captured over time. It's used for tasks like action recognition, motion analysis, and object tracking.
* Depth Data: This type of data provides information about the distance of objects from the camera. It's used in tasks like 3D reconstruction and augmented reality.
* Point Clouds: Point cloud data represents a 3D scene by a collection of points in space. It's used extensively in applications like robotics and autonomous navigation.

**Data Annotation in Computer Vision:**

Data annotation involves labeling or tagging the visual data with relevant information. This annotation process provides ground truth labels that serve as the target output during training. Annotations are crucial for supervised learning, where the model learns to associate visual patterns with specific labels.

**Types of Annotations:**

* Bounding Boxes: Used in object detection tasks, bounding boxes are rectangular regions that enclose objects of interest in an image.
* Segmentation Masks: For tasks like semantic segmentation, each pixel in an image is labeled with the corresponding object or class.
* Key points : Key point annotations mark specific points of interest on objects. This is used in tasks like pose estimation.
* Class Labels: In image classification tasks, each image is labeled with a specific category or class.
* Depth Annotations: These annotations provide the depth information for objects in the scene, complementing color images.
* Temporal Annotations (for videos):Annotations for videos may include object trajectories, action labels, or event timestamps.

**Challenges in Data Annotation:**

* Subjectivity and Ambiguity: Interpreting images can be subjective, leading to potential discrepancies in annotations, especially for complex scenes.
* Scale and Volume: Annotating large datasets can be time-consuming and labor-intensive, necessitating efficient annotation tools and workflows.
* Consistency and Accuracy: Ensuring consistent and accurate annotations across a dataset is crucial for training reliable models.
* Specialized Tools: Complex annotations may require specialized tools or software designed for specific tasks.

**4.2 Tool For Data Annotation (** [**roboflow.com**](https://roboflow.com/) **) :**

We will be using [Roboflow.com](https://roboflow.com/) for data collection, annotation , preprocessing in this project

Roboflow is an online platform that provides a suite of tools and functionalities for data annotation and preprocessing, particularly in the field of computer vision. It is designed to streamline the process of preparing and augmenting visual data for machine learning projects. Here's an overview of Roboflow:

**Key Features:**

1. Annotation Tools: Roboflow offers a variety of annotation tools for different tasks, including bounding boxes, polygonal segmentation, key point detection, and image classification.

2. Import and Export Options: It supports various file formats for importing and exporting annotated data. This includes popular formats like COCO JSON, Pascal VOC XML, YOLO TXT, and more.

3. Data Augmentation: Roboflow provides tools for augmenting images, enabling users to generate additional training data by applying transformations like rotations, flips, brightness adjustments, and more.

4. Collaboration and Teamwork: The platform allows multiple users to collaborate on annotation projects, making it suitable for team-based workflows.

5. Quality Control and Review: It includes features for reviewing annotations, ensuring accuracy and consistency in the labeling process.

6. Integration with Machine Learning Frameworks: Roboflow offers integrations with popular machine learning frameworks like TensorFlow, PyTorch, and others, simplifying the process of training models with annotated data.

7. Version Control:It provides version control features, allowing users to track changes and revert to previous versions of their projects.

**Use Cases:**

1. Object Detection: Roboflow is particularly well-suited for tasks like object detection, where bounding boxes or other annotations are required.

2. Image Segmentation: It supports polygonal segmentation, making it suitable for tasks like instance segmentation.

4. Image Classification : Roboflow provides tools for annotating images with class labels, making it suitable for image classification tasks.

Note :- We will be using opensource annotated datasets for face mask detection having three classes “Proper Mask”, “No Mask” and “Improper Mask”.

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

**4.3 Data Exploration and Preprocessing in Computer Vision :**

**Data Exploration in Computer Vision:**

Data exploration in computer vision involves examining and understanding the characteristics, properties, and content of the visual data that will be used to train and test machine learning models. This process is crucial for gaining insights into the dataset, identifying potential challenges, and making informed decisions about how to preprocess the data.

Key Steps in Data Exploration:

1. Data Visualization: Generate visual representations of the images or videos in the dataset to understand their content, diversity, and complexity. This can include viewing sample images, exploring histograms, and visualizing labels or annotations.

2.Statistical Analysis: Conduct basic statistical analyses to gather information about the distribution of image properties such as size, aspect ratio, color channels, and pixel intensity values.

3. Class Distribution: Examine the distribution of classes or categories in the dataset. This helps identify potential class imbalances, which can impact model performance.

4. Anomaly Detection: Look for anomalies or outliers in the dataset that may need special handling during preprocessing. Anomalies could be caused by incorrect labels or noisy data.

5. Data Quality Assessment: Check for issues like corrupted images, missing labels, or poorly annotated data. Identifying and addressing these issues early on can prevent problems during model training.

**Data Preprocessing in Computer Vision:**

Data preprocessing involves a series of steps taken to prepare the visual data for training a machine learning model. It aims to enhance the quality and relevance of the data, making it more suitable for the specific computer vision task at hand.

Common Data Preprocessing Techniques:

1. Resizing and Standardization: Ensure that all images have a consistent size and format. This is important for compatibility with the chosen model architecture.

2. Normalization and Scaling: Adjust pixel values to a standardized range (e.g., [0, 1] or [-1, 1]) to improve convergence during training.

3. Data Augmentation: Apply transformations (e.g., rotations, flips, zooms) to generate additional training samples, increasing the diversity of the dataset.

4. Noise Reduction: Apply filters or techniques to reduce noise in images, which can improve model accuracy.

5. Label Encoding: Convert class labels into a numerical format that can be used by the machine learning algorithm.

6. Handling Missing Data: Address any missing annotations or labels in the dataset through imputation or removal.

7. Data Balancing: If there is class imbalance, techniques like oversampling, under sampling, or using class weights can be applied.

8. Feature Extraction (Optional): For specific tasks, extract relevant features from the images using techniques like edge detection, texture analysis, or deep feature extraction.

9. Dimensionality Reduction (Optional): Use techniques like PCA to reduce the complexity of high-dimensional image data while preserving meaningful information.

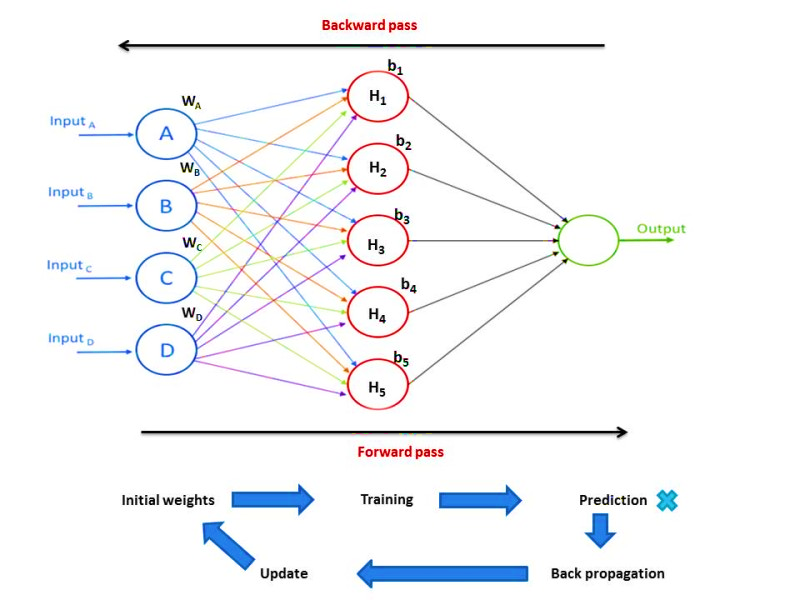
**4.4 Deep Learning :**

Deep learning is a subfield of machine learning that focuses on training highly complex neural networks to perform tasks without being explicitly programmed. It is inspired by the structure and functioning of the human brain, particularly the interconnected neurons that process information.

In deep learning, the term "deep" refers to the presence of multiple layers in a neural network. These layers allow the network to learn hierarchical representations of data. Each layer in the network processes the data in a progressively abstract manner, enabling it to understand increasingly complex features.

**Key Concepts and Components:**

1. Neural Networks: Neural networks are the foundation of deep learning. They are composed of artificial neurons (nodes) arranged in layers. These neurons are connected with weighted edges, and they process information and pass it along to the next layer.
2. Deep Neural Networks: Deep learning involves training networks with multiple hidden layers, which is what distinguishes it from traditional machine learning. The depth of the network allows it to learn intricate patterns and relationships in the data.
3. Activation Functions: Neurons in a neural network apply activation functions to the weighted sum of their inputs. These functions introduce non-linearity, allowing the network to model complex relationships.
4. Backpropagation: Backpropagation is a fundamental training algorithm in deep learning. It involves propagating errors backward through the network to adjust the weights and biases. This process allows the network to learn and improve its predictions over time.
5. Loss Functions: Loss functions measure the discrepancy between the predicted outputs of the network and the actual targets. The goal is to minimize this discrepancy during training.
6. Optimization Algorithms: Optimization algorithms, such as Gradient Descent, are used to adjust the parameters of the network (weights and biases) based on the calculated gradients of the loss function. This guides the network towards better performance.



**4.5 Convolutional Neural Networks :**

Convolutional Neural Networks (CNNs) are a class of deep neural networks specifically designed for processing grid-like data, such as images and videos. They are particularly powerful in computer vision tasks due to their ability to automatically and adaptively learn spatial hierarchies of features from the data. Here's how CNNs work:

1. Convolution: The core operation in CNNs is the convolution operation. It involves sliding a small window called a kernel (also referred to as a filter) across the input image. At each position, the kernel multiplies its values with the pixel values in the corresponding region of the image and sums the results to produce a single output value. This process is repeated for every position in the image.

2. Feature Maps: The output of the convolution operation is referred to as a feature map or activation map. Each feature map represents the presence of a particular feature or pattern in the input. Multiple feature maps are generated by using different kernels, each designed to detect different types of features.

3. Non-Linearity (ReLU): After the convolution operation, a non-linear activation function is applied element-wise to the feature map. The Rectified Linear Unit (ReLU) is a common choice, as it introduces non-linearity and helps the network learn complex patterns.

4. Pooling (Downsampling): Pooling layers reduce the spatial dimensions of the feature maps while retaining important information. Common pooling operations include max pooling (selecting the maximum value in a region) and average pooling (calculating the average value).

5. Stacking Layers: Multiple convolutional layers, followed by activation and pooling layers, are stacked on top of each other. This creates a hierarchy of increasingly abstract features, allowing the network to learn complex patterns.

6. Fully Connected Layers: After several convolutional layers, fully connected layers are introduced. These layers perform high-level reasoning and decision-making based on the features learned by the previous layers.

7. Output Layer: The final layer of the CNN depends on the specific task. For tasks like image classification, it typically consists of a softmax activation function to produce class probabilities.

**Working of a CNN in Image Classification:**

1. Input Image: The raw input image is fed into the network.

2. Convolutional Layers: Convolutional layers apply multiple filters to the input image to detect various features like edges, textures, and shapes.

3. ReLU Activation: The ReLU activation function introduces non-linearity, helping the network learn more complex patterns.

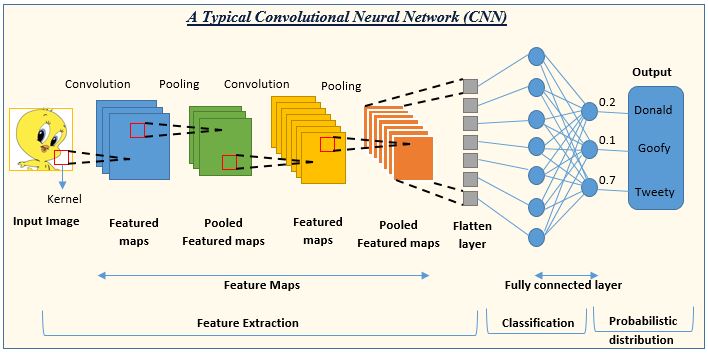
4. Pooling Layers: Pooling layers reduce the spatial dimensions of the feature maps while retaining important information.

5. Fully Connected Layers: The feature maps from the convolutional layers are flattened and passed through fully connected layers for high-level reasoning.

6. Output Layer: The final layer produces class probabilities using an appropriate activation function (e.g., softmax for classification).

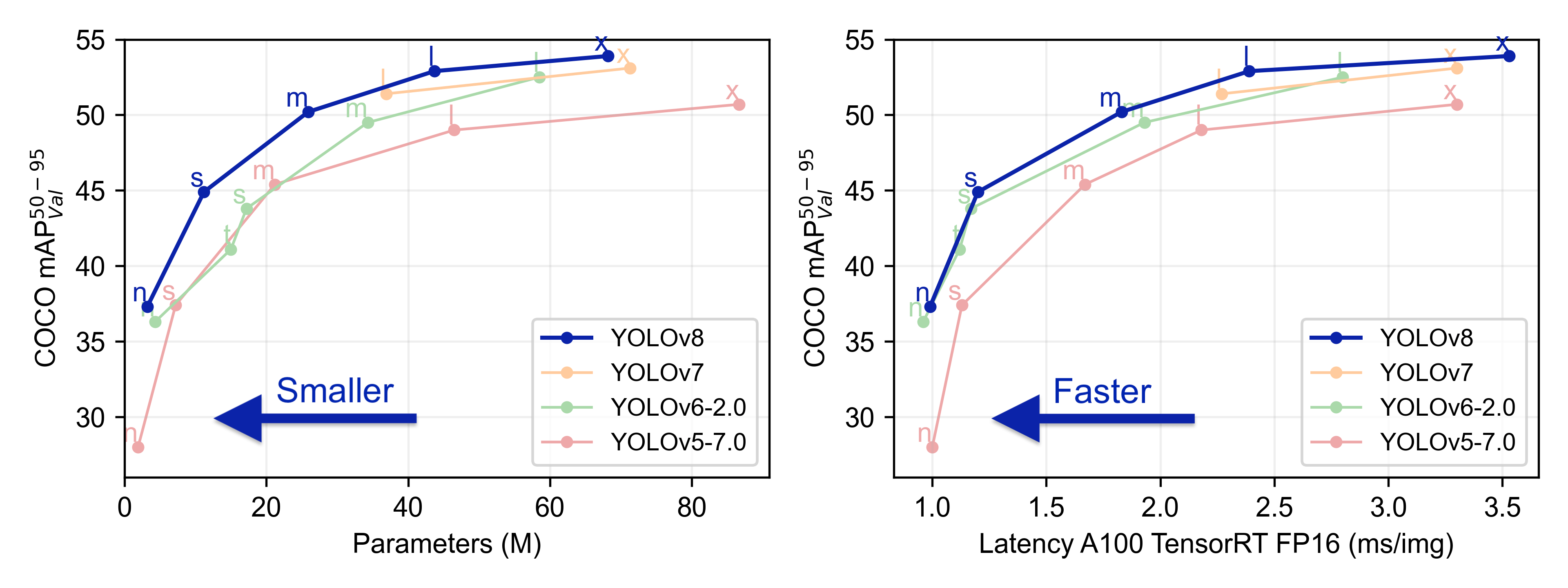
7. Training and Optimization: The network is trained using backpropagation and an optimization algorithm to minimize the loss function.

8. Prediction: After training, the CNN can make predictions on new, unseen images.



**4.6 YOLOv8 (Object Detection Algorithm) :**

* 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>

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.

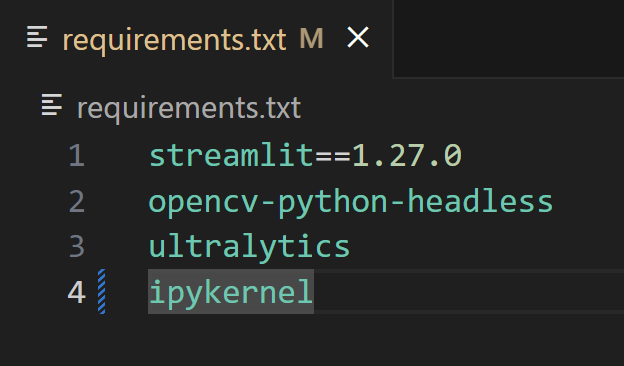
Create conda environment

    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



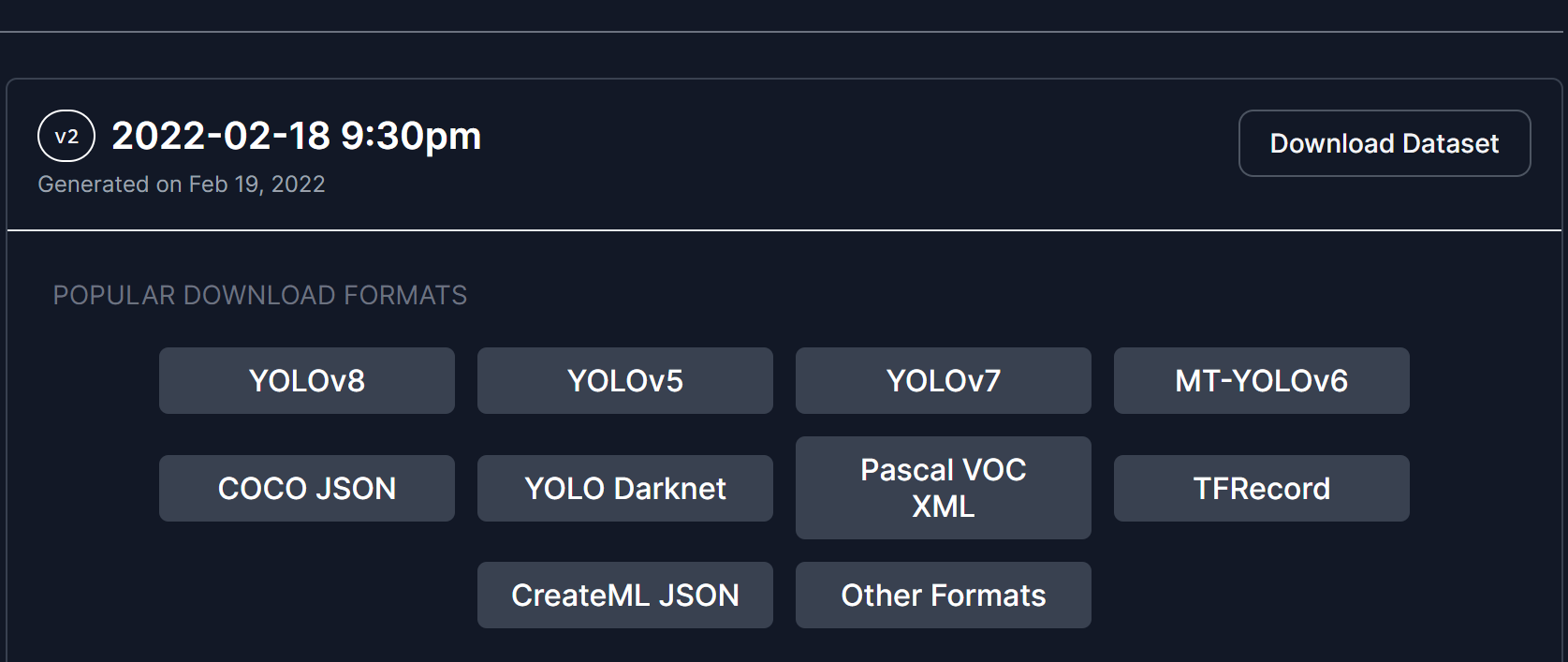
pip install -r requirements.txt

**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>

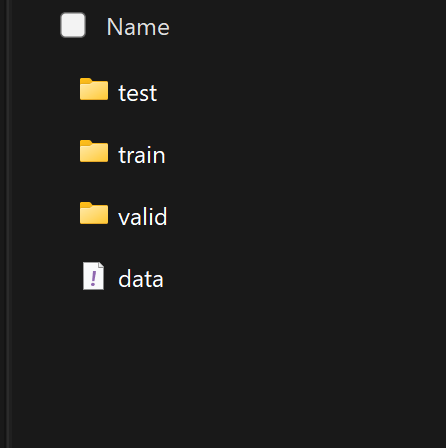
* Clink on button Download Dataset button



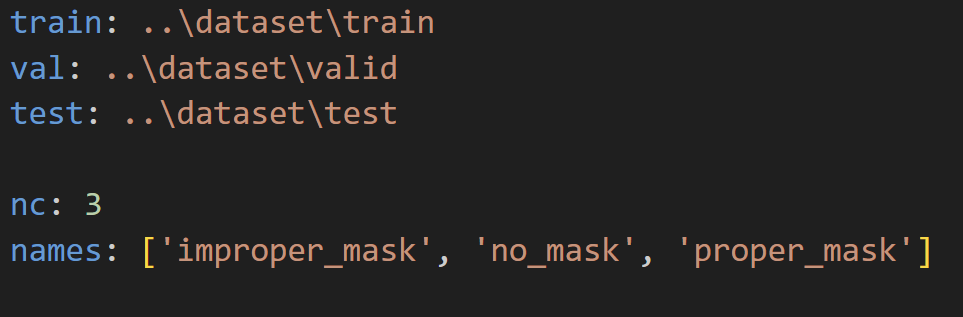
* Select export format YOLOv8 and select download ZIP to computer



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



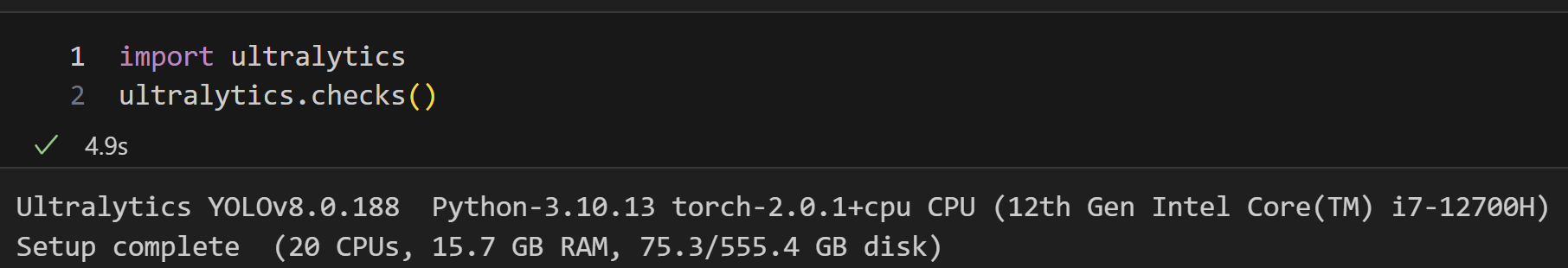
* Configure data.yaml file inside dataset folder. Enter correct paths for train, valid and test sets. data.yaml file will look like this:



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

import ultralytics

ultralytics.checks()

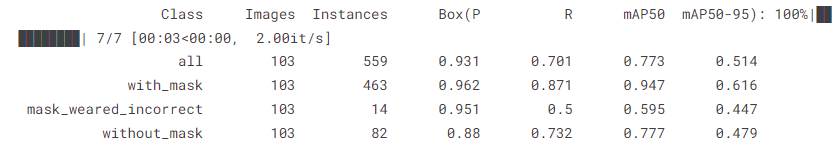


from ultralytics import YOLO

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

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

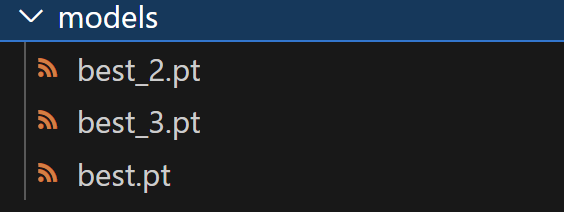
* Above code will start model training on out custom dataset on pretrained YOLOv8n.pt model. After completion metrics will generated like below:



* 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:

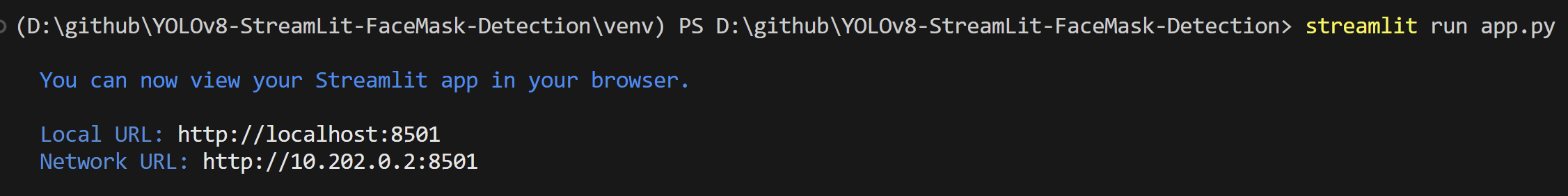
        main()

    except Exception as e:

        raise Exception(e)

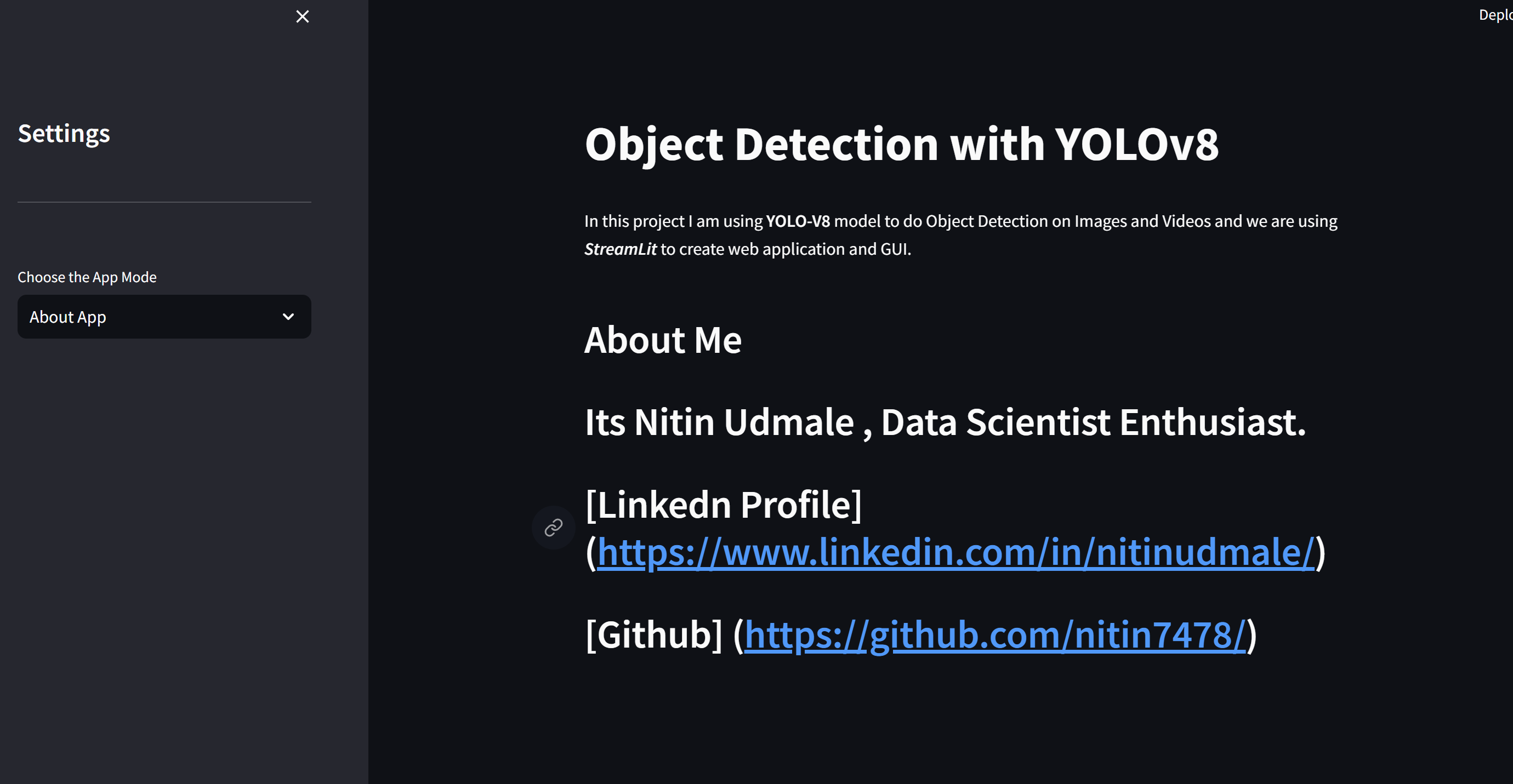
1. **Result**

**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 :**



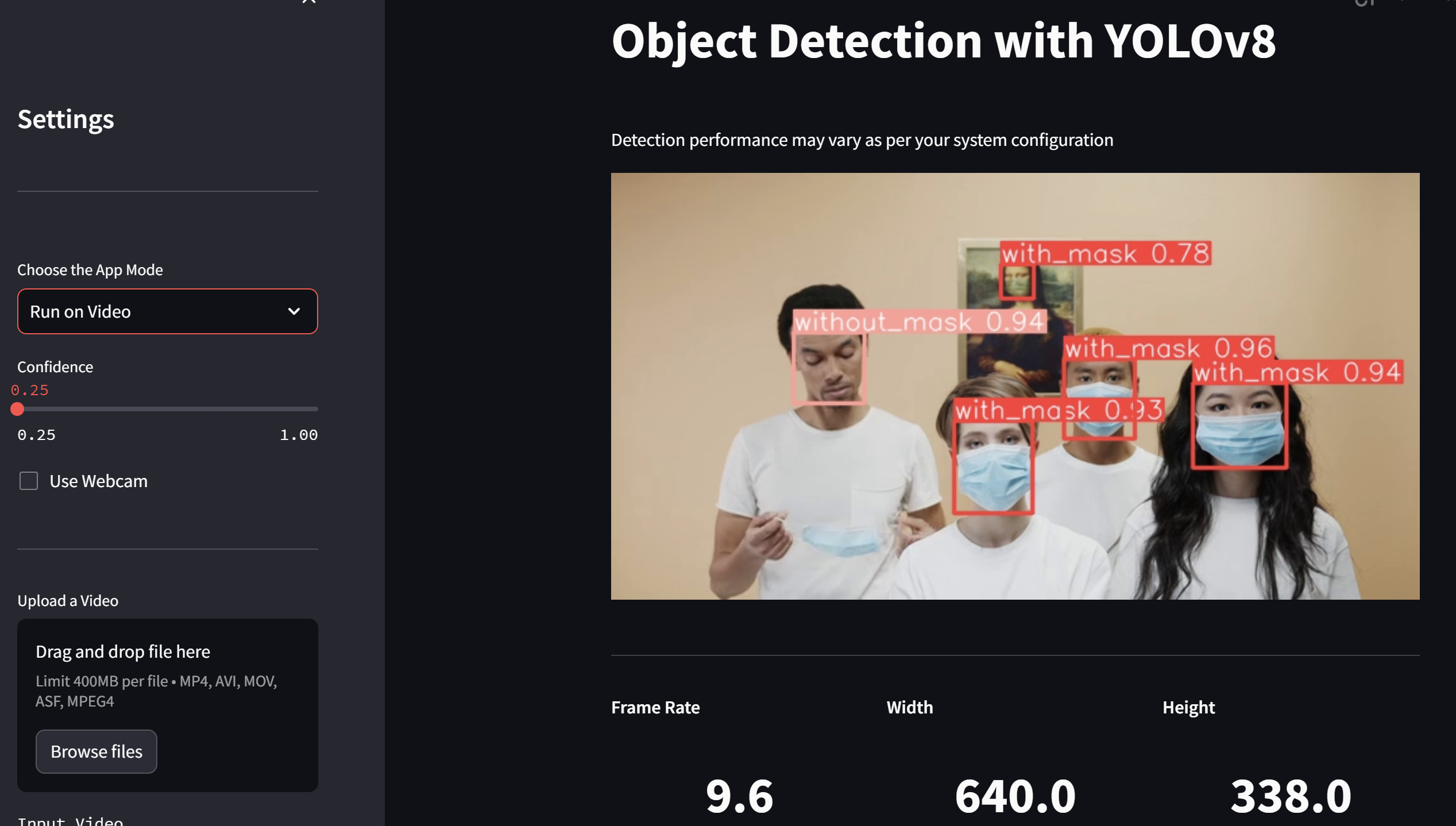
**Run on Image section :**



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 :**



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)