ConstructGuard_ YOLO-Based Safety Gear Surveillance

Introduction:

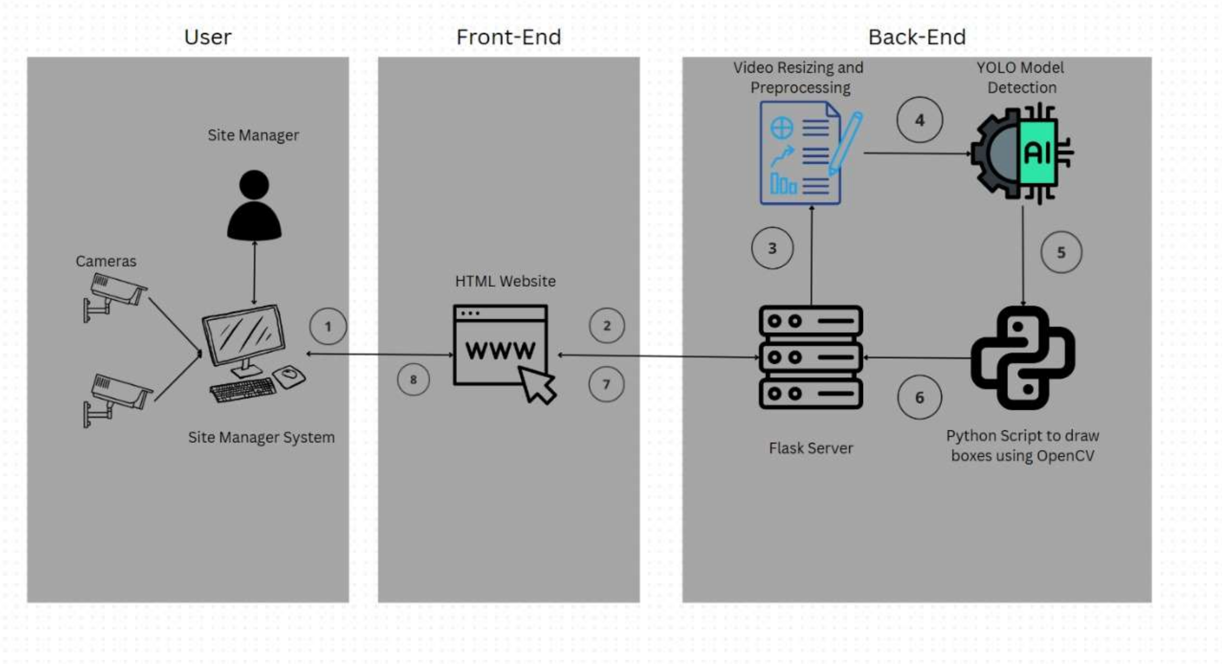
"ConstructGuard: YOLO-Based Safety Gear Surveillance" is an innovative application of computer vision and artificial intelligence aimed at enhancing safety and security on construction sites. This advanced system utilizes YOLO (You Only Look Once), a state-of-the-art object detection algorithm, to accurately identify and verify the presence of essential safety gear worn by construction workers.

With this technology in place, the system can swiftly and accurately detect safety gear such as hard hats, reflective vests, safety goggles, gloves, and more in real-time. It operates seamlessly, even in challenging environmental conditions, ensuring that every worker is properly equipped for the job.

YOLOv8, the latest version of the YOLO (You Only Look Once) model series, is a popular set of object detection models used for real-time object detection and classification in computer vision.

The key feature of YOLOv8 is its single-stage detection approach, which is designed to detect objects in real time and with high accuracy.YOLO processes the entire image in a single pass, making it faster and more efficiently.Processing images with YOLOv8 is simple and straightforward. The system resizes the input image to 448 × 448, runs a single convolutional network on the image, and thresholds the resulting detections by the model's confidence1

Technical Architecture:



Pre-requisites

To complete this project, you must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, Spyder, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

1. To build Machine learning models you must require the following packages

- OpenCv :OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products
- Ultralytics: Ultralytics is a company that creates artificial intelligence models. They offer cutting-edge solutions for a wide range of AI tasks, including detection, segmentation, classification, tracking, and pose estimation
- Flask: Web framework used for building Web applications
- Python packages:
  - ✓ open cmd prompt as administrator
  - ✓ Type pip install ultralytics and click enter.
  - ✓ Type “pip install opencv and click enter.
  - ✓ Type “pip install scikit-learn” and click enter.
  - ✓ Type “pip install flask” and click enter.

Deep Learning Concepts

- Object Detection :Object detection is a computer vision technique that identifies and classifies a particular object in a particular setting1. The main goal of object detection is to scan digital images or real-life scenarios to locate instances of every object, separate them, and analyze their necessary features for real-time predictions1

2. **YoloV8**: YOLOv8 is the latest version of the YOLO (You Only Look Once) algorithm, developed by Ultralytics1. It is a state-of-the-art model that can be used for object detection, image classification, and instance segmentation tasks**.**https://www.youtube.com/watch?v=ag3DLKsl2vk

3. **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications

Flask Basics

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives:

By the end of this project, you will:

4. Know fundamental concepts and techniques of Convolutional Neural Network.
5. Gain a broad understanding of image data.
6. Know how to pre-process/clean the data using different data pre-processing techniques.
7. know how to build a web application using the Flask framework.

Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.

To accomplish this, we have to complete all the activities and tasks listed below

 Data Collection.

✓ Create Train and Test Folders.

✓ Create data.yaml file

 Training and testing the model

✓ Save the Model

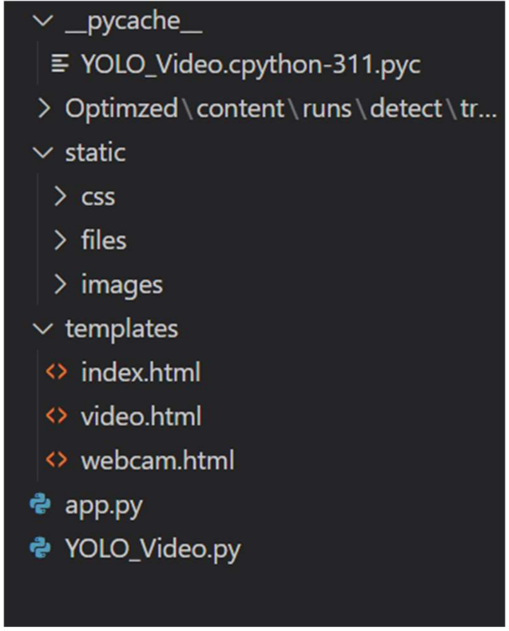
✓ Application Building

✓ Create an HTML file

✓ Build Python Code

Project Structure:

Create a Project folder which contains files as shown below



```
⌄ __pycache__
    YOLO_Video.cpython-311.pyc
› Optimzed\content\runs\detect\tr...
⌄ static
    › css
    › files
    › images
⌄ templates
    index.html
    video.html
    webcam.html
  app.py
  YOLO_Video.py
```

- The Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting
- We need the model which is saved and the saved model in this content is a seanaimal.h5 templates folder containing base.html, index.html pages.

Milestone 1: Collection of Data

Dataset has 3 classes of plastic :

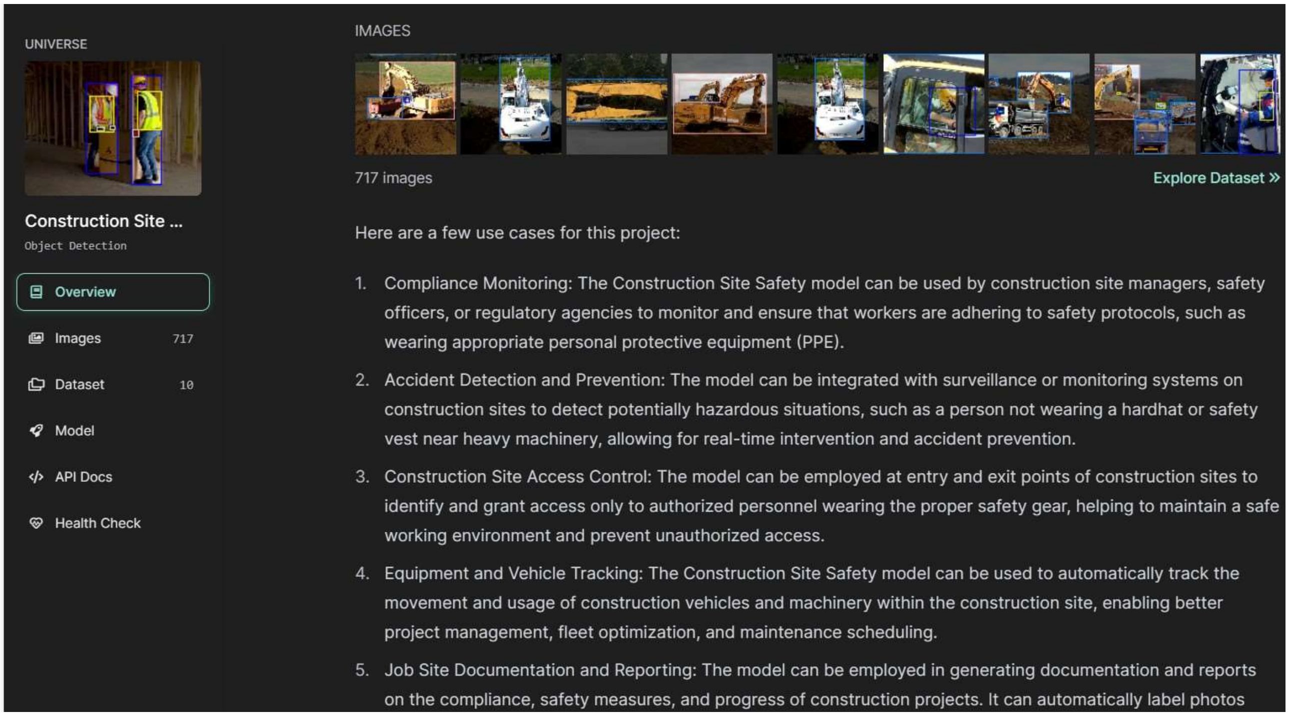
The given dataset has 3 different types of plastics they are following:

- Plastic Bottles

- Plastic Food packaging
- Plastic Bag

Download the Dataset-

[Construction Site Safety Dataset > Overview (roboflow.com)](roboflow.com)



Here are a few use cases for this project:

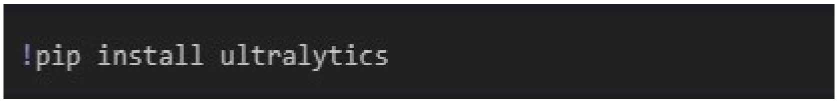
1. Compliance Monitoring: The Construction Site Safety model can be used by construction site managers, safety officers, or regulatory agencies to monitor and ensure that workers are adhering to safety protocols, such as wearing appropriate personal protective equipment (PPE).
2. Accident Detection and Prevention: The model can be integrated with surveillance or monitoring systems on construction sites to detect potentially hazardous situations, such as a person not wearing a hardhat or safety vest near heavy machinery, allowing for real-time intervention and accident prevention.
3. Construction Site Access Control: The model can be employed at entry and exit points of construction sites to identify and grant access only to authorized personnel wearing the proper safety gear, helping to maintain a safe working environment and prevent unauthorized access.
4. Equipment and Vehicle Tracking: The Construction Site Safety model can be used to automatically track the movement and usage of construction vehicles and machinery within the construction site, enabling better project management, fleet optimization, and maintenance scheduling.
5. Job Site Documentation and Reporting: The model can be employed in generating documentation and reports on the compliance, safety measures, and progress of construction projects. It can automatically label photos

Milestone 2: Image Pre-processing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

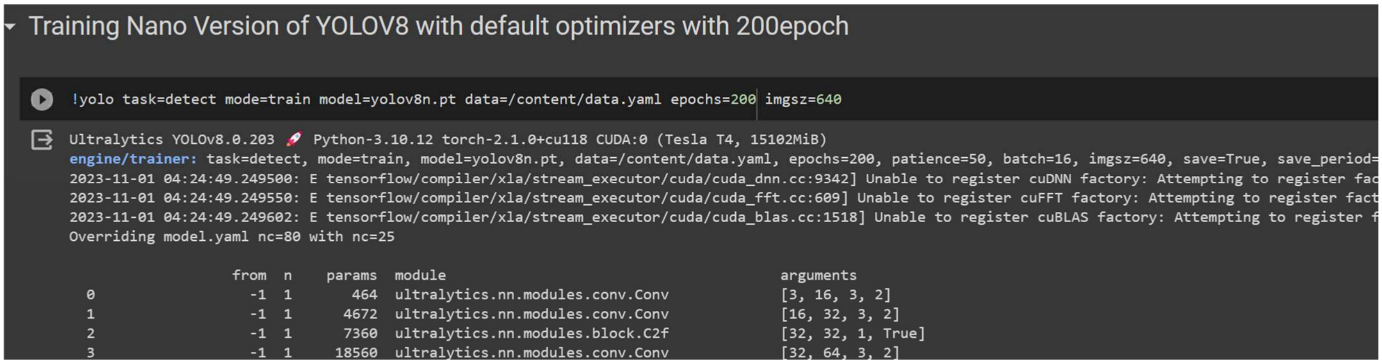
Activity 1: Import the required libraries

We import and install ultralytics



```
!pip install ultralytics
```

Activity 2: Load pre-trained model with OpenCV



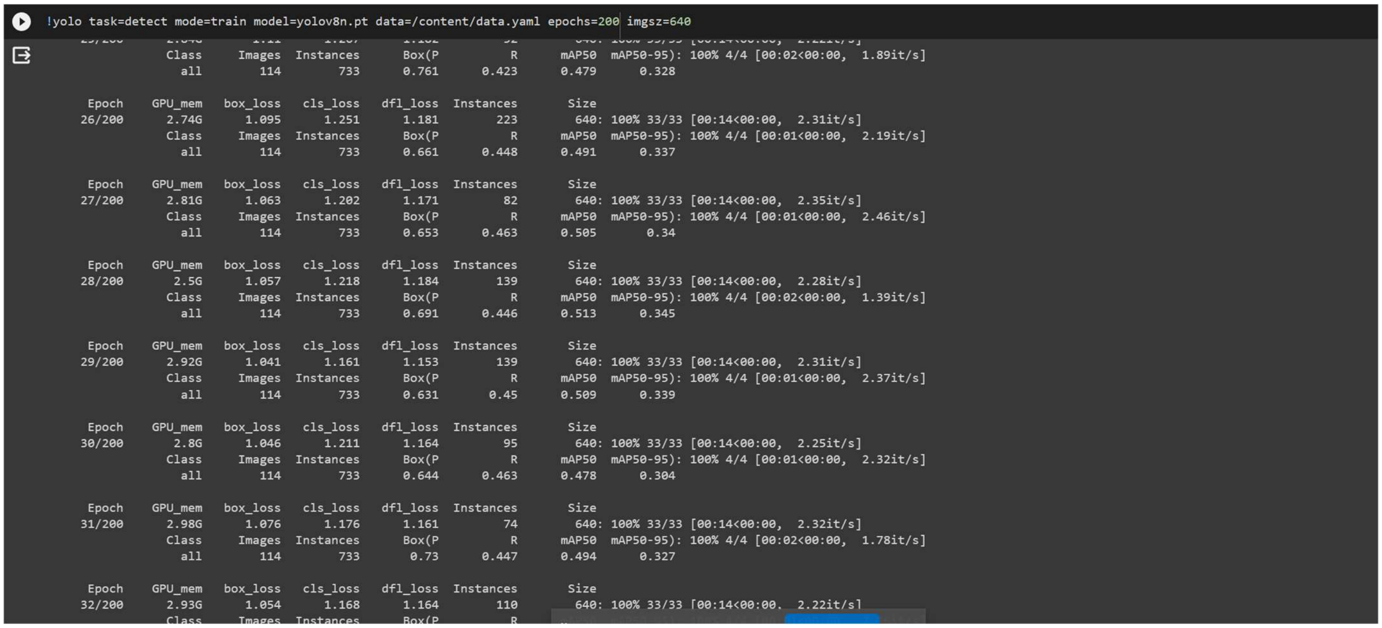
Training Nano Version of YOLOV8 with default optimizers with 200epoch

```
!yolo task=detect mode=train model=yolov8n.pt data=/content/data.yaml epochs=200 imgsz=640
```

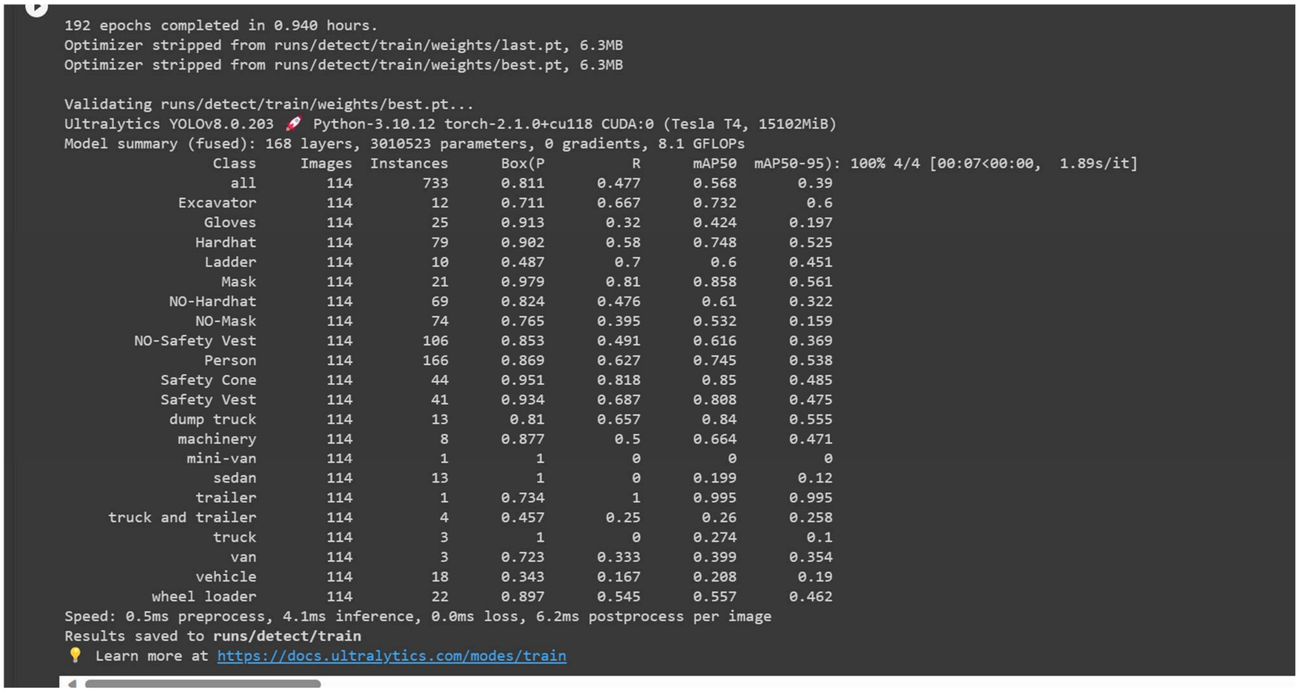
```
Ultralytics YOLOv8.0.203 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8n.pt, data=/content/data.yaml, epochs=200, patience=50, batch=16, imgsz=640, save=True, save_period=
2023-11-01 04:24:49.249500: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to register cuDNN factory: Attempting to register fac
2023-11-01 04:24:49.249550: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register cuFFT factory: Attempting to register fact
2023-11-01 04:24:49.249602: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to register cuBLAS factory: Attempting to register f
Overriding model.yaml nc=80 with nc=25

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
```

Milestone 3: training

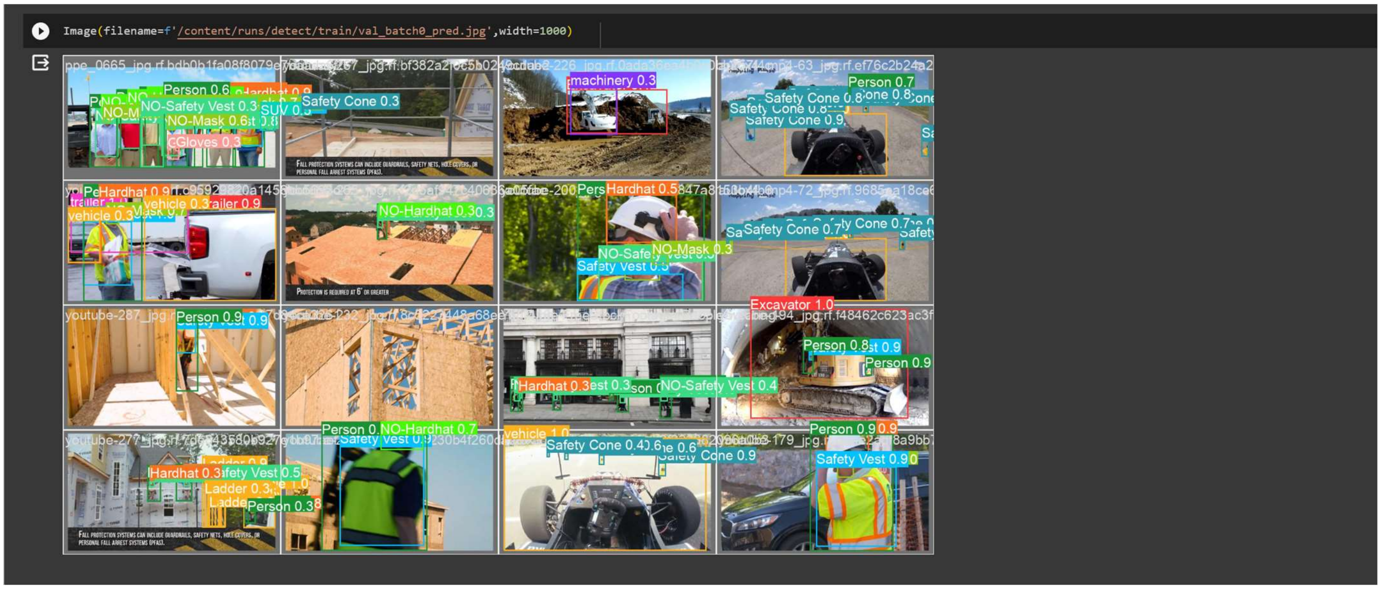
Now it's time to train our yolo model :



```
!yolo task=detect mode=train model=yolov8n.pt data=/content/data.yaml epochs=200 imgsz=640
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.89it/s]
                   all        114        733      0.761      0.423      0.479      0.328

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     26/200      2.74G      1.095      1.251      1.181        223        640: 100% 33/33 [00:14<00:00,  2.31it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:01<00:00,  2.19it/s]
                   all        114        733      0.661      0.448      0.491      0.337

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     27/200      2.81G      1.063      1.202      1.171         82        640: 100% 33/33 [00:14<00:00,  2.35it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:01<00:00,  2.46it/s]
                   all        114        733      0.653      0.463      0.505       0.34

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     28/200       2.5G      1.057      1.218      1.184        139        640: 100% 33/33 [00:14<00:00,  2.28it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.39it/s]
                   all        114        733      0.691      0.446      0.513      0.345

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     29/200      2.92G      1.041      1.161      1.153        139        640: 100% 33/33 [00:14<00:00,  2.31it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:01<00:00,  2.37it/s]
                   all        114        733      0.631       0.45      0.509      0.339

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     30/200       2.8G      1.046      1.211      1.164         95        640: 100% 33/33 [00:14<00:00,  2.25it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:01<00:00,  2.32it/s]
                   all        114        733      0.644      0.463      0.478      0.304

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     31/200      2.98G      1.076      1.176      1.161         74        640: 100% 33/33 [00:14<00:00,  2.32it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.78it/s]
                   all        114        733       0.73      0.447      0.494      0.327

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     32/200      2.93G      1.054      1.168      1.164        110        640: 100% 33/33 [00:14<00:00,  2.22it/s]
                 Class     Images  Instances      Box(P          R
```



```
192 epochs completed in 0.940 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.3MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.3MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.203 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3010523 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:07<00:00,  1.89s/it]
                   all        114        733      0.811      0.477      0.568       0.39
             Excavator        114         12      0.711      0.667      0.732        0.6
                Gloves        114         25      0.913       0.32      0.424      0.197
               Hardhat        114         79      0.902       0.58      0.748      0.525
                Ladder        114         10      0.487        0.7        0.6      0.451
                  Mask        114         21      0.979       0.81      0.858      0.561
            NO-Hardhat        114         69      0.824      0.476       0.61      0.322
               NO-Mask        114         74      0.765      0.395      0.532      0.159
        NO-Safety Vest        114        106      0.853      0.491      0.616      0.369
                Person        114        166      0.869      0.627      0.745      0.538
           Safety Cone        114         44      0.951      0.818       0.85      0.485
           Safety Vest        114         41      0.934      0.687      0.808      0.475
            dump truck        114         13       0.81      0.657       0.84      0.555
             machinery        114          8      0.877        0.5      0.664      0.471
              mini-van        114          1          1          0          0          0
                 sedan        114         13          1          0      0.199       0.12
               trailer        114          1      0.734          1      0.995      0.995
     truck and trailer        114          4      0.457       0.25       0.26      0.258
                 truck        114          3          1          0      0.274        0.1
                   van        114          3      0.723      0.333      0.399      0.354
               vehicle        114         18      0.343      0.167      0.208       0.19
          wheel loader        114         22      0.897      0.545      0.557      0.462
Speed: 0.5ms preprocess, 4.1ms inference, 0.0ms loss, 6.2ms postprocess per image
Results saved to runs/detect/train
💡 Learn more at https://docs.ultralytics.com/modes/train
```



```
Image(filename=f'/content/runs/detect/train/val_batch0_pred.jpg',width=1000)
```

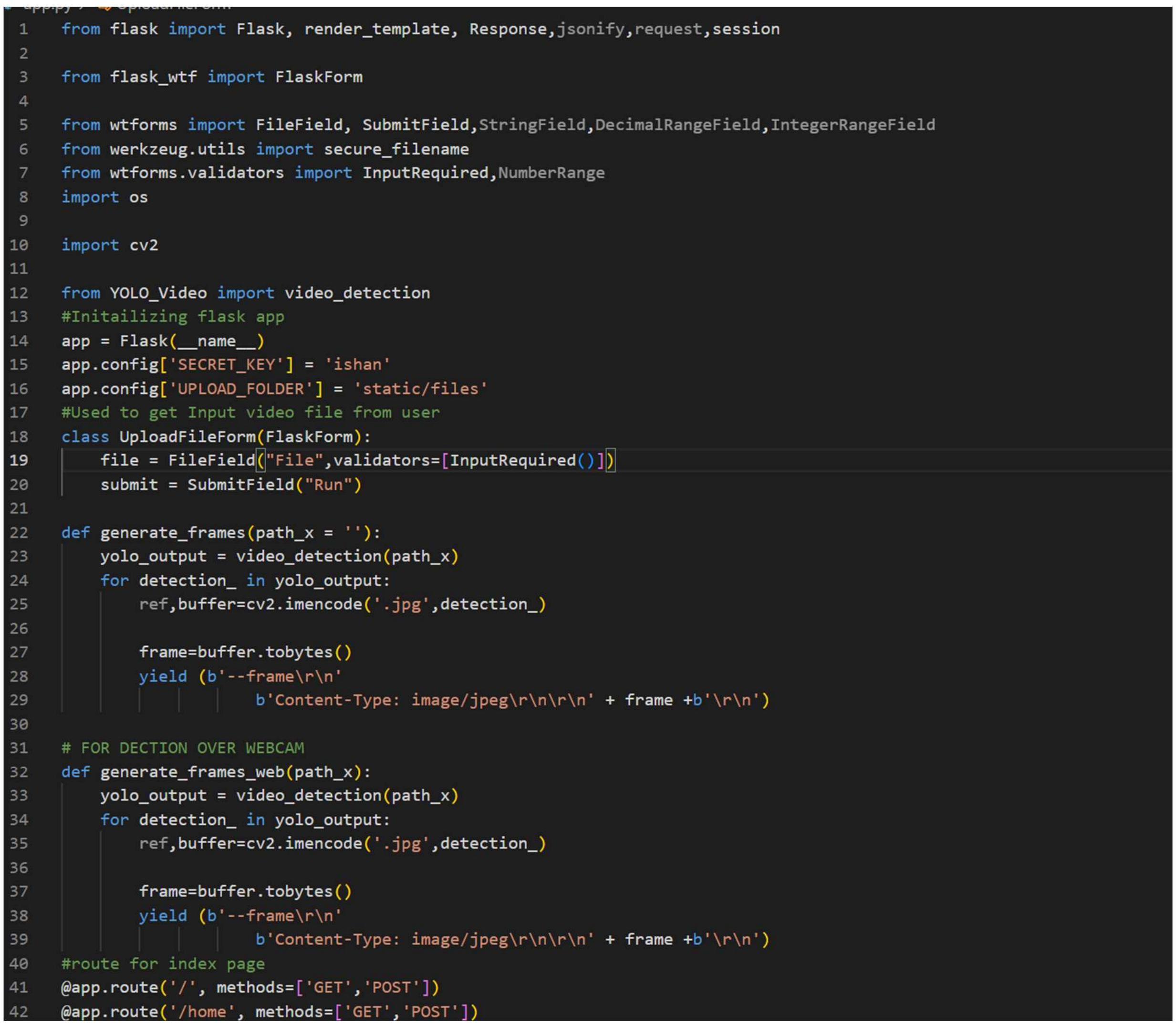
Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

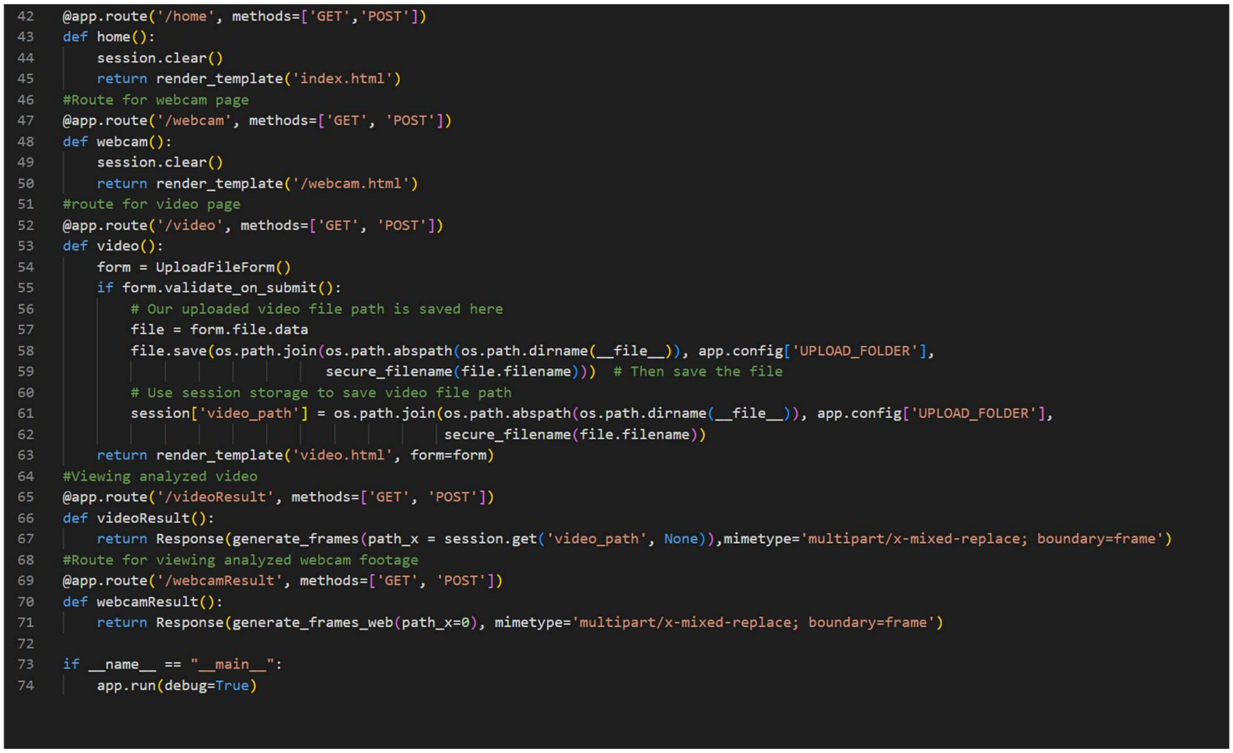
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Create app.py (Python Flask) file: -

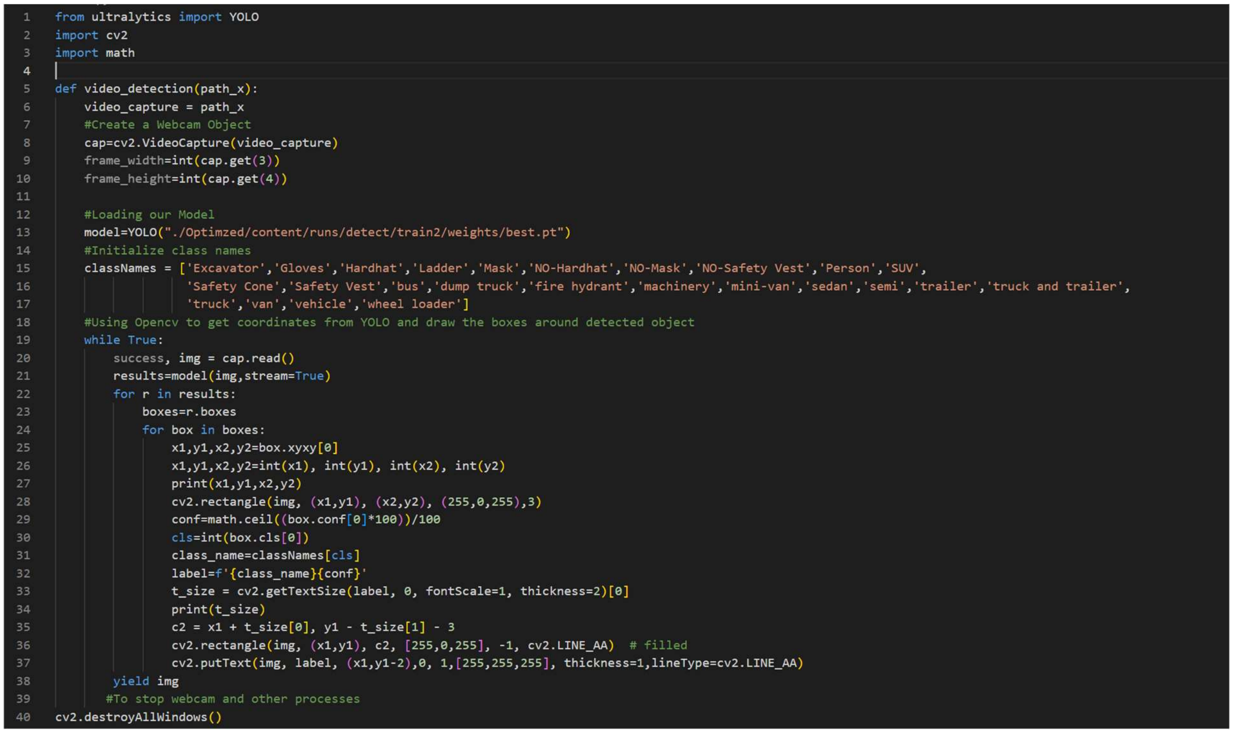
Write below code in Flask app.py python file script to run Object detection Project.



```python
from flask import Flask, render_template, Response,jsonify,request,session

from flask_wtf import FlaskForm

from wtforms import FileField, SubmitField,StringField,DecimalRangeField,IntegerRangeField
from werkzeug.utils import secure_filename
from wtforms.validators import InputRequired,NumberRange
import os

import cv2

from YOLO_Video import video_detection
#Initailizing flask app
app = Flask(__name__)
app.config['SECRET_KEY'] = 'ishan'
app.config['UPLOAD_FOLDER'] = 'static/files'
#Used to get Input video file from user
class UploadFileForm(FlaskForm):
    file = FileField("File",validators=[InputRequired()])
    submit = SubmitField("Run")

def generate_frames(path_x = ''):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref,buffer=cv2.imencode('.jpg',detection_)

        frame=buffer.tobytes()
        yield (b'--frame\r\n'
                    b'Content-Type: image/jpeg\r\n\r\n' + frame +b'\r\n')

# FOR DECTION OVER WEBCAM
def generate_frames_web(path_x):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref,buffer=cv2.imencode('.jpg',detection_)

        frame=buffer.tobytes()
        yield (b'--frame\r\n'
                    b'Content-Type: image/jpeg\r\n\r\n' + frame +b'\r\n')
#route for index page
@app.route('/', methods=['GET','POST'])
@app.route('/home', methods=['GET','POST'])
```

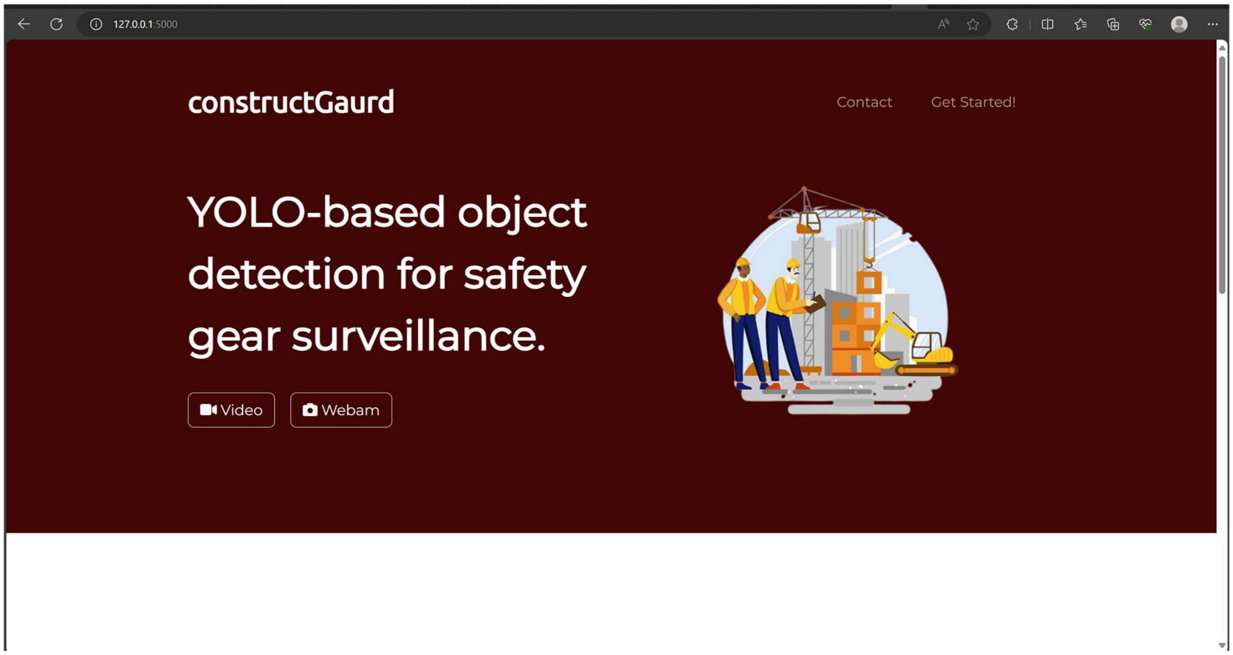


```python
@app.route('/home', methods=['GET','POST'])
def home():
    session.clear()
    return render_template('index.html')
#Route for webcam page
@app.route('/webcam', methods=['GET', 'POST'])
def webcam():
    session.clear()
    return render_template('/webcam.html')
#route for video page
@app.route('/video', methods=['GET', 'POST'])
def video():
    form = UploadFileForm()
    if form.validate_on_submit():
        # Our uploaded video file path is saved here
        file = form.file.data
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'],
                               secure_filename(file.filename)))  # Then save the file
        # Use session storage to save video file path
        session['video_path'] = os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'],
                                             secure_filename(file.filename))
    return render_template('video.html', form=form)
#Viewing analyzed video
@app.route('/videoResult', methods=['GET', 'POST'])
def videoResult():
    return Response(generate_frames(path_x = session.get('video_path', None)),mimetype='multipart/x-mixed-replace; boundary=frame')
#Route for viewing analyzed webcam footage
@app.route('/webcamResult', methods=['GET', 'POST'])
def webcamResult():
    return Response(generate_frames_web(path_x=0), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == "__main__":
    app.run(debug=True)
```



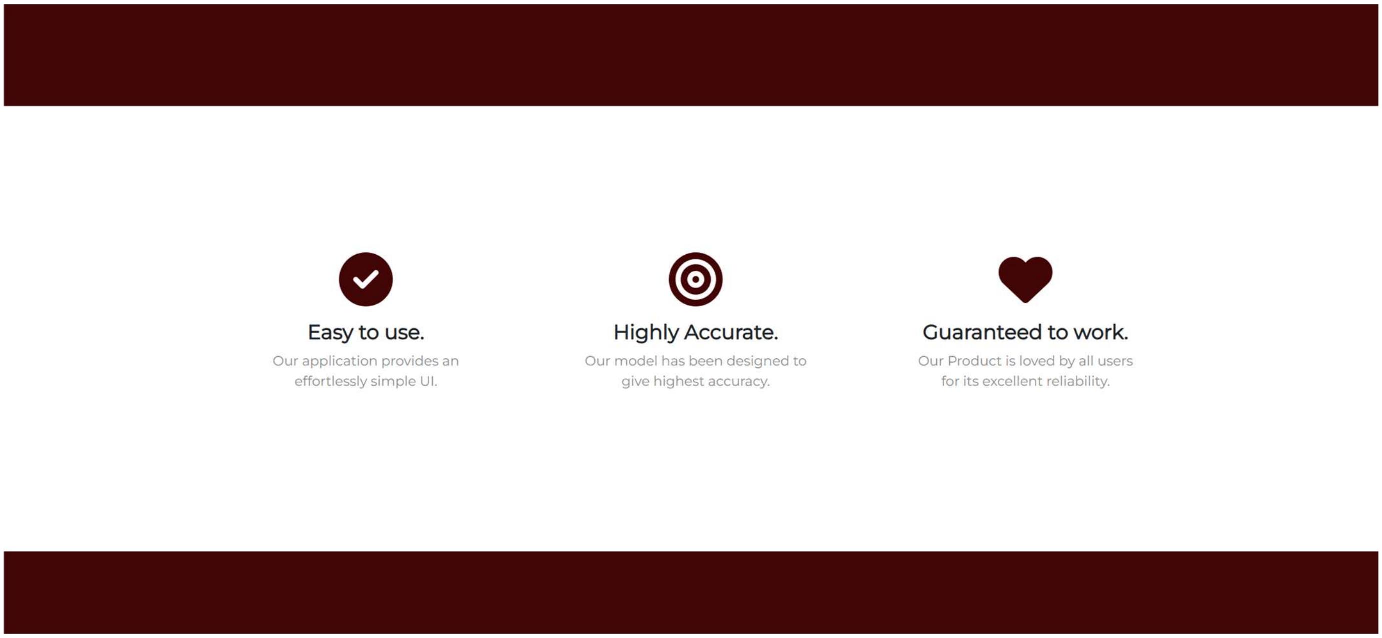
```python
from ultralytics import YOLO
import cv2
import math

def video_detection(path_x):
    video_capture = path_x
    #Create a Webcam Object
    cap=cv2.VideoCapture(video_capture)
    frame_width=int(cap.get(3))
    frame_height=int(cap.get(4))

    #Loading our Model
    model=YOLO("./Optimzed/content/runs/detect/train2/weights/best.pt")
    #Initialize class names
    classNames = ['Excavator','Gloves','Hardhat','Ladder','Mask','NO-Hardhat','NO-Mask','NO-Safety Vest','Person','SUV',
                  'Safety Cone','Safety Vest','bus','dump truck','fire hydrant','machinery','mini-van','sedan','semi','trailer','truck and trailer',
                  'truck','van','vehicle','wheel loader']
    #Using Opencv to get coordinates from YOLO and draw the boxes around detected object
    while True:
        success, img = cap.read()
        results=model(img,stream=True)
        for r in results:
            boxes=r.boxes
            for box in boxes:
                x1,y1,x2,y2=box.xyxy[0]
                x1,y1,x2,y2=int(x1), int(y1), int(x2), int(y2)
                print(x1,y1,x2,y2)
                cv2.rectangle(img, (x1,y1), (x2,y2), (255,0,255),3)
                conf=math.ceil((box.conf[0]*100))/100
                cls=int(box.cls[0])
                class_name=classNames[cls]
                label=f'{class_name}{conf}'
                t_size = cv2.getTextSize(label, 0, fontScale=1, thickness=2)[0]
                print(t_size)
                c2 = x1 + t_size[0], y1 - t_size[1] - 3
                cv2.rectangle(img, (x1,y1), c2, [255,0,255], -1, cv2.LINE_AA)  # filled
                cv2.putText(img, label, (x1,y1-2),0, 1,[255,255,255], thickness=1,lineType=cv2.LINE_AA)
        yield img
       #To stop webcam and other processes
cv2.destroyAllWindows()
```

Index.html is displayed below:



YOLO-based object detection for safety gear surveillance.

about Section is displayed below:


Easy to use.

Our application provides an effortlessly simple UI.

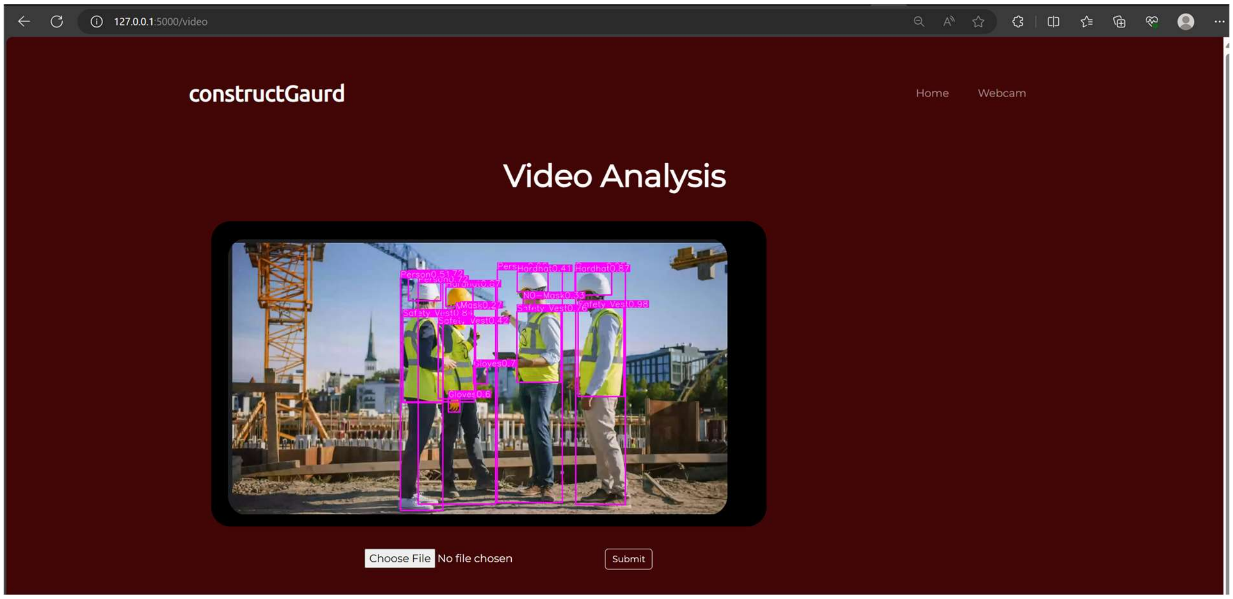
Highly Accurate.

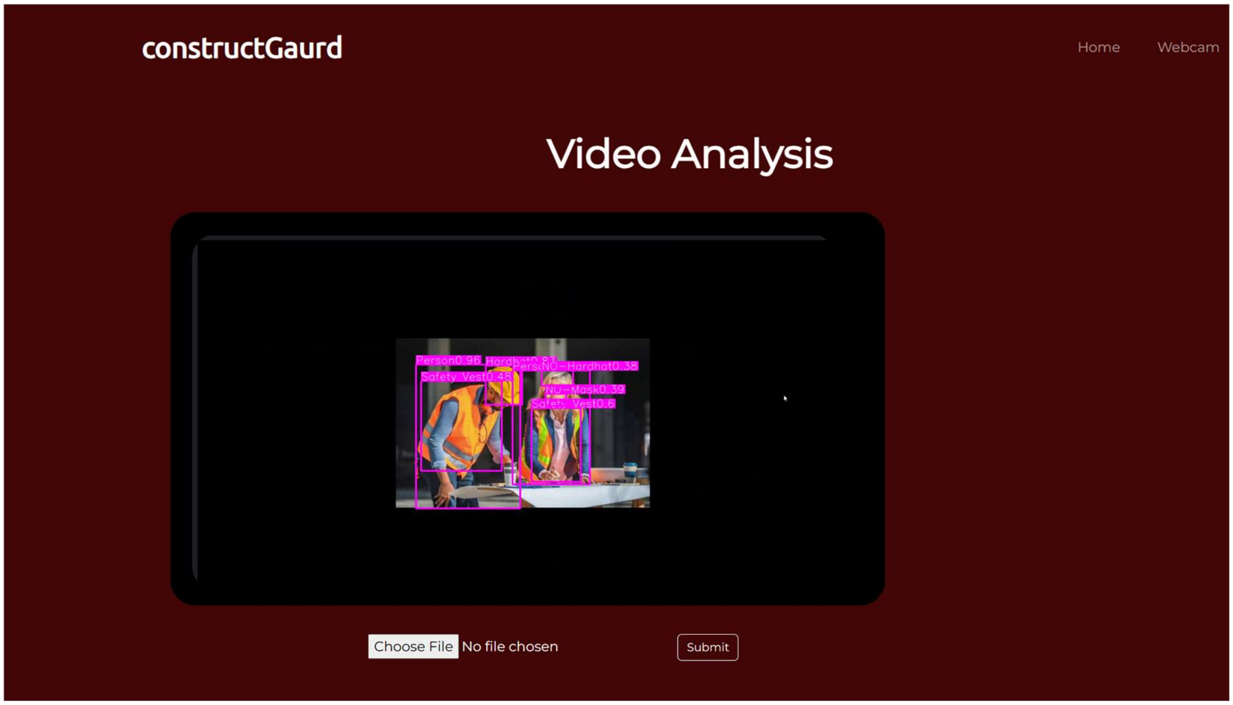
Our model has been designed to give highest accuracy.

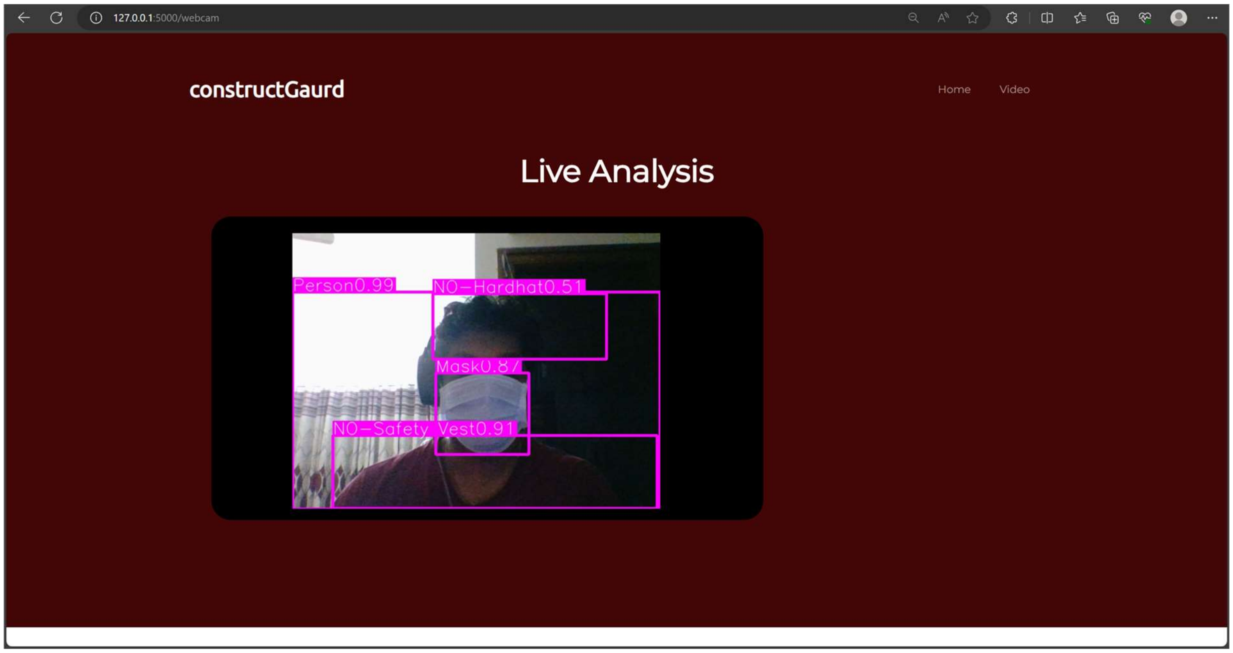
Guaranteed to work.

Our Product is loved by all users for its excellent reliability.

Final MediaOutput (after you click on submit) is displayed as follows:







127.0.0.1:5000/webcam
constructGaurd
Home
Video
Live Analysis
Person0.99
NO−Hardhat0.51
Mask0.87
NO−Safety Vest0.91