Real-time Bone Fracture Detection with YOLO-V8 Using X-ray Images

INTRODUCTION:

The project aims to develop a real-time bone fracture detection system using YOLO-V8, a state-of-the-art object detection algorithm, and X-ray images. This system serves as a crucial tool for medical professionals, facilitating rapid and accurate diagnosis of bone fractures, thereby expediting treatment and improving patient outcomes.

In this system, YOLO-V8 is trained on a dataset of X-ray images annotated with fracture locations. Upon deployment, the model processes X-ray images in real-time, identifying regions indicative of fractures with high accuracy. The detected fractures are then highlighted for easy visualization by medical practitioners.

Scenario 1: Emergency Room Diagnosis

A patient arrives at the emergency department with a suspected bone fracture. X-ray images are taken and instantly analyzed by the YOLO-V8-powered detection system. Within seconds, the system identifies the fracture's location and severity, helping the physician quickly plan appropriate treatment and reduce diagnostic delays.

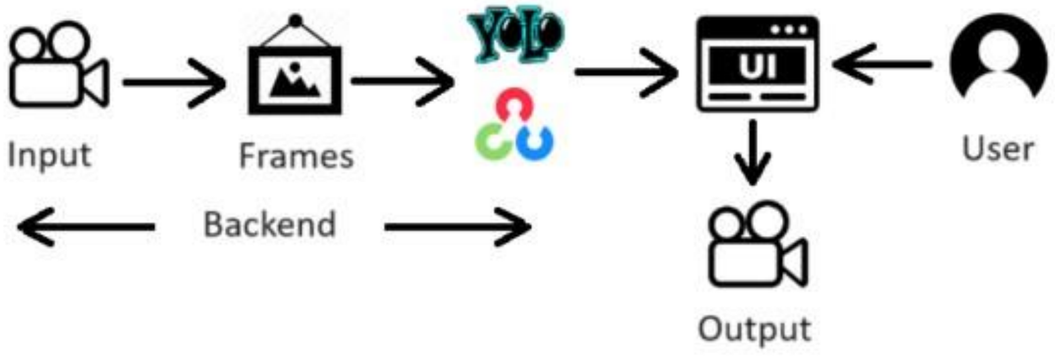
Scenario 2: Streamlined Clinic Workflow

In a busy orthopedic clinic, a physician faces a high volume of X-ray scans for review. By leveraging the YOLO-V8-based detection system, X-rays are analyzed in real time, with fractures automatically flagged. This streamlines the diagnostic process, allowing the physician to focus on treatment decisions rather than manual image inspection.

Scenario 3: Disaster Response Triage

During a mass casualty incident, such as a natural disaster, medical teams face resource constraints. The real-time fracture detection system aids in triaging patients by rapidly analyzing X-ray images. It identifies confirmed fractures, enabling medical personnel to prioritize critical cases and optimize resource allocation efficiently.

Technical Architecture:



Pre-requisites:

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

1. IDE Installation:

Spyder/ PyCharm IDE is Ideal to complete this project

To install Spyder IDE, please refer to Spyder IDE Installation Steps

To install PyCharm IDE, please refer to the PyCharm IDE Installation Steps

2. Python Packages

If you are using the vs code, follow the below steps to download the required packages:

Open the vs code.

- Type “pip install ultralytics” and click enter.
- Type “pip install numpy” and click enter
- Type “pip install flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- Flask - https://www.youtube.com/watch?v=lj4I_CvBnt
- Yolov8 - https://www.youtube.com/watch?v=ag3DLKsl2vk

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for computer vision.
- Gain knowledge in the pre-trained model yolov8.

Project Flow:

The user interacts with the UI (User Interface) to choose the image.

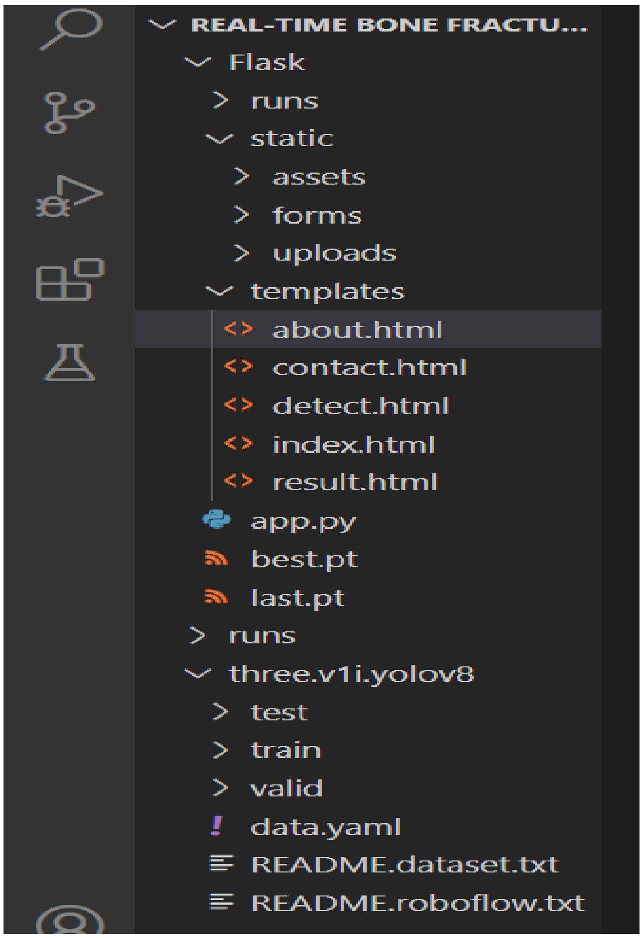
The chosen image is analyzed by the model which is integrated with the flask application.

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.



```
REAL-TIME BONE FRACTU...
  Flask
    runs
    static
      assets
      forms
      uploads
    templates
      about.html
      contact.html
      detect.html
      index.html
      result.html
    app.py
    best.pt
    last.pt
  runs
  three.v1i.yolov8
    test
    train
    valid
    data.yaml
    README.dataset.txt
    README.roboflow.txt
```

● The Dataset folder contains the training, testing, and val images for training our model.

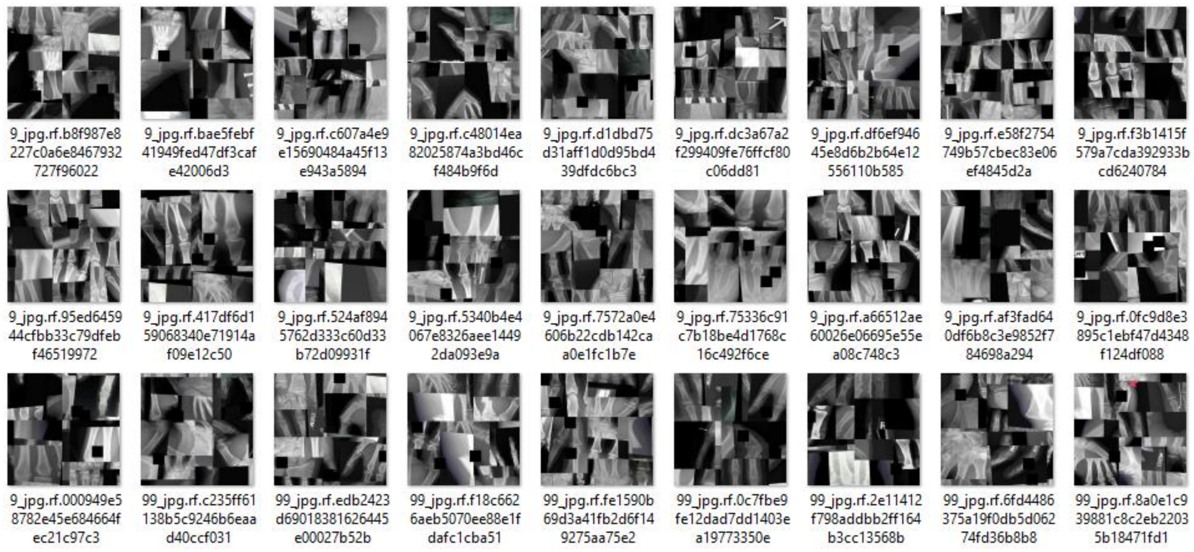
● We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting

● We need the model which is saved and the saved model in this content there is a templates folder containing index.html and inner-page.html pages.

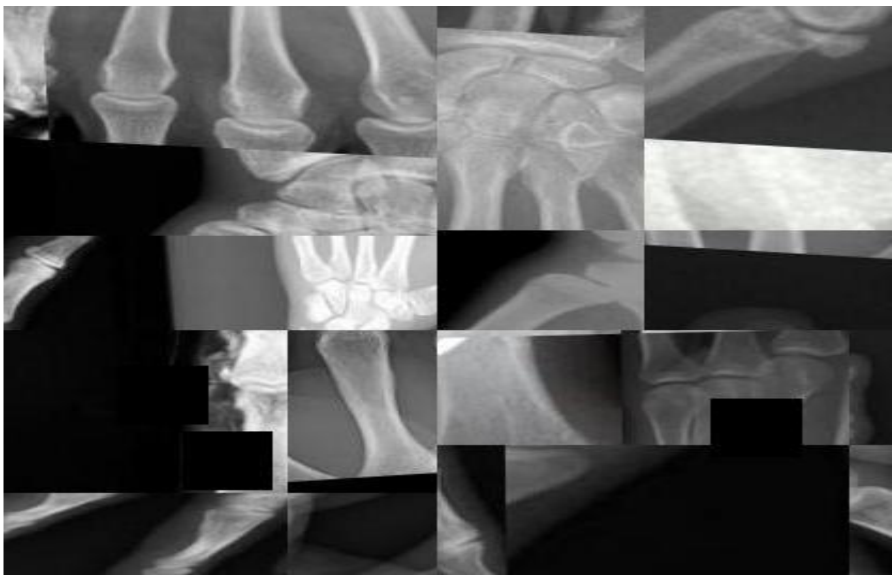
Milestone 1: Collection of Data

Dataset has 3 classes Which are Test, Train and Valid

Download the Dataset-


9_jpg.rf.b8f987e8227c0a6e8467932727f96022
9_jpg.rf.bae5febf41949fed47df3cafe42006d3
9_jpg.rf.c607a4e9e15690484a45f13e943a5894
9_jpg.rf.c48014ea82025874a3bd46cf484b9f6d
9_jpg.rf.d1dbd75d31aff1d0d95bd439dfdc6bc3
9_jpg.rf.dc3a67a2f299409fe76ffcf80c06dd81
9_jpg.rf.df6ef94645e8d6b2b64e12556110b585
9_jpg.rf.e58f2754749b57cbec83e06ef4845d2a
9_jpg.rf.f3b1415f579a7cda392933bcd6240784
9_jpg.rf.95ed645944cfbb33c79dfebf46519972
9_jpg.rf.417df6d159068340e71914af09e12c50
9_jpg.rf.524af8945762d333c60d33b72d09931f
9_jpg.rf.5340b4e4067e8326aee14492da093e9a
9_jpg.rf.7572a0e4606b22cdb142caa0e1fc1b7e
9_jpg.rf.75336c91c7b18be4d1768c16c492f6ce
9_jpg.rf.a66512ae60026e06695e55ea08c748c3
9_jpg.rf.af3fad640df6b8c3e9852f784698a294
9_jpg.rf.0fc9d8e3895c1ebf47d4348f124df088
9_jpg.rf.000949e58782e45e684664fec21c97c3
99_jpg.rf.c235ff61138b5c9246b6eaad40ccf031
99_jpg.rf.edb2423d69018381626445e00027b52b
99_jpg.rf.f18c6626aeb5070ee88e1fdafc1cba51
99_jpg.rf.fe1590b69d3a41fb2d6f149275aa75e2
99_jpg.rf.0c7fbe9fe12dad7dd1403ea19773350e
99_jpg.rf.2e11412f798addbb2ff164b3cc13568b
99_jpg.rf.6fd4486375a19f0db5d06274fd36b8b8
99_jpg.rf.8a0e1c939881c8c2eb22035b18471fd1

Sample Data:

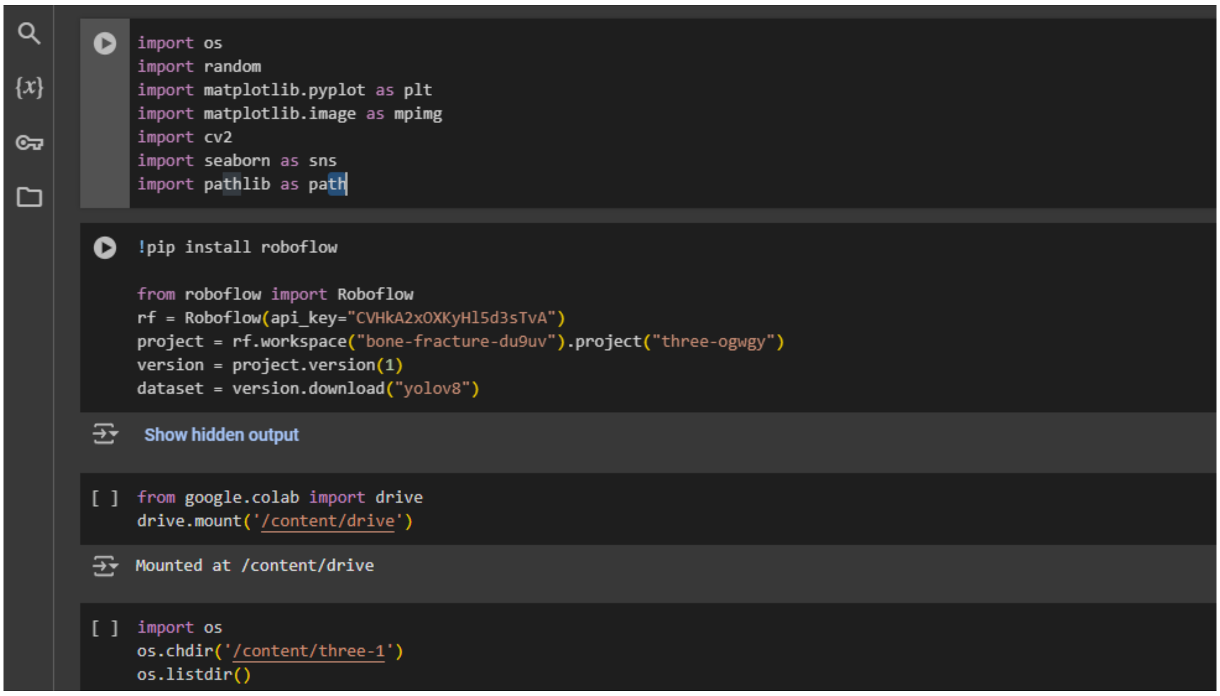


Milestone 2: Image Pre-processing

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 need to download yoloV8 from the ultralytics. https://universe.roboflow.com/bone-fracture-du9uv/three-ogwgy/dataset/1



```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import seaborn as sns
import pathlib as path
```

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="CVHkA2xOXKyHl5d3sTvA")
project = rf.workspace("bone-fracture-du9uv").project("three-ogwgy")
version = project.version(1)
dataset = version.download("yolov8")
```

Show hidden output

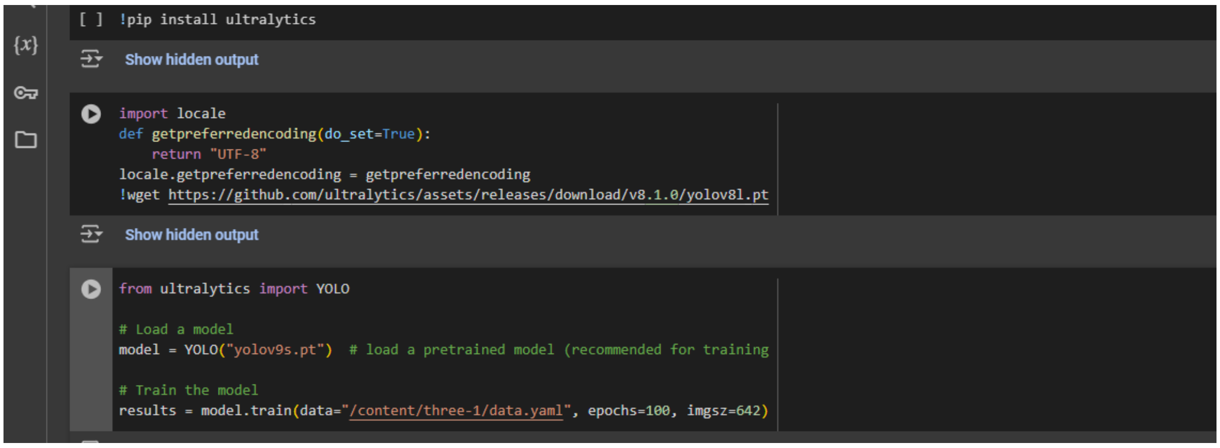
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
os.chdir('/content/three-1')
os.listdir()
```

Activity 2: Download the pretrained weights loading

pre-trained model yolov8 weights from ultralytics



```
!pip install ultralytics
```

Show hidden output

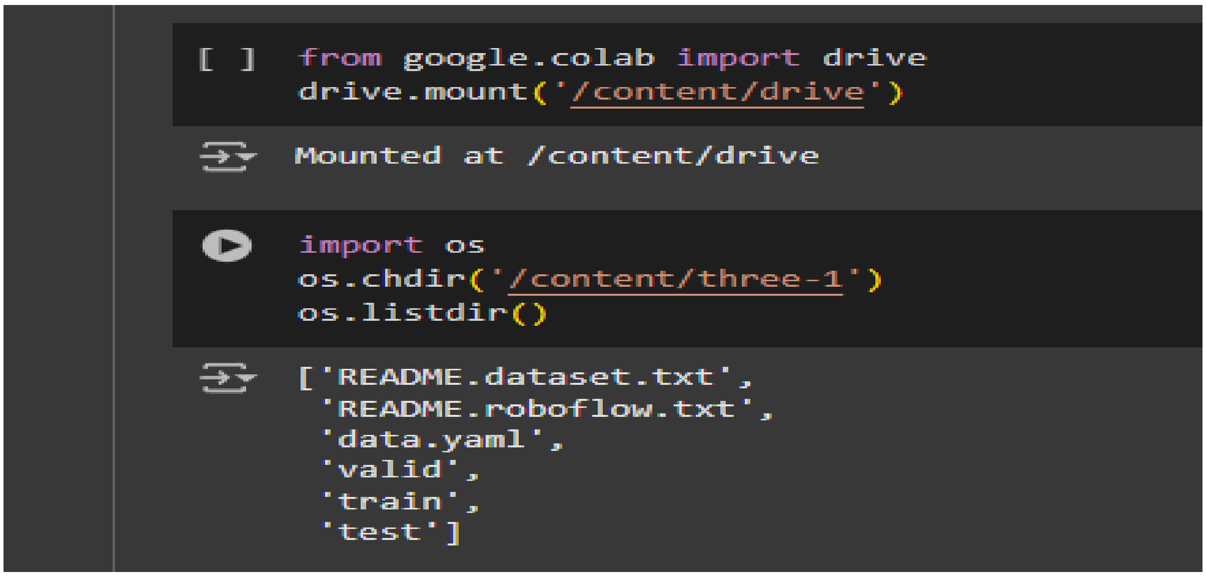
```
import locale
def getpreferredencoding(do_set=True):
    return "UTF-8"
locale.getpreferredencoding = getpreferredencoding
!wget https://github.com/ultralytics/assets/releases/download/v8.1.0/yolov8l.pt
```

Show hidden output

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov9s.pt")  # load a pretrained model (recommended for training

# Train the model
results = model.train(data="/content/three-1/data.yaml", epochs=100, imgsz=642)
```

Activity 3: Load the Dataset:

Installing roboflow and connect our dataset from google drive

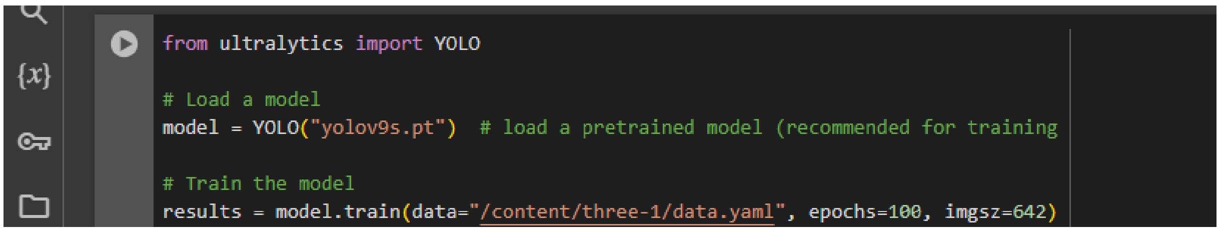


```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

     import os
     os.chdir('/content/three-1')
     os.listdir()

     ['README.dataset.txt',
      'README.roboflow.txt',
      'data.yaml',
      'valid',
      'train',
      'test']
```

Milestone 3: training

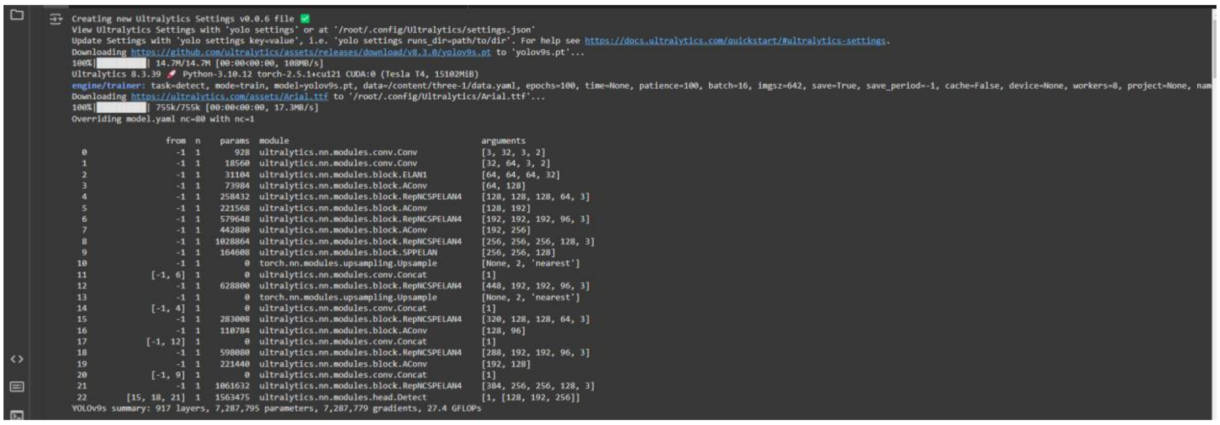
Now it's time to train our Yolo model:

We have to create data.yaml:

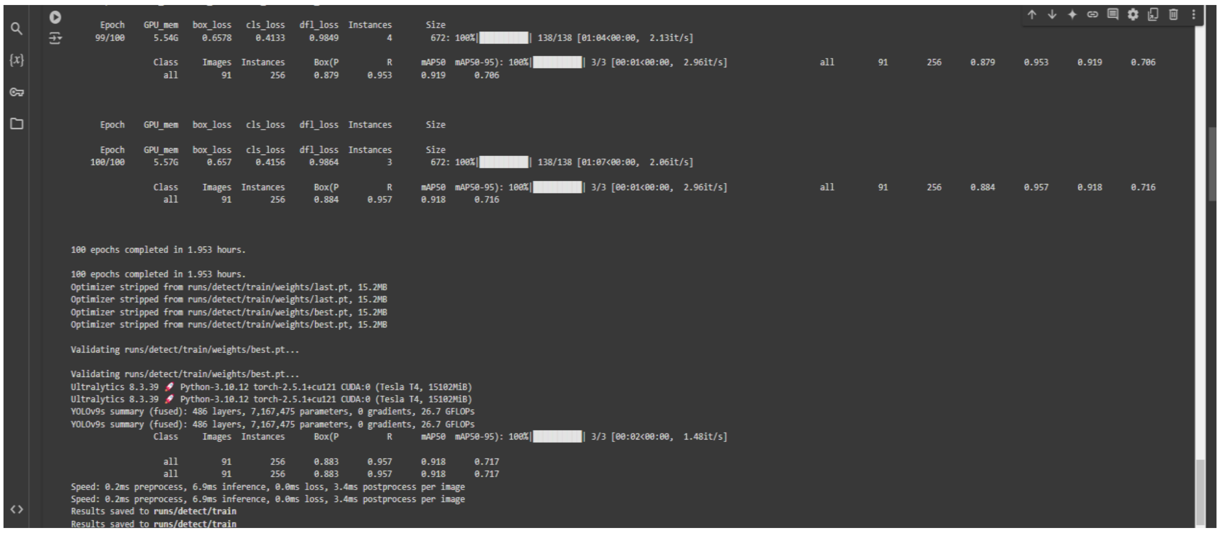


```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov9s.pt")  # load a pretrained model (recommended for training

# Train the model
results = model.train(data="/content/three-1/data.yaml", epochs=100, imgsz=642)
```

Training yolo v8 model on a custom dataset.



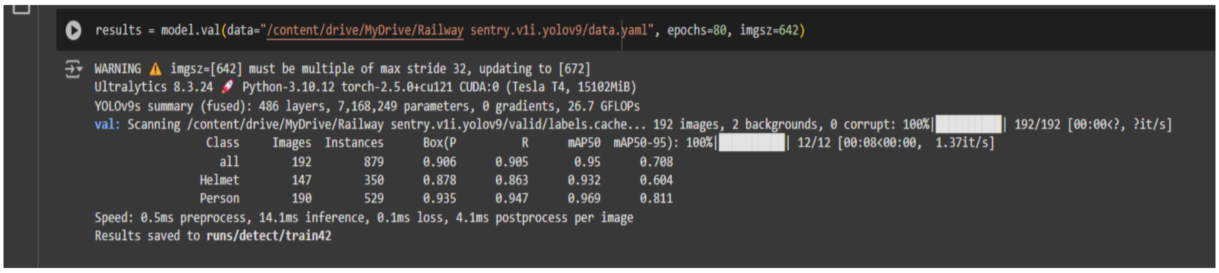
```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov9s.pt to 'yolov9s.pt'...
100%|██████████| 14.7M/14.7M [00:00<00:00, 108MB/s]
Ultralytics 8.3.39 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov9s.pt, data=/content/three-1/data.yaml, epochs=100, time=None, patience=100, batch=16, imgsz=642, save=True, save_period=-1, cache=False, device=None, workers=8, project=None, nam
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100%|██████████| 755k/755k [00:00<00:00, 17.3MB/s]
Overriding model.yaml nc=80 with nc=1

                 from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     31104  ultralytics.nn.modules.block.ELAN1           [64, 64, 64, 32]
  3                  -1  1     73984  ultralytics.nn.modules.block.AConv           [64, 128]
  4                  -1  1    258432  ultralytics.nn.modules.block.RepNCSPELAN4    [128, 128, 128, 64, 3]
  5                  -1  1    221568  ultralytics.nn.modules.block.AConv           [128, 192]
  6                  -1  1    579648  ultralytics.nn.modules.block.RepNCSPELAN4    [192, 192, 192, 96, 3]
  7                  -1  1    442880  ultralytics.nn.modules.block.AConv           [192, 256]
  8                  -1  1   1028864  ultralytics.nn.modules.block.RepNCSPELAN4    [256, 256, 256, 128, 3]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPELAN         [256, 256, 128]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    628800  ultralytics.nn.modules.block.RepNCSPELAN4    [448, 192, 192, 96, 3]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1    283008  ultralytics.nn.modules.block.RepNCSPELAN4    [320, 128, 128, 64, 3]
 16                  -1  1    110784  ultralytics.nn.modules.block.AConv           [128, 96]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    598080  ultralytics.nn.modules.block.RepNCSPELAN4    [288, 192, 192, 96, 3]
 19                  -1  1    221440  ultralytics.nn.modules.block.AConv           [192, 128]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1061632  ultralytics.nn.modules.block.RepNCSPELAN4    [384, 256, 256, 128, 3]
 22        [15, 18, 21]  1   1563475  ultralytics.nn.modules.head.Detect           [1, [128, 192, 256]]
YOLOv9s summary: 917 layers, 7,287,795 parameters, 7,287,779 gradients, 27.4 GFLOPs
```



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      5.54G     0.6578     0.4133     0.9849          4        672: 100%|██████████| 138/138 [01:04<00:00,  2.13it/s]

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:01<00:00,  2.96it/s]          all         91        256      0.879      0.953      0.919      0.706
                   all         91        256      0.879      0.953      0.919      0.706


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      5.57G      0.657     0.4156     0.9864          3        672: 100%|██████████| 138/138 [01:07<00:00,  2.06it/s]

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:01<00:00,  2.96it/s]          all         91        256      0.884      0.957      0.918      0.716
                   all         91        256      0.884      0.957      0.918      0.716



100 epochs completed in 1.953 hours.

100 epochs completed in 1.953 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 15.2MB
Optimizer stripped from runs/detect/train/weights/last.pt, 15.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 15.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 15.2MB

Validating runs/detect/train/weights/best.pt...

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.39 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Ultralytics 8.3.39 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv9s summary (fused): 486 layers, 7,167,475 parameters, 0 gradients, 26.7 GFLOPs
YOLOv9s summary (fused): 486 layers, 7,167,475 parameters, 0 gradients, 26.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:02<00:00,  1.48it/s]

                   all         91        256      0.883      0.957      0.918      0.717
                   all         91        256      0.883      0.957      0.918      0.717
Speed: 0.2ms preprocess, 6.9ms inference, 0.0ms loss, 3.4ms postprocess per image
Speed: 0.2ms preprocess, 6.9ms inference, 0.0ms loss, 3.4ms postprocess per image
Results saved to runs/detect/train
Results saved to runs/detect/train
```

Validation:

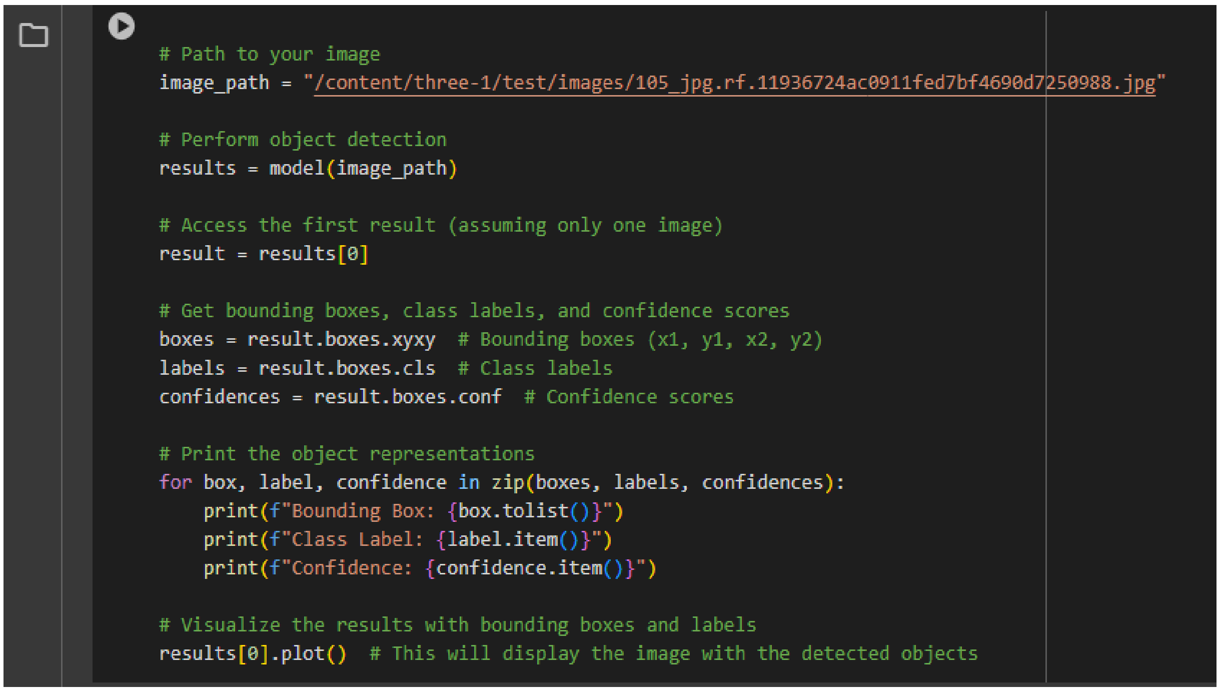
Validating our model with a random image from the test folder. Also, we have saved our best.pt



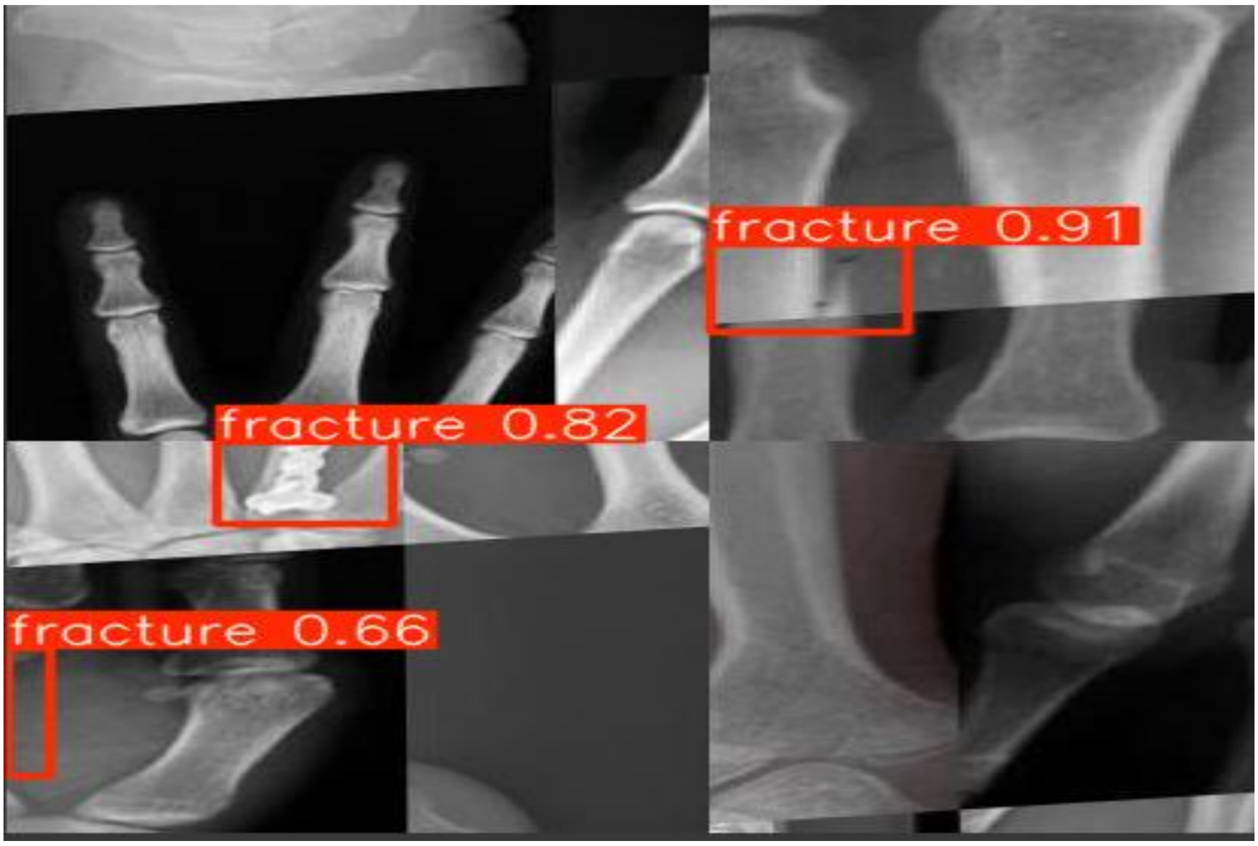
```
results = model.val(data="/content/drive/MyDrive/Railway sentry.v1i.yolov9/data.yaml", epochs=80, imgsz=642)
```

```
WARNING ⚠ imgsz=[642] must be multiple of max stride 32, updating to [672]
Ultralytics 8.3.24 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv9s summary (fused): 486 layers, 7,168,249 parameters, 0 gradients, 26.7 GFLOPs
val: Scanning /content/drive/MyDrive/Railway sentry.v1i.yolov9/valid/labels.cache... 192 images, 2 backgrounds, 0 corrupt: 100%|██████████| 192/192 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:08<00:00,  1.37it/s]
                   all        192        879      0.906      0.905       0.95      0.708
                Helmet        147        350      0.878      0.863      0.932      0.604
                Person        190        529      0.935      0.947      0.969      0.811
Speed: 0.5ms preprocess, 14.1ms inference, 0.1ms loss, 4.1ms postprocess per image
Results saved to runs/detect/train42
```

Displaying detected image in training notebook:



```python
# Path to your image
image_path = "/content/three-1/test/images/105_jpg.rf.11936724ac0911fed7bf4690d7250988.jpg"

# Perform object detection
results = model(image_path)

# Access the first result (assuming only one image)
result = results[0]

# Get bounding boxes, class labels, and confidence scores
boxes = result.boxes.xyxy  # Bounding boxes (x1, y1, x2, y2)
labels = result.boxes.cls  # Class labels
confidences = result.boxes.conf  # Confidence scores

# Print the object representations
for box, label, confidence in zip(boxes, labels, confidences):
    print(f"Bounding Box: {box.tolist()}")
    print(f"Class Label: {label.item()}")
    print(f"Confidence: {confidence.item()}")

# Visualize the results with bounding boxes and labels
results[0].plot()  # This will display the image with the detected objects
```



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.

Activity 1: Create HTML Pages

● We use HTML to create the front-end part of the web page.

● Here, we have created 3 HTML pages- home.html, intro.html, and upload.html

● home.html displays the home page.

● Intro.html displays an introduction about the project

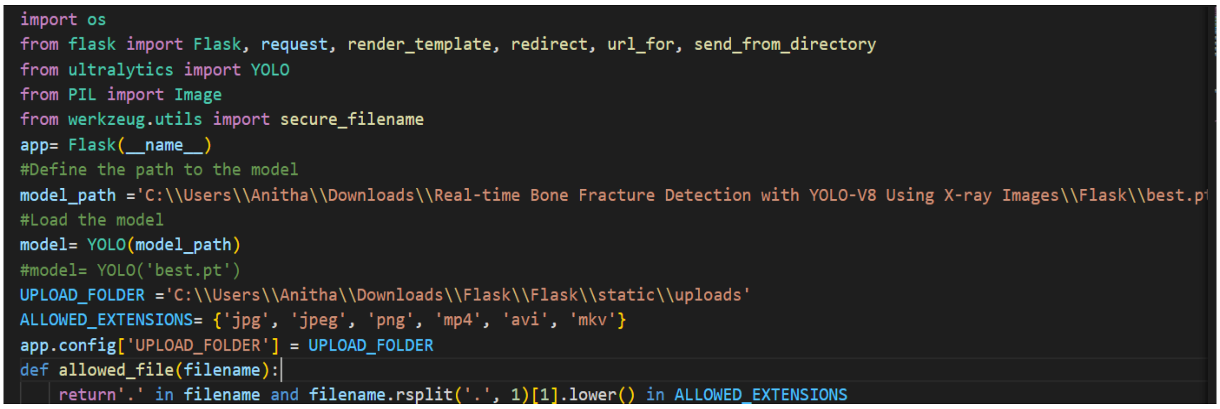
● upload.html gives the emergency alert for more information regarding HTML

● We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

● Link: CSS, JS

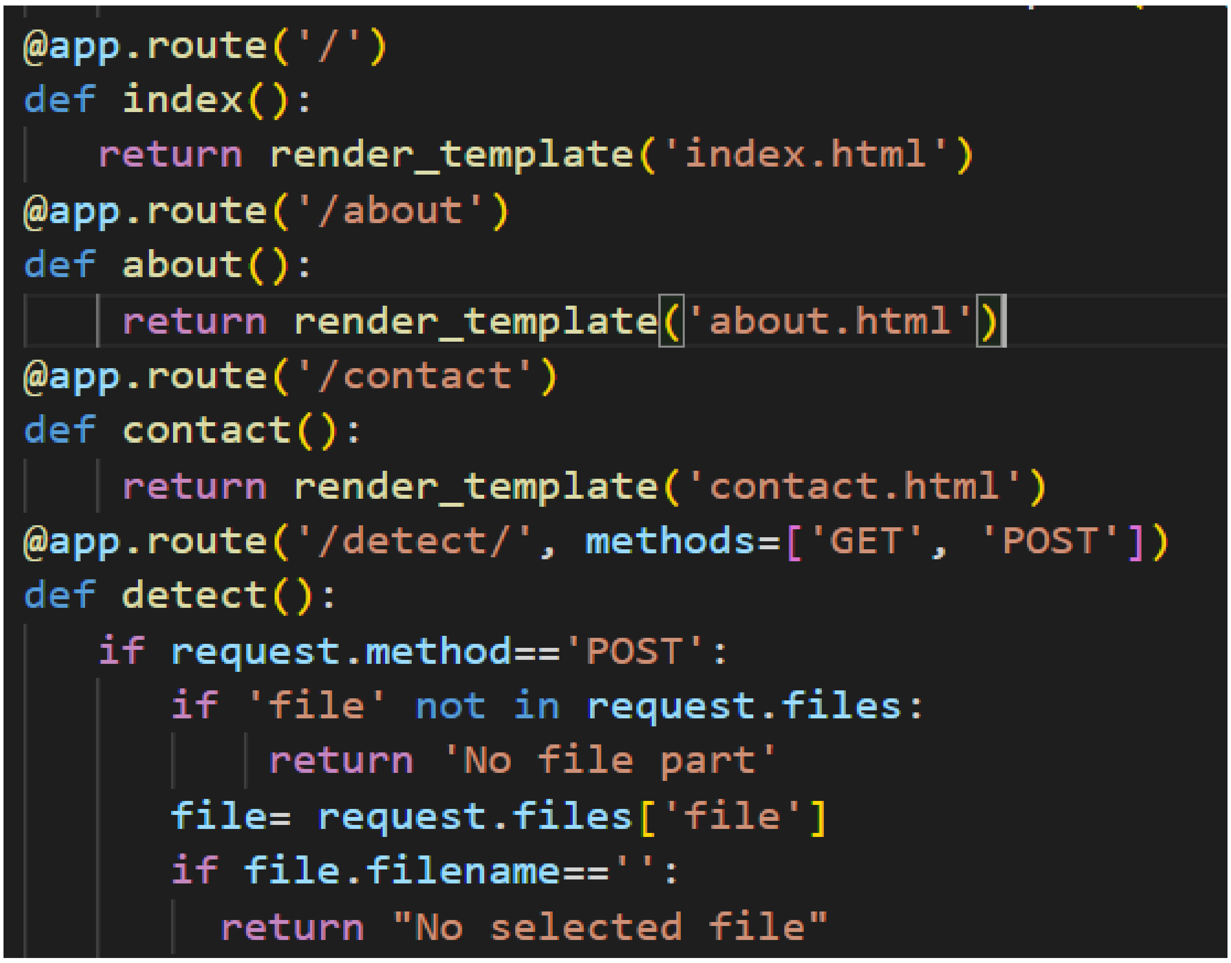
Create app.py (Python Flask) file: -

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



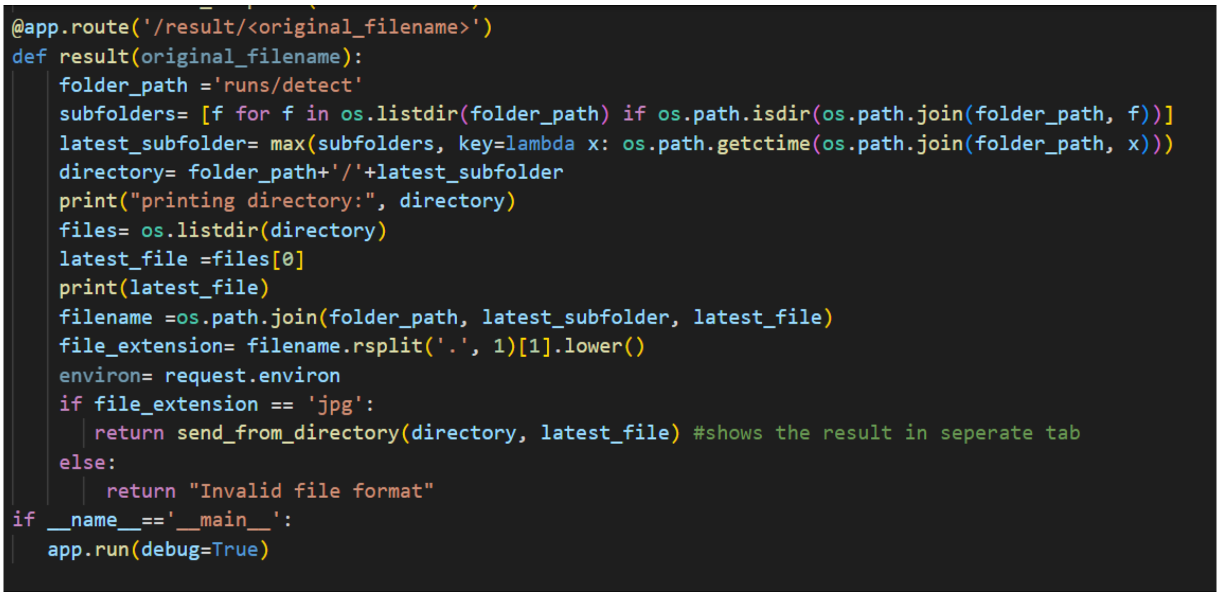
```python
import os
from flask import Flask, request, render_template, redirect, url_for, send_from_directory
from ultralytics import YOLO
from PIL import Image
from werkzeug.utils import secure_filename
app= Flask(__name__)
#Define the path to the model
model_path ='C:\\Users\\Anitha\\Downloads\\Real-time Bone Fracture Detection with YOLO-V8 Using X-ray Images\\Flask\\best.p
#Load the model
model= YOLO(model_path)
#model= YOLO('best.pt')
UPLOAD_FOLDER ='C:\\Users\\Anitha\\Downloads\\Flask\\Flask\\static\\uploads'
ALLOWED_EXTENSIONS= {'jpg', 'jpeg', 'png', 'mp4', 'avi', 'mkv'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
def allowed_file(filename):
    return'.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

To upload image in UI:



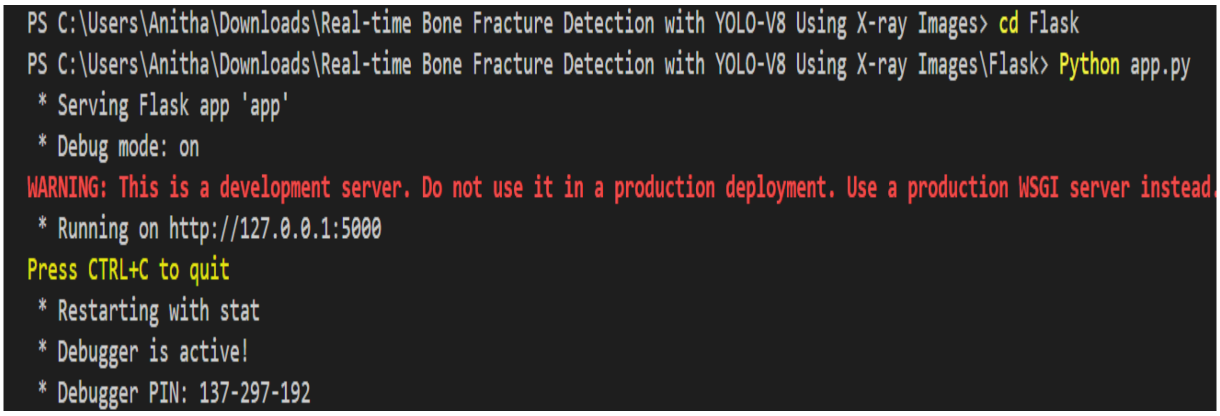
```python
@app.route('/')
def index():
   return render_template('index.html')
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/contact')
def contact():
    return render_template('contact.html')
@app.route('/detect/', methods=['GET', 'POST'])
def detect():
   if request.method=='POST':
      if 'file' not in request.files:
          return 'No file part'
      file= request.files['file']
      if file.filename=='':
        return "No selected file"
```

To display the image in UI:



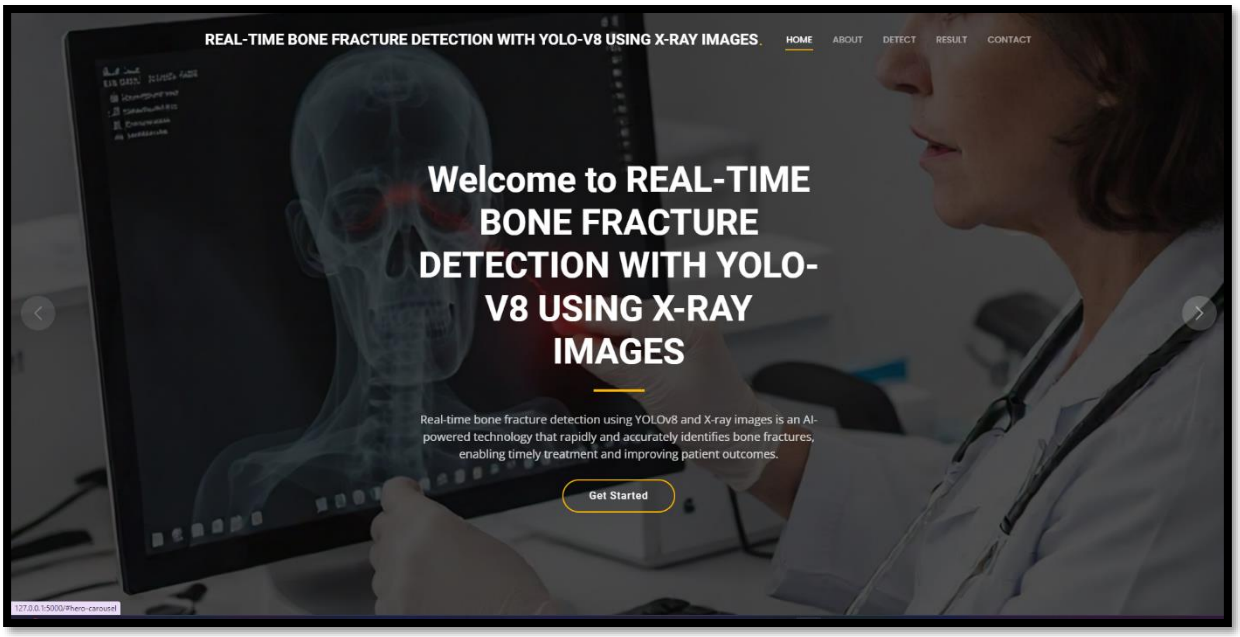
```
@app.route('/result/<original_filename>')
def result(original_filename):
    folder_path ='runs/detect'
    subfolders= [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    latest_subfolder= max(subfolders, key=lambda x: os.path.getctime(os.path.join(folder_path, x)))
    directory= folder_path+'/'+latest_subfolder
    print("printing directory:", directory)
    files= os.listdir(directory)
    latest_file =files[0]
    print(latest_file)
    filename =os.path.join(folder_path, latest_subfolder, latest_file)
    file_extension= filename.rsplit('.', 1)[1].lower()
    environ= request.environ
    if file_extension == 'jpg':
        return send_from_directory(directory, latest_file) #shows the result in seperate tab
    else:
        return "Invalid file format"
if __name__=='__main__':
    app.run(debug=True)
```

Getting local host in the terminal while running app.py:



```
PS C:\Users\Anitha\Downloads\Real-time Bone Fracture Detection with YOLO-V8 Using X-ray Images> cd Flask
PS C:\Users\Anitha\Downloads\Real-time Bone Fracture Detection with YOLO-V8 Using X-ray Images\Flask> Python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 137-297-192
```

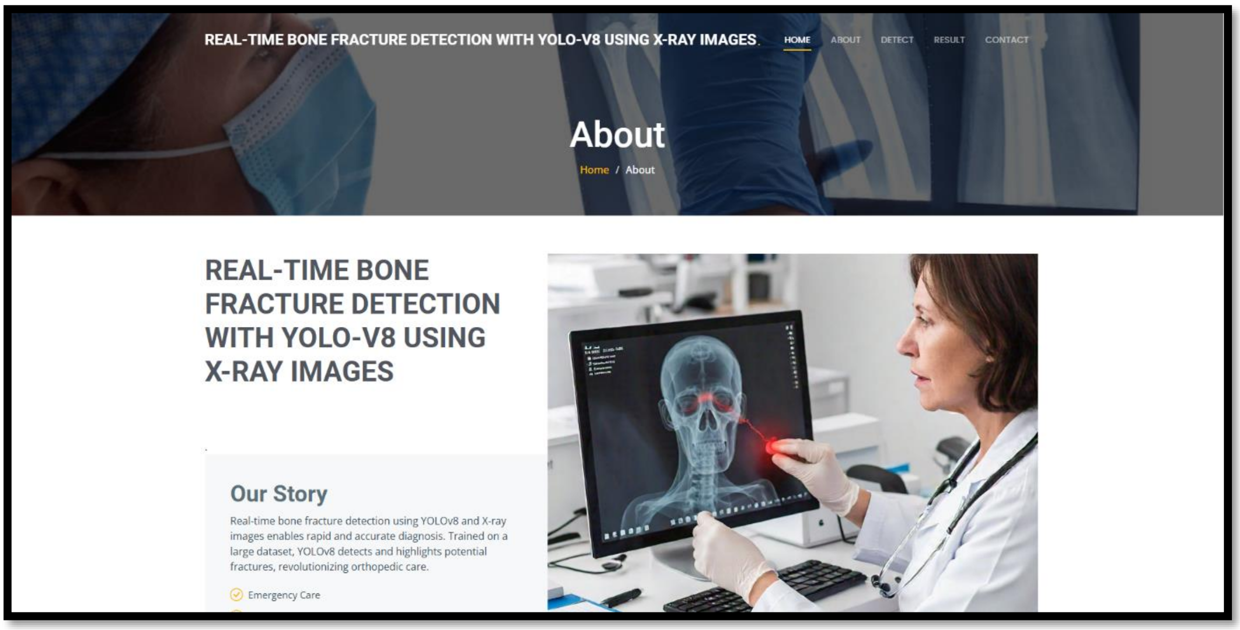
Index.html is displayed below:


REAL-TIME BONE FRACTURE DETECTION WITH YOLO-V8 USING X-RAY IMAGES
HOME
ABOUT
DETECT
RESULT
CONTACT
Get Started
127.0.0.1:5000/#hero-carousel

Welcome to REAL-TIME BONE FRACTURE DETECTION WITH YOLO-V8 USING X-RAY IMAGES

Real-time bone fracture detection using YOLOv8 and X-ray images is an AI-powered technology that rapidly and accurately identifies bone fractures, enabling timely treatment and improving patient outcomes.

About Section is displayed below:


REAL-TIME BONE FRACTURE DETECTION WITH YOLO-V8 USING X-RAY IMAGES
HOME
ABOUT
DETECT
RESULT
CONTACT
Home / About
Emergency Care

About

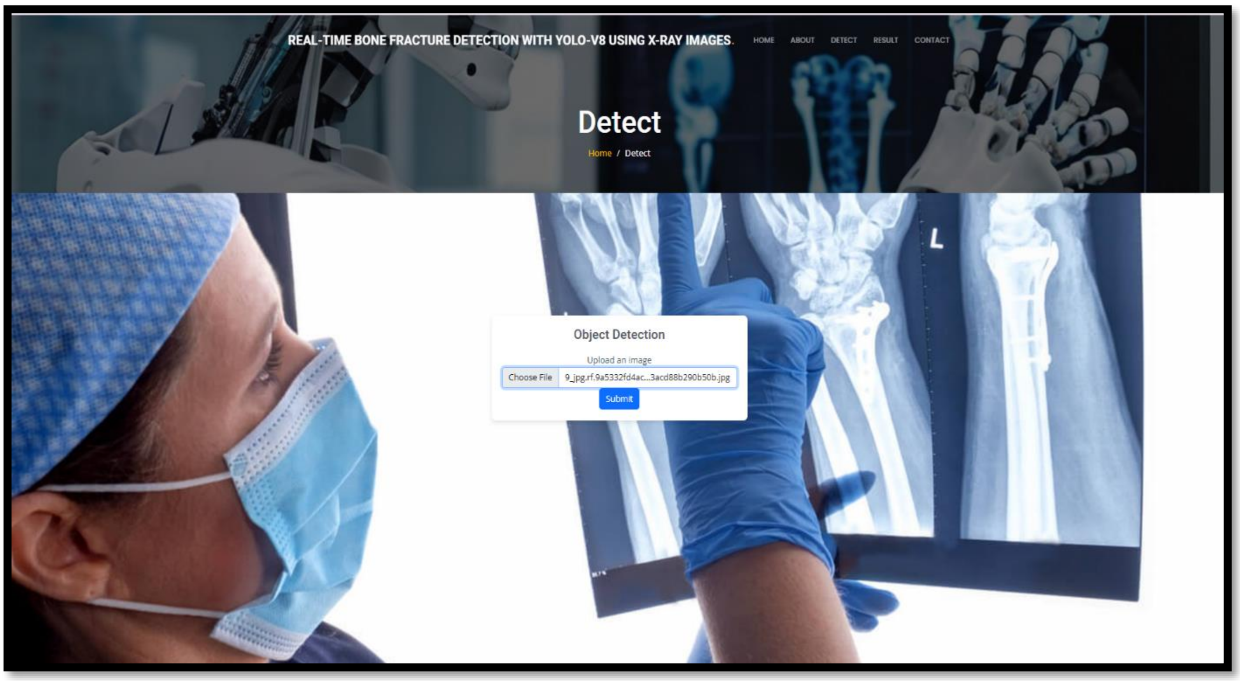
REAL-TIME BONE FRACTURE DETECTION WITH YOLO-V8 USING X-RAY IMAGES

Our Story

Real-time bone fracture detection using YOLOv8 and X-ray images enables rapid and accurate diagnosis. Trained on a large dataset, YOLOv8 detects and highlights potential fractures, revolutionizing orthopedic care.

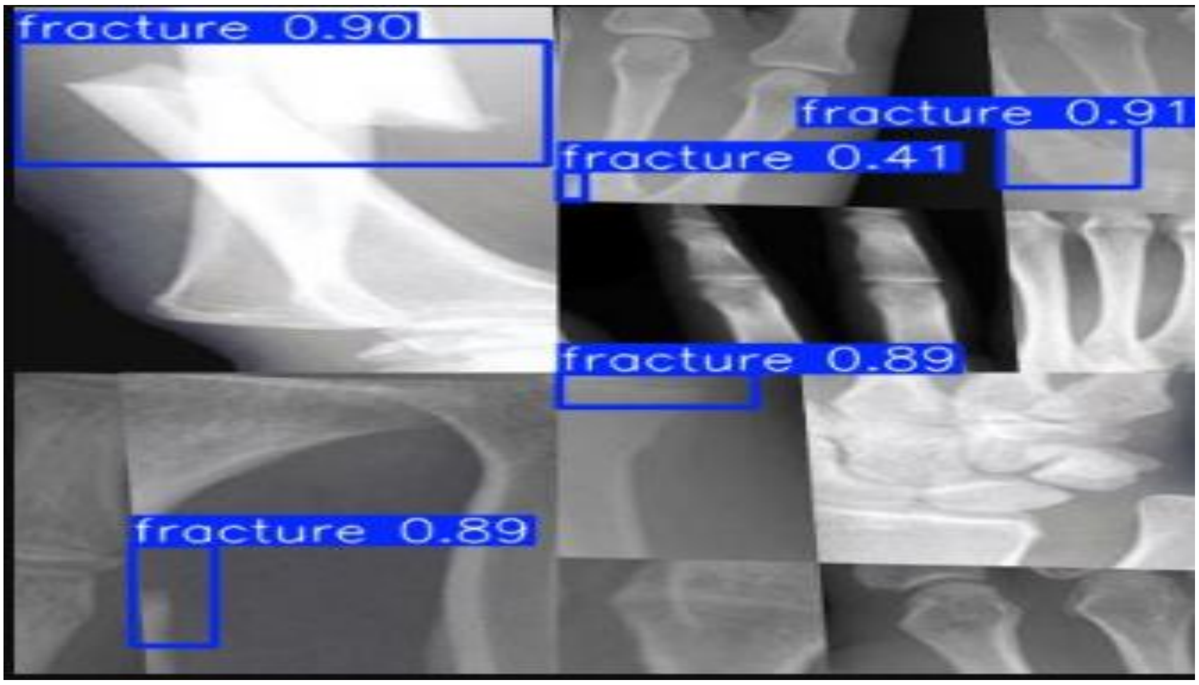
Upload and Output Page is displayed below

Input:1

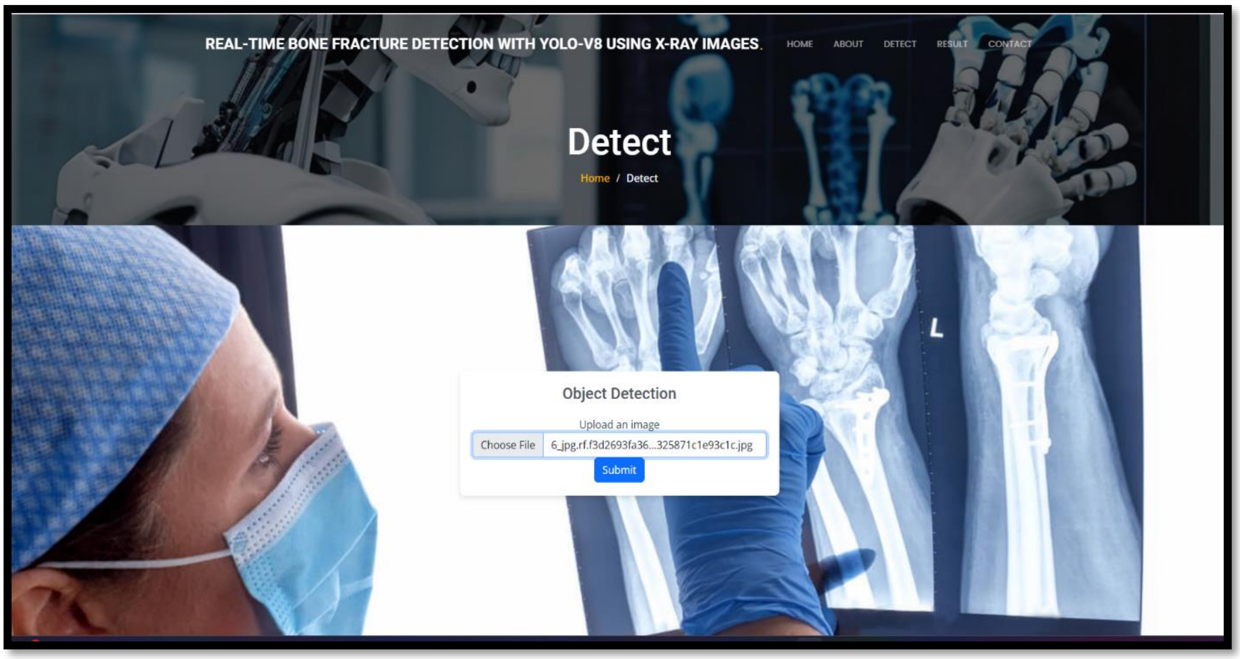

REAL-TIME BONE FRACTURE DETECTION WITH YOLO-V8 USING X-RAY IMAGES.
HOME
ABOUT
DETECT
RESULT
CONTACT
Detect
Home / Detect
Object Detection
Upload an image
Choose File
9_jpg.rf.9a5332fd4ac...3acd88b290b50b.jpg
Submit

Final Output (after you click on Upload) is displayed as follows:

Output: 1



Input:2


REAL-TIME BONE FRACTURE DETECTION WITH YOLO-V8 USING X-RAY IMAGES
HOME
ABOUT
DETECT
RESULT
CONTACT
Detect
Home / Detect
Object Detection
Upload an image
Choose File
6_jpg.rf.f3d2693fa36...325871c1e93c1c.jpg
Submit

Output:2