Raja Ramanna Centre for Advanced Technology

Indore(M.P.)-452012

Report of Internship

Deep Learning based Object detection in Images using YOLOv8

Submitted to:-
Shri. Gaurav Saxena
SO/G

Submitted by:-
Parth Gome

Abstract

This report investigates the implementation and evaluation of the YOLOv8 model on a custom dataset tailored for fire and smoke detection. Fire and smoke detection hold critical significance across diverse sectors, including fire prevention, surveillance, and safety monitoring. Leveraging the YOLOv8 model, renowned for its real-time object detection capabilities, the study customizes and trains the model using the Ultralytics framework. The methodology encompasses comprehensive stages of data preprocessing, model configuration, training, and evaluation.

The implementation of a smoke and fire detection model using YOLO was imperative to enhance fire safety measures. Its real-time processing capability and high accuracy enable swift identification of fire hazards, facilitating timely response and minimizing the risk of property damage and loss of life.

In our project the dataset has been taken from the muliple sources which are fire images and smoke images with corresponding labels of their annotations. There were around 5000+ Images in the model that has been used. The dataset was not ideal as there were many corrupt images and empty labels , some images are been corrected while making of project. The dataset then been curated in the directory of folders according to the needs of the YOLOv8 model .

For creation of model then a data.yaml file was created to store the path of the directory of the images and the classes stored in the same folder similar to the python notebok is saved then the model is been trained for 100 epoches in the batch size of 8 . That gave us the mAP value of overall as 0.945 , precision of 93.6%, recall of 91.3%, and an F1 score of 73.0% on the test dataset

Furthermore the past training infrencing been done to check whether the model is working properly or not, so the images are tested on the model then it is exported in the deployable form which was ONNX in this project. Deployment has been done using Flask Framework through which the model then deploed on the local host web page which were created using HTML .

In addition to the model's performance evaluation, this report delves into the challenges encountered and the strategies employed to address them during the implementation process. Factors such as dataset quality, class imbalance, and model optimization techniques are thoroughly examined to provide a comprehensive understanding of the model's behavior in fire and smoke detection scenarios. Validation has been done for the evaluation of the nodel on the different matrices. Furthermore, Benchmarking has been dones on the model to check whether th model is upto the mark on the ideal set taken from the research papers .

Table of Contents

1. Introduction	4
2. Literature Review	5
3. Technology Used	6
4. Development Platform	7
4.1 Object Detection	
4.2 Ultralytics & YOLOv8	
4.3 YOLO	
4.4 Python Flask	8
4.5 PyCharm	
4.6 Python	
4.7 Deep Learning	
5. Methodology	9
5.1 Training	
5.2 Inferencing	13
5.3 Validation	14
5.4 Testing/Benchmarking	16
5.5 Performance Metrices	17
5.6 Export	18
5.7 Deployment	19
6. Results	24
7. Future Use	28
8. Discussion	29
9. Conclusions	30
10. References	31

1. Introduction

During this 6-month internship, the main focus was on Object Detection via YOLO. The Ultralytics YOLOv8 model served as a role model for our custom trained model.

Addressing the limitations of traditional fire detection methods, we focus on developing an efficient smoke and fire detection system using deep learning technology.

In this report, we delve into the exploration and implementation of YOLOv8 for object detection tasks during a six-month internship. We investigate the underlying principles of the algorithm, explore its capabilities, and evaluate its performance in real-world scenarios. Through practical experimentation and analysis, we aim to gain insights into the strengths and limitations of YOLOv8 and its implications for the broader field of computer vision.

A smoke and fire detection model via YOLO is essential for timely identification and mitigation of fire hazards. Its real-time processing capabilities enable swift detection, aiding in the prevention of property damage, loss of life, and facilitating prompt emergency response measures, ultimately enhancing overall safety.YOLO offers real-time, accurate detection across diverse environments, enhancing fire safety measures significantly.

Current methods for smoke and fire detection often involve the use of traditional smoke detectors, heat detectors, and flame detectors. Additionally, advanced systems may integrate sensors such as infrared cameras and photoelectric sensors to detect smoke particles and changes in temperature or radiation levels, enhancing detection accuracy.

YOLOv8, developed by Ultralytics, represents a significant evolution in the realm of object detection. Building upon the success of its predecessors, YOLOv8 integrates cutting-edge deep learning techniques to achieve remarkable accuracy and speed in detecting objects within images and videos.[1] By employing a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation, YOLOv8 offers unparalleled efficiency compared to traditional multi-stage detection pipelines.[2]

Furthermore, YOLOv8 features a user-friendly interface and comprehensive documentation, facilitating its adoption and customization for various use cases. Its open-source nature encourages collaboration and community-driven improvements.

We have depoyed our model via python flask as YOLOv8's seamless integration with Python Flask further enhances its utility by enabling easy deployment of object detection models as web services. Leveraging the power of Flask, developers can create robust and scalable applications that harness the capabilities of YOLOv8 for real-time object detection tasks.

This deployment approach opens up new possibilities for incorporating object detection functionalities into web-based systems, such as e-commerce platforms, smart surveillance systems, and image analysis tools

2. Literature Review

Several studies have explored the use of deep learning techniques for smoke and fire detection. Zhanchi Liu created on smoke and fire only in 2023 YOLOv8 for Fire and Smoke Recognition Algorithm Integrated with the Convolutional Block Attention Module that employs the latest YOLOv8 for object recognition. Subsequently, the integration of CBAM enhances its feature extraction capabilities. [8]

Fenggang Liu et al. in 2017 proposed a deep convolutional neural network (CNN) for real-time smoke detection in video surveillance systems.First, we calculate the similarity of the overall structure and the average square error (MSE) for the detection of the smoke movement of the entry surveillance camera. The regions that are candidates for the delay were extracted by a deep learning algorithm (faster R-CNN). [9]

Similarly, Snehitvaddi et al. in 2020 developed a fire detection system using YOLOv3, achieving high accuracy in detecting flames in video streams.[10]

In the field of forest fire monitoring, standard methods require the utilization of manual observation techniques , Moulay A. Akhloufi in 2023 created fire and smoke Detection using fine-tuned YOLOv8 and YOLOv7 deep models.[11]

Train YOLOv8 On A Custom Dataset For Fire And Smoke Detection by Kazi Mushfiqur Rahman in July 2023 completed its work on it.[12]

Wildfire and Smoke Detection Using Staged YOLO Model and Ensemble CNN that helps minimize mortality and harm to ecosystems and forest life created by Chayma Bahhar in January 2023.[13]

Recent advancements in deep learning & object detection algorithms like YOLO (You Only Look Once), have shown promise in various computer vision tasks, including smoke and fire detection. YOLO is known for its real-time processing capabilities and high accuracy, making it well-suited for applications requiring swift detection of objects in images or video streams.

Overall, the highlights the potential of deep learning, particularly YOLO-based methods, in improving the efficiency and accuracy of smoke and fire detection systems. By using these advancements, our project aims to develop a robust detection system capable of timely identification of fire hazards, thereby enhancing fire safety measures in various environments.

3. Technology Used

In this internship project , the utilization of several key technologies that developed and deployed a smoke and fire detection system:

i. Software Components:

PyCharm as an Integrated Development Environment (IDE): PyCharm provided us with a comprehensive development environment for writing, testing, and debugging our Python code. Its fillbodied features, including intelligent code completion, version control integration, and multiple plugins, significantly facilitated our development workflow.With a PyCharm version of 2023.2.5.Also the python language was 3.11.9 version of it.

Ultralytics YOLOv8: We used YOLOv8, developed by Ultralytics, as the core deep learning model for smoke and fire detection. YOLOv8 is a SOTA object detection algorithm whicch is known for its real-time processing capabilities and high accuracy. Using the pre-trained weights and flexible architecture of YOLOv8, we customized the model to detect smoke and fire instances within images.

Flask Framework for Deployment: Python Flask work as the framework for deploying our trained YOLOv8 model as a web service. Flask is a lightweight and versatile micro-framework that enables the creation of web applications with minimal boilerplate code. By using Flask with our YOLOv8 model, we created an efficient system that is capable of processing incoming image data in real-time and providing timely smoke and fire detection results via a web interface.

ii. Hardware Components: In addition to software tools, our project utilized specific hardware components to support the development and deployment of the detection system. This included high-performance GPUs (Graphics Processing Units) for accelerating the training and inference processes of the YOLOv8 model. The parallel processing capabilities of GPUs significantly enhanced the efficiency of model training and real-time detection.The Intel i7 12700 chip was utilized as the processor, with 32GB of RAM for enhanced user experience speed. The nVIDIA Quadro T100 w/4GB was employed as our GPU. Additionally, a 1TB HDD was incorporated for extensive dataset handling on Windows 11 Professional.

The combination of all these provided us with a powerful toolkit and a platform for developing and deploying our smoke and fire detection system. These technologies facilitated us to implement our project and also enabled us to achieve high accuracy and efficiency in detecting fire hazards, showcasing us the potential of deep learning and web development in addressing real-world challenges in fire safety and emergency response.

4. Development Platform

4.1 Object Detection

Object detection is a computer vision technique that loactes the objects in images or videos. Object detection algorithms typically uses machine learning or deep learning to produce meaningful results. The main goal of object detection is aswe human detect the feature in an image or video within fraction of seconds.Similarly the goal of object detection is to replicate this intelligence using a computer.[1]

4.2 Ultralytics & Yolov8

Ultralytics is a software company that focuses on developing computer vision and deep learning solutions. Their latest product is YOLOv8 that stands for "You Only Look Once version 8". YOLO is a popular object detection algorithm known for its speed and accuracy. Ultralytics' YOLOv8 is an upgraded version of the original YOLO algorithm, optimized for efficiency and performance. It is used world wide in various applications such as autonomous vehicles, surveillance camera's, and image analysis. Ultralytics provides pre-trained models, training scripts, and tools to facilitate the deployment of YOLOv8 in real-world scenarios.

Ultralytics YOLOv8 is the successor model of previous YOLO versions and with new features and improvements to further boost performance and flexibility. YOLOv8 is now upgraded and designed to be fast, accurate, and easy to use, making it an excellent choice for object detection and tracking, instance segmentation, image classification and pose estimation tasks.[2]

4.3 YOLO

You Only Look Once (YOLO) is an end-toend neural network that makes predictions by making bounding boxes at object and class probabilities all at once. It uses different approach as compared to previous object detection algorithms, which repurposed classifiers to perform detection.

Following a fundamentally different approach to object detection, YOLO achieved state-of-the-art results, beating other real-time object detection algorithms by a large margin.

Faster RCNN is an algorithm works by detecting possible regions of interest using the RPN (Region Proposal Network) and then performs recognition on those regions separately, YOLO performs all of its predictions with the help of a single fully connected layer.

Methods that use Region Proposal Networks perform multiple iterations for the same image, while YOLO gets away with a single iteration.

Multiple new versions of YOLO model have been proposed since the initial release of YOLO in 2015, each building on and improving its predecessor.[3]

4.4 Python Flask

Flask is an API of Python that allows us to build up web-applications. Flask's framework is more explicit than Django's framework and is also easier to learn because it has less base code to implement a simple web-Application. It is a Web-Application Framework which has collection of modules and libraries that helps the developer to write applications without writing the low-level codes such as protocols, thread management, etc. Flask is a WSGI(Web Server Gateway Interface) based toolkit and Jinja2 template engine.

Flask is a web framework that developes lightweight web applications quickly and easily with Flask Libraries. It was developed by Armin Ronacher, leader of the International Group of Python Enthusiasts(POCCO). It is basically based on the WSGI toolkit and Jinja2 templating engine.[4]

4.5 PyCharm

PyCharm is an IDE used for programming python codes. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and supports web development with Django. PyCharm is developed by Jet Brains a Czech Company.

It works cross-platform, Windows, linux, macOS. PyCharm has a Professional Edition, released under a proprietary license and a Community Edition which was released under the Apache License. PyCharm Community Edition is less extensive than the Professional Edition.[5]

We have used PyChaarm as the IDE for the development of the machie learning model.

4.6 Python

Python is a programming language that is interpreted, object-oriented, and considered to be high-level too.Python is the simplest yet most useful programming languages which is widely used in the software industry. Python is used in Competitive programming, Web Development, and creating software. It has easiest syntax, it is recommended for beginners who are new to the software engineering field. The growing demand of it at a very rapid pace due to its vast use cases in Modern Technological fields like Data Science, Machine learning and Automation Tasks. For many years now, it has been ranked among the top Programming languages. [4]

4.7 Deep learning

Deep learning is a class of machine learning algorithm thatuses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human, such as digits, letters, or faces.

Deep learning is a subset of machine learning that is based on artificial neural networks (ANNs). Deep learning uses multiple layers in the network. Methods used in Deep Learning are supervised, semi-supervised or unsupervised any one of the three can be used. [5]

5. Methodology

5.1 Training

In the process of customizing YOLOv8 to train a model on a custom dataset, the initial step involves creating the dataset. For this purpose, a folder named "data" needs to be established. Within this folder, three subfolders are required: one for training, another for validation, and a third one for the test set. Each of these subfolders should contain two additional folders: "image" for storing images and "labels" for storing annotations corresponding to the images.

The images in this dataset should be annotated according to the YOLO format, which includes class, x center, y center, width, and height (c, x, y, w, h). These annotations are stored in text files accompanying each image. Each line in the text file corresponds to one object instance in the image, and the format for each line is: "class x_center y_center width height".

So the directory of the dataset would looklike this :

```
data/
│
├── train/
│   ├── images/
│   │   ├── image1.jpg
│   │   ├── image2.jpg
│   │   └── ...
│   └── labels/
│       ├── image1.txt
│       ├── image2.txt
│       └── ...
│
├── test/
│   ├── images/
│   │   ├── image1.jpg
│   │   ├── image2.jpg
│   │   └── ...
│   └── labels/
│       ├── image1.txt
│       ├── image2.txt
│       └── ...
│
└── val/
    ├── images/
    │   ├── image1.jpg
    │   ├── image2.jpg
    │   └── ...
    └── labels/
        ├── image1.txt
        ├── image2.txt
        └── ...
```

We've taken significant strides in our project, focusing on object detection. Our foremost task was to compile a diverse dataset comprising images pertinent to our objectives. In this endeavor, we've meticulously curated images representing various objects relevant to our detection goals.During the dataset curation process, we've encountered the need to preprocess both images and labels. We've diligently handled corrupted or empty labels, ensuring the integrity of our dataset. An essential aspect of our work involved annotating images with precise bounding boxes delineating the objects for detection by our model.

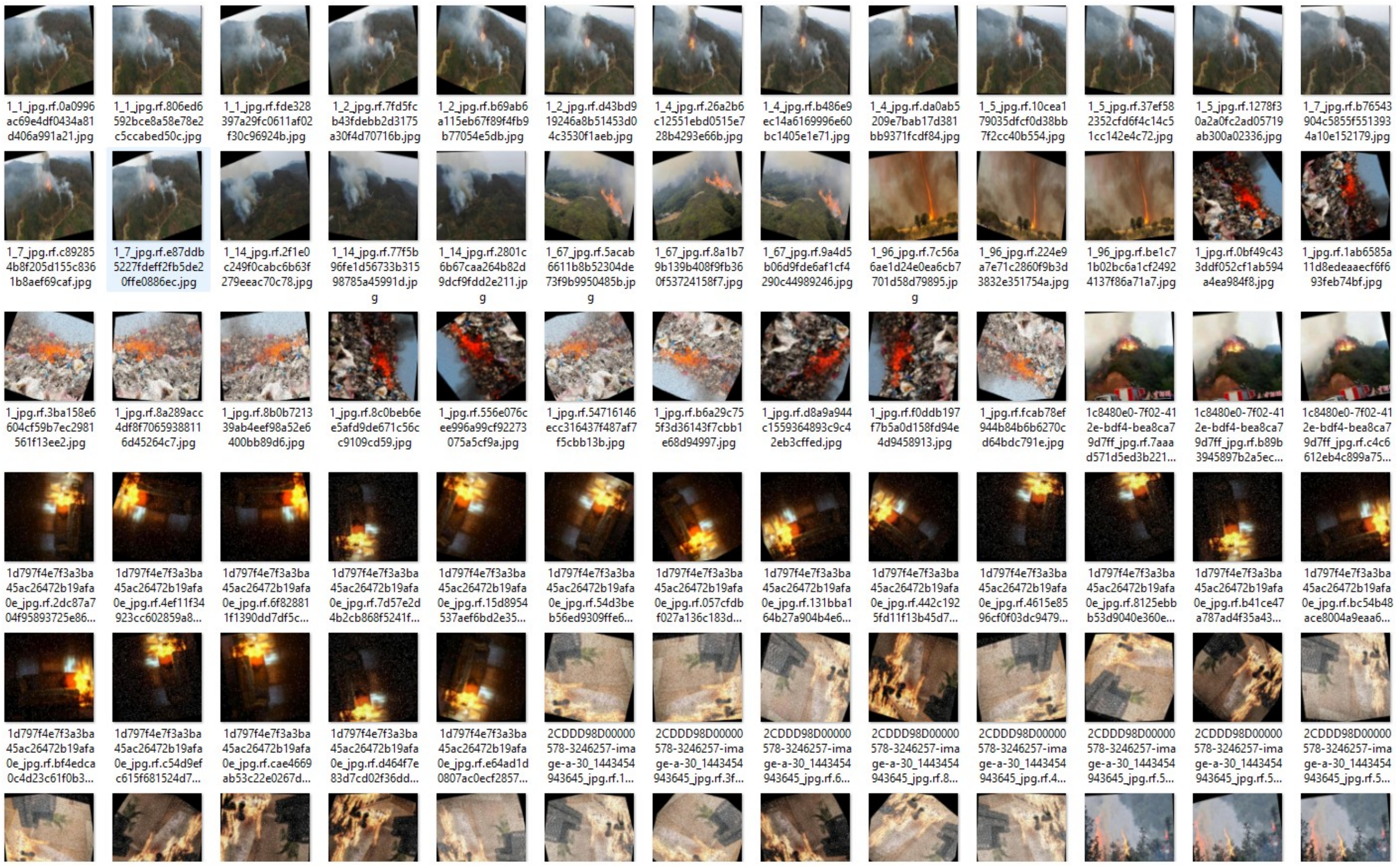


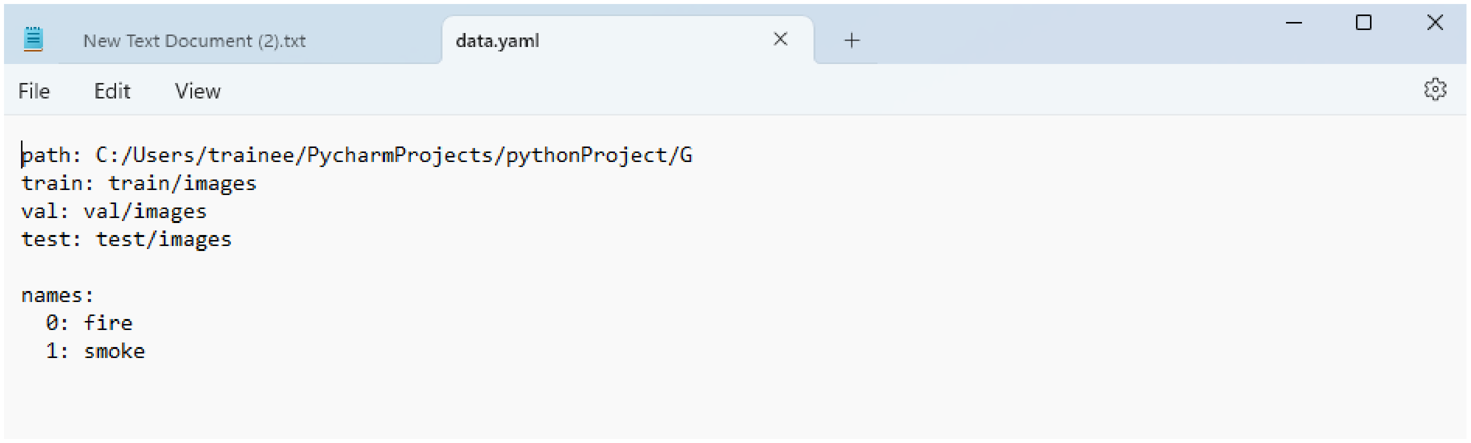
Fig.1 The training data images used for model training

As we progress, we recognize the importance of data preprocessing to optimize our model's performance. Tasks such as resizing images, normalizing pixel values, and converting annotations into the YOLO format have been integral to our preprocessing pipeline.To further bolster the efficacy of our model, we've embarked on downloading additional datasets from reputable sources on the internet. This strategic move aims to fortify our dataset, thereby minimizing error rates and enhancing the accuracy of object detection. With our dataset meticulously organized into the prescribed directory structure, we're poised to advance to the next phase of our project with confidence and determination.

The next step involves dividing the dataset of 5000+ images into three parts with a ratio of 70:20:10 for training(4181), validation(556), and testing(375) respectively. The majority of the dataset will be allocated for training the model, followed by a smaller portion for validation, and the least amount for testing to evaluate the model's performance post-training.

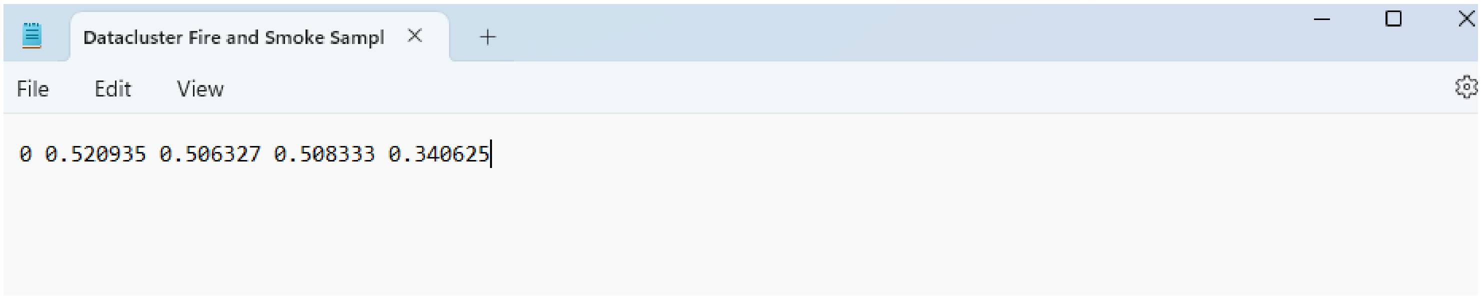
To streamline this process, a YAML file needs to be created. This file contains the paths to the directories where the images are stored for training, testing, and validation. Additionally, it includes the class names and the corresponding numbers assigned to them in the labels.Creating this YAML file will provide the necessary configuration for the model training process, ensuring smooth execution and accurate evaluation.

In our project, the YAML configuration file must be placed in the same directory as the Python file (main.py). To ensure the seamless access to the configuration parameters .



```
path: C:/Users/trainee/PycharmProjects/pythonProject/G
train: train/images
val: val/images
test: test/images

names:
  0: fire
  1: smoke
```

Fig.2 Is the data.yaml file of the smoke and fire dataset model.

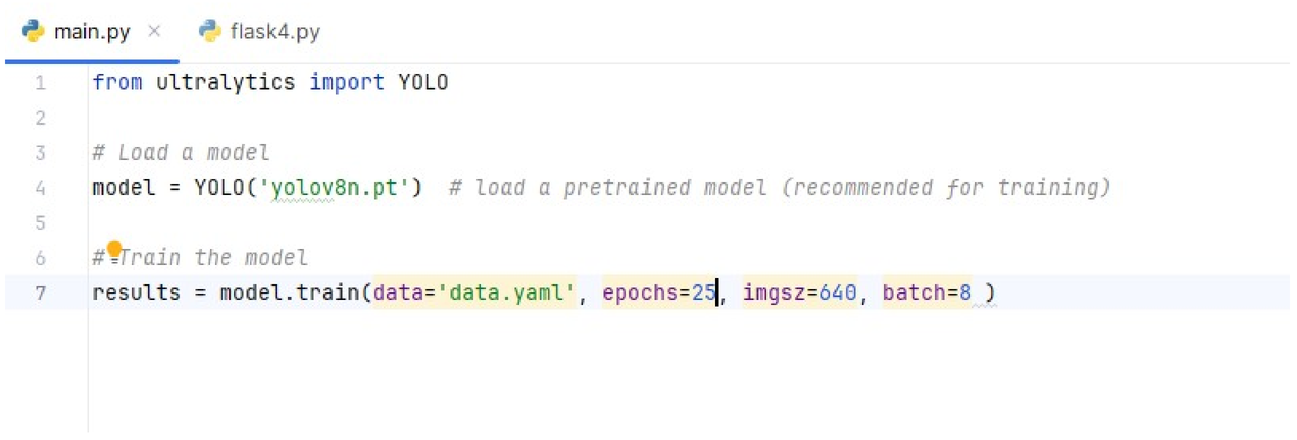


```
0 0.520935 0.506327 0.508333 0.340625
```

Fig.3 Here is the example of the images after been annotated , their labels are saved in the text file in this format

In main.py, the establishment of Ultralytics environment and import the requisite libraries is done. Following this setup, crucial decisions need to be made regarding the training process. Specifically, we determine the number of batches through which images will cycle during training, as well as the number of epochs the model will undergo to enhance accuracy.

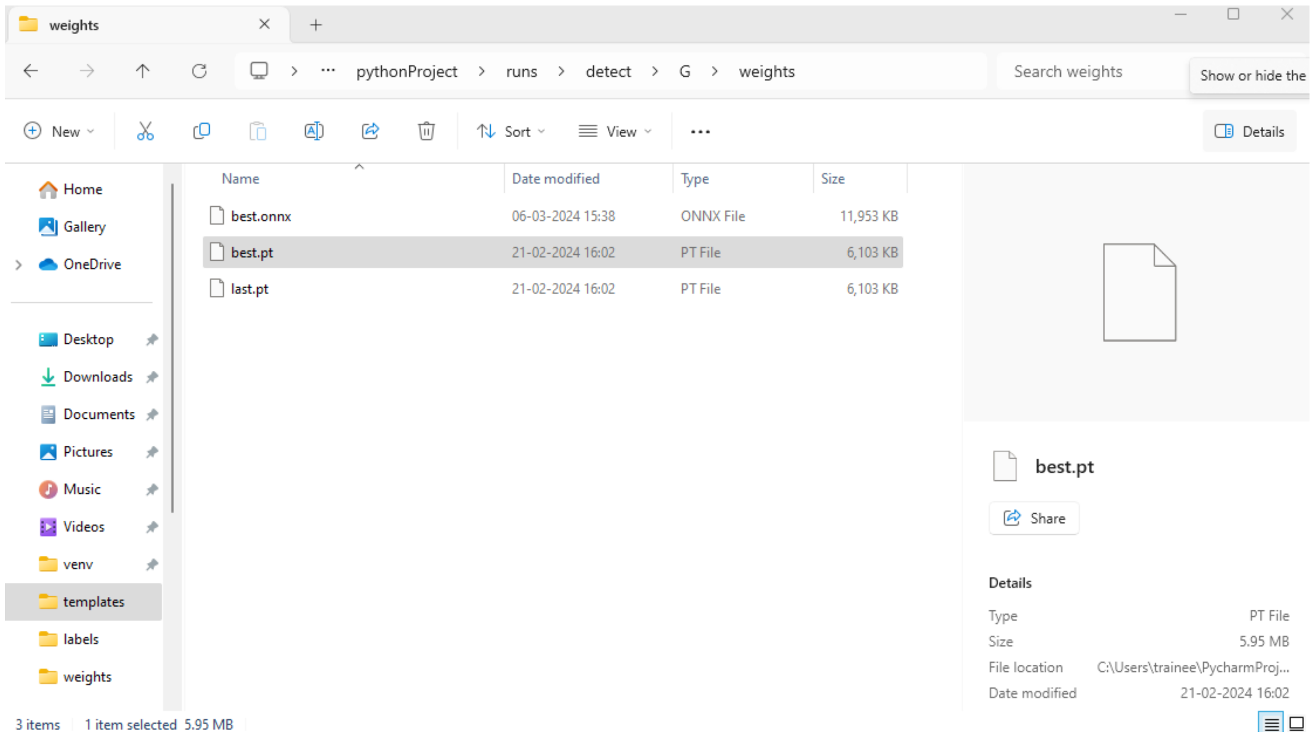
With these parameters defined, we proceed to execute the YOLOv8 code for model training.



```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load a pretrained model (recommended for training)

# Train the model
results = model.train(data='data.yaml', epochs=25, imgsz=640, batch=8 )
```

Fig.4 Code for model training & model's directory after been trained.

Now the model has been trained it will be saved in the runs/detect/model/weights/best.pt. The "best.pt" is our model file , trained on the custom dataset of smoke & fire.



Name	Date modified	Type	Size
best.onnx	06-03-2024 15:38	ONNX File	11,953 KB
best.pt	21-02-2024 16:02	PT File	6,103 KB
last.pt	21-02-2024 16:02	PT File	6,103 KB

best.pt
Share

Details

Type	PT File
Size	5.95 MB
File location	C:\Users\trainee\PycharmProj...
Date modified	21-02-2024 16:02

3 items 1 item selected 5.95 MB

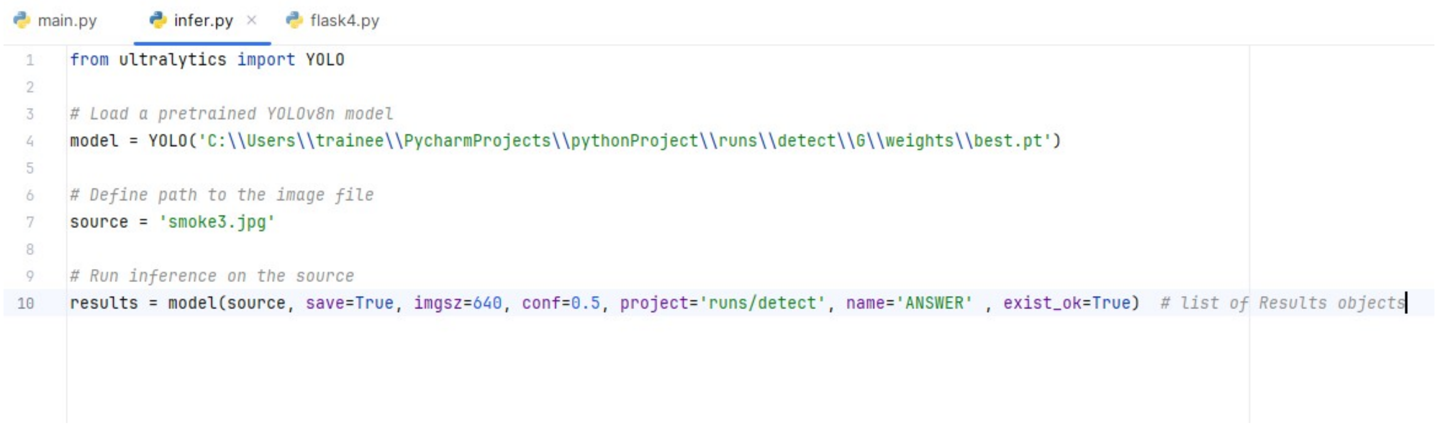
5.2 Inferencing

Inference is the application of the trained machine learning model on new data to create a result. This is the point that the model is performing the task it was designed to do in the live business environment.[6]

The process of making sense out of visual data is called 'inference' or 'prediction'. [2]

After training the model, it's imperative to ensure its functionality. To assess its performance, we conduct inference testing by predicting on images/videos to verify its accuracy.

In this process, the trained model is been loaded and test it on randomly selected images from the classes of interest. By loading the model into the environment, we evaluate its confidence scores and observe whether it accurately detects and assigns proper classes. The output of this testing phase manifests as an image or video, depending on the input file type provided for detection. The results of the inference process are stored in the directory path: runs/detect/ANSWER. Furthermore, we employ arguments and methods to tailor the results to specific requirements or modifications.



```python
from ultralytics import YOLO

# Load a pretrained YOLOv8n model
model = YOLO('C:\\Users\\trainee\\PycharmProjects\\pythonProject\\runs\\detect\\6\\weights\\best.pt')

# Define path to the image file
source = 'smoke3.jpg'

# Run inference on the source
results = model(source, save=True, imgsz=640, conf=0.5, project='runs/detect', name='ANSWER' , exist_ok=True)  # list of Results objects
```

Fig.5 Code For Inferencing

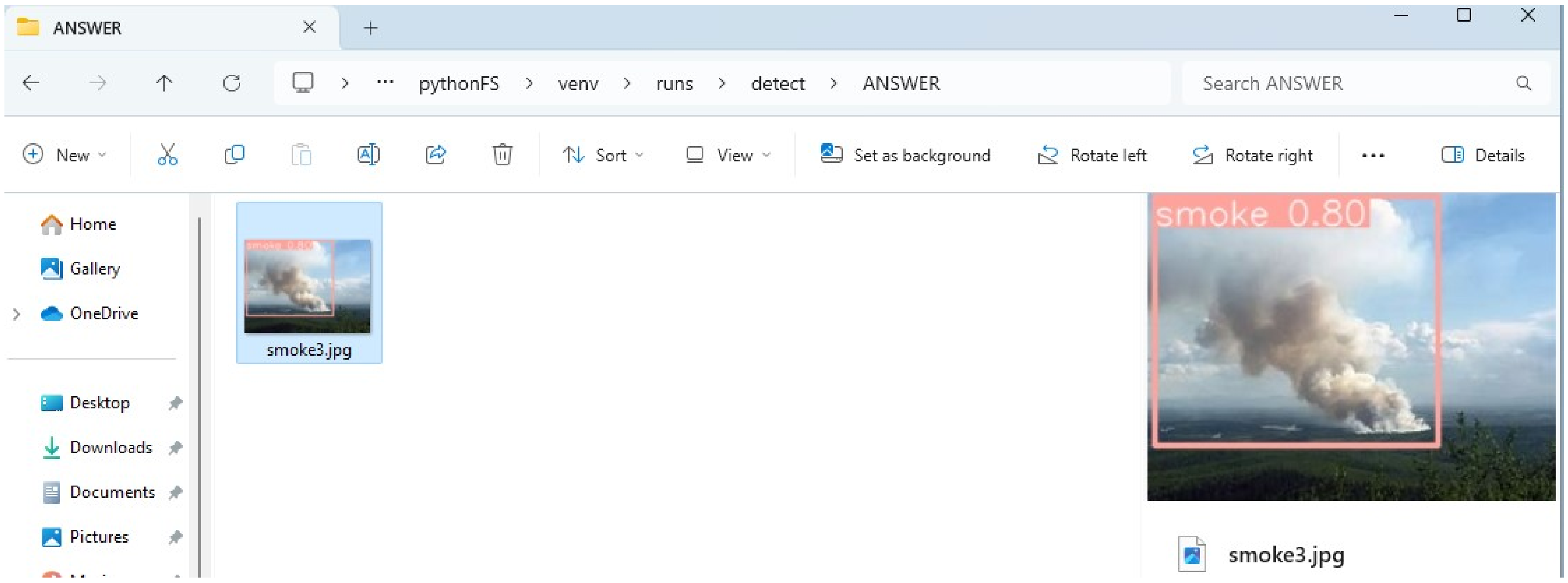


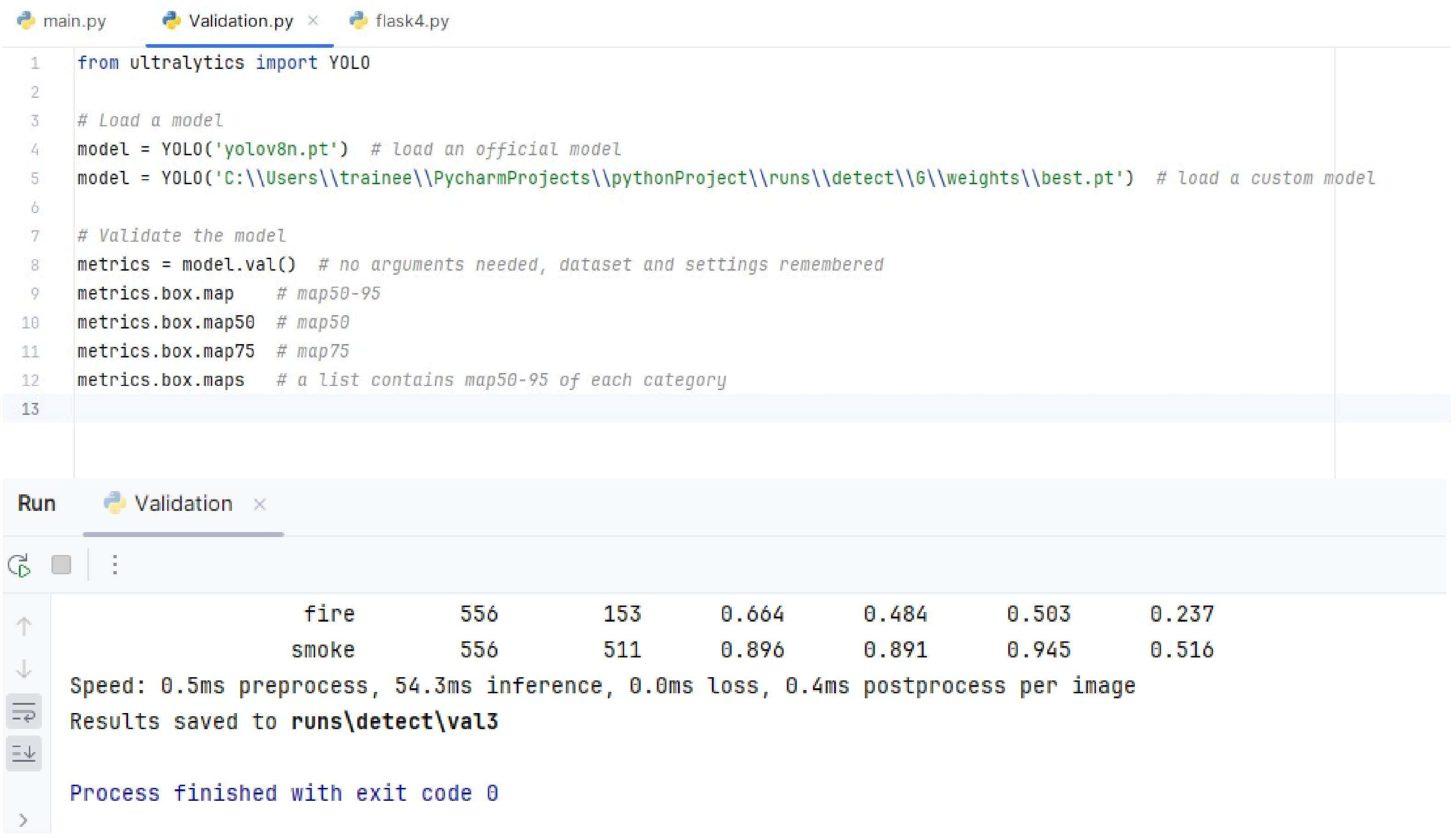
Fig.6 Result of inferencing

5.3 Validation

Validation is a critical step in the machine learning pipeline, allowing you to assess the quality of your trained models. Val mode in Ultralytics YOLOv8 provides a robust suite of tools and metrics for evaluating the performance of your object detection models. **[2]**

During validation, we assess various metrics to gauge the performance of our model, including Mean Average Precision (mAP), Intersection over Union (IoU), Confusion Matrix, Precision-Recall (P-R) Curve, and Receiver Operating Characteristic (ROC) Curve. These metrics provide valuable insights into the effectiveness of our model's training process. By analyzing these metrics, we can evaluate and judge whether our model's training has been conducted effectively. Additionally, they enable us to identify areas for improvement or configuration adjustments. Common failure modes or instances of lower accuracy can be pinpointed through this evaluation, informing future training iterations and optimization strategies.

Validation is an iterative process, and continuous monitoring and improvement are essential for robust object detection models.



```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('C:\\Users\\trainee\\PycharmProjects\\pythonProject\\runs\\detect\\6\\weights\\best.pt')  # load a custom model

# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings remembered
metrics.box.map    # map50-95
metrics.box.map50  # map50
metrics.box.map75  # map75
metrics.box.maps   # a list contains map50-95 of each category
```

```
                   fire        556        153      0.664      0.484      0.503      0.237
                  smoke        556        511      0.896      0.891      0.945      0.516
Speed: 0.5ms preprocess, 54.3ms inference, 0.0ms loss, 0.4ms postprocess per image
Results saved to runs\detect\val3

Process finished with exit code 0
```

Fig.7 Code & Results of Validation of Smoke & fire model

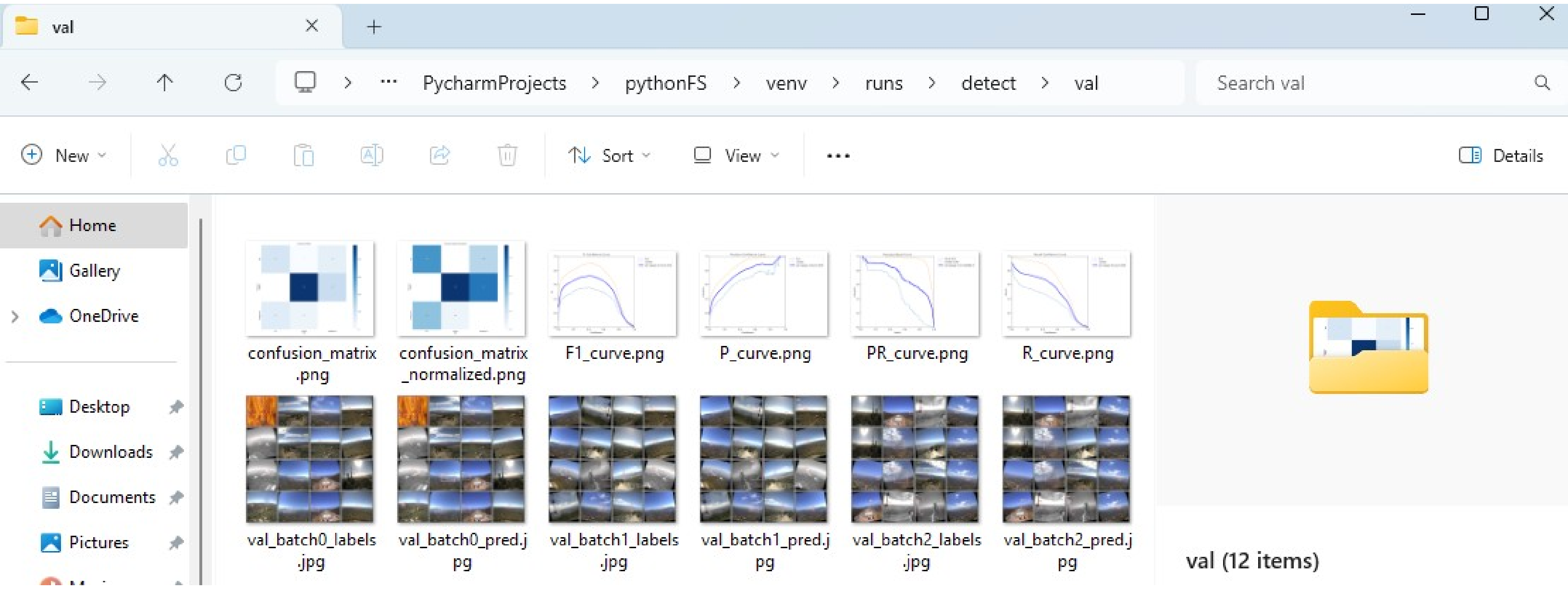


Fig.8 Validation Scores of the model

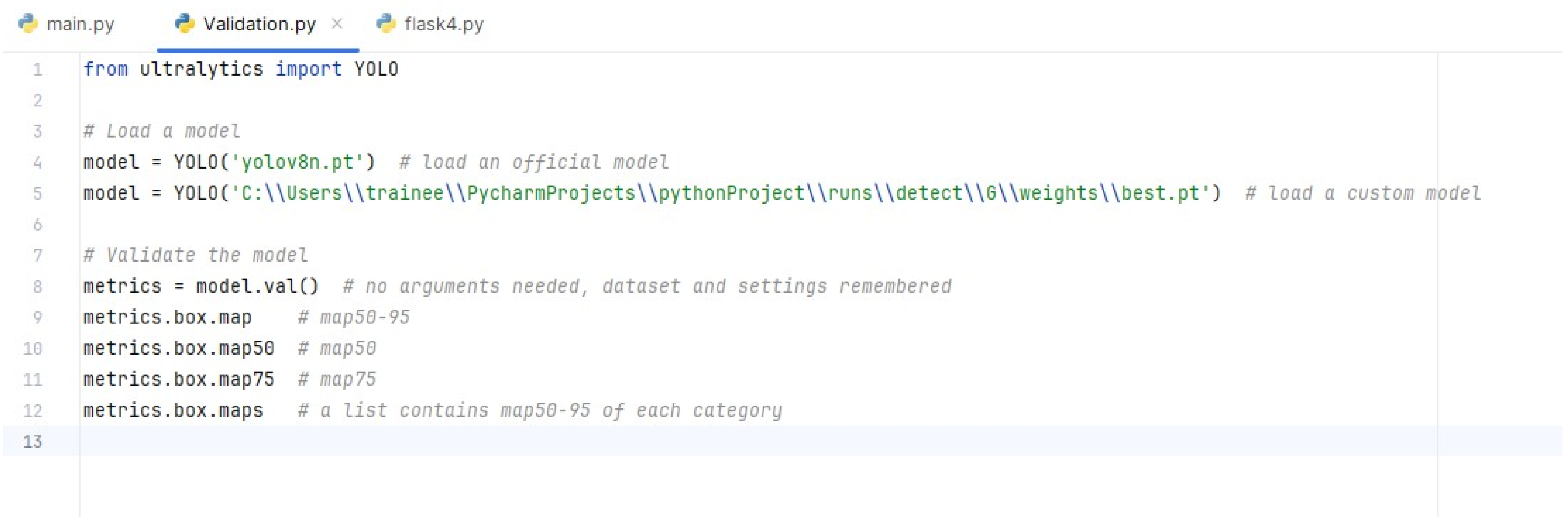
5.4 Testing/Benchmarking

A benchmark is a predetermined standard, and benchmarking/testing is the process of setting those standards. To determine benchmarks, you need to measure your work against something else which is idealized for its work. [7]

To validate the accuracy of our model and ensure its compliance with industry standards, we conduct testing on an established benchmark dataset. This dataset aligns with previously published research papers on customization and adheres to established norms and standards within the field.

Notably, the benchmark dataset is distinct from the one used during the model training phase. It is meticulously curated to ensure that none of the images or videos included for benchmarking were part of our training dataset. This approach ensures the integrity and impartiality of our evaluation process.

For this we have downloaded an ideal set from the research papers which are been idealized and used perform the validation on them.



```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('C:\\Users\\trainee\\PycharmProjects\\pythonProject\\runs\\detect\\G\\weights\\best.pt')  # load a custom model

# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings remembered
metrics.box.map    # map50-95
metrics.box.map50  # map50
metrics.box.map75  # map75
metrics.box.maps   # a list contains map50-95 of each category
```

Fig.9 Preforming Benchmarking on an ideal set

For this we have changed the dataset directory path in the data.yaml file so it took the dataset from there and perfomed the validaton from our model. That gave us the mAP values and confusion matrix and other graphes that will help us to compare the and evaluate our model.

5.5 Performance Metrices

Evaluating the performance of a Machine learning model is an important step while building an effective machine learning model. To analyze the performance or quality of the model, multiple metrics are used, and these metrics are known as performance metrics or evaluation metrics. These performance metrics help us understand how well our model has performed for the given data. We can improve the model's performance by tuning the hyper-parameters. Each ML model aims to generalize well on unseen/new data, and performance metrics help determine how well the model generalizes on the new dataset. [14]

Precision - Precision is the ratio of accurately predicted positive observations to the overall predicted positive observations. It measures classifier's exactness. High precision relates to a low false positive rate.

$$Precision=TP/(TP+FP)$$

where (TP) is the number of true positives and (FP) is the number of false positives.

Recall – Recall/Sensitivity is the ratio of correctly predicted positive observations to all observations in the actual class. It is an estimate of a classifier's completeness. High recall relates to a low false negative rate.

$$Recall=TP/(TP+FN)$$

where (TP) is the number of true positives and (FN) is the number of false negatives.

F1 score - The F1 score is a metric used in machine learning to evaluate the accuracy of a classification model, especially when the class distribution is uneven. It considers both the precision and the recall of the test to compute the score. The F1 score is the harmonic mean of precision and recall, providing a balance between the two by penalizing extreme values.

$$F1= 2 \times precision*recall/(precision+recall)$$

Confidence score - A confidence score is a parameter that reflects the probability that a given prediction made by a model is correct. It's a numerical value written in percentage or a number between 0 and 1, where a higher score indicates greater confidence in the prediction.

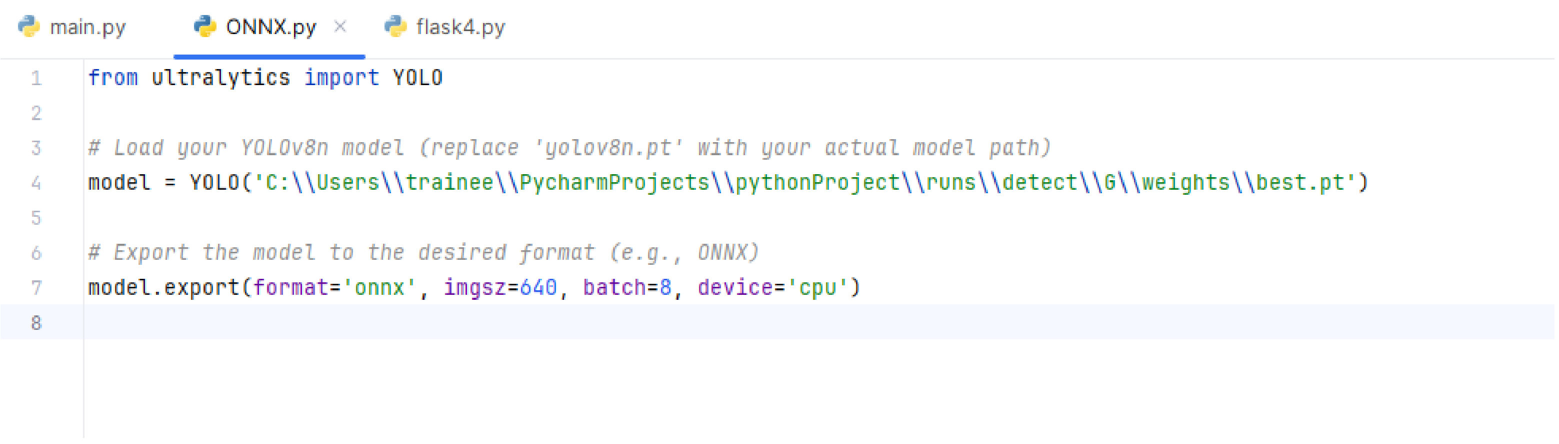
mAP value - In machine learning, particularly in the context of object detection and segmentation, mAP stands for Mean Average Precision. It is a parameter used to analyze the performance of models on these tasks. This is the mean of the Average and Precision scores for all classes or over different IoU thresholds. The mAP score provides a single value that captures both the precision and recall of the model, which is important when the model is detecting multiple objects and classes. A higher mAP score shows better performance of the model in terms of accuracy and reliability.

5.6 Export

Following the completion of training, the YOLOv8 model is exported into a deployable format, streamlining the deployment process. Specifically, our smoke & fire dataset model has been exported into the ONNX format, which facilitates seamless deployment.

Throughout this process, we have leveraged a range of arguments and methods to customize the exportation procedure according to our specific requirements and objectives.

ONNX (Open Neural Network Exchange) is an open format for representing deep learning models. It's designed to allow interoperability between different deep learning frameworks. ONNX enables models to be trained in one framework and then transferred to another for inference or further processing.[5]



```python
from ultralytics import YOLO

# Load your YOLOv8n model (replace 'yolov8n.pt' with your actual model path)
model = YOLO('C:\\Users\\trainee\\PycharmProjects\\pythonProject\\runs\\detect\\G\\weights\\best.pt')

# Export the model to the desired format (e.g., ONNX)
model.export(format='onnx', imgsz=640, batch=8, device='cpu')
```

Fig.10 Code for exporting the model into ONNX format

5.7 Deployment

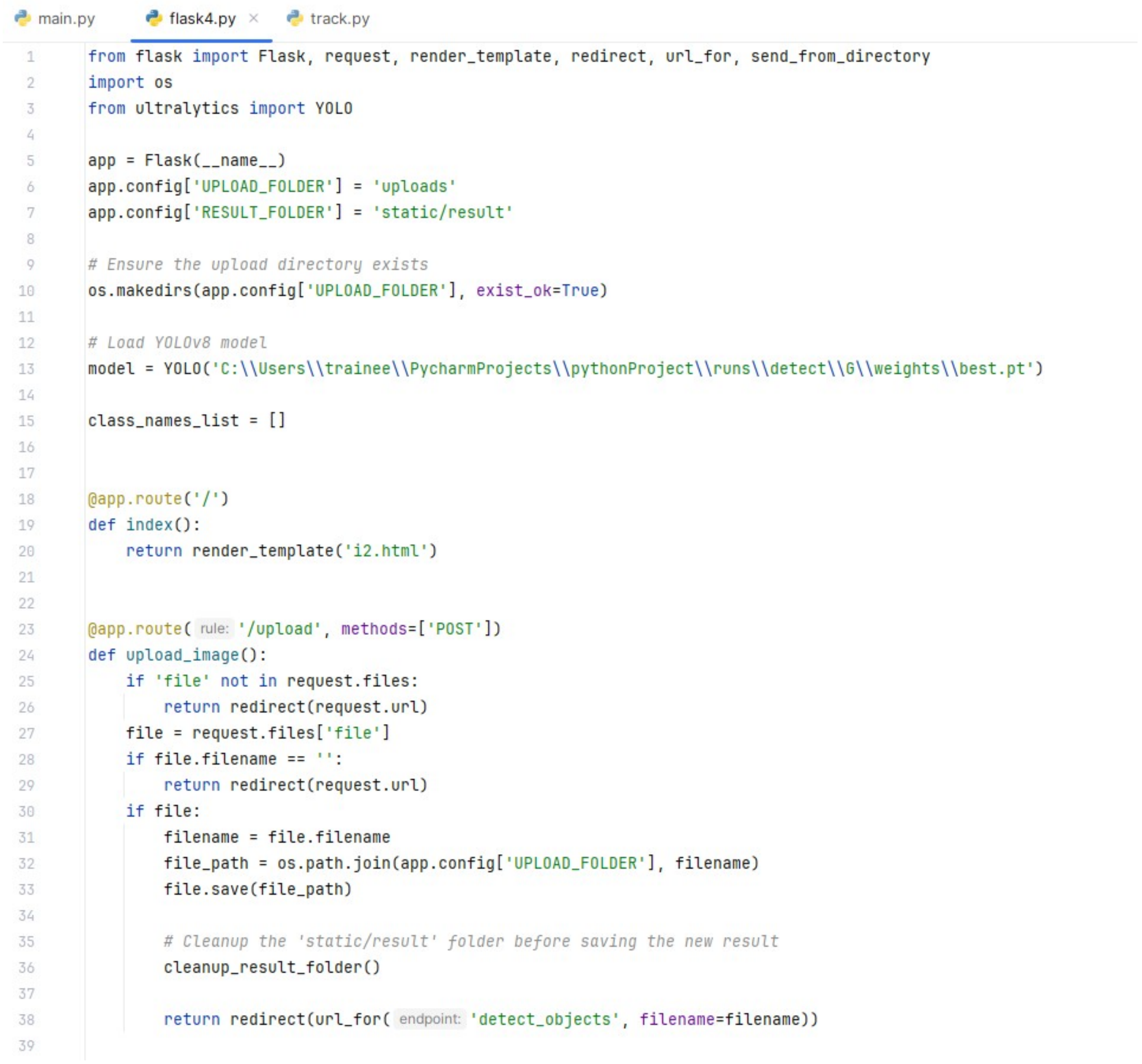
Deployment involves making a software application or system available for use in a production environment. It encompasses various steps, including configuring, testing, and releasing the software to ensure it meets desired requirements and functions correctly.

For deploying our YOLOv8 model on the server, we've opted for Python Flask. The deployment process involves creating two HTML files: index.html and result.html, both stored in the 'templates' folder within the virtual environment (venv). index.html serves as the main interaction page, where users upload files for detection. Conversely, result.html displays the detection results.

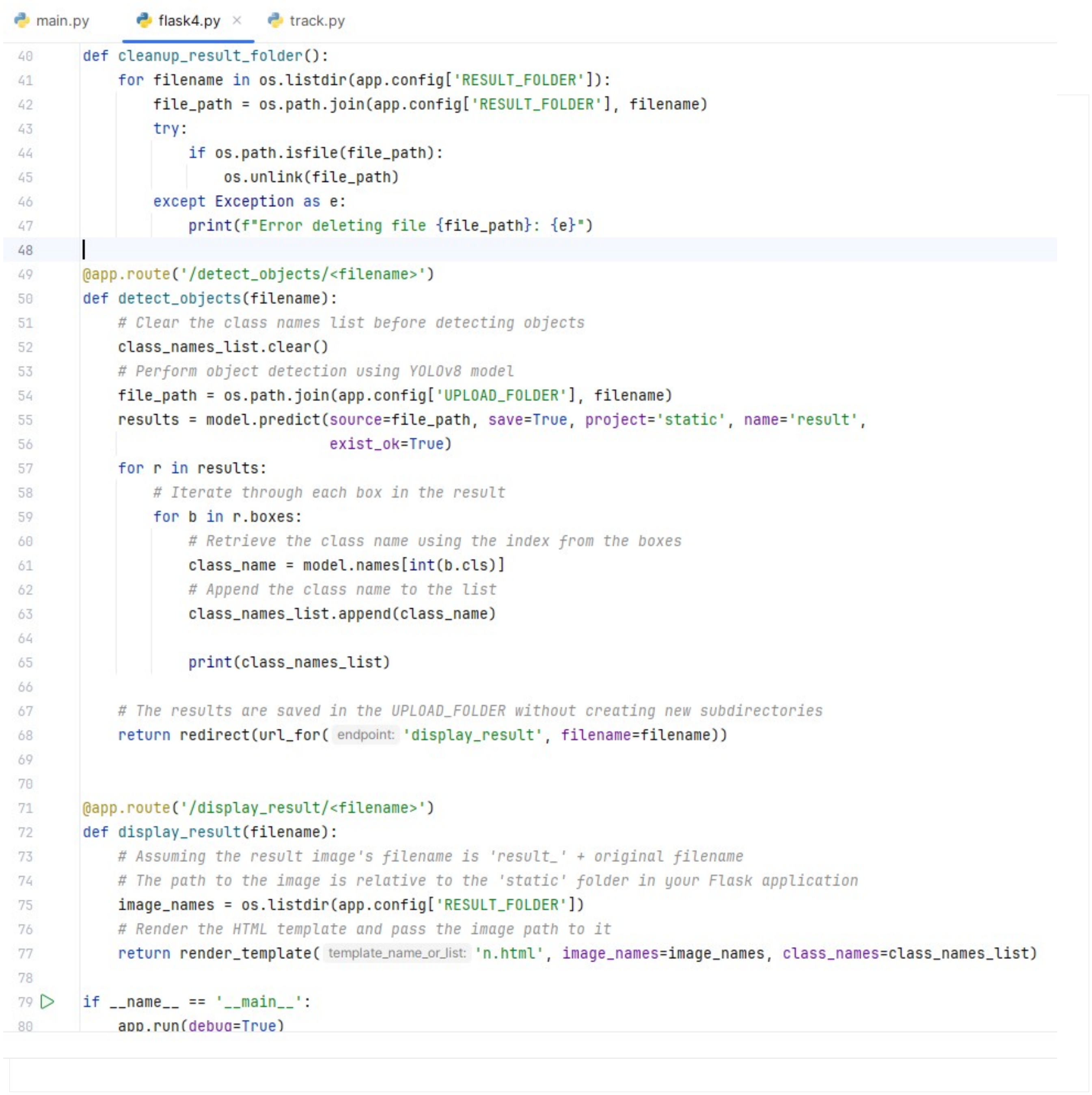
We've introduced two additional folders: 'static', which includes a subfolder named result. Detected images are stored here and accessed by Flask and HTML files for result display. The second folder, uploads, serves as an input repository for user-uploaded files.

Deployment Workflow

1. Import necessary libraries.
2. Start with the index page, where users provide input and inference occurs.
3. Upon clicking the detect button, users are redirected to the result page.
4. The result page displays the detected image.



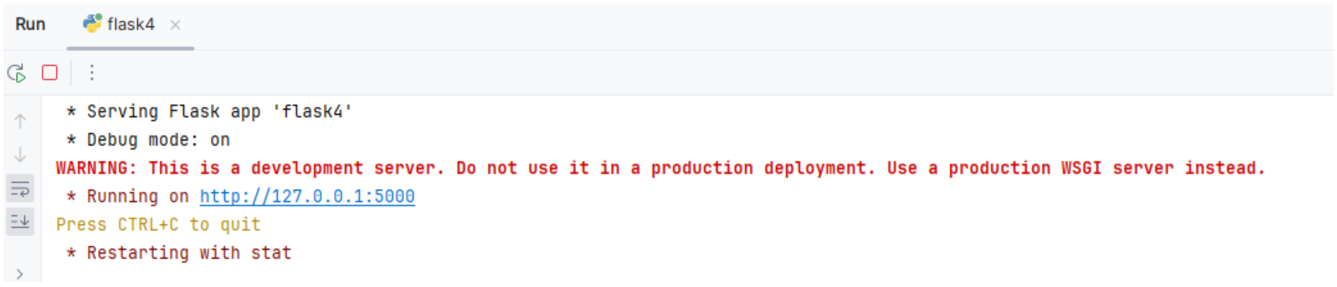
```python
from flask import Flask, request, render_template, redirect, url_for, send_from_directory
import os
from ultralytics import YOLO

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['RESULT_FOLDER'] = 'static/result'

# Ensure the upload directory exists
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# Load YOLOv8 model
model = YOLO('C:\\Users\\trainee\\PycharmProjects\\pythonProject\\runs\\detect\\6\\weights\\best.pt')

class_names_list = []


@app.route('/')
def index():
    return render_template('i2.html')


@app.route( rule: '/upload', methods=['POST'])
def upload_image():
    if 'file' not in request.files:
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        return redirect(request.url)
    if file:
        filename = file.filename
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        # Cleanup the 'static/result' folder before saving the new result
        cleanup_result_folder()

        return redirect(url_for( endpoint: 'detect_objects', filename=filename))

```



```
main.py    flask4.py ×    track.py
```

```python
def cleanup_result_folder():
    for filename in os.listdir(app.config['RESULT_FOLDER']):
        file_path = os.path.join(app.config['RESULT_FOLDER'], filename)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
        except Exception as e:
            print(f"Error deleting file {file_path}: {e}")

@app.route('/detect_objects/<filename>')
def detect_objects(filename):
    # Clear the class names list before detecting objects
    class_names_list.clear()
    # Perform object detection using YOLOv8 model
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    results = model.predict(source=file_path, save=True, project='static', name='result',
                            exist_ok=True)
    for r in results:
        # Iterate through each box in the result
        for b in r.boxes:
            # Retrieve the class name using the index from the boxes
            class_name = model.names[int(b.cls)]
            # Append the class name to the list
            class_names_list.append(class_name)

            print(class_names_list)

    # The results are saved in the UPLOAD_FOLDER without creating new subdirectories
    return redirect(url_for( endpoint: 'display_result', filename=filename))


@app.route('/display_result/<filename>')
def display_result(filename):
    # Assuming the result image's filename is 'result_' + original filename
    # The path to the image is relative to the 'static' folder in your Flask application
    image_names = os.listdir(app.config['RESULT_FOLDER'])
    # Render the HTML template and pass the image path to it
    return render_template( template_name_or_list: 'n.html', image_names=image_names, class_names=class_names_list)

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

Fig.11 Code of depolying the model via python flask



```
Run    flask4 ×
```

```
 * Serving Flask app 'flask4'
 * 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
```

Fig.12 On ruuning this code this creates a localhost

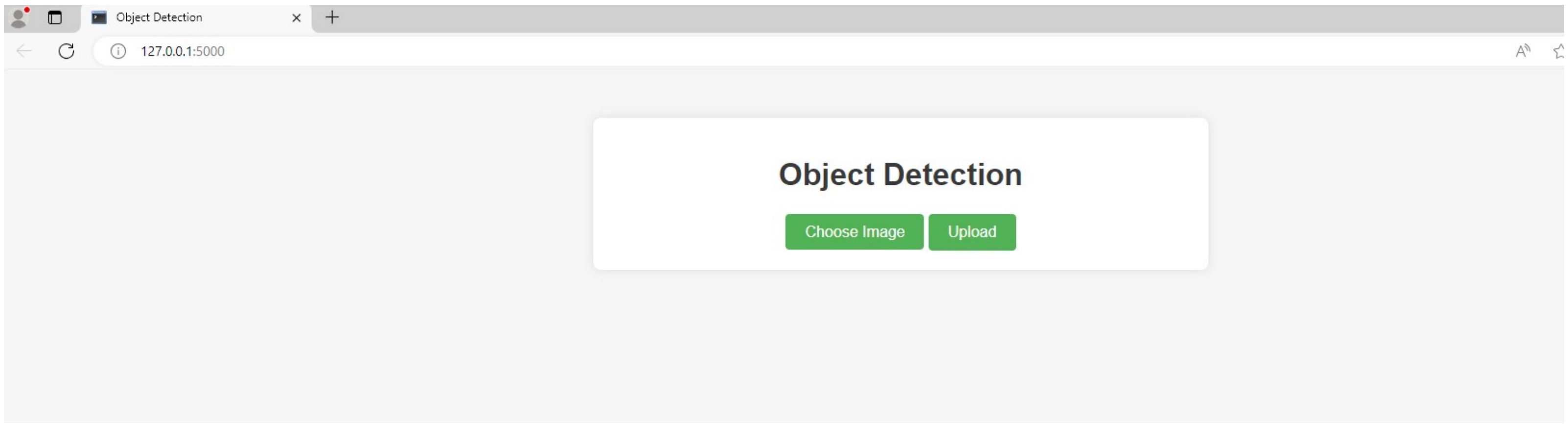
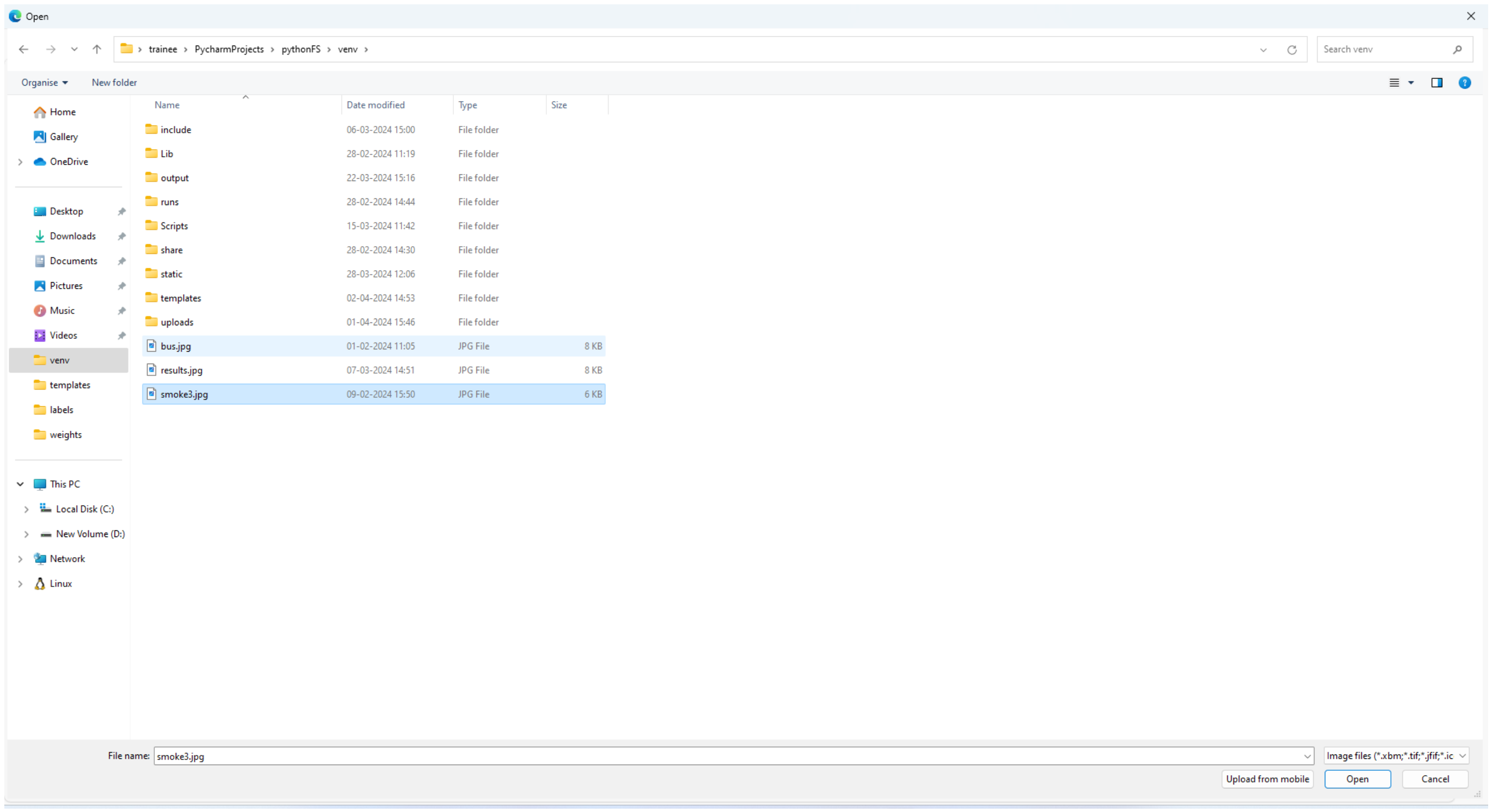


Fig.13 On opening the local host this HTML file is opened where the detection is done

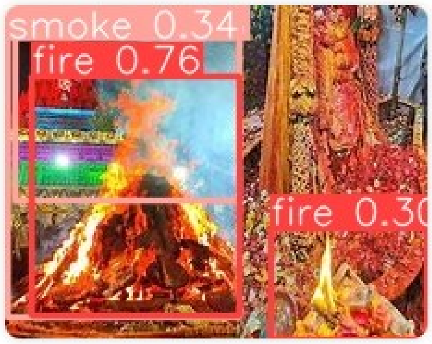


Name	Date modified	Type	Size
include	06-03-2024 15:00	File folder	
Lib	28-02-2024 11:19	File folder	
output	22-03-2024 15:16	File folder	
runs	28-02-2024 14:44	File folder	
Scripts	15-03-2024 11:42	File folder	
share	28-02-2024 14:30	File folder	
static	28-03-2024 12:06	File folder	
templates	02-04-2024 14:53	File folder	
uploads	01-04-2024 15:46	File folder	
bus.jpg	01-02-2024 11:05	JPG File	8 KB
results.jpg	07-03-2024 14:51	JPG File	8 KB
smoke3.jpg	09-02-2024 15:50	JPG File	6 KB

File name: smoke3.jpg

Fig.14 Uploading the image which is to be detected

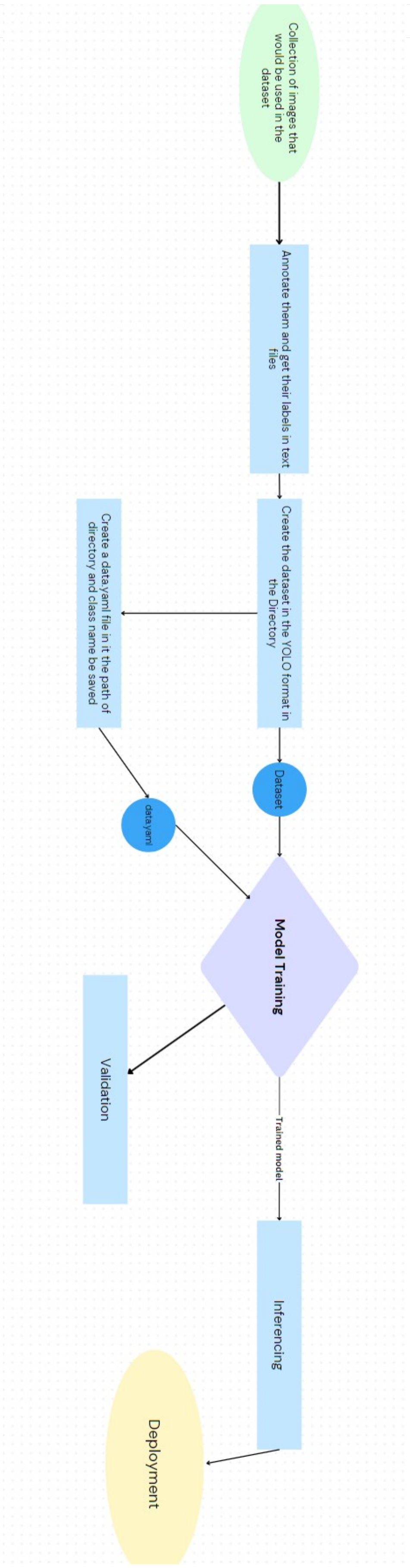
Detected Object



- Detected Class: ['fire', 'smoke', 'fire', 'smoke']

Fig.15 Result output

Fig.16 Workflow of the model


Collection of images that would be used in the dataset
Annotate them and get their labels in text files
Create the dataset in the YOLO format in the Directory
Create a data.yaml file in it the path of directory and class name be saved
Dataset
data.yaml
Model Training
Validation
Trained model
Inferencing
Deployment

6. Results

Our experiments yielded promising results, with the YOLOv8 model achieving a precision of 93.6%, recall of 91.3%, mAP score of 94.5% , and an F1 score of 73.0% on the test dataset. These metrics indicate the model's ability to effectively identify smoke and fire regions while minimizing false positives.

A qualitative assessment has been conducted that visually inspects the model's predictions on sample images from the test set. The YOLOv8 model consistently demonstrated accurate localization and classification of smoke and fire instances across various environmental conditions and image complexities.

Real-Time Deployment Performance

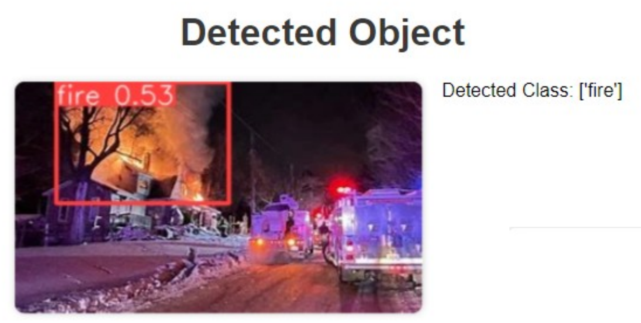
To evaluate the models performance offline, we deployed the trained YOLOv8 model using Python Flask to assess its real-time performance. The deployed system successfully processed incoming image data and provided timely smoke and fire detection results via a web interface.

The tests showed us that the deployed system could handle multiple concurrent requests efficiently, it demonstrate its suitability for real-world applications requiring rapid detection and response to fire incidents.

Comparative Analysis

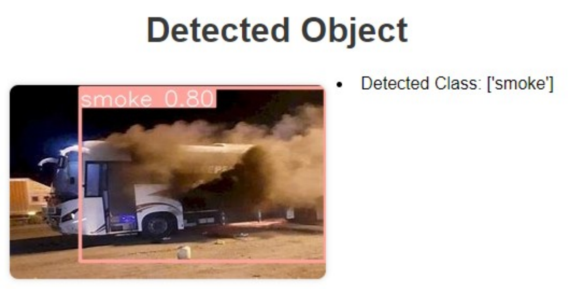
We conducted a comparative analysis with other state-of-the-art object detection models for smoke and fire detection. The results indicate that YOLOv8 performing excellently on several baseline models in terms of both accuracy and speed, reaffirming its effectiveness for smoke and fire detection tasks.

Below are some graphes and matrices through which the model could be compared on the standards and the accuracy and efficency of it be checked.



Detected Object

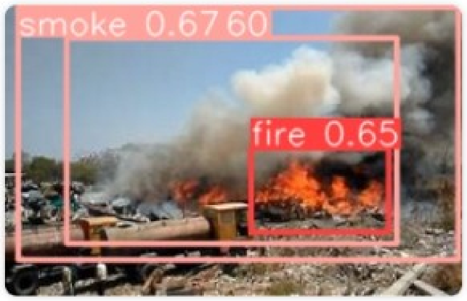
Detected Class: ['fire']



Detected Object

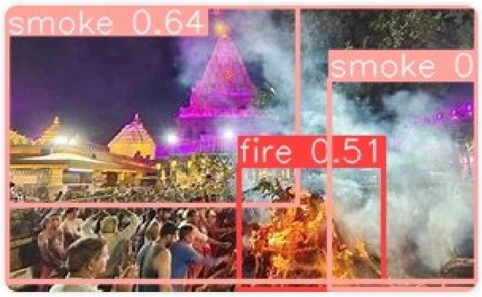
- Detected Class: ['smoke']

Detected Object



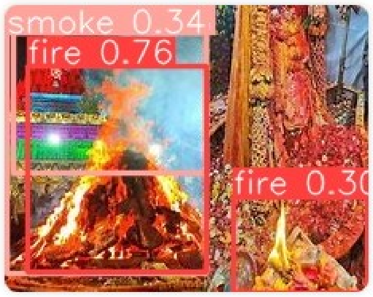
- Detected Class: ['smoke', 'fire', 'smoke']

Detected Object



- Detected Class: ['smoke', 'smoke', 'fire', 'smoke']

Detected Object



- Detected Class: ['fire', 'smoke', 'fire', 'smoke']

Fig.17 Result images from our model

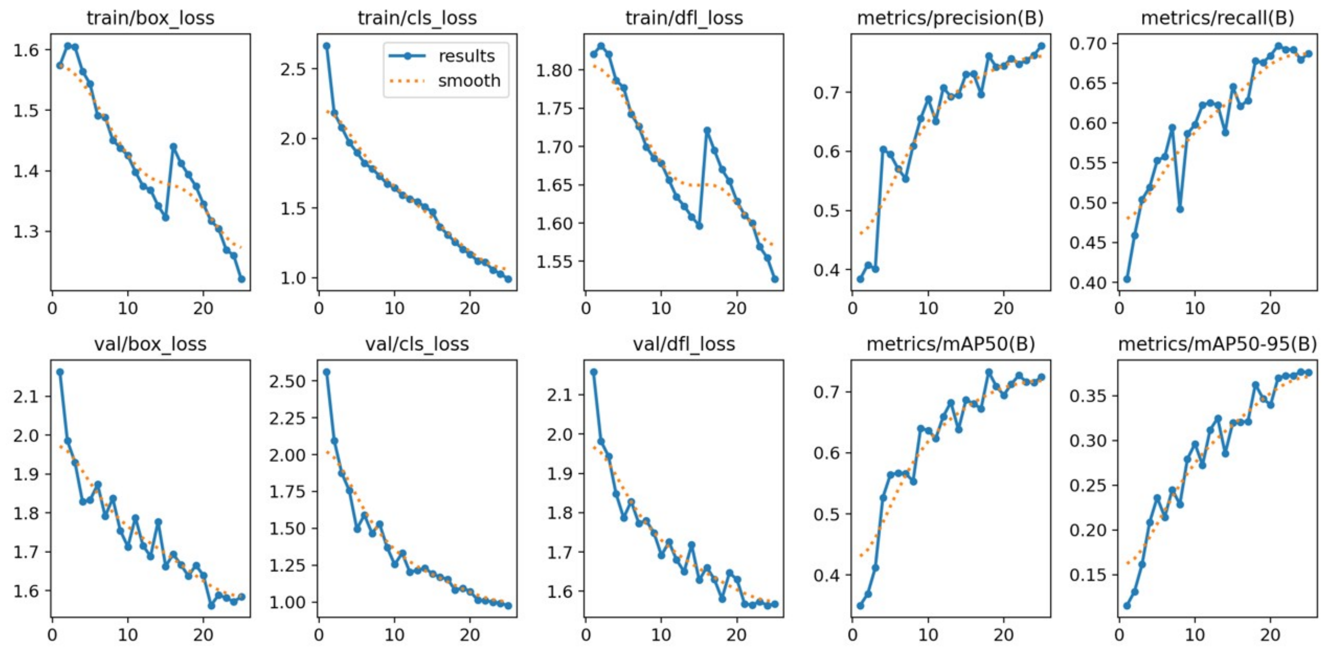


Fig.18 Result Graphes of the trained model

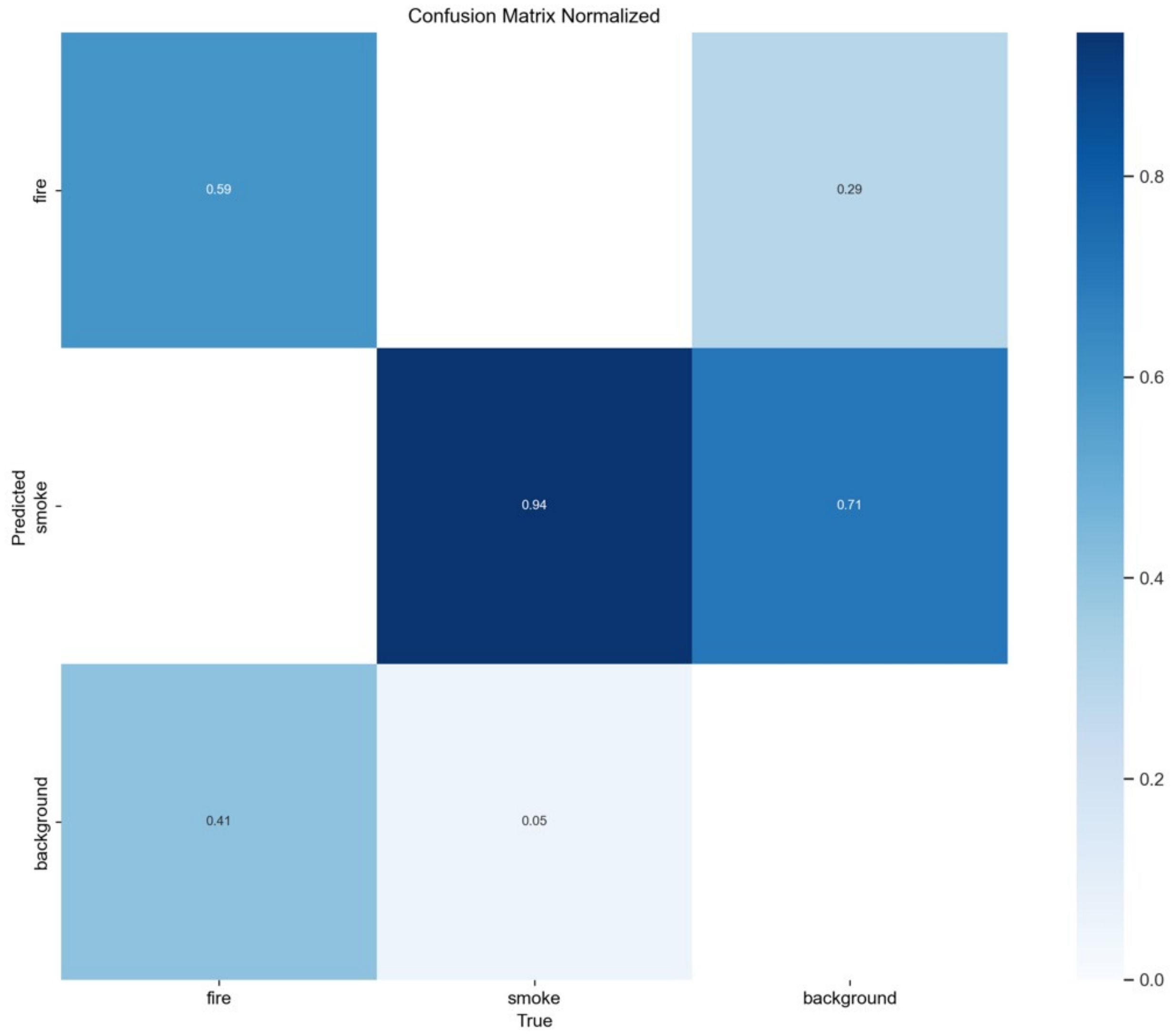


Fig.19 Normalized Confusion matrix of the trained model

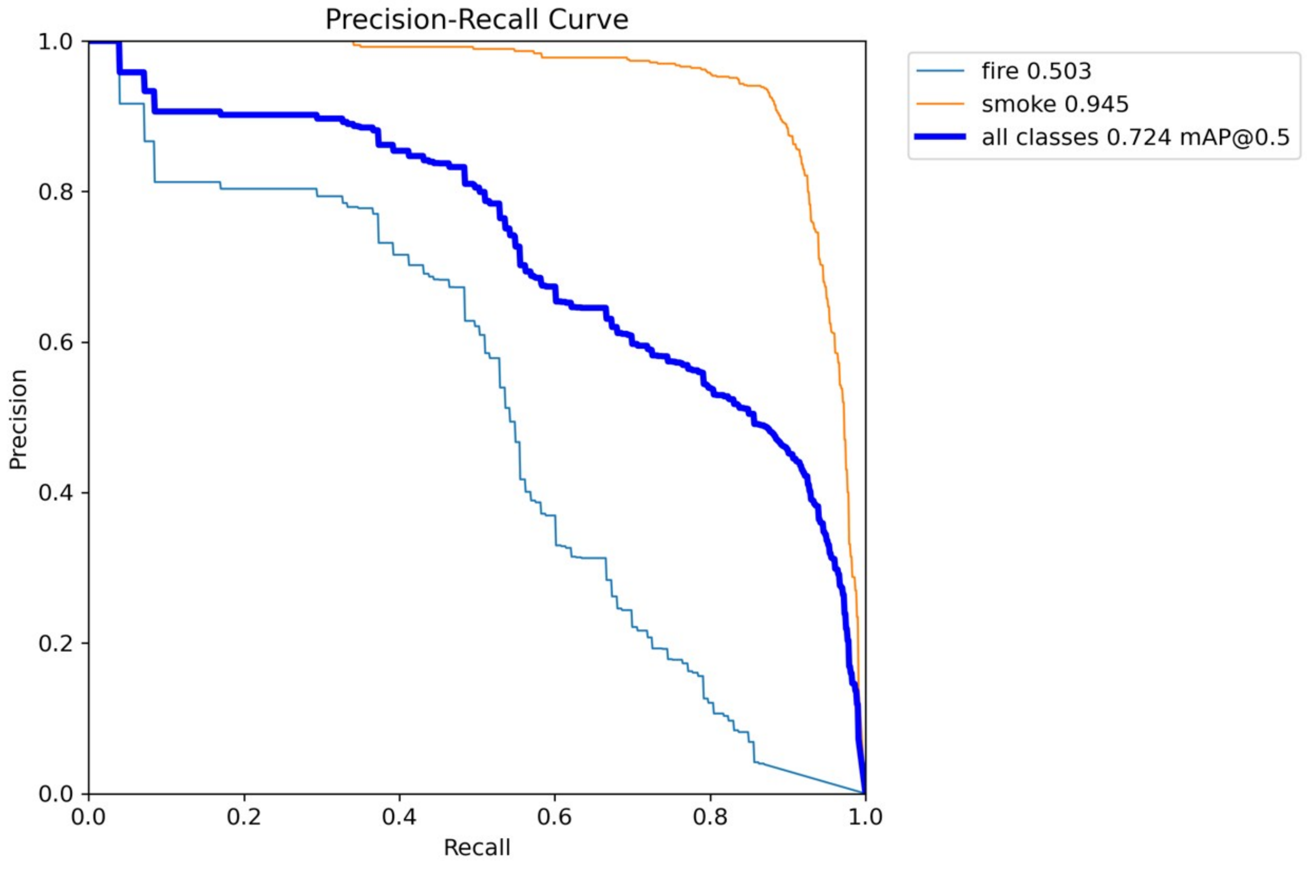


Fig.20 PR Curve

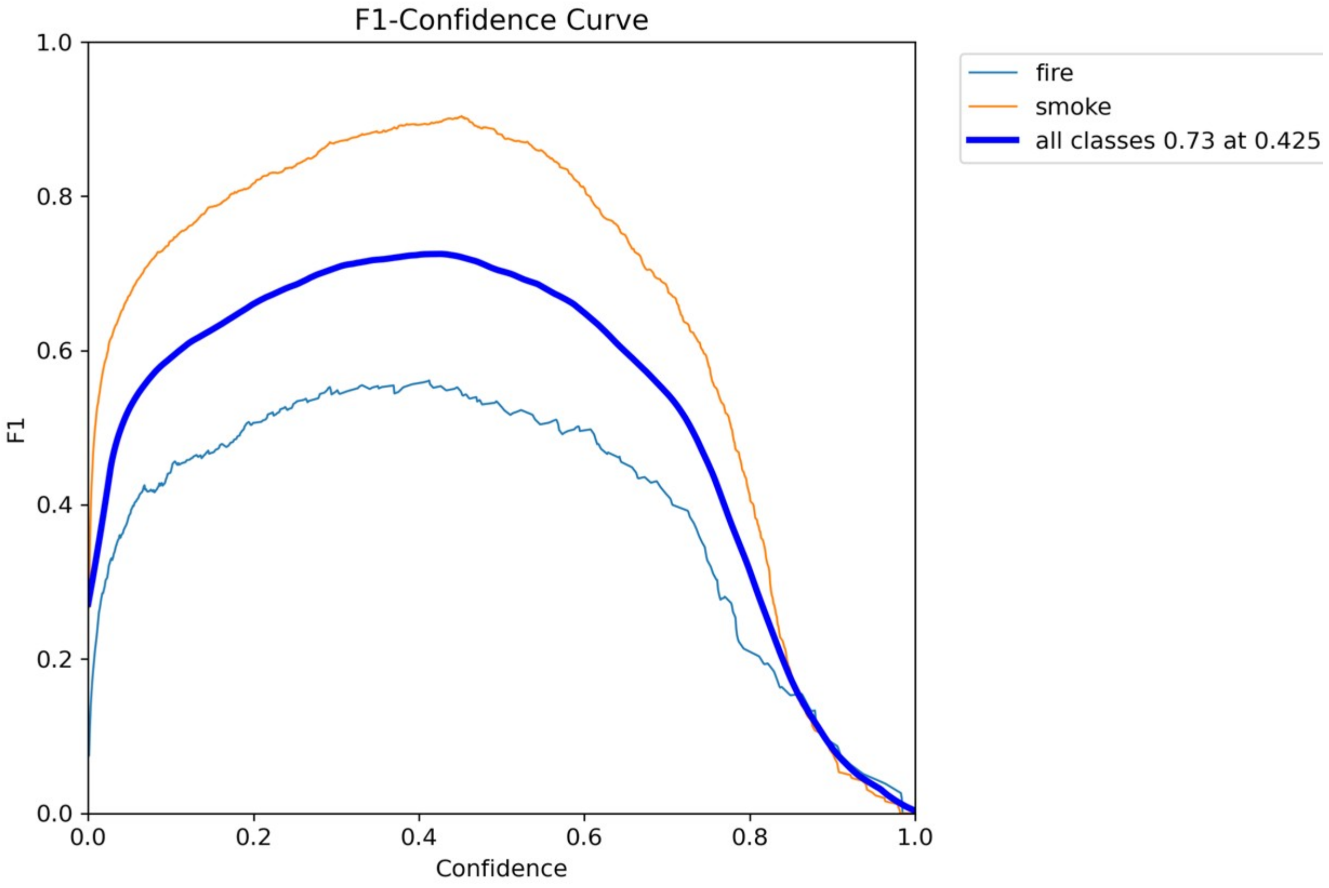


Fig.21 F1-Confidence Curve

7. Future Use

In the future this model could be used to make smart homes even smarter. This technology could help to detect fires before they spread, keeping families safe and homes secure. It could be handy in industrial settings, helping to prevent accidents and save lives. With this model, we could make workplaces safer and protect workers from potential hazards.

1. Integration with IoT Devices: We can integrate our model with the fire detection system with Internet of Things (IoT) devices. By integrating sensors and cameras into IoT networks, the system can provide real-time monitoring of fire hazards in smart buildings, homes, and industrial facilities. This integration enables dynamic fire prevention measures and improves overall safety and security.

2. Autonomous Firefighting Systems: After Further optimization and refining, the smoke and fire detection system could serve as a key component of autonomous firefighting systems. By adding the detection capabilities with robotic platforms equipped with fire suppression tools, such as water cannons or extinguishing agents, the system can autonomously detect and extinguish fires in hazardous environments, reducing the risk to human firefighters and minimizing property damage.

3. Application in Urban Planning and Disaster Management: The data collected by the smoke and fire detection system can be valuable for urban planning and disaster management initiatives. By analyzing historical fire incident data, city planners can identify high-risk areas and implement preventive measures to mitigate the impact of future fire events. Additionally, the system can aid emergency responders in coordinating evacuation efforts and allocating resources more effectively during fire emergencies.

The immense potential for future advancements and diverse applications is held by the smoke and fire detection system developed in our internship project. Through leveraging emerging technologies and innovative approaches, fire safety measures can be further enhanced, the impact of fire incidents can be minimized, and contributions can be made to the resilience of communities and infrastructure worldwide.

8. Discussion

Object detection using YOLOv8, developed by Ultralytics, has demonstrated remarkable performance in detecting smoke and fire in images. Throughout our internship, we trained the YOLOv8 model on a dataset specifically curated for smoke and fire detection.

One of the key strengths of YOLOv8 is its real-time processing capability, making it highly suitable for applications requiring timely detection, such as fire monitoring systems. The model's architecture, which employs a single convolutional neural network to simultaneously predict bounding boxes and class probabilities, contributes to its efficiency and accuracy.

Our experiments revealed that YOLOv8 achieved impressive results in detecting smoke and fire instances within images, even in challenging environmental conditions such as varying lighting and occlusions. This robustness is crucial for real-world deployment, where the detection system must perform reliably under diverse circumstances.

However, it's important to note that while YOLOv8 excels in real-time detection, there is still room for improvement, particularly in scenarios with small or heavily occluded smoke and fire instances. Fine-tuning the model architecture and augmenting the dataset with additional diverse examples could potentially enhance its performance further.

Furthermore, the deployment of YOLOv8 in real-world settings may pose challenges related to computational resources and model optimization, especially when considering embedded systems or resource-constrained environments. Addressing these challenges will be essential for the widespread adoption of YOLOv8-based fire detection systems.

In our internship project, we leveraged the capabilities of Python Flask to deploy the trained YOLOv8 model as a web service. This deployment setup enables seamless integration of the fire and smoke detection system into existing applications and frameworks. The use of Flask provides flexibility and scalability, allowing the detection system to handle multiple simultaneous requests efficiently.

In conclusion, our experience with YOLOv8 for smoke and fire detection, coupled with the deployment using Python Flask, has been highly promising. The model's efficiency, accuracy, and real-time processing capabilities, combined with the flexibility of Flask deployment, make it a valuable tool for fire monitoring and emergency response systems, with the potential to save lives and mitigate property damage.

9. Conclusion

In this internship project has been contributed in exploring the potential of YOLOv8 for smoke and fire detection, by its real-time deployment using Python Flask. Through a combination of rigorous experiments and practical implementations, we've gained valuable insights into the capabilities of deep learning in addressing critical challenges in fire monitoring and emergency response systems.

The results of our experiments underscore the effectiveness of YOLOv8 in accurately detecting smoke and fire instances within images. The successful creation of the model the YOLOv8 model achieving a precision of 93.6%, recall of 91.3%, and an F1 score of 73.0% on the test dataset & mAP score of 94.5% gave us high accuracy in the objected detection of the dmoke and fire. And with the not much large dataset but of 5000+ images with 4181 in training, 556 in validation, and 375 in testing, images were used to create the optimized model.

Furthermore, the real-time deployment of the model using Python Flask provides a scalable and efficient solution for integrating fire detection capabilities into existing applications and frameworks. The system's ability to handle multiple simultaneous requests in a timely manner enhances its suitability for deployment in diverse real-world scenarios requiring rapid detection and response.

This led us to see and acheieve greater heights in the field of fire safety that created a great awareness , enlighten & gave inspiration so the journey of creation of this model got started.

In conclusion, our internship journey has been both enlightening and rewarding, providing us with valuable practical experience and insights that will undoubtedly shape our future endeavors in the field of computer vision and artificial intelligence.

10. References

[1] Object Detection - MATLAB & Simulink - MathWorks Switzerland

https://ch.mathworks.com/help/vision/object-detection.html

[2] Home - Ultralytics YOLOv8 Docs

https://docs.ultralytics.com/

[3]YOLO Algorithm for Object Detection Explained [+Examples] (v7labs.com)

https://www.v7labs.com/blog/yolo-object-detection

[4] Introduction to Web development using Flask - GeeksforGeeks

https://www.geeksforgeeks.org/python-introduction-to-web-development-using-flask/

[5] PyCharm - Wikipedia

https://en.wikipedia.org/wiki/PyCharm

[6] What is AI inferencing? | IBM Research Blog

https://research.ibm.com/blog/AI-inference-explained

[7] Use Benchmarking to Set Your Standards for Success [2024] • Asana

https://asana.com/resources/benchmarking

[8] YOLOv8 for Fire and Smoke Recognition Algorithm Integrated with the Convolutional Block Attention Module (scirp.org)

https://www.scirp.org/journal/paperinformation?paperid=130929

[9] Real-time smoke detection with Faster R-CNN | 2021 2nd International Conference on Artificial Intelligence and Information Systems (acm.org)

https://dl.acm.org/doi/abs/10.1145/3469213.3471342

[10] YOLOv3-Cloud-Based-Fire-Detection/YOLOv3_Train_Fire.ipynb at master · snehitvaddi/YOLOv3-Cloud-Based-Fire-Detection · GitHub

https://github.com/snehitvaddi/YOLOv3-Cloud-Based-Fire-Detection/blob/master/YOLOv3_Train_Fire.ipynb

[11] (PDF) An Improved Wildfire Smoke Detection Based on YOLOv8 and UAV Images (researchgate.net)

https://www.researchgate.net/publication/374591996_An_Improved_Wildfire_Smoke_Detection_Based_on_YOLOv8_and_UAV_Images

[12] Train YOLOv8 On A Custom Dataset For Fire And Smoke Detection | by Kazi Mushfiqur Rahman | Medium

https://medium.com/@KaziMushfiq1234/train-yolov8-on-a-custom-dataset-for-fire-and-smoke-detection-d9d0696fdcd7

[13] Electronics | Free Full-Text | Wildfire and Smoke Detection Using Staged YOLO Model and Ensemble CNN (mdpi.com)

https://www.mdpi.com/2079-9292/12/1/228

[14] Precision and Recall in Machine Learning - Javatpoint

https://www.javatpoint.com/precision-and-recall-in-machine-learning