

FireGuardian: YOLOv8 Empowered Wildfire Smoke Surveillance

FireGuardian is a state-of-the-art project that integrates the YOLOv8 (You Only Look One-level) algorithm to empower wildfire smoke surveillance. The system employs real-time object detection to identify and monitor the presence of wildfire smoke, providing timely information for effective response and mitigation strategies.

Scenario 1: Early Detection and Alerting

FireGuardian can be deployed in wildfire-prone areas to detect smoke at its early stages. The YOLOv8 algorithm enables quick and accurate identification of smoke patterns, triggering automated alerts to emergency services and residents. This early warning system can significantly reduce response times, allowing for more efficient evacuation and firefighting efforts.

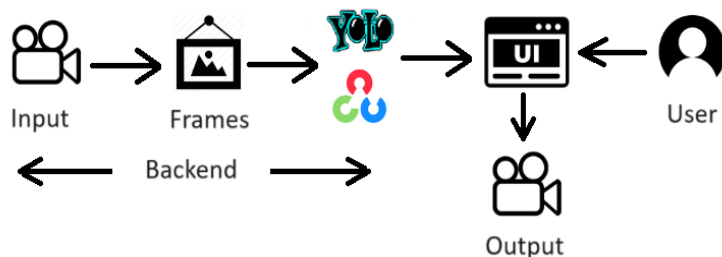
Scenario 2: Aerial Monitoring

Utilizing drones equipped with FireGuardian, authorities can conduct aerial surveillance of large forested areas. The YOLOv8 algorithm excels in real-time object detection, enabling the identification of smoke plumes even in challenging terrains. This aerial monitoring can enhance situational awareness and facilitate targeted firefighting strategies.

Scenario 3: Resource Allocation Optimization

FireGuardian contributes to efficient resource allocation during wildfire incidents. By continuously monitoring the progression of smoke patterns, emergency responders can make informed decisions about deploying firefighting resources. This targeted approach ensures that resources are utilized where they are most needed, maximizing the effectiveness of firefighting efforts and minimizing damage.

Technical Architecture:



Pre-requisites

To complete this project, you must require the following software, 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 many very nice tools like JupyterLab, Jupyter Notebook, Spyder, and 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

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:

3. open cmd prompt as administrator
4. Type pip install ultralytics and click enter.
5. Type “pip install opencv and click enter.
6. Type “pip install scikit-learn” and click enter.
7. 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 predictions.

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>

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.

Prior Knowledge

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

- OpenCV - <https://www.youtube.com/watch?v=WQeoO7MI0Bs>
- Flask - https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives

By the end of this project, you will:

1. Know fundamental concepts and techniques of Convolutional Neural Networks.
2. Gain a broad understanding of image data.
3. Know how to pre-process/clean the data using different data pre-processing techniques.
4. 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.

Image preprocessing

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
- Image Preprocessing
 - Import the required libraries
 - Load pre-trained model with OpenCV
- Training Model
- Application Building

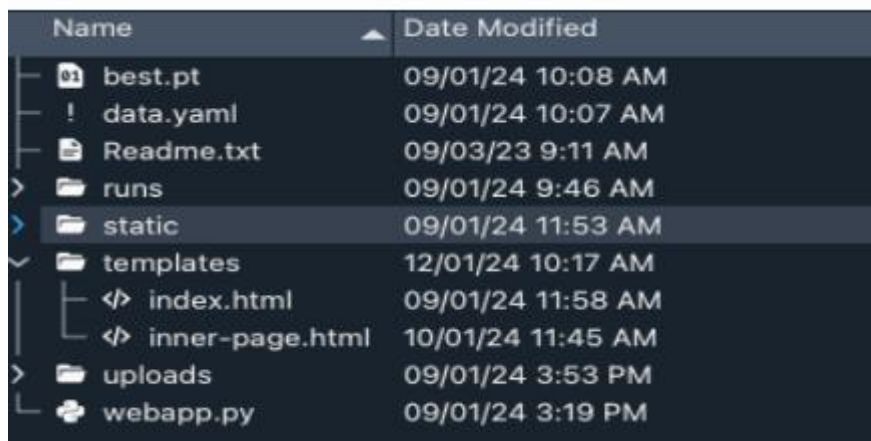
Create HTML Pages

Building Python Code

Run The Web Application

Project structure

Create a Project folder that contains files as shown below



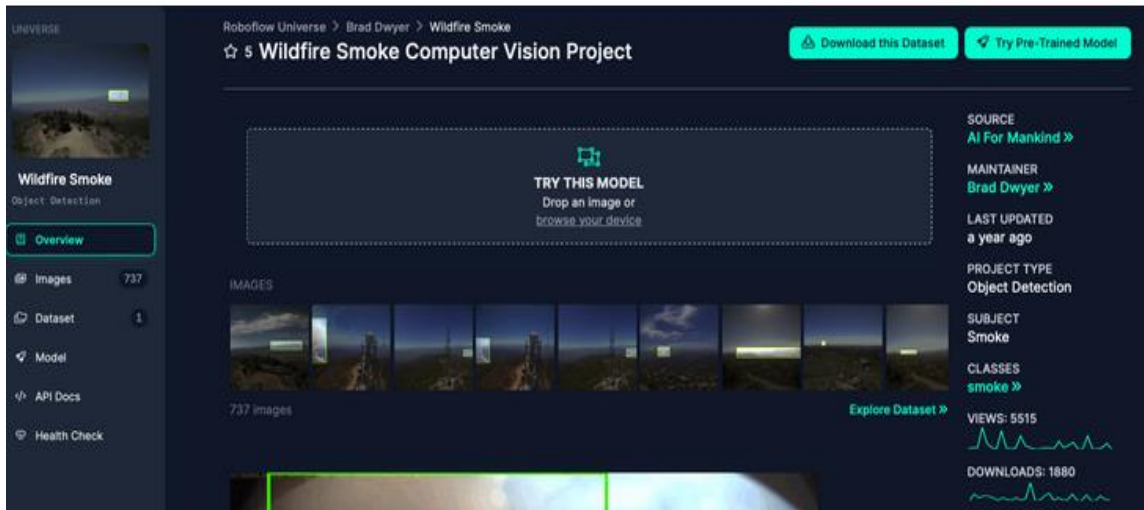
Name	Date Modified
01 best.pt	09/01/24 10:08 AM
! data.yaml	09/01/24 10:07 AM
Readme.txt	09/03/23 9:11 AM
> runs	09/01/24 9:46 AM
> static	09/01/24 11:53 AM
✓ templates	12/01/24 10:17 AM
</> index.html	09/01/24 11:58 AM
</> inner-page.html	10/01/24 11:45 AM
> uploads	09/01/24 3:53 PM
webapp.py	09/01/24 3:19 PM

1. The upload folder contains the training and testing images for training our model.
2. We are building a Flask Application that needs HTML pages stored in the templates folder and a python script webapp.py for server-side scripting

3. We need the model that is saved and the saved model in this content is best.pt, templates folder containing index.html, inner_page.html pages.

Collection of Data

This dataset is released by AI for Mankind in collaboration with HPWREN under a Creative Commons by Attribution Non-Commercial Share Alike license.



Dataset

Refer to the Dataset Link in the below activity.



Wildfire Smoke Object Detection Dataset and Pre-Trained Model by AI Fo..

737 open source Smoke images plus a pre-trained Wildfire Smoke model and API.
Created by AI For Mankind..

<https://universe.roboflow.com/brad-dwyer/wildfire-smoke>

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.

Import the required libraries

We import and install ultralytics

```
!pip install ultralytics
```

- The code uses the `!pip` command to install the ultralytics Python package.
- This package is likely related to deep learning or computer vision tasks, as indicated by its name.
- It enables additional functionalities or tools that are provided by Ultralytics, a company specializing in computer vision and AI solutions.
- After installation, the code may use functionalities provided by the Ultralytics package for tasks such as object detection, image classification, or other computer vision tasks.
- However, the specific functionalities and usage of the package cannot be determined from this code snippet alone, as it only installs the package but does not demonstrate its usage.

Load pre-trained model with OpenCV

Ultralytics:

- This is a computer vision library that provides implementations of various state-of-the-art deep learning models for object detection, segmentation, and classification tasks.

- It is commonly used for tasks such as object detection, instance segmentation, and image classification.
- The library offers easy-to-use APIs for training, inference, and evaluation of deep learning models.

YOLO (You Only Look Once):

- YOLO is a popular object detection algorithm known for its speed and accuracy.
- It processes the entire image in a single pass through a neural network, making it faster compared to other object detection algorithms.
- YOLO divides the image into a grid and predicts bounding boxes and class probabilities for objects within each grid cell.

```
from ultralytics import YOLO

!yolo task=detect mode=predict model=yolov8l.pt conf=0.25 source='https://ultralytics.com/images/bus.jpg'

Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8l.pt to 'yolov8l.pt'...
100% 83.7M/83.7M [00:00<00:00, 289MB/s]
Ultralytics YOLOv8.0.166 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l summary (fused): 268 layers, 43668288 parameters, 0 gradients

Downloading https://ultralytics.com/images/bus.jpg to 'bus.jpg'...
100% 476k/476k [00:00<00:00, 23.2MB/s]
image 1/1 /content/bus.jpg: 640x480 5 persons, 1 bicycle, 1 bus, 91.2ms
Speed: 15.1ms preprocess, 91.2ms inference, 35.0ms postprocess per image at shape (1, 3, 640, 480)
Results saved to runs/detect/predict
```

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="R3pDSrUAYZwnY83bjj7x")
project = rf.workspace("brad-dwyer").project("wildfire-smoke")
dataset = project.version(1).download("yolov8")

Collecting roboflow
```

The code installs the roboflow Python package using `!pip install roboflow`. This package allows you to interact with Roboflow, a platform for managing and preprocessing datasets for computer vision tasks.

-

- It imports the Roboflow class from the roboflow package.
- It creates an instance of the Roboflow class with an API key provided as an argument (`api_key="R3pDSrUAYZwnY83bjj7x"`).
- It accesses a project named wildfire-smoke within the workspace of a user named brad-dwyer using the workspace and project methods.
- It specifies version 1 of the project and downloads the dataset associated with this version using the download method with the model yolov8.

Training

Now it's time to train our yolo model :

```
!yolo task=detect mode=train model=yolov8n.pt data=/content/Wildfire-Smoke-1/data.yaml epochs=51 imgsz=256
```

New <https://pypi.org/project/ultralytics/8.0.238> available 🥳 Update with 'pip install -U ultralytics'
 Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
 engine/trainer: task=detect, mode=train, model=yolov8n.pt, data=/content/Wildfire-Smoke-1/data.yaml, epochs=51, patience=50, batch=16, imgsz=256, sa
 Downloading <https://ultralytics.com/assets/Arial.ttf> to '/root/.config/Ultralytics/Arial.ttf'...
 100% 755k/755k [00:00<00:00, 24.4MB/s]
 2024-01-09 04:20:40.454497: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register
 2024-01-09 04:20:40.454562: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register

- The code snippet likely uses the YOLO (You Only Look Once) object detection model for a task labeled as "detect".
- It specifies the training mode, indicating that the model will be trained rather than used for inference.
- The specific YOLO variant being used is "yolov8n.pt", which refers to a version of the YOLO model.
- The training data for the model is provided from the directory specified by the "data" parameter, which is likely a YAML file located at "/content/Wildfire-Smoke-1/data.yaml".
- The model will be trained for a total of 51 epochs, which represents the number of times the entire training dataset is passed through the model.
- The imgsz parameter sets the size of the input images to 256x256 pixels during training.
- Overall, this code snippet initiates the training of a YOLO object detection model using specific parameters such as model type, training data, number of epochs, and image size.


```
!yolo task=detect mode=train model=yolov8n.pt data=/content/Wildfire-Smoke-1/data.yaml epochs=51 imgsz=256
Transferred 319/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train4', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
WARNING ⚠ NMS time limit 0.550s exceeded
AMP: checks passed ✅
train: Scanning /content/Wildfire-Smoke-1/train/labels... 516 images, 0 backgrounds, 0 corrupt: 100% 516/516 [00:00<00:00 2177.0s
```

- Logging results to runs/detect/train4: It indicates the directory where the training results are being logged.

```
#now cheking the model
!yolo task=detect mode=predict model='/content/runs/detect/train4/weights/best.pt' conf=0.25 source='/content/Wildfire-Smoke-1/test/images/ck0kcoc8ik6ni084
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
```

- The code snippet specifies a YOLO task labeled as detect.
- It operates in predict mode, implying inference on new data.
- The model used for prediction is located at /content/runs/detect/train4/weights/best.pt.
- The confidence threshold for detected objects is set to 0.25.
- The input source for inference is an image file located at /content/Wildfire-Smoke-1/test/images/ck0kcoc8ik6ni084clxs0vif_jpeg.rf.8b4629777ffe1d349cc970ee8af59eac.jpg.
- This code snippet executes object detection on the specified image using the YOLO model, with a confidence threshold of 0.25 for detected objects.

```
image = Image.open('/content/runs/detect/predict/ck0kcoc8ik6ni0848clxs0vif_peg.rf.8b4629777ffe1d349cc970ee8af59eac.jpg')
```

image



- It uses the `Image` module from an external library (likely PIL or Pillow) in Python.
- The `Image.open()` function is used to open an image file located at the specified path:
'/content/runs/detect/predict/ck0kcoc8ik6ni0848clxs0vif_peg.rf.8b4629777ffe1d349cc970ee8af59eac.jpg'.
- Once opened, the image is loaded into memory and can be accessed for further processing or analysis.

```
#checking other image
```

```
!yolo task=detect mode=predict model='/content/runs/detect/train4/weights/best.pt' conf=0.25 source='/content/Wildfire-Smoke-1/test/images/ck0kd4afx8g47070' 15102MiB
```

```
Ultralytics YOLOv8.0.196 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
```

```
image 1/1 /content/Wildfire-Smoke-1/test/images/ck0kd4afx8g470701watkwxut_.jpeg.rf.bb5a1f2c2b04be20c948fd3c5cec33ff.jpg: 192x256 1 smoke, 90.4ms  
Speed: 1.2ms preprocess, 90.4ms inference, 589.3ms postprocess per image at shape (1, 3, 192, 256)  
Results saved to runs/detect/predict2
```

```
Learn more at https://docs.ultralytics.com/modes/predict
```

```
image2 = Image.open('/content/runs/detect/predict2/ck0kd4afx8g470701watkwxut_.jpeg.rf.bb5a1f2c2b04be20c948fd3c5cec33ff.jpg')
```

image2



```
!yolo task=detect mode=predict model='/content/runs/detect/train4/weights/best.pt' conf=0.25 source='/content/Wildfire-Smoke-1/test/images/ck0kdhyrna0b1072' 15102MiB
```

```
Ultralytics YOLOv8.0.196 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
```

```
WARNING ⚠ NMS time limit 0.550s exceeded
```

```
image 1/1 /content/Wildfire-Smoke-1/test/images/ck0kdhyrna0b10721v4wntit8_.jpeg.rf.a08e34d04fb672ce6cf8e94e810ec81d.jpg: 192x256 1 smoke, 200.8ms  
Speed: 1.4ms preprocess, 200.8ms inference, 1607.1ms postprocess per image at shape (1, 3, 192, 256)  
Results saved to runs/detect/predict3
```

```
Learn more at https://docs.ultralytics.com/modes/predict
```

```
image3 = Image.open('/content/runs/detect/predict3/ck0kdhyrna0b10721v4wntit8_.jpeg.rf.a08e34d04fb672ce6cf8e94e810ec81d.jpg')
```

image3




```
!yolo task=detect mode=predict model='/content/runs/detect/train4/weights/best.pt' conf=0.25 source='/content/Wildfire-Smoke-1/test/images/ck0kepbs9kdym0848hgpcf3y9.jpeg.rf.d0a63becb54a83b6b026f4b38a42933b.jpg'

Ultralytics YOLOv8.0.196 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs

WARNING ⚠ NMS time limit 0.550s exceeded
image 1/1 /content/Wildfire-Smoke-1/test/images/ck0kepbs9kdym0848hgpcf3y9.jpeg.rf.d0a63becb54a83b6b026f4b38a42933b.jpg: 192x256 1 smoke, 152.1ms
Speed: 1.5ms preprocess, 152.1ms inference, 807.0ms postprocess per image at shape (1, 3, 192, 256)
Results saved to runs/detect/predict4
Learn more at https://docs.ultralytics.com/modes/predict

image4 = Image.open('/content/runs/detect/predict4/ck0kepbs9kdym0848hgpcf3y9.jpeg.rf.d0a63becb54a83b6b026f4b38a42933b.jpg')

image4
```



```
!zip -r runs.zip runs
```

```
from google.colab import files
files.download('runs.zip')
```

This code snippet lets you download a file named 'runs.zip' from your Google Colab environment to your local device. It's done using a special function provided by Google Colab called `files.download()`. When you run this code, a download prompt will appear in your browser, allowing you to save the 'runs.zip' file to your computer.

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 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 “inspect” button, the next page is opened where the user chooses the image and predicts the output.

Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we have created 2 HTML pages- index.html, inner-page.html
- index.html displays the home page.
- Inner-page.html displays an output about the project
- We also use CSS-main.css to enhance our functionality and view of HTML pages.
- Link :CSS , JS

Index.html displays as below:

```
from flask import Flask, request, render_template, send_from_directory
from PIL import Image
import os
from ultralytics import YOLO

app = Flask(__name__)
model = YOLO("best.pt")

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

@app.route("/inspect")
def inspect():
    return render_template('inner-page.html')

@app.route('/upload', methods=["POST"])
def detect():
    if request.method == "POST":
        f = request.files['file']
        basepath = os.path.dirname(__file__)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        f.save(filepath)

        im1 = Image.open(filepath)
        results = model.predict(source=im1, save=True)
        print(results)

    return display()
```

```

@app.route('/display')
def display():
    folder_path = 'runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path,
    f))]

    if subfolders:
        latest_subfolder = max(subfolders, key=lambda x: os.path.getctime(os.path.join(folder_path, x)))
        directory = os.path.join(folder_path, latest_subfolder)
        files = os.listdir(directory)

        if files:
            latest_file = files[0]
            filename = os.path.join(folder_path, latest_subfolder, latest_file)
            file_extension = filename.rsplit('.', 1)[1].lower()

            if file_extension in {'jpg', 'jpeg', 'png'}:
                return send_from_directory(directory, latest_file)

    return "No valid files found."

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

```

PROJECT FIREGUARDIAN

[Home](#)

[About](#)

[Contact](#)

[Get Started](#)

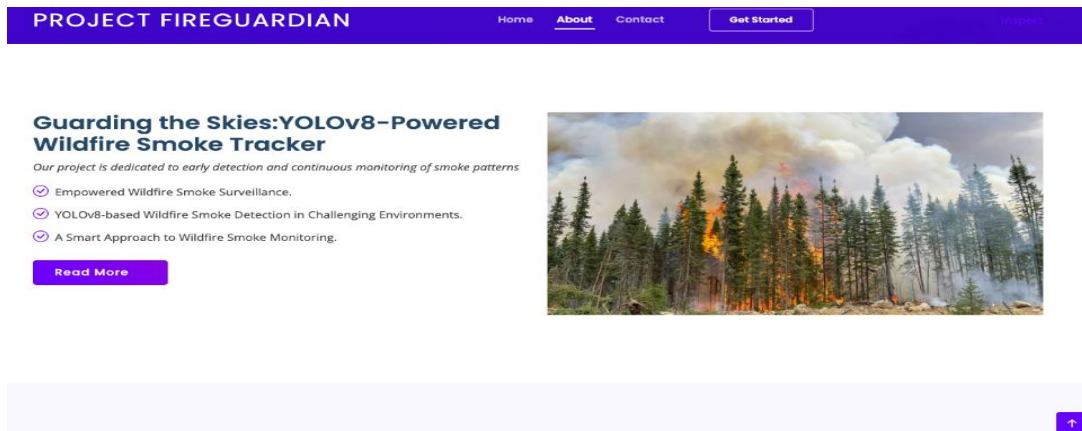
FireGuardian:Wildfire Smoke Detection using YOLOv8

YOLOv8's Vision for Early Wildfire Smoke Identification

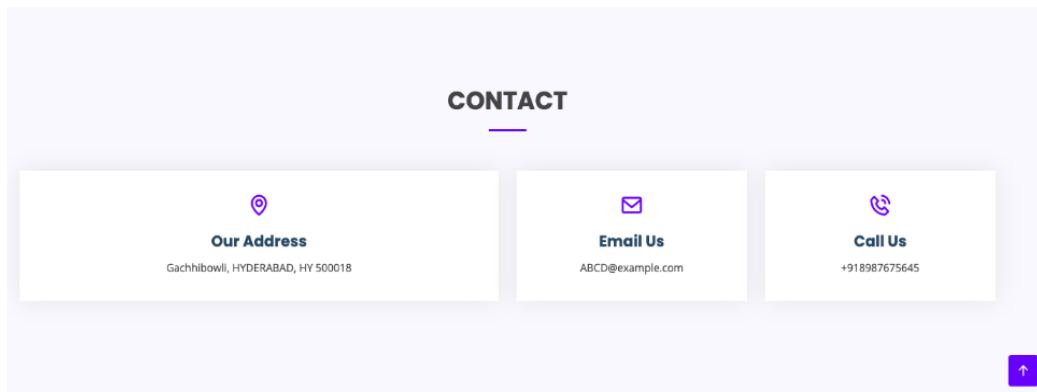
[Get Started](#)



About Section:



Contact Section:



Building Python Code

Create web.py (Python Flask) file: - Write the below code in Flask web.py python file script to run the Object detection project

```
from flask import Flask, request, render_template, send_from_directory
from PIL import Image
import os
from ultralytics import YOLO

app = Flask(__name__)

model = YOLO("best.pt")

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

@app.route("/inspect")
def inspect():
    return render_template('inner-page.html')

@app.route('/upload', methods=["POST"])
def detect():
    if request.method == "POST":
        f = request.files['file']
        basepath = os.path.dirname(__file__)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        f.save(filepath)

        im1 = Image.open(filepath)
        results = model.predict(source=im1, save=True)
        print(results)

    return display()
```

- Imports: We're bringing in tools for building a website and working with images.
- Setting Up Flask: We're getting Flask ready to create our website.
- Preparing the YOLO Model: We're loading a smart tool (YOLO) that recognizes objects in images.
- Creating Paths for Our Website: We're deciding what people see when they visit different parts of our website.
- Handling Image Uploads: When someone uploads an image, we save it and ask YOLO to recognize objects in it. Then, we print out what YOLO finds. Finally, we should show this information on the website


```

@app.route('/display')
def display():
    folder_path = 'runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path,
    f))]

    if subfolders:
        latest_subfolder = max(subfolders, key=lambda x: os.path.getctime(os.path.join(folder_path, x)))
        directory = os.path.join(folder_path, latest_subfolder)
        files = os.listdir(directory)

        if files:
            latest_file = files[0]
            filename = os.path.join(directory, latest_file)
            file_extension = filename.split('.', 1)[1].lower()

            if file_extension in {'jpg', 'jpeg', 'png'}:
                return send_from_directory(directory, latest_file)

        return "No valid files found."

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

```

Route Setup:

The `@app.route('/display')` decorator sets up a route for displaying images on a webpage.

Display Function:

The `display()` function:

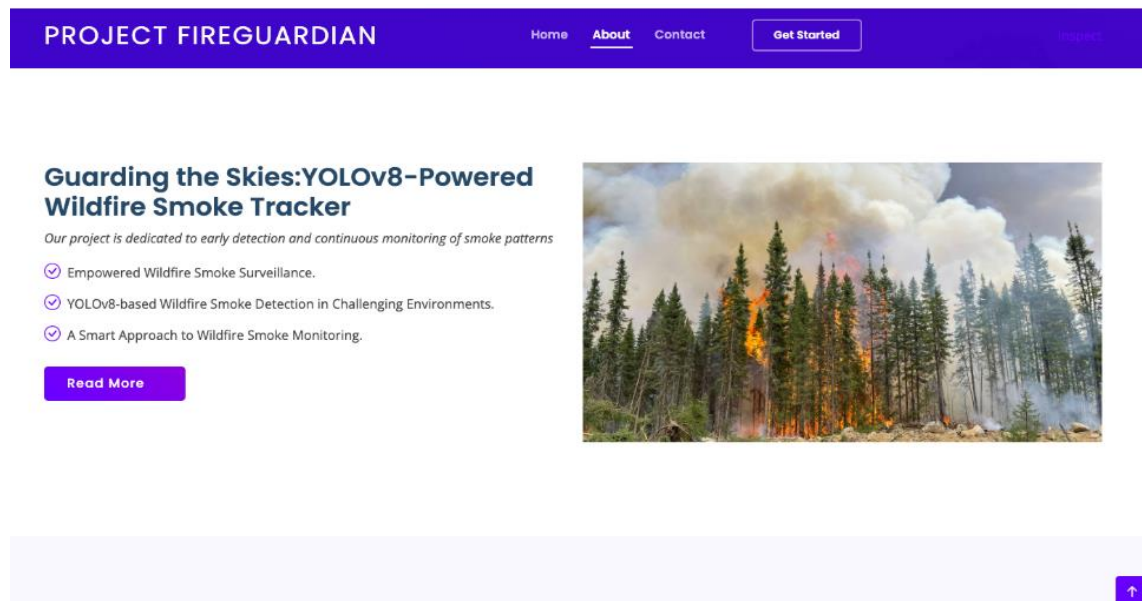
- Looks for the latest image file generated by an object detection process.
- Checks if the file is a valid image (with extensions like jpg, jpeg, or png).
- If a valid image is found, it sends it to be displayed on the webpage.
- If no valid image is found, it returns a message saying so.

Running the App:

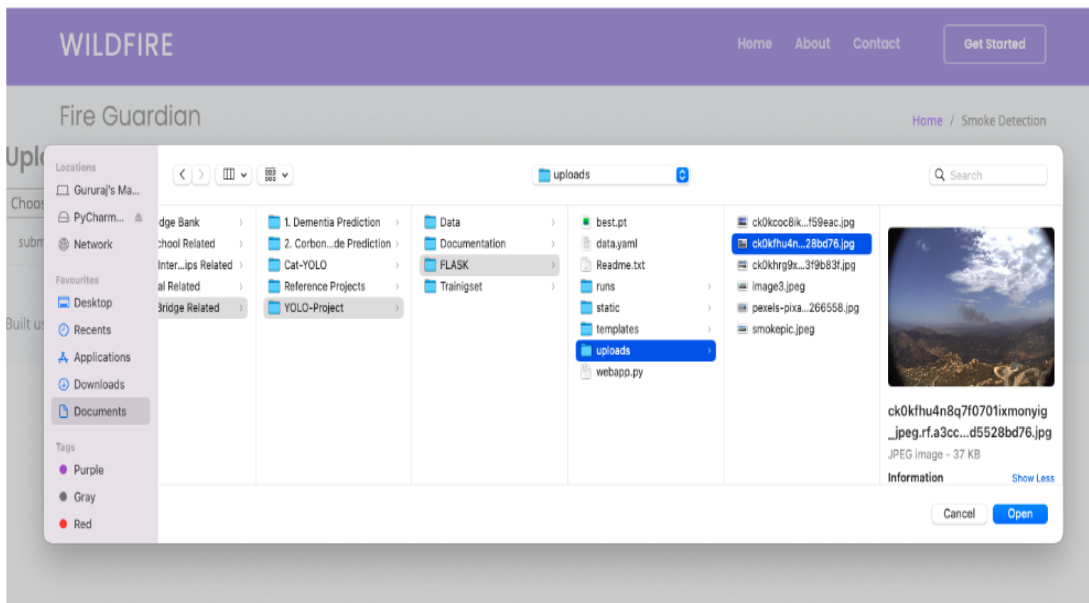
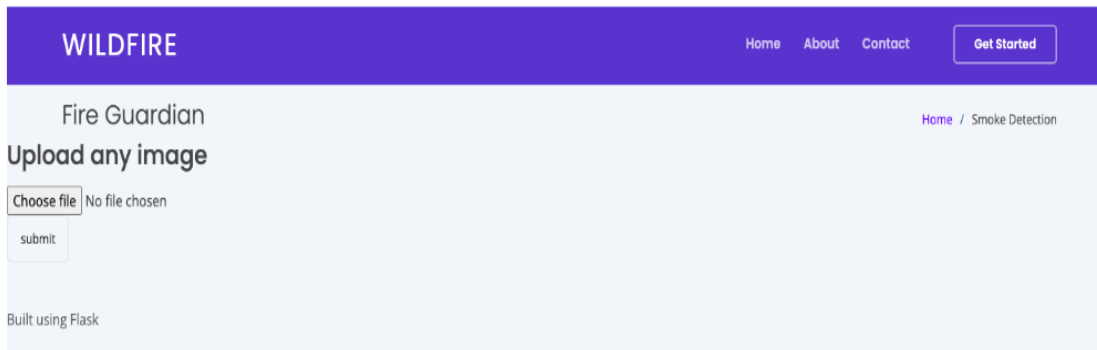
- The `if name == 'main':` block ensures the app only runs when this script is executed directly.
- It starts the Flask app in debug mode, useful for catching and displaying errors during development.

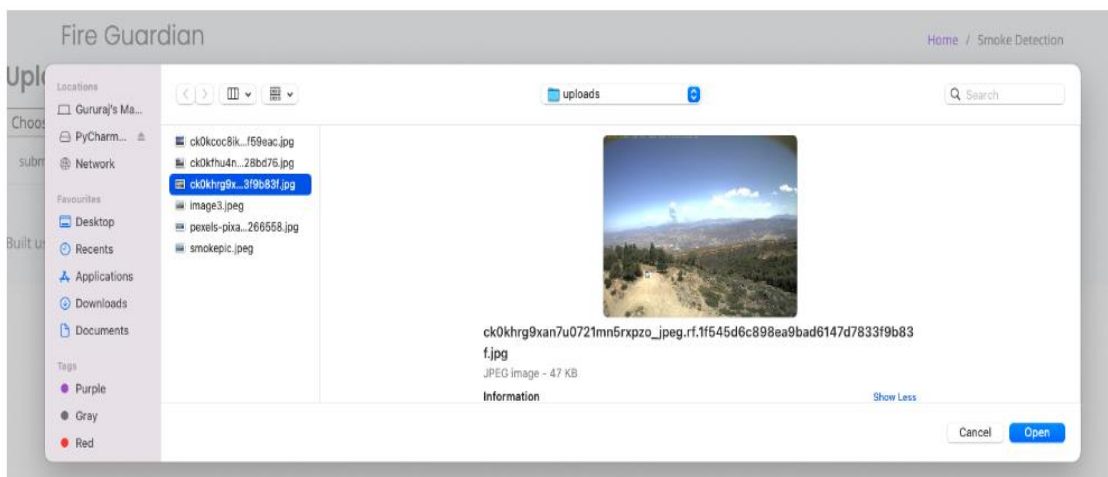
Run The Web Application

- Open the anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type “app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



Inspect button:







Our model predicts wildfire smoke and shows boundaries on the output page.