Project Documentation

Title : Project Documentation: Smart Safety Gear Detection System

Introduction:

Workplace safety is a critical aspect of any industrial or construction environment. Non-compliance with safety protocols, particularly the use of essential personal protective equipment (PPE), can lead to severe accidents, injuries, or even fatalities. Ensuring that workers adhere to safety standards is a challenge, especially in large-scale operations where manual monitoring can be resource-intensive and error-prone.

This project, **Smart Safety Gear Detection**, presents an automated solution for detecting missing safety gear in real-time. Leveraging the power of **YOLOv8 (You Only Look Once)**, a state-of-the-art object detection model, the system can identify individuals not wearing critical safety equipment, such as helmets, masks, or safety vests. The solution is further enhanced with an intuitive **Streamlit-based interface**, enabling seamless image and video processing, along with automated email alerts and a historical record system powered by MongoDB.

By addressing the need for real-time detection and notification, this project aims to reduce workplace hazards, promote safety compliance, and improve operational efficiency.

Project Scope:

Inclusions

1. **Detection of Safety Gear**:
   The system is capable of detecting various safety gear items, including helmets, masks, and safety vests, as well as identifying when these items are missing. It supports detection across multiple classes such as Hardhat, Mask, NO-Hardhat, NO-Mask, NO-Safety Vest, Person, Safety Cone, Safety Vest, Machinery, and Vehicle.

2. **User-Friendly Interface**:
   An interactive **Streamlit-based interface** enables users to upload images or videos, view annotated outputs, and manage settings such as detection confidence levels.

3. **Notification System**:
   The project incorporates an automated email alert system, allowing supervisors to be notified of missing safety gear with detailed reports.

4. **Data Storage and History Management**:
   Detection results are stored in a **MongoDB database**, allowing users to review past records and manage historical data efficiently.

Limitations and Constraints

1. **Model Performance**:
   The YOLOv8 model has been trained on a specific dataset. Detection accuracy depends on the dataset quality and may vary with unseen data.

2. **Hardware Requirements**:
   The system's performance can benefit from high-spec hardware, particularly for faster processing of larger files. However, it is functional on standard configurations as well.
   GPU acceleration significantly improves detection times, but it is not mandatory for the current implementation.

3. **Manual Configuration**:
   Users need to configure certain paths (e.g., dataset directories, model weights) and email credentials before running the application.
   The system depends on a functional MongoDB instance for data storage, requiring users to have MongoDB installed and configured.

4. **Scalability**:
   The system is designed for small to medium-scale deployments. Large-scale monitoring involving thousands of images or videos might require further optimization.

By focusing on these aspects, the project ensures efficient safety gear detection while providing a foundation for further enhancements in the future.

Requirements:

Functional Requirements

1. **Safety Gear Detection**:
   The system must identify the presence or absence of safety gear, such as helmets, masks, and safety vests, from the input files.
   The detection should be capable of recognizing multiple predefined classes, including Hardhat, Mask, Safety Vest, and others.

2. **Input File Processing**:
   The system should allow users to upload images or video files for analysis via the Streamlit interface.
   Uploaded files should be processed for detecting safety gear, and the output should include annotated visuals highlighting detected or missing items.

3. **Results Display**:
   The application must display results on the interface with annotations and class-specific bounding boxes.
   Missing items should be highlighted for better visibility and immediate action.

4. **Notification System**:
   An automated email alert system must notify supervisors about missing safety gear.
   The notification should include a detailed report of the missing items to ensure clarity.

5. **History Management**:
   The system should store detection results in a **MongoDB database**.
   Users must be able to view detection history through the interface and delete all history when needed.

6. **Export and Reporting**:
   Detection results must be exportable in visual formats (e.g., annotated images/videos).
   The system should log training metrics (e.g., losses) during model training and save them for analysis.

Non-Functional Requirements

1. **Performance**:
   The system should process inputs efficiently to provide results in a reasonable time frame.
   The YOLOv8 model must maintain high accuracy across all tested datasets, minimizing false positives and negatives.

2. **Usability**:
   The Streamlit interface should be intuitive and user-friendly, ensuring ease of use for non-technical users.
   Parameters such as confidence thresholds should be adjustable via the interface.

3. **Scalability**:
   The system should handle a moderate number of files without significant performance degradation.
   Additional classes or safety gear types can be incorporated with minor adjustments to the code.

4. **Reliability**:
   The system must function consistently, ensuring accurate detection and proper notification delivery under different scenarios.

5. **Security**:
   User credentials for email and database interactions must be protected and not

exposed in the system logs or code.
MongoDB must be configured to ensure secure storage of detection results.

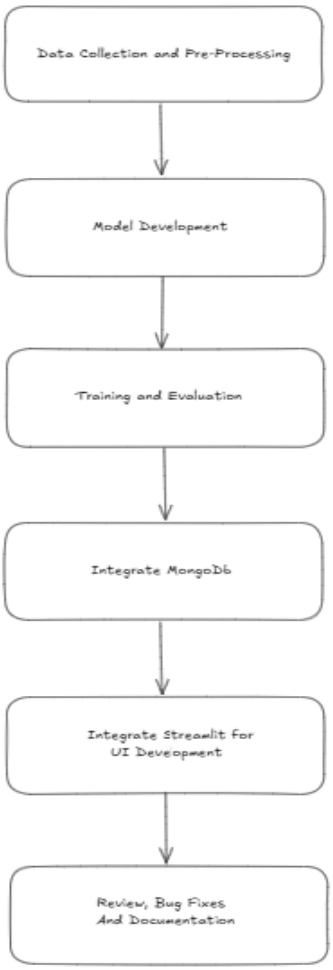
6. **Portability**:
   The application should run on any environment supporting Python, with dependencies specified in the requirements.txt file.

By adhering to these functional and non-functional requirements, the system ensures a reliable and efficient solution for safety gear detection.

Technical Stack:

- **Programming Language**: Python
- **Frameworks/Libraries:** Streamlit, OpenCV, YOLOv8, Matplotlib, Seaborn, Pandas, PyMongo
- **Database:** MongoDB
- **Tools/Platform:** Google Colab, VS Code, and Streamlit Web App

Architecture/Design:



Development:

Description of the Tech and Frameworks Used

1. **YOLOv8:**

   Leveraged for real-time object detection and classification of safety gear in images and videos.

   Provided pre-trained weights and flexibility for fine-tuning on custom datasets, ensuring accurate detection.

2. **Streamlit:**

   Used as a framework to build a user-friendly and interactive web application.

   Enabled seamless integration of image and video upload options, parameter adjustments (e.g., confidence thresholds), and dynamic result display.

3. **OpenCV and PIL:**

   OpenCV: For processing video frames, drawing bounding boxes, and annotating detected objects.

   Pillow (PIL): Used for handling image input/output and conversions between formats.

4. **MongoDB:**

   Acted as a NoSQL database to store and retrieve detection results, including file names, detected items, and timestamps.

   Allowed retrieval of detection history for user review.

5. **Matplotlib & Seaborn:**

   Matplotlib: Used to plot model training logs and annotate detection outputs.

   Seaborn: Enhanced dataset visualizations, such as class distributions across training, validation, and test datasets, with more aesthetic and intuitive plots.

Overview of Coding Standards and Best Practices

- Code structured into logical sections: preprocessing, model training, Model's Performance, application interface, and MongoDB operations.
- Extensive use of comments and function modularity for clarity and reusability.
- Parameters like confidence thresholds and file paths are configurable to make the application more flexible and adaptable to different use cases.
- Integration of error handling for email functionality and database operations.

Challenges Encountered and Solutions

1. **Integration of YOLO with Streamlit:**

   Challenge: Combining YOLOv8 detection with Streamlit's interactive interface required handling diverse inputs such as images and videos dynamically.

   Solution: Streamlined input processing using modular functions and temporary storage to manage uploaded files seamlessly.

2. **Visualizing Detection Results Dynamically:**

   Challenge: Making detection results visually intuitive and user-friendly.

   Solution: Applied consistent color mapping for bounding boxes and annotations, enhancing interpretability and user interaction.

3. **Email Alerts:**

   Challenge: Users need to provide their email password to enable email notifications securely.

   Solution: Introduced the concept of generating an app password in the user's email account (e.g., Gmail) as an alternative to using their original password. This ensures better security and compliance with modern email authentication standards.

4. **Issue with Video Not Being Displayed as Output:**

   Challenge: Real-time display of processed video frames was not feasible within the Streamlit framework.

   Solution: Implemented a download button that allows users to save and view the processed video locally after detection. This ensures accessibility to the output while circumventing display limitations.

User Guide:

Instructions for Using the Application

1. **Setup:**

- Install the required dependencies listed in the requirements.txt file.
- Configure MongoDB by ensuring the service is running and updating the connection string if using a remote instance.
- Place the YOLOv8 model weights (best.pt) in the specified path in the code.
- Run the application via Streamlit (streamlit run app.py).
- To send an Email you need to have one app password which you can get by creating one in your mail go to mail -> Manage your Google Account -> Search -> App Passwords -> Give Name for which app You are Creating -> Get the Password.

2. **Using the Application:**

- **Image Mode:** Upload an image to detect safety gear and view annotated results.
- **Video Mode:** Upload a video to detect safety gear across frames, with a downloadable processed video.
- **Email Alerts:** Option to send automated alerts when missing safety gear is detected.

3. **Detection History:**
   - View previous detections, including timestamps, detected items, and missing items.
   - Clear detection history if needed.
4. **Detection Logs:**
   - Get a CSV File where in You can get all the Data Stored in your DB

Troubleshooting Tips

- **Issue:** Email alert fails.
  Solution: Verify email credentials and app password.
- **Issue:** YOLO model not loading.
  Solution: Ensure the model weight file is in the correct path.
- **Issue:** MongoDB connection failure.
  Solution: Verify the MongoDB service is running and the connection URI is correct.
- **Issue:** Ultralytics, CV2, Streamlit etc.. error

  Solution: Verify the Installation of the Modules and if not installed use the requirements.txt and install in Venv.

Conclusion:

Summary of Project Outcomes and Achievements

- Successfully implemented a safety gear detection system using YOLOv8 and Streamlit.
- The application effectively detects and highlights missing safety gear in both images and videos.
- Provided alerts via email, Downloadable Logs and maintained a comprehensive history in MongoDB.

Reflections on Lessons Learned

- **Strengths:** Efficient use of YOLO for real-time object detection; interactive and user-friendly Streamlit interface. Access for the DB and Downloadable logs and Adjustable confidence threshold.

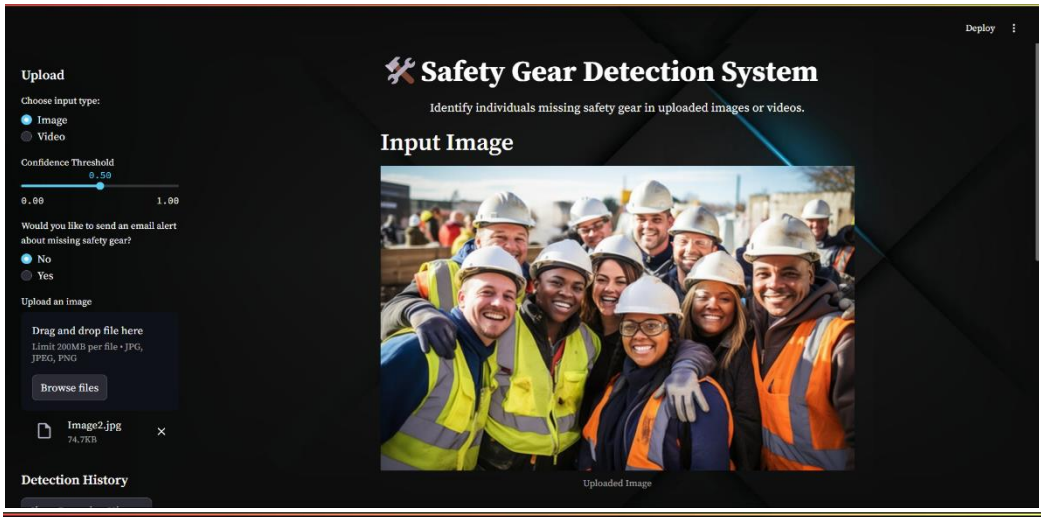
- **Areas for Improvement:** Incorporate additional safety gear classes and further optimize model performance for large datasets. More secure way to handle the Mail sending Part.

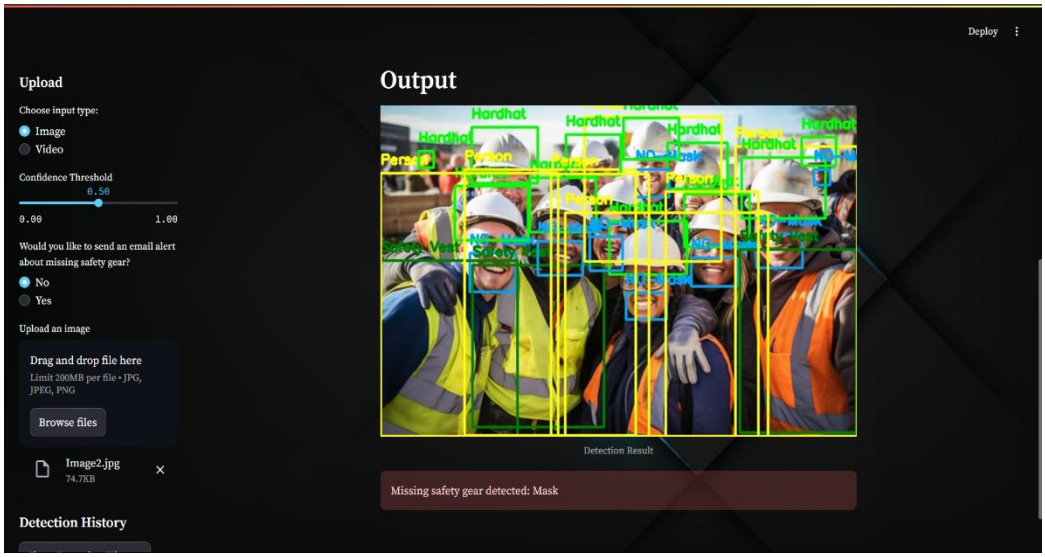
Future Enhancements

- Add support for live video stream detection.
- Implement role-based access control for secure usage.
- Expand detection capabilities to cover more safety regulations and industrial scenarios.
- Deployment in Cloud Like AWS, GCP, Azure for more Scalability.

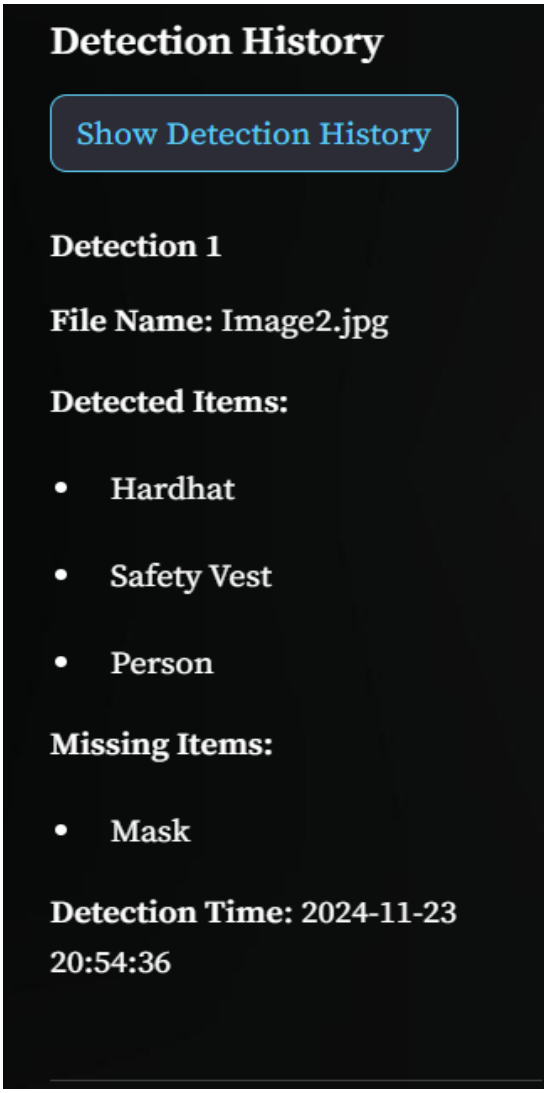
Output:

Sample Input and Output:

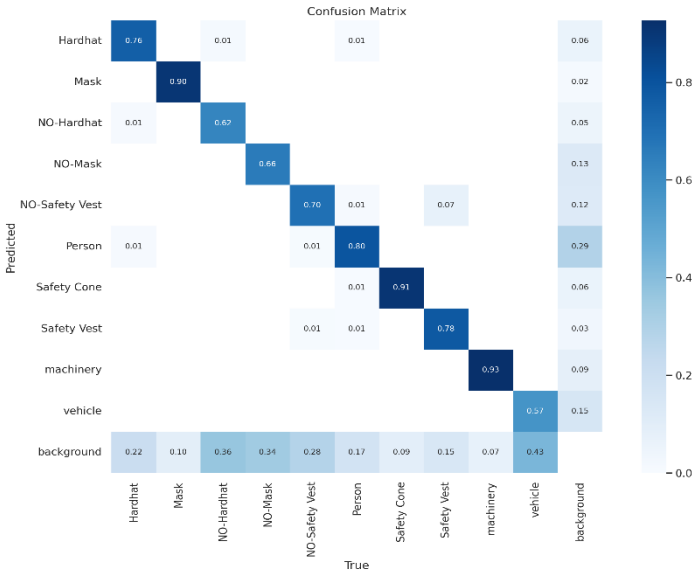


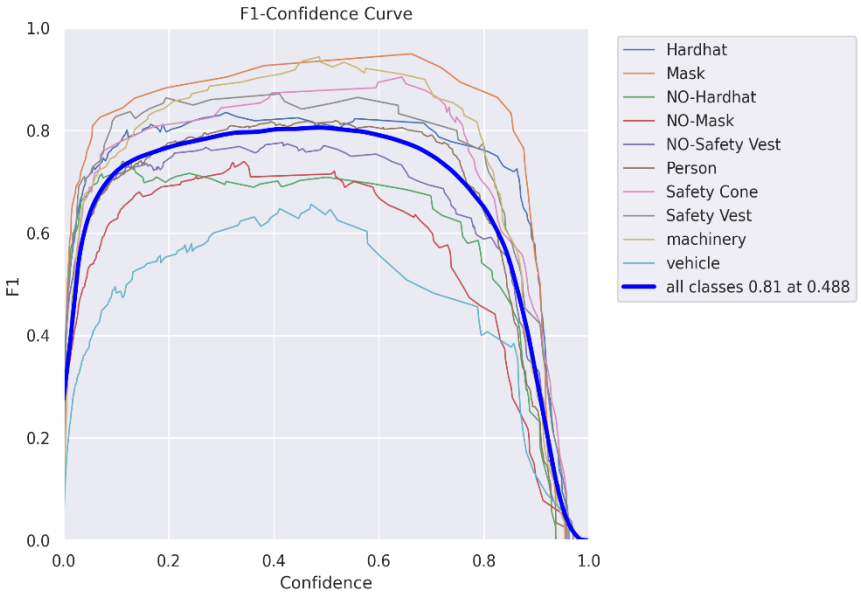


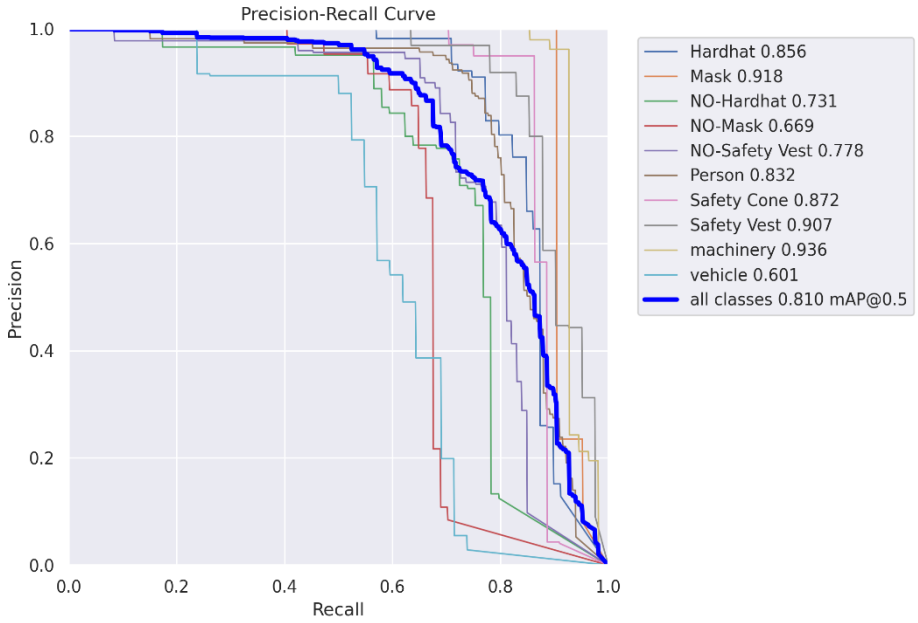
Sample Detection History:



Graphs:







References:

- **Dataset:** https://www.kaggle.com/datasets/snehilsanyal/construction-site-safety-image-dataset-roboflow/data
- **OpenCV Library:** https://opencv.org/
- **Streamlit Framework:** https://streamlit.io/
- **Python SMTP Library:** https://docs.python.org/3/library/smtplib.html
- **MongoDb:** https://www.mongodb.com/
- **YOLOv8 Documentation:** https://docs.ultralytics.com