Final Project Report Template

1.Introduction

1.1 Project overviews

The primary goal of the "Virtual Eye – Lifeguard for Active Swimming Drowning" project is to architect, develop, and ultimately deploy a highly robust, exceptionally reliable, and truly real-time automated drowning detection system. This sophisticated system is specifically engineered to identify and respond to active swimming drowning incidents, leveraging cutting-edge computer vision techniques, with a particular focus on the YOLOv5 (You Only Look Once, version 5) deep learning model. The fundamental aim is to significantly and demonstrably enhance the safety protocols and overall security within diverse swimming pool environments by introducing an unprecedented layer of intelligent, continuous surveillance and early warning capabilities. By achieving this, the project endeavors to dramatically reduce the critical response times required during life-threatening drowning events, thereby increasing the probability of successful rescues and, most importantly, ultimately saving human lives. This system aims to significantly enhance swimming pool safety by providing an additional layer of surveillance and early warning capabilities, thereby reducing response times during critical drowning incidents and ultimately saving lives. This project leverages the YOLO (You Only Look Once) object detection architecture to perform accurate and efficient visual analysis. By training YOLOv5 on a custom dataset of underwater and poolside footage, the system learns to classify actions as either "Swimming" or "Drowning" based on human posture, motion patterns, and position in water. Virtual Eye is an AI-powered real-time surveillance system designed to enhance water safety by automatically detecting drowning incidents in swimming pools using deep learning. The system acts as a virtual lifeguard, continuously analyzing video footage to distinguish between normal swimming behavior and signs of distress, such as erratic movements, prolonged inactivity, or submersion. In essence, the "Virtual Eye - Lifeguard" project's primary goal is a commitment to leveraging the power of artificial intelligence and computer vision to forge safer aquatic environments. It seeks to create a technological safety net that vigilantly watches over swimmers, provides an intelligent early warning, and ultimately contributes to the profound objective of saving lives by ensuring that critical drowning incidents are identified and addressed with unprecedented speed and accuracy. This goal encompasses not just the technical implementation, but the broader societal impact of fostering greater peace of mind for swimmers, their families, and pool operators alike.

1.2 Objectives

- To develop and implement a real-time, AI-powered drowning detection system capable of identifying active swimming drowning events.
- To establish and curate a comprehensive dataset of active drowning behaviors, normal swimming, and non-drowning distress signals, comprising at least 100 hours of annotated video footage, within the first 3 months of the project.
- To design and implement a robust automated alert system that can notify designated lifeguards or pool management via multiple channels (e.g., audible alarms, visual alerts on a dashboard, mobile notifications).
- To develop an intuitive and user-friendly monitoring dashboard that displays real-time video feeds, highlights detected drowning events, and provides controls for alert management, accessible via a web interface.

2.Project Initialization and Planning Phase

2.1 Define Problem statement

Drowning remains a pervasive and tragic global public health issue, disproportionately affecting vulnerable populations, particularly children. According to the World Health Organization (WHO), drowning is the third leading cause of unintentional injury death worldwide, accounting for 7% of all injury-related deaths. Swimming pools, while offering recreational opportunities, are significant sites for these incidents. The inherent risks associated with aquatic environments necessitate highly vigilant and effective safety measures.

The core problem lies in the limitations of current swimming pool safety protocols, which primarily rely on human lifeguards and often lack the autonomous, real-time intelligence required to consistently and immediately detect active drowning events as they unfold.

Lifeguards, despite rigorous training, are susceptible to fatigue, momentary lapses in concentration, and distractions, especially during long shifts, peak hours, or in high-stress environments. This can lead to delayed detection or, in tragic cases, missed incidents.

In crowded pools, water turbulence, glare from sunlight, reflections, or even architectural elements can create significant visual obstructions, making it extremely difficult for lifeguards to maintain continuous, clear sight of all swimmers, particularly those at the bottom or far end of the pool.

VirtualEye problem statement template :- click here

2.2 Project Proposal(Proposed Solution)

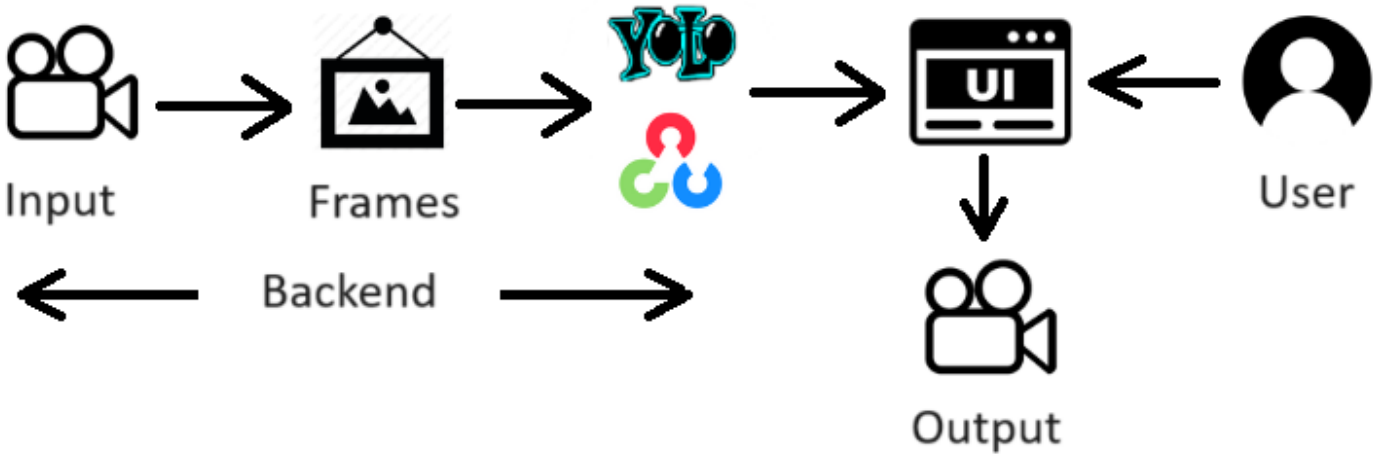
The objective of the "Virtual Eye" project is to provide real-time underwater safety detection by leveraging a deep learning model (YOLO) to analyze and detect potential hazards in underwater environments. The system allows users to upload images and videos for automated safety assessments, enhancing monitoring and response to underwater threats.

The model is a real-time object detection in underwater environments. It uses Flask to host a web application that allows users to upload images or videos. These uploads are processed using the YOLO model, which scans for safety hazards. After processing, the system provides immediate feedback, including hazard alerts and detected object labels. The project improves underwater safety by providing immediate analysis of visual data, helping to detect potential hazards in realtime.

Utilizes the YOLO deep learning model for fast and accurate identification of objects within the underwater images and videos. Simple and intuitive Flask-based web application to upload and display the results of drowning detection. Supports both images and videos, ensuring flexibility in usage. By automating the detection of underwater threats, it reduces the response time and enhances the overall safety of swimmers.

Project proposal Template:- **Click Here**

2.3 Initial Project Planning



The provided diagram depicts a flowchart of a system for processing video input and delivering output to a user. Here's a description of each step in the flow:

1. Input: The process begins with a video input, represented by a camera icon, which could be a live video feed or pre-recorded video.

2. Frames: The video input is converted into individual frames for processing, depicted by the picture frame icon.

3. YOLO/OpenCV: The frames are passed to object detection or image processing frameworks such as YOLO (You Only Look Once) or OpenCV for analysis and detection tasks.

4. Backend: The processed data from the detection frameworks is sent to the backend for further processing, logic implementation, and integration.

5. UI: The results from the backend are sent to the user interface, where they are displayed in a structured format for the end-user to view and interact with.

Project Planning Template:- **Click Here**

3. Data Collection and Preprocessing Phase

3.1Data Collection Plan and Raw Data Sources Identified

The goal of this data collection plan is to gather a diverse and highquality dataset of swimming activities and potential drowning scenarios to train, test, and evaluate the YOLOv3-based drowning detection system. The dataset should include various environmental conditions swimmer demographics, body poses, and behavior patterns to ensure the model can generalize effectively and accurately detect distress across a wide range of real-world situations. Raw Data Sources Identified Public Surveillance Footage: Open-source or publicly shared swimming pool surveillance videos and drowning datasets available online are used to gather realistic scenarios for training and evaluation.

Data Collection plan & Raw Data Sources Identification Template :- **Click Here**

3.2 Data Quality Report

Incorrect or missing labels for 'drowning' vs 'swimming' objects in annotation files (YOLO format). High Review and validate all annotations using tools like Roboflow. Ensure bounding boxes are tight and class labels are accurate. FrameSet4 Dataset is imbalanced with far more 'swimming' cases than 'drowning'. Medium Augment minority class (drowning) via flipping, brightness variation, and synthetic data techniques. Consider oversampling techniques or weight adjustments during training.

Data Quality Report Template:- **Click Here**

3.3 Data Preprocessing

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

The following steps were implemented:

1. **Image collection:** Images are collected from various sources.
2. **Resizing**: All the images are resized to a standard YOLOV5 compatible size to ensure efficient processing.
3. **Image Labelling**: Images are labelled as Swimming and drowning.

Data Preprocessing Template:- **Click Here**

4. Model Development Phase

4.1 Model Selection Report

The Model Selection Report for Virtual Eye: Active Swimming Drowning Detection Using YOLOv5 focuses on applying real-time object detection techniques to improve water safety monitoring. By leveraging deep learning and convolutional neural networks (CNNs), specifically the YOLO architecture, this system identifies swimmers and potential drowning scenarios from surveillance footage efficiently and accurately. YOLO (You Only Look Once) is well-suited for this task due to its balance between speed and accuracy, making it capable of operating in real time environments like swimming pools or aquatic centers.

Model Selection Report Template:- **Click Here**

4.2 Initial Model Training Code,Model Validation and Evaluation Report:

Development and Execution Environment:

The initial model training code for Virtual Eye: Active Swimming Drowning Detection uses the YOLOv5 architecture, trained on a dataset of swimming and drowning images. The training pipeline involves loading a pre-trained model, defining custom data via a yaml file, and training it using the Ultralytics training framework. During training, no manual preprocessing was applied, as YOLO's data loader internally handles image resizing, normalization, and augmentation. The model was trained over 70+ epochs with adjustable confidence thresholds to improve detection quality. Model validation and evaluation metrics were generated by the Ultralytics framework. These metrics help quantify the model's ability to detect drowning scenarios under varied lighting, motion, and angle conditions. The evaluation is visualized via automated plots and detection

overlays on test images.

Tools:

- **Python**: Core programming language for data processing and model training.
- **Flask**: Provides a web-based user interface (UI) for resume uploads and displaying results.

Training Process:

- Validation was conducted using a sample image to evaluate prediction accuracy.

User Interface (UI)

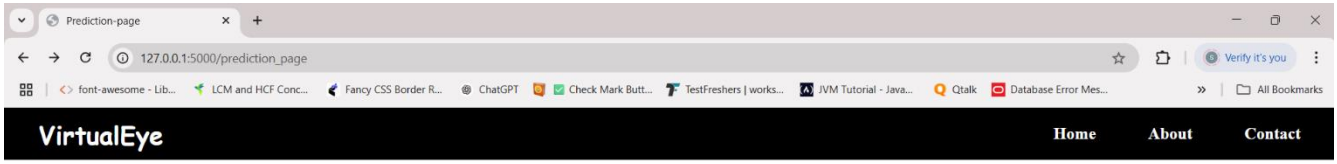
Flask-Based Web Application:

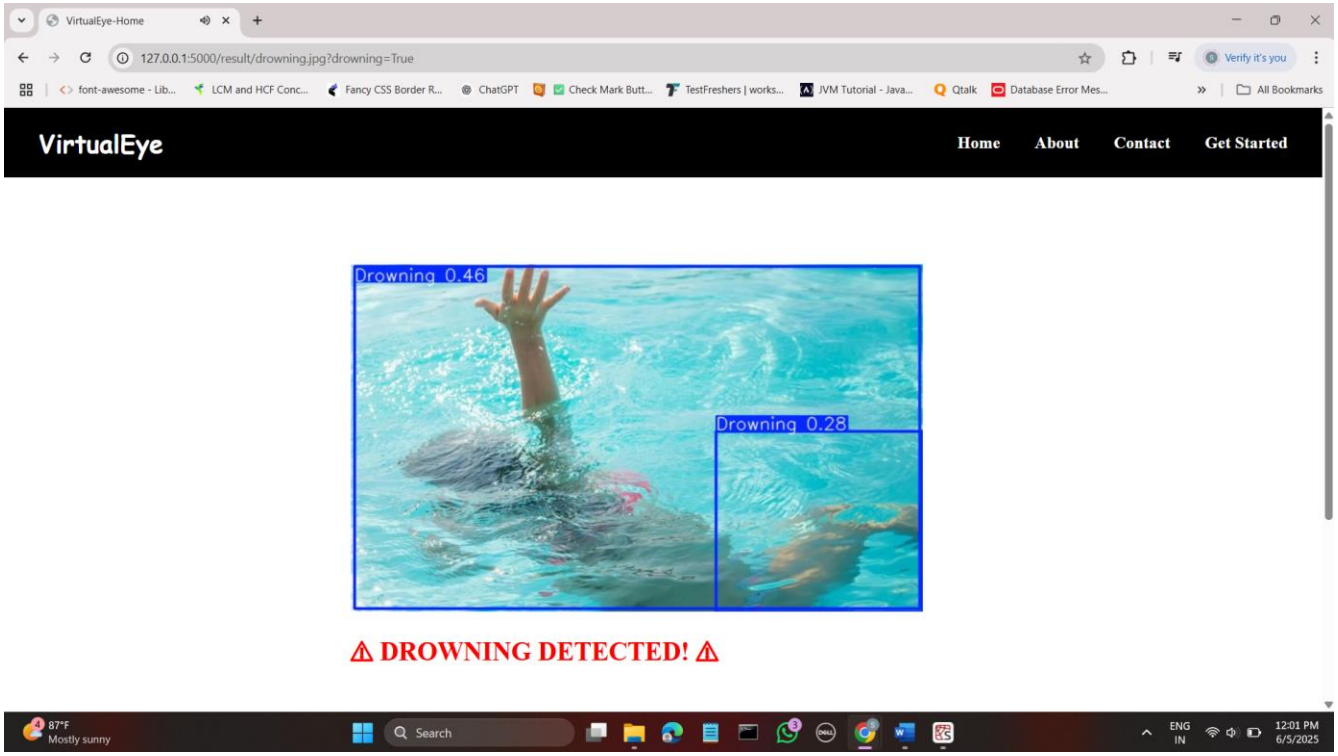
- Allows users to upload images in jpeg, jpg, png, webp formats.
- Displays the results if the drowning is detected it alerts the users with alarms.

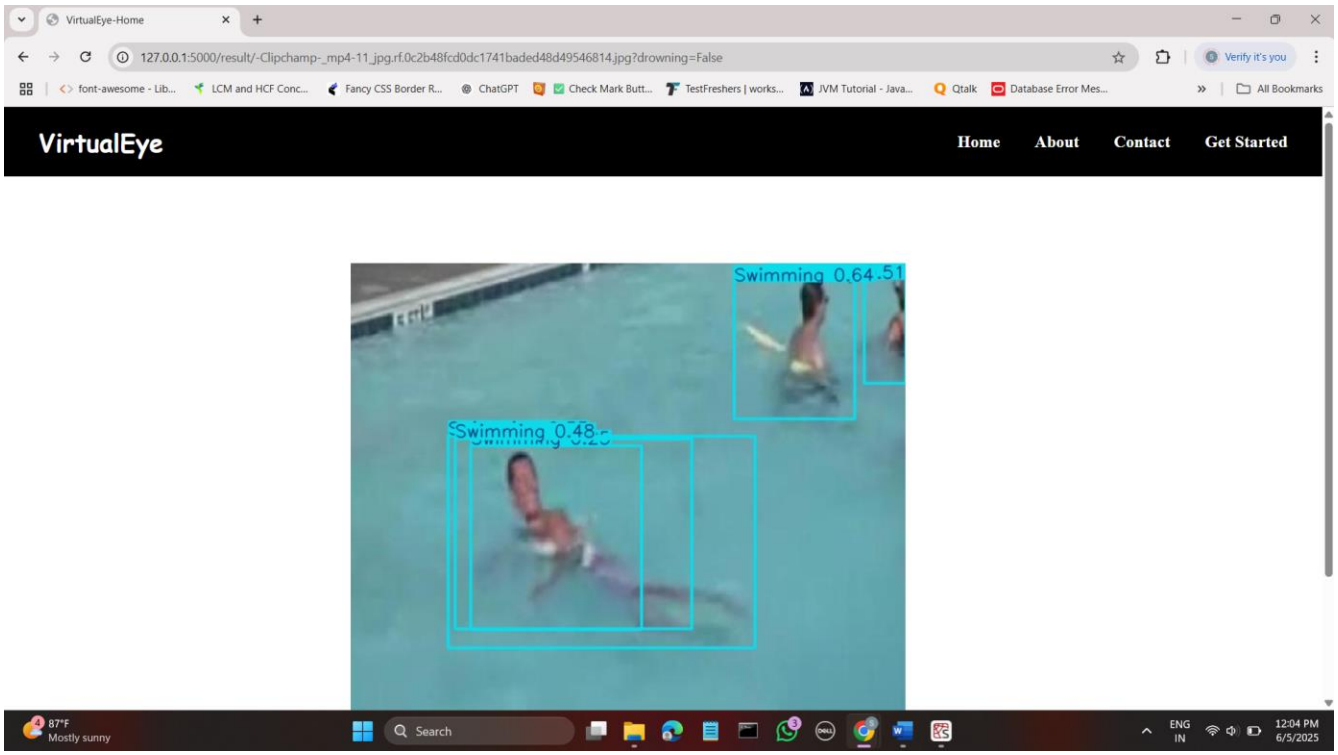
Initial Model Training Code, Model Validation and Evaluation Report: Click Here

6.RESUlLT

6.1 Output Screenshots




VirtualEye-Home
127.0.0.1:5000/result/drowning.jpg?drowning=True
VirtualEye
Home
About
Contact
Get Started
Drowning 0.46
Drowning 0.28
⚠ DROWNING DETECTED! ⚠


VirtualEye-Home
127.0.0.1:5000/result/-Clipchamp-_mp4-11_jpg.rf.0c2b48fcd0dc1741baded48d49546814.jpg?drowning=False
VirtualEye
Home
About
Contact
Get Started
Swimming 0.64
Swimming 0.48

7.Advantages & Disadvantages

Advantages:

- **Real-time Monitoring:** Continuously monitors swimming activity and identifies potential drowning incidents instantly.
- **Enhanced Safety:** Reduces human error and improves response time in emergency situations, potentially saving lives.
- **Automated Alerts:** Automatically sends alerts to lifeguards or emergency services, ensuring quick intervention.
- **Non-Intrusive Monitoring:** Does not require swimmers to wear any devices; works using computer vision.
- **Cost-effective in Long Run:** Reduces the need for a large team of lifeguards and ensures constant vigilance.

. **Disadvantages:**

- **False Positives/Negatives:** May sometimes misinterpret actions, such as play behavior as drowning, or miss actual incidents.
- **Limited Detection in Crowded Areas:** In heavily crowded pools or beaches, it may be difficult for the system to track and analyze individual swimmers accurately.
- **Adaptation to Different Swimming Styles:** The system might misclassify certain swimming techniques or diving actions as drowning behavior.
- **Environmental Interference:** Reflections, splashes, and waves can interfere with video clarity, affecting detection accuracy.
- **Dependency on Camera Angle and Placement:** Poor camera positioning can lead to blind spots, reducing the effectiveness of detection.

8. CONCLUSION

The Virtual Eye - Lifeguard for Swimming to Detect Active Drowning represents a groundbreaking advancement in water safety, leveraging cutting-edge technologies YOLO V5 to address the challenges of drowning prevention. By providing real-time monitoring, proactive detection of distress signals, and immediate alerts to responders, this system significantly enhances the ability to prevent drowning incidents and save lives. The integration of this technology as a supplementary tool to traditional lifeguarding ensures greater efficiency, reduces the risk of human error, and allows for faster intervention in emergencies. Its adaptability to various aquatic environments, coupled with its ability to operate under challenging conditions, makes it a reliable and scalable solution for improving safety standards in swimming facilities worldwide. By combining human expertise with intelligent automation, the Virtual Eye has the potential to redefine the future of water safety, creating a safer environment for swimmers and reducing the global burden of drowning-related accidents.

9. Future Scope

The "Virtual Eye – Lifeguard for Active Swimming Drowning Detection" system holds immense potential for future enhancement and large-scale deployment. The system can also be extended to incorporate drone surveillance for wider and more flexible monitoring, especially in open water environments. Real-time location tracking of the drowning individual using GPS or indoor positioning systems can be implemented to guide rescuers precisely and quickly. Mobile application support can allow lifeguards, parents, or pool administrators to receive instant alerts, live video feeds, and swimmer analytics. Voice-based multi-language alert systems and automatic sirens can ensure faster communication in public spaces.

10.Appendix

10.1 Source Code

```
!wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.pt
```

```
[ ] !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/
    !chmod 600 ~/.kaggle/kaggle.json
```

```
[ ] !kaggle datasets download alanoudawaji/swimming-and-drowning-dataset
```

```
[ ] !unzip /content/swimming-and-drowning-dataset.zip
```

```
[ ] !pip install ultralytics
```

```
[ ] from ultralytics import YOLO

    #Loading the model
    model = YOLO('yolov5s.pt')

    print("Model loaded Successfully")
```

1. **Loading the dataset and model**

```
#Training the model
results = model.train(
    data="/content/data.yaml",
    epochs=100,
    imgsz=640,
    augment=True,
    patience=20
)
```

2. **Training the model**

```
[ ] #validating the model
    model.val(data = "/content/data.yaml")
```

3. **Validating the model**

```
# Predict on a test image
model.predict(
    "/content/test/images/-Clipchamp-_mp4-11_jpg.rf.0c2b48fcd0dc1741baded48d49546814.jpg",
    save=True,
    imgsz=320,
    conf=0.6
)
```

4. **Predicting the model**

Create app.py (Python Flask) file:

```python
from flask import Flask, request, render_template, redirect, url_for, send_from_directory
import os
from ultralytics import YOLO
from PIL import Image
from werkzeug.utils import secure_filename

app = Flask(__name__)
model = YOLO(r'D:\VirtualEye\Flask\best.pt')  # Load the YOLO model

# Define upload and prediction folders
UPLOAD_FOLDER = 'static/uploads'
PREDICTION_FOLDER = 'static/predictions'
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'mp4', 'avi', 'mkv'}

# Configure Flask app folders
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['PREDICTION_FOLDER'] = PREDICTION_FOLDER

# Ensure the necessary directories exist
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(PREDICTION_FOLDER, exist_ok=True)

# Helper function to check allowed file types
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# Route: Home page
@app.route('/')
def index():
    return render_template('index.html')
```

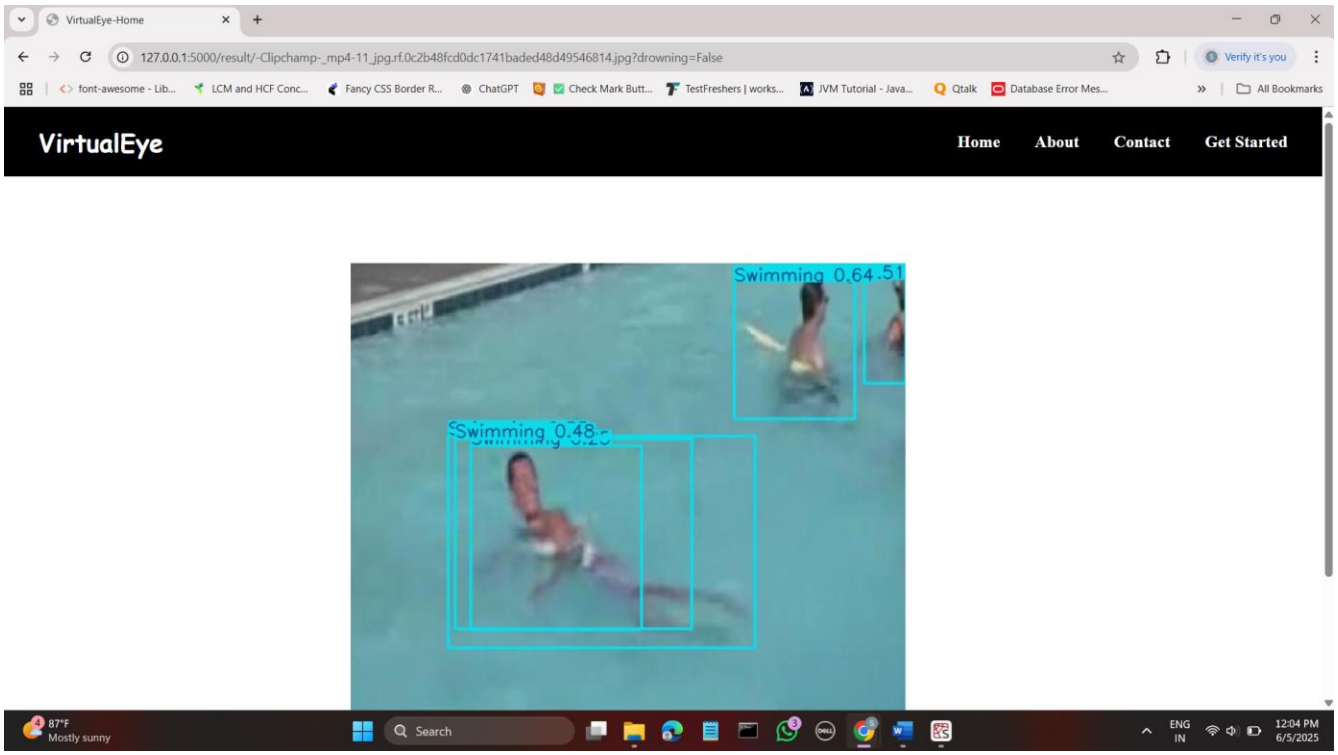
```python

@app.route('/static/<path:filename>')
def static_files(filename):
    return send_from_directory('static', filename)

# Route: About page
@app.route('/about')
def about():
    return render_template('about.html')

# Route: Contact page
@app.route('/contact')
def contact():
    return render_template('contact.html')

# Route: Prediction page
@app.route('/prediction_page', endpoint='prediction_page')
def prediction_page():
    return render_template('prediction-page.html')

# Route: Results page (static)
@app.route('/results')
def results():
    return render_template('results.html')

# Route: Handle file uploads and YOLO predictions
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return "No file part"
```

```python

    file = request.files['file']

    if file.filename == '':
        return "No selected file"

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

        # Initialize detection flag
        drowning_detected = False
```

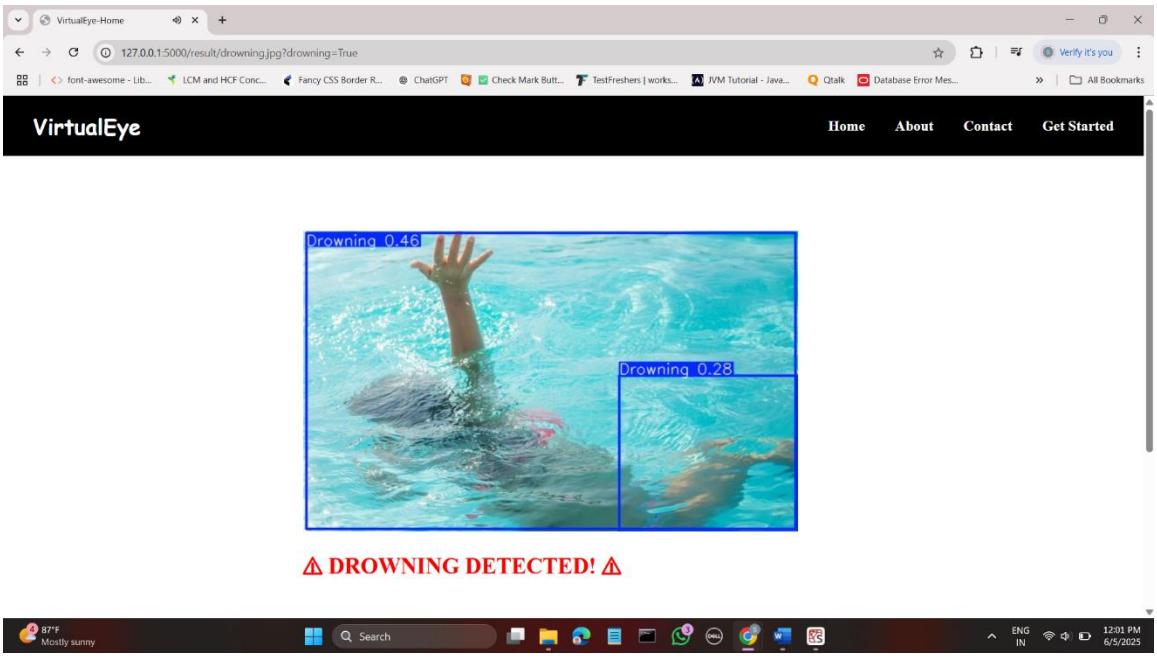
```
        # Run YOLO prediction based on file type
        results = model(filepath, save=True, project=app.config['PREDICTION_FOLDER'],
                        exist_ok=True)

        for result in results:
            if result.names and result.boxes:
                classes = [result.names[int(cls)] for cls in result.boxes.cls]

                if any(cls.lower() == 'drowning' for cls in classes):
                    drowning_detected = True
                    break  # Exit early if drowning is detected

        # Redirect with detection flag as query param
        return redirect(url_for('result', original_filename=filename,
                                drowning=drowning_detected))


    return "Invalid file type"

# Route: Display prediction results
@app.route('/result/<original_filename>')
def result(original_filename):
    drowning = request.args.get('drowning', 'False') == 'True'
    # Find the latest prediction directory
    prediction_dirs = [
        os.path.join(app.config['PREDICTION_FOLDER'], d)
        for d in os.listdir(app.config['PREDICTION_FOLDER'])
        if os.path.isdir(os.path.join(app.config['PREDICTION_FOLDER'], d))
    ]
    if not prediction_dirs:
        return "No prediction directories found"

    latest_folder = max(prediction_dirs, key=os.path.getctime)

    # Find the first prediction file
    prediction_files = os.listdir(latest_folder)
    if not prediction_files:
        return "No predictions found"

    # Return the prediction result
    latest_file = prediction_files[0]

    result_file_url = url_for('static',
            filename=f'predictions/{os.path.basename(latest_folder)}/{latest_file}')
    return render_template('results.html',
                           result_file=result_file_url, drowning_detected=drowning)


# Run the Flask app
if __name__ == '__main__':
    app.run(debug=False)
```

Output Screenshots

Output1:Swimming:



-Output2: Drowning detected:



10.2 GitHub &Project Demo Link:- Project execution Files **Click Here**