

Virtual Eye - Life Guard for Swimming Pools to Detect Active Drowning

Introduction:

In modern urban lifestyles, swimming serves as a popular exercise, offering numerous health benefits. However, beginners, particularly children, often face challenges like difficulty breathing underwater, leading to potential drowning incidents. Worldwide, drowning is a leading cause of accidental death, notably among children under six, accounting for approximately 1.2 million fatalities annually. To address this, a robust AI-based safety system is essential in swimming pools to alert lifeguards to potential drowning situations and prevent accidents.

The Virtual Eye system aims to detect drowning in real-time using a single underwater camera. By leveraging the YOLOv5 model, this solution detects three classes:

- 0: Drowning
- 1: Swimming
- 2: Out of Water

When the system identifies a high drowning probability, it generates an alert to attract the lifeguard's attention.

Scenario 1: Child Floating Motionless

Situation: A young child is seen floating motionless underwater for more than a few seconds.

System Detection: The camera detects the child's still position and classifies it as a high-risk "Drowning" event.

Scenario 2: Active Swimmer Struggling

Situation: An adult swimmer starts struggling to keep their head above water, with irregular movements and a noticeable lack of control.

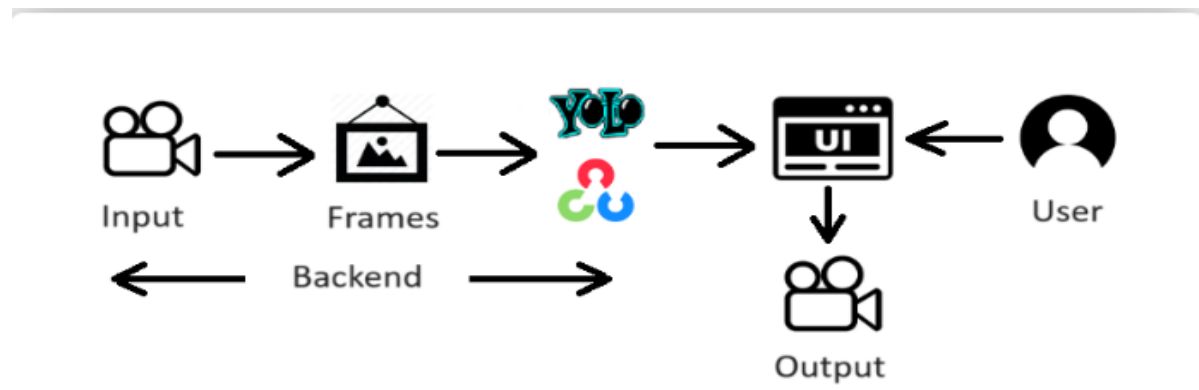
System Detection: The system picks up the swimmer's erratic motion and categorizes it as "Drowning."

Scenario 3: Normal Pool Activity

Situation: Multiple swimmers are in the pool, with some underwater and others swimming on the surface, all in controlled and stable movements.

System Detection: The system classifies these as "Swimming" or "Out of Water" based on individual behaviours. It continually monitors for any shift in behaviour but identifies no immediate risk.

Technical Architecture:



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

Yolov5 - <https://docs.ultralytics.com/models/yolov5/>

Prerequisites:

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

IDE Installation:

Spyder/ PyCharm IDE is Ideal to complete this project

To install Spyder IDE, please refer to [Spyder IDE Installation Steps](#)

To install PyCharm IDE, please refer to the [PyCharm IDE Installation Steps](#)

Python Packages:

If you are using the Anaconda navigator, follow the below steps to download the required packages:

Open the Anaconda prompt.

- Type “pip install opencv-python==4.9.0.80” and click enter.
- Type “pip install numpy ==1.26.3” and click enter
- Type “pip install flask” and click enter.

Project Objectives:

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

- Gain knowledge of OpenCV.
- Gain knowledge of Web Integration.

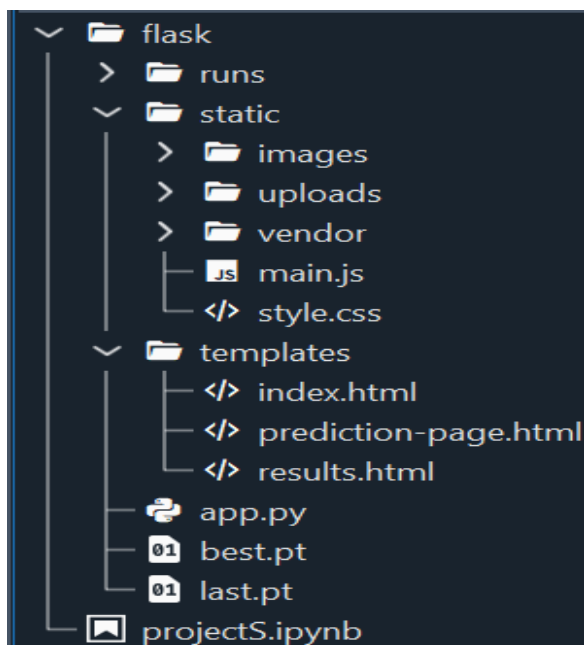
Project Flow:

The user interacts with the UI (User Interface) to choose the image. The chosen image is analyzed by the model which is integrated with the flask application. To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
- Create Train and Test Folders.
- Create data.yaml file
- Training and testing the model
- Save the Model
- Application Building
- Create an HTML file
- Build Python Code

Project Structure:

Create a Project folder which contains files as shown below



- ☐ The Dataset folder contains the training, testing, and val images for training our model.
- ☐ We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting
- ☐ We need the model which is saved and the saved model in this content there is a templates folder containing index.html and inner-page.html pages.

Milestone 1: Collection of Data

Dataset has three classes swimming, drowning, out of water.

Download the Dataset-

<https://www.kaggle.com/datasets/alanoudawaji/swimming-and-drowning-dataset>



Sample data:



Milestone 2: Image Pre-processing

In this milestone, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Import the required libraries

We need to download yoloV5 from the github

```
!wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5n.pt

--2024-11-15 01:26:09-- https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5n.pt
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/264818686/311b8f6d-3436-41e7-96b5-7996c29acc88?X-Amz-Algorithm=
--2024-11-15 01:26:09-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/264818686/311b8f6d-3436-41e7-96b5-7996c29acc88?X
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.111.133, 185.199.110.133, 185.199.108.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3952441 (3.8M) [application/octet-stream]
Saving to: 'yolov5n.pt'

yolov5n.pt          100%[=====] 3.77M  --.-KB/s  in 0.09s

2024-11-15 01:26:10 (39.7 MB/s) - 'yolov5n.pt' saved [3952441/3952441]
```

Activity 2: Loading the Dataset

Download the dataset from the kaggle using CLI it will download zip file and unzip the file to get train, valid, test folders.

```
#!/bin/bash
!kaggle datasets download alanoudawaji/swimming-and-drowning-dataset

Dataset URL: https://www.kaggle.com/datasets/alanoudawaji/swimming-and-drowning-dataset
License(s): unknown
Downloading swimming-and-drowning-dataset.zip to /content
97% 358M/371M [00:03<00:00, 137MB/s]
100% 371M/371M [00:03<00:00, 98.9MB/s]

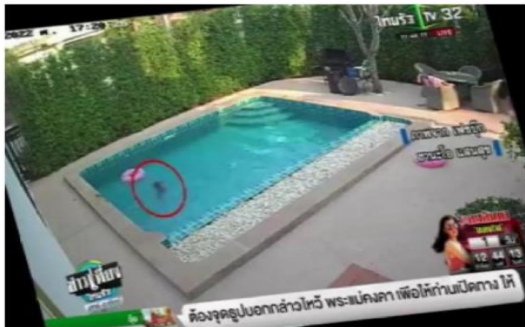
[2] !unzip /content/swimming-and-drowning-dataset.zip

inflating: valid/labels/drowning1_mp4-27_jpg.rf.1f3835c6ac855a859bdcaccc4bac6ddd.txt
inflating: valid/labels/drowning1_mp4-27_jpg.rf.23281b8179ea382be7877a63f0bc922d.txt
inflating: valid/labels/drowning1_mp4-27_jpg.rf.2384323280b96662d6f9dfad128860a0.txt
inflating: valid/labels/drowning1_mp4-27_jpg.rf.520f066deda6ae4c450fe7815a017716.txt
inflating: valid/labels/drowning1_mp4-27_jpg.rf.bdbb93e4e99f73f3fa0e51cacd6bed8b.txt
inflating: valid/labels/drowning1_mp4-28_jpg.rf.43a9f59e1a9fb902fa2583be7884fab7.txt
inflating: valid/labels/drowning1_mp4-28_jpg.rf.4f172699460b9f7b9dc8257ead574070.txt
inflating: valid/labels/drowning1_mp4-28_jpg.rf.6da8643ff01428d67e70f80ab04f012d.txt
inflating: valid/labels/drowning1_mp4-28_jpg.rf.d66f7f0dc9771049d3824dee0c3bea16.txt
inflating: valid/labels/drowning1_mp4-28_jpg.rf.dcf96086b620a95c9dce066675dc7c91.txt
inflating: valid/labels/drowning1_mp4-28_jpg.rf.f247901af628cf154c15a8e60106b599.txt
inflating: valid/labels/drowning1_mp4-29_jpg.rf.1536ac5a8cc5ac7597579d105e8ccfffd.txt
```

Pre-processing:

```
#Data Augmentation
augmented_images = []
for img in resized_images:
    flipped_img = cv2.flip(img, 1) # Horizontal flip
    rotated_img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE) # 90-degree rotation
    augmented_images.extend([flipped_img, rotated_img])

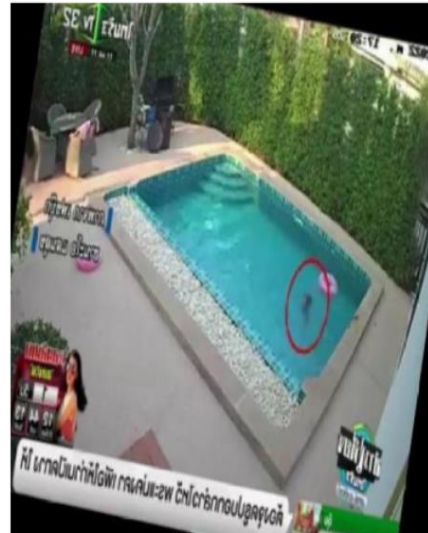
# Display an augmented image as a sample
plt.imshow(cv2.cvtColor(augmented_images[0], cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```




```
#loading data
import cv2
import glob
import matplotlib.pyplot as plt

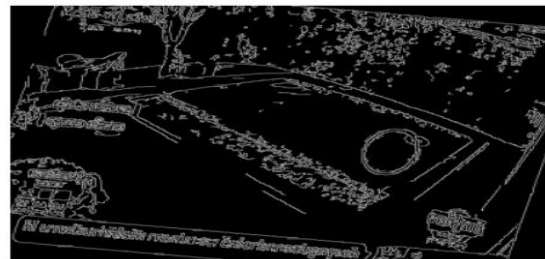
# Load images from a directory - Update with the correct path to your images
image_paths = glob.glob('/content/test/images/*.jpg') # Replace with the actual path
images = [cv2.imread(img_path) for img_path in image_paths]

# Check if any images were loaded
if images:
    # Display first loaded image as a sample
    plt.imshow(cv2.cvtColor(images[0], cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
else:
    print("No images found in the specified directory.")
```



```
#edge detection
edge_detected_images = [cv2.Canny(img, 100, 200) for img in resized_images]

# Display an edge-detected image as a sample
plt.imshow(edge_detected_images[0], cmap='gray')
plt.axis('off')
plt.show()
```



```
#color space
grayscale_images = [cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in resized_images]

# Display a grayscale image as a sample
plt.imshow(grayscale_images[0], cmap='gray')
plt.axis('off')
plt.show()
```



Milestone 3: training

Now it's time to train our Yolo model:

Activity 1: Building the Model

Training yolo v5 model on a custom dataset.

```
[2] Inflatig: valid/labels/drowning20_mp4-17.jpg.rf.15c90e0d9705334e8ec/8308a403409.txt
Inflatig: valid/labels/drowning20_mp4-17.jpg.rf.478da77eab1ba3098f9dd541db22c44d.txt
Inflatig: valid/labels/drowning20_mp4-17.jpg.rf.b3376c6d13fcb0403f5835c5c6494a145c.txt

!pip install ultralytics

Collecting ultralytics
  Downloading ultralytics-8.3.27-py3-none-any.whl.metadata (35 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.26.4)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.8.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.8.1)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (10.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.31.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.13.1)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.5.0+cu121)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.19.0+cu121)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.66.6)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.2.2)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.13.2)
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.10-py3-none-any.whl.metadata (9.4 kB)
Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.27 ultralytics-thop-2.0.10

[3] Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.27 ultralytics-thop-2.0.10

!ls
data.yaml      README.roboflow.txt  swimming-and-drowning-dataset.zip  train
README.dataset.txt  sample_data         test                                valid
+ Code + Text

[5] !ls swimming-and-drowning-dataset/
ls: cannot access 'swimming-and-drowning-dataset/': No such file or directory

[6] !mkdir swimming-and-drowning-dataset

[7] !mv data.yaml /content/swimming-and-drowning-dataset/

[8] !wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5n.pt

[8] yolov5n.pt 100%[=====] 3.77M --.-KB/s in 0.06s
2024-11-05 12:31:52 (64.5 MB/s) - 'yolov5n.pt' saved [3952441/3952441]

from ultralytics import YOLO

# Load the model using the correct filename
model = YOLO("yolov5n.pt") # Ensure this file is in the current directory

# Step 3: Verify successful loading
print("Model loaded successfully.")
# Train the model
results = model.train(data="/content/swimming-and-drowning-dataset/data.yaml", epochs=70, imgsz=640)

Class Images Instances Box(P R mAP50 mAP50-95): 100% 62/62 [00:17<00:00, 3.51it/s]

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
63/70 2.22G 1.284 0.8229 1.397 25 640: 100% 376/376 [01:50<00:00, 3.42it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 62/62 [00:17<00:00, 3.52it/s]

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
70/70 2.26G 1.232 0.759 1.357 34 640: 100% 376/376 [01:50<00:00, 3.41it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 62/62 [00:18<00:00, 3.43it/s]
all 1977 3624 0.809 0.779 0.835 0.488

70 epochs completed in 2.600 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 5.3MB
Optimizer stripped from runs/detect/train/weights/best.pt, 5.3MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.27 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv5n summary (fused): 193 layers, 2,503,529 parameters, 0 gradients, 7.1 GFLOPs
Class Images Instances Box(P R mAP50 mAP50-95): 100% 62/62 [00:20<00:00, 2.98it/s]
all 1977 3624 0.813 0.781 0.838 0.489
Drowning 1657 2130 0.856 0.847 0.901 0.547
Swimming 555 1377 0.806 0.838 0.85 0.471
out of water 80 117 0.778 0.658 0.762 0.448
Speed: 0.3ms preprocess, 2.2ms inference, 0.0ms loss, 2.1ms postprocess per image
Results saved to runs/detect/train
```

Activity 2: Validation

Validating our model using the ‘val’ folder. Also, we have saved our best.pt Download the model weights from run/detect/train/weights/best.pt

```

✓ [12] model.val(data = "/content/swimming-and-drowning-dataset/data.yaml")
25s
0.64565, 0.64665, 0.64765, 0.64865, 0.64965, 0.65065, 0.65165, 0.65265, 0.65365, 0.65465, 0.65566, 0.65666, 0.65766,
names: {0: 'Drowning', 1: 'Swimming', 2: 'out of water'}
✓ [12] plot: True
25s
results_dict: {'metrics/precision(B)': 0.8123269156632354, 'metrics/recall(B)': 0.7835984545581999, 'metrics/mAP50(B)': 0.8368440733851998,
'metrics/mAP50-95(B)': 0.4881379161239458, 'fitness': 0.5230085318500712}
save_dir: PosixPath('runs/detect/val2')
speed: {'preprocess': 0.30939097831390333, 'inference': 4.432705597981215, 'loss': 0.0013969857462379869, 'postprocess': 1.486679008649827}
task: 'detect'

✓ [13] from ultralytics import YOLO
0s
# Load the model using the correct filename
model = YOLO("/content/runs/detect/train/weights/best.pt") # Ensure this file is in the current directory

✓ model.predict("/content/test/images/-Clipchamp-_mp4-11_jpg.rf.0c2b48fcd0dc1741baded48d49546814.jpg", save=True, imgsz=320, conf=0.5)
0s
keypoints: None
masks: None
names: {0: 'Drowning', 1: 'Swimming', 2: 'out of water'}
obb: None
orig_img: array([[106, 105, 107],
[105, 104, 106],
[103, 102, 104],
...,
[157, 149, 112],
[155, 147, 110],

```

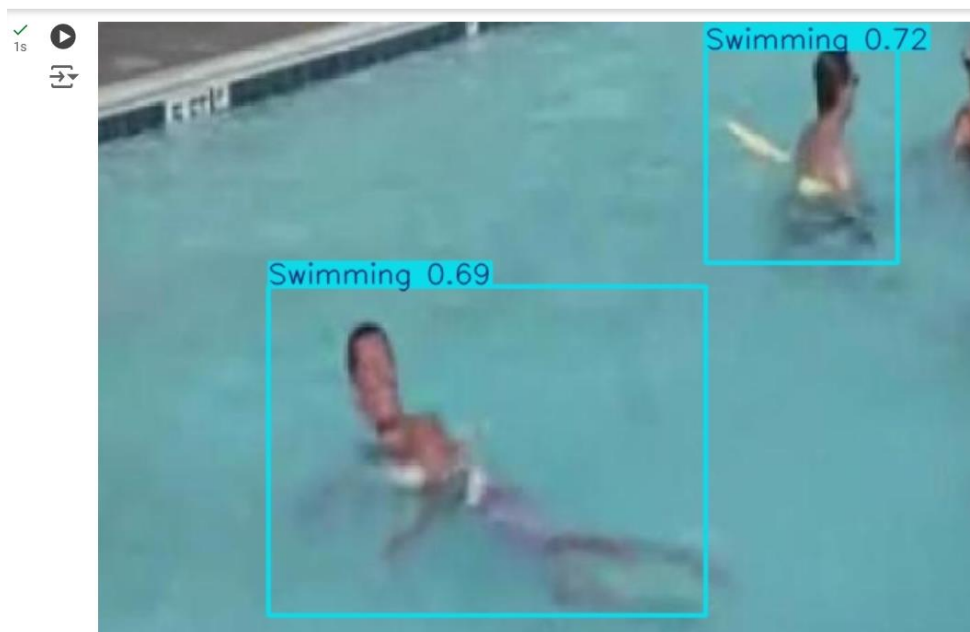
Displaying detected image in training notebook:

```

✓ from PIL import Image
1s
from IPython.display import display
im = Image.open("/content/runs/detect/predict/-Clipchamp-_mp4-11_jpg.rf.0c2b48fcd0dc1741baded48d49546814.jpg")
display(im)

```

displaying a detected image from the saved folder runs/detect/predict4



Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface. In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts

with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

- We use HTML to create the front-end part of the web page.
- Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- home.html displays the home page.
- Intro.html displays an introduction about the project
- upload.html gives the emergency alert For more information regarding HTML <https://www.w3schools.com/html/>
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- Link:CSS, JS

Create app.py (Python Flask) file:

Write the below code in Flask app.py python file script to run the Object Detection Project. To upload image in UI, To display the image in UI:

```
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__)

# Define the path to the model
model_path = 'best.pt'

# Load the model
model = YOLO(model_path)

# model = YOLO('best.pt')

UPLOAD_FOLDER = 'static/uploads'
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'mp4', 'avi', 'mkv'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

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

@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
```

```

if request.method == 'POST':
    if 'file' not in request.files:
        return 'No file part'

    file = request.files['file']

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

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

        if filename.lower().endswith(('jpg', 'jpeg', 'png')):
            img = Image.open(filepath)
            model(img, save=True)

        elif filename.lower().endswith(('mp4', 'avi', 'mkv')):
            model(filepath, save=True)

        return redirect(url_for('result', original_filename=filename))

return render_template('prediction-page.html')

```

```

@app.route('/result/<original_filename>')
def result(original_filename):
    folder_path = 'runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    latest_subfolder = max(subfolders, key=lambda x: os.path.getctime(os.path.join(folder_path, x)))
    directory = folder_path+'/'+latest_subfolder
    print("printing directory: ",directory)
    files = os.listdir(directory)
    latest_file = files[0]

    print(latest_file)

    filename = os.path.join(folder_path, latest_subfolder, latest_file)

    file_extension = filename.rsplit('.', 1)[1].lower()

    environ = request.environ
    if file_extension == 'jpg':
        return send_from_directory(directory,latest_file) #shows the result in seperate tab

    else:
        return "Invalid file format"

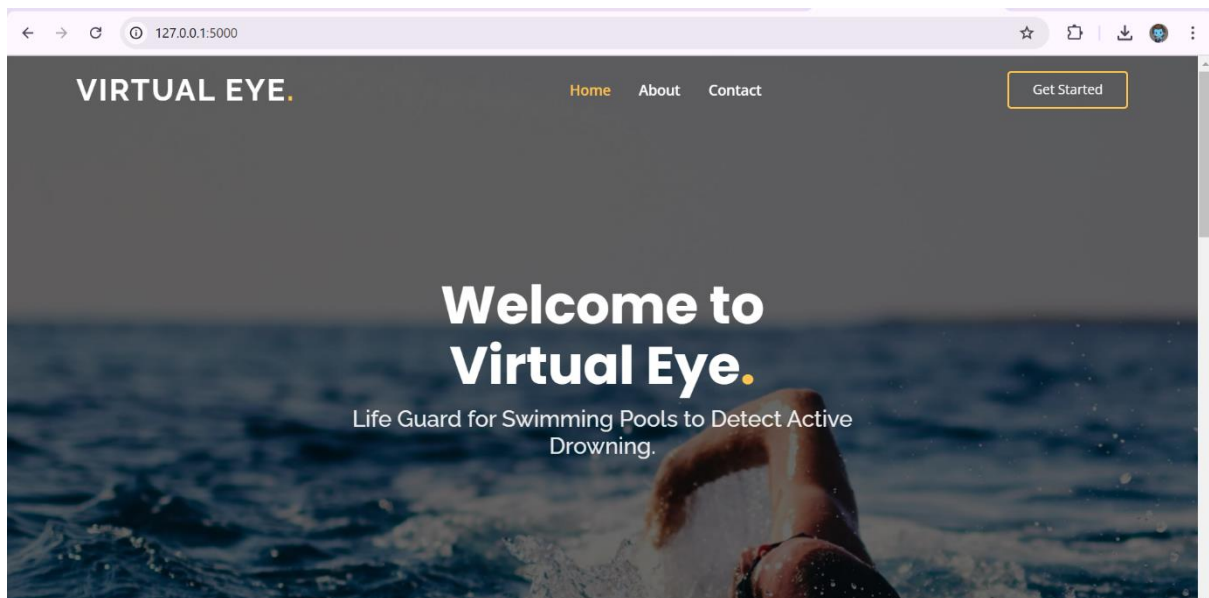
if __name__ == '__main__':
    app.run(debug=True)

```

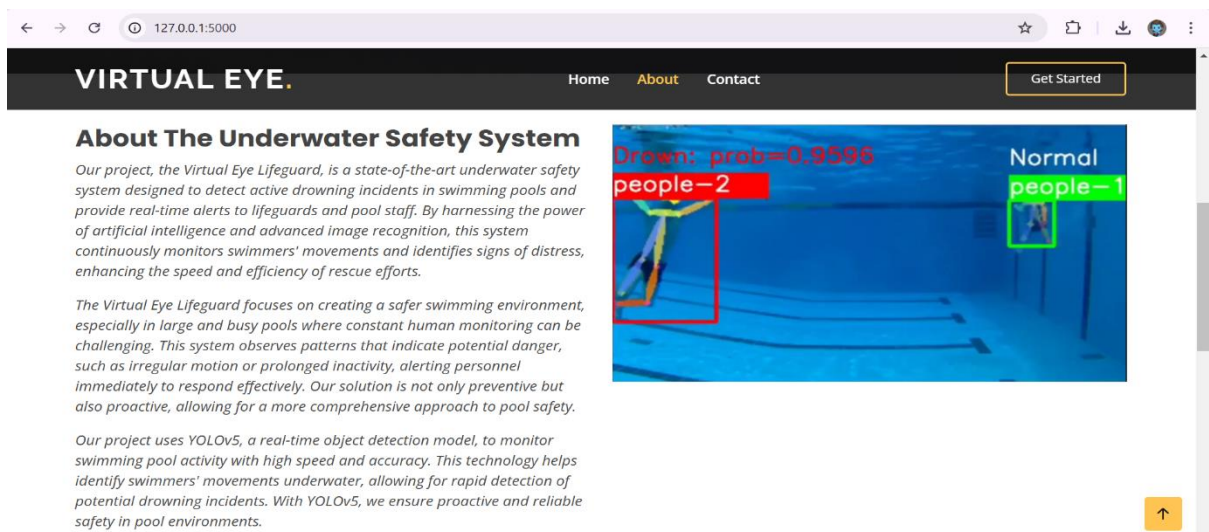
Getting local host in the terminal while running app.py:

```
Console 1/A X
In [1]: runfile('C:/Users/rudro/OneDrive/Desktop/VirtualEye-
Life guard for swimmingpools to detect active drowning/flask/
app.py', wdir='C:/Users/rudro/OneDrive/Desktop/VirtualEye-Life
guard for swimmingpools to detect active drowning/flask')
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
```

Index.html is displayed below:



About Section is displayed below:



Upload and Output Page is displayed below Input:1

VIRTUAL EYE.[Home](#)[About](#)[Contact](#)

Prediction Page[Home](#) / [Prediction Page](#)

Virtual Eye

Upload an image

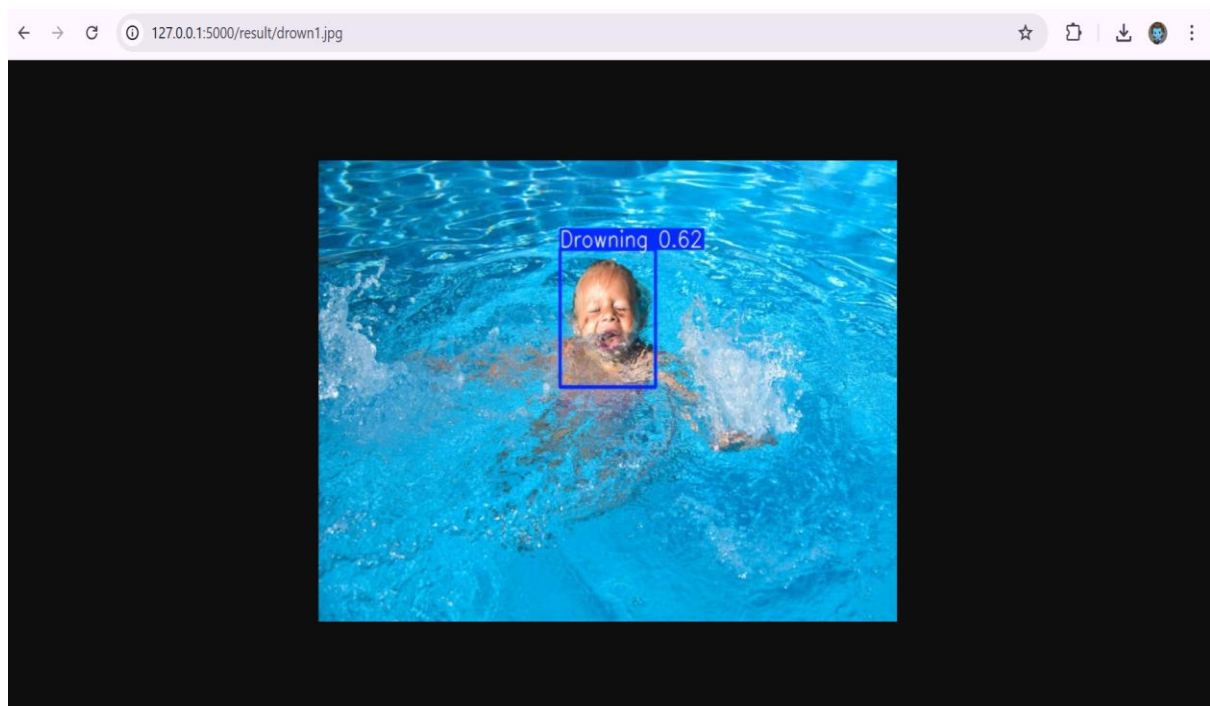
Choose File

drown1.jpg

Submit

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

Output1:



Input:2

127.0.0.1:5000/predict

VIRTUAL EYE. Home About Contact

Prediction Page [Home](#) / [Prediction Page](#)

Virtual Eye

Upload an image

Choose File youtube-8.jpg.rf.8da14f7f6fcc92ef0829a7ace8a56e4c.jpg

Submit

Output: 2

