Arming Against Violence - YOLO-Based Weapon Detection

Introduction :

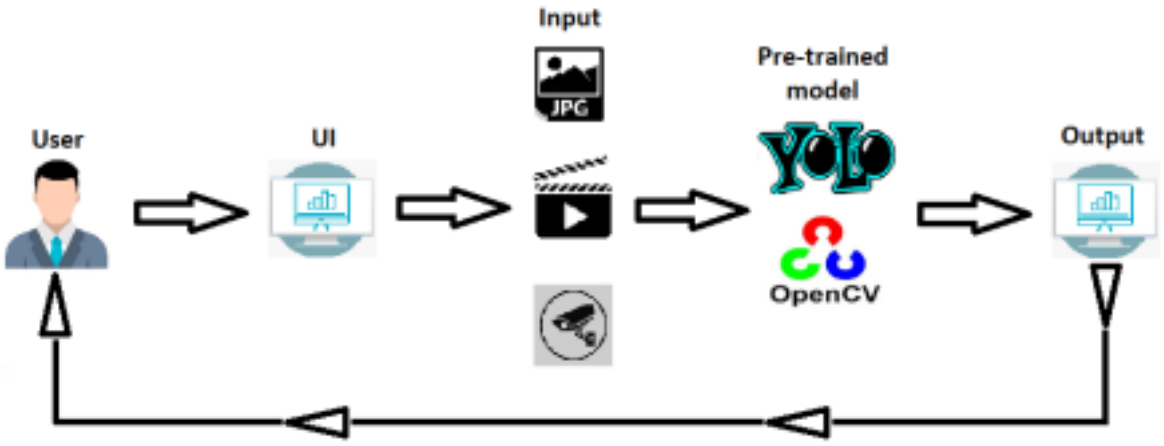
In a contemporary landscape riddled with concerns surrounding public safety and security, the imperative for innovative technologies to address the dynamic challenges posed by acts of violence is more evident than ever. This project endeavors to harness the capabilities of the You Only Look Once (YOLO) algorithm, a real-time object detection system, to create an advanced weapon detection solution.

The YOLO algorithm stands out for its prowess in real-time object detection, making it particularly well-suited for situations demanding swift identification. This project specifically aims to leverage YOLO for the detection of weapons within images and video streams. By training a YOLO model on a diverse dataset encompassing various instances of weapons, the project seeks to develop a robust system capable of real-time weapon detection.

Beyond mere technological innovation, the primary objective of this project is to provide a tool that empowers security personnel, law enforcement agencies, and the wider public with a proactive means to identify and prevent potential acts of violence. Through harnessing the YOLO algorithm, the project endeavors to enhance situational awareness and response times, contributing to a safer and more secure environment.

Ethical and responsible technology application is at the forefront of this initiative. Balancing innovation with societal well-being, the YOLO-based weapon detection project strives to lead in leveraging artificial intelligence for the greater good. Through this undertaking, the aim is to contribute to the broader discourse on public safety and showcase the positive impact that technological advancements can have on collective security.

Technical Architecture:



Pre-requisites:

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

1. **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

2. **Python Packages**

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

Open the Anaconda prompt and create a virtual environment.

- Type “conda create –n weapon python=3.7”. Then activate the environment.
- Type “conda activate weapon” and click enter.
- Type "pip install opencv-python==4.7.0.68” and click enter.
- Type "pip install playsound==1.3.0” and click enter.
- Type "pip install flask” and click enter.

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for computer vision.
- Gain a broad understanding of the YOLO.
- Gain knowledge of OpenCV.

Project Flow:

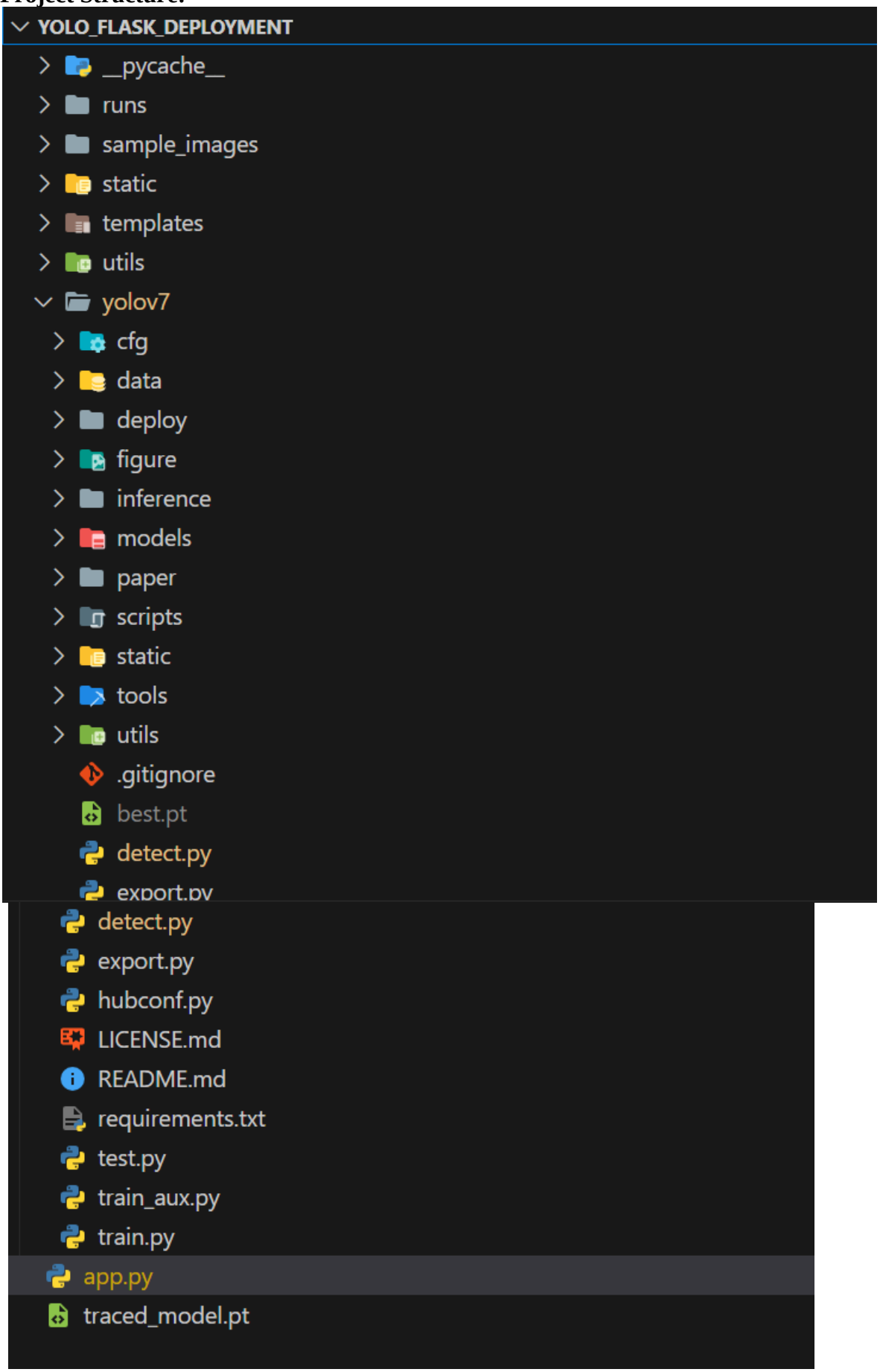
- The user interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated. ● Once the model analyses the input the summary is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

Create detector.py file
- Import the required libraries
- Load Pre-Trained weights
- Load Pre-Trained model with OpenCV
- Weapon detection with bbox

Application building
- Building HTML page
- Build Python code
- Run the application

Project Structure:



The Template folder contains HTML pages. The app.py contains the flask code used to detect the weapon in image, video & live input and if the weapon is detected an alarm will be raised. Alarm sounds and demo videos are presented in project folders. Under training folder detector.py is the python script used to detect weapons.

Milestone 1: Create detect.py python file

Activity 1: Initialization and Setup

```
# Importing necessary libraries and modules
import argparse
import time
from pathlib import Path
import cv2
import torch
from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_s
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized, Trac

# Parsing command line arguments
parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str, default='yolov7.pt', help='mode
# ... (other command line arguments)
opt = parser.parse_args()

# Setting up directories for saving results
save_dir = Path(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))
(save_dir / 'labels' if opt.save_txt else save_dir).mkdir(parents=True, exist_ok=True
```

```
# Setting up directories for saving results
save_dir = Path(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))
(save_dir / 'labels' if opt.save_txt else save_dir).mkdir(parents=True, exist_ok=True

# Initializing logging and selecting computing device
set_logging()
device = select_device(opt.device)
half = device.type != 'cpu'  # half precision only supported on CUDA

# Loading YOLOv7 model
model = attempt_load(opt.weights, map_location=device)
stride = int(model.stride.max())
imgsz = check_img_size(opt.img_size, s=stride)

# ... (other initializations and setups)
```

This step involves setting up the script by importing necessary libraries, parsing command line arguments, initializing directories for saving results, setting up logging, selecting the computing device, and loading the YOLOv7 model.

Activity 2: Model Loading and Configuration

```
# More configurations and loading second-stage classifier if needed
classify = False
if classify:
    modelc = load_classifier(name='resnet101', n=2)
    modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['m

# Setting up data loader based on source (image, video, webcam)
vid_path, vid_writer = None, None
if webcam:
    view_img = check_imshow()
    cudnn.benchmark = True
    dataset = LoadStreams(source, img_size=imgsz, stride=stride)
else:
    dataset = LoadImages(source, img_size=imgsz, stride=stride)

# Getting class names and colors
names = model.module.names if hasattr(model, 'module') else model.names
colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]
```

This step involves additional configurations, loading a second-stage classifier (if needed), and setting up a data loader based on the input source (image, video, or webcam). It also initializes class names and colors for result visualization.

Activity 3: Load pre-trained model with OpenCV

```
# Main inference loop
t0 = time.time()
for path, img, im0s, vid_cap in dataset:
    # ... (preprocessing and preparing input data)

    # Running inference
    with torch.no_grad():
        pred = model(img, augment=opt.augment)[0]

    # Applying Non-Maximum Suppression (NMS)
    pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.class

    # ... (other post-processing steps and result visualization)

# ... (saving results if required)
```

This step involves the main inference loop. It processes each input image or frame, runs inference using the YOLOv7 model, applies non-maximum suppression (NMS), and performs other post-processing steps. The loop iterates over the entire dataset.

Activity 4: Weapon detection with box

```
# Finalizing and cleaning up
if save_txt or save_img:
    s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / '
print(f'Done. ({time.time() - t0:.3f}s)')
```

This step involves finalizing the script and printing information about the execution, such as the number of labels saved (if text saving is enabled) and the total execution time.

Milestone 2: Application Building

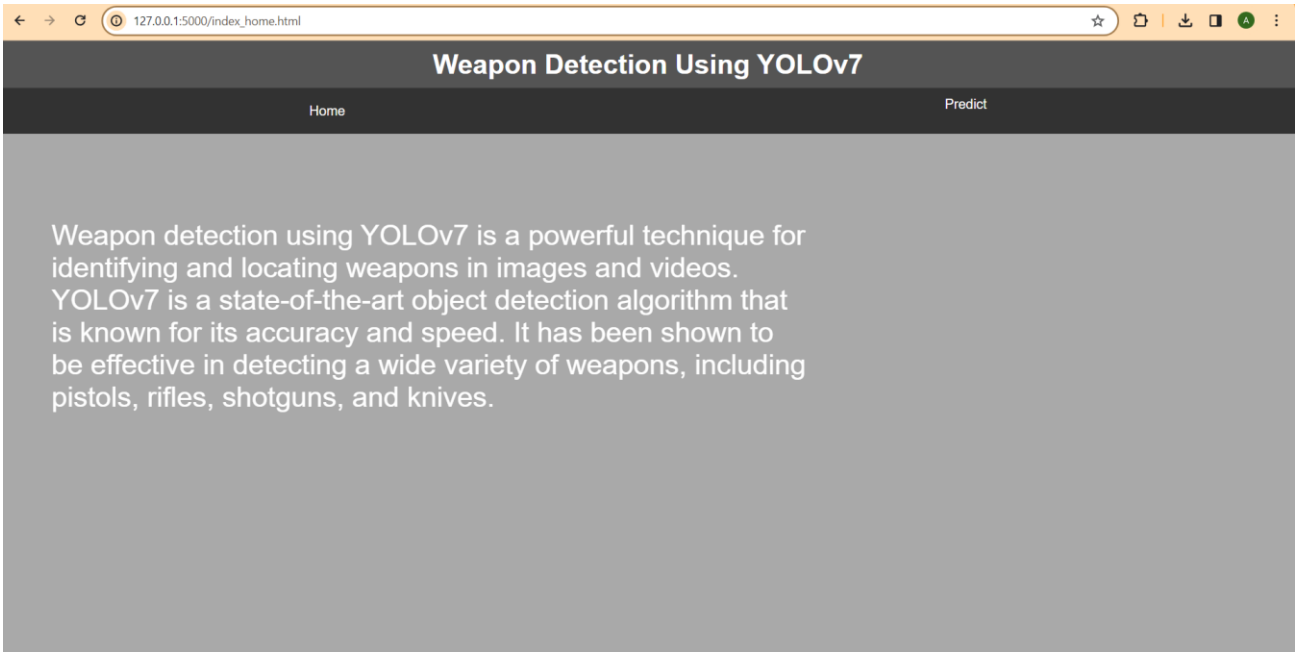
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he/she has to navigate to detect button. Then the video will be showcased on the UI.

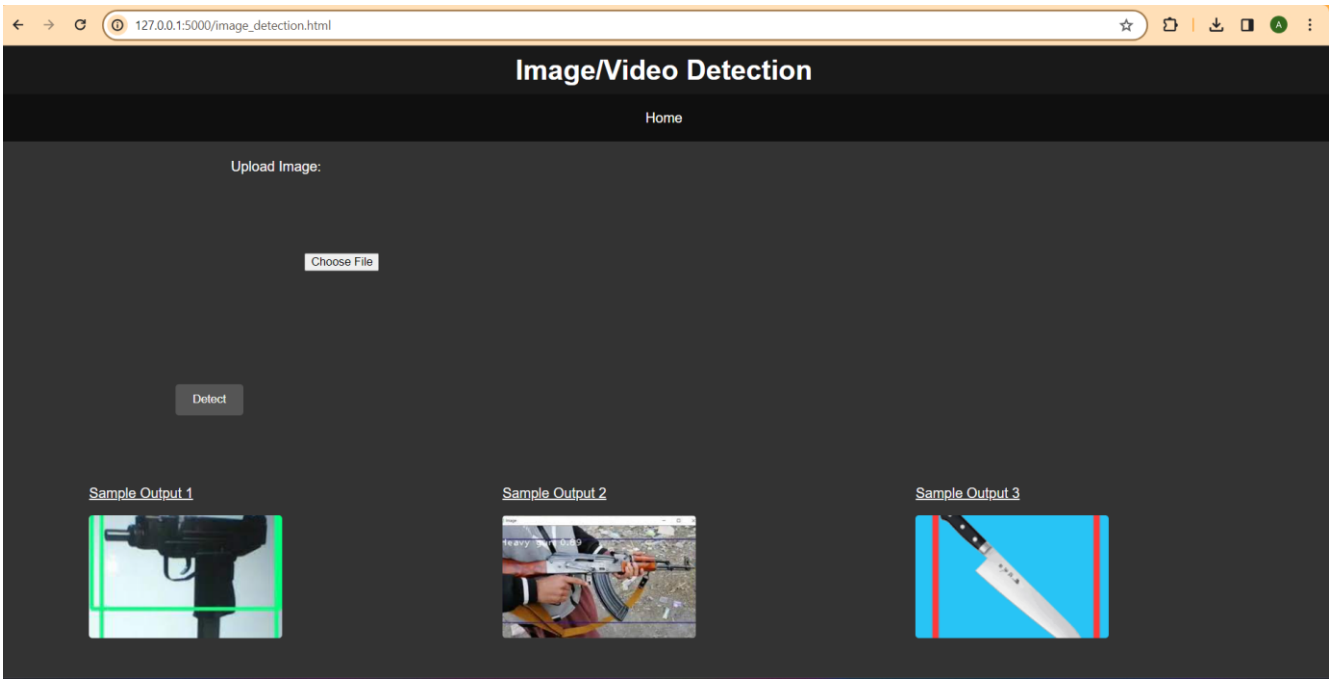
This section has the following tasks

- Building HTML Pages
- Building server-side script

Activity1: Building Html Pages:

For this project we have created 2 HTML files and saved them in the templates folder. Let's see how those html pages looks like:





Activity 2: Build Python code:

- Import the libraries

```
from flask import Flask, render_template, request, redirect, url_for, jsonify
import os
import subprocess
from werkzeug.utils import secure_filename
```

- Importing the flask module into the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as an argument.

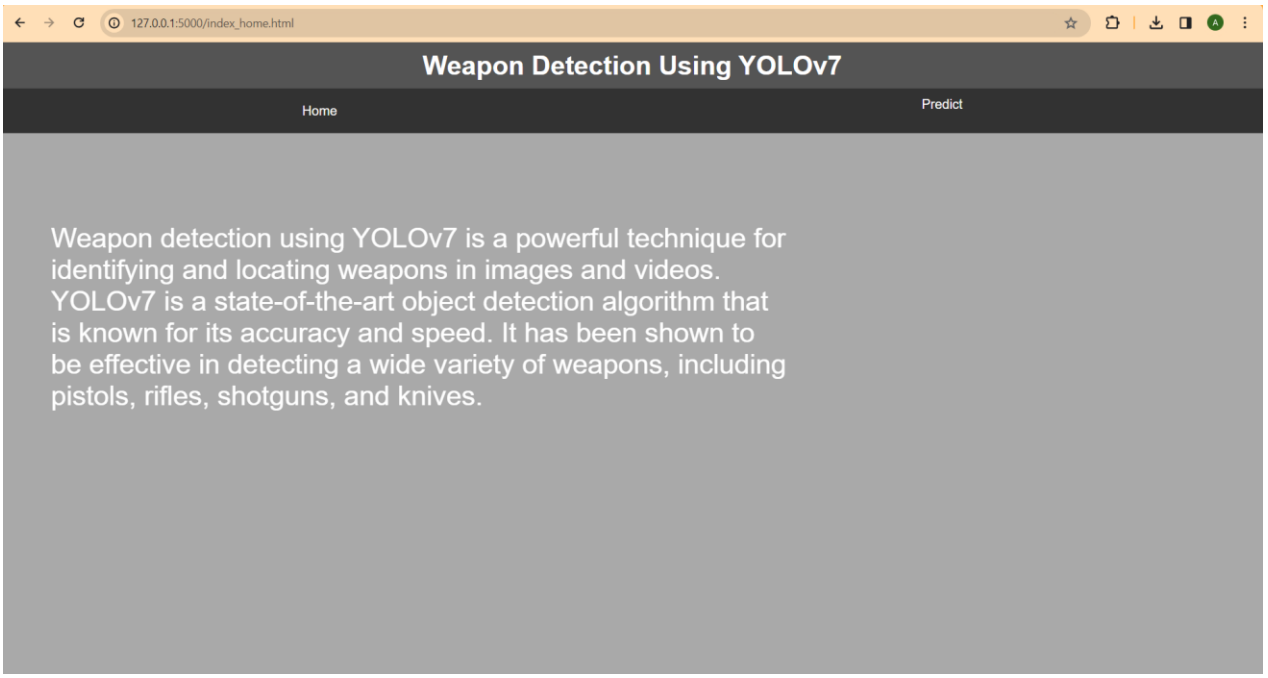
```
app = Flask(__name__)
```

- As discussed in milestone 1, create variables to pass the path of weights file and path of cfg file. Now read the file by opencv and set the input parameters.

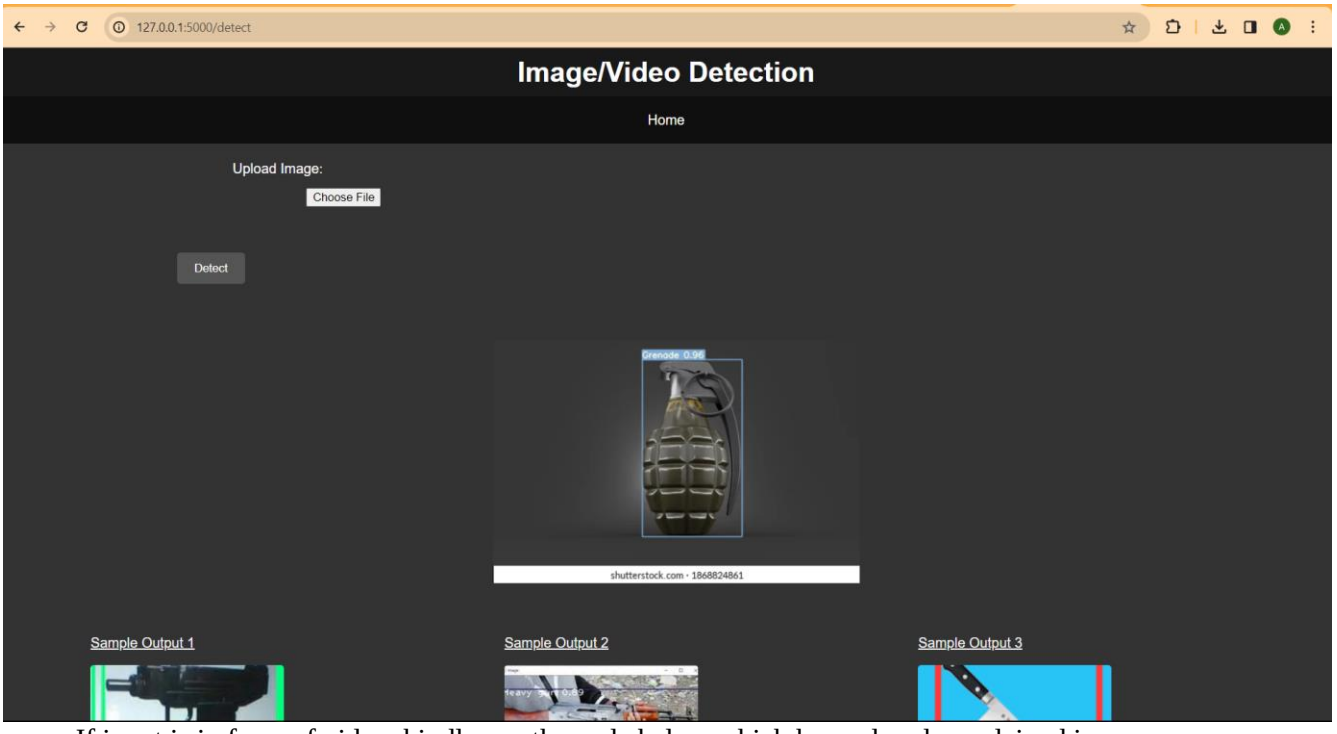
```
subprocess.run([
    'python', yolo_script_path,
    '--weights', 'yolov7/best.pt',
    '--img-size', '640',
    '--conf', '0.25',
    '--source', temp_image_path
])
```

Render HTML page:

- Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.



- If input is in form of image, kindly use the code below which have already explained in milestone 1. Change 2 lines of code, instead of VideoCapture() method kindly go with imread() method.



- If input is in form of video, kindly use the code below which have already explained in milestone 1.

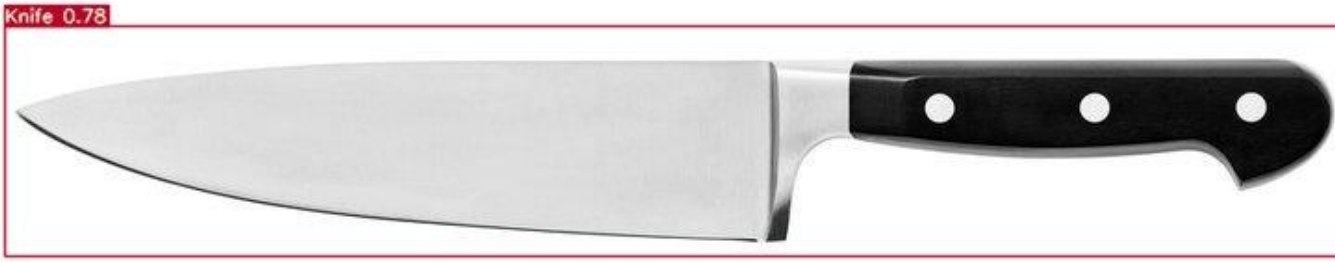
Activity 3: Run the application

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

```
>>
 * 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 stat
 * Debugger is active!
 * Debugger PIN: 249-700-862
127.0.0.1 - - [22/Nov/2023 23:15:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Nov/2023 23:15:33] "GET /static/styles1.css HTTP/1.1" 304 -
127.0.0.1 - - [22/Nov/2023 23:15:37] "GET /image_detection.html HTTP/1.1" 200 -
```

Output:

Output for Image as Input



Outut for Video as Input

hand_-_15563_360p.mp4