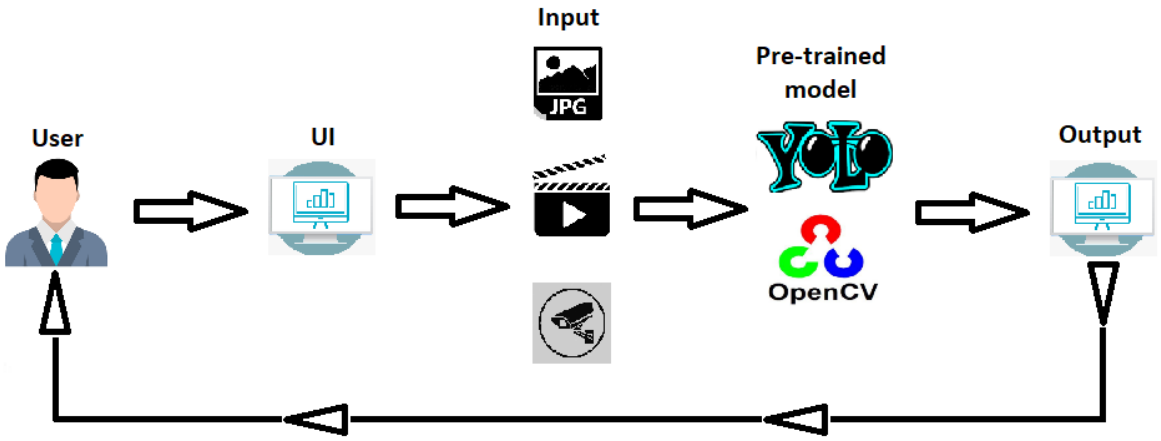
Arming Against Violence - YOLO-Based Weapon Detection

Project Description:

A YOLO-based weapon detection project would involve using the YOLO (You Only Look Once) algorithm to detect weapons in images or video streams. The YOLO algorithm is a real-time object detection system that is particularly well-suited for detecting multiple objects in an image or video frame. The project would likely involve training a YOLO model on a dataset of images and videos that contain weapons, and then using the trained model to detect weapons in new images or videos. The goal of the project would likely be to provide a tool that can help to identify and prevent acts of violence by detecting weapons in real-time.

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.

Prior Knowledge:

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

- YOLO - https://www.youtube.com/watch?v=ag3DLKsl2vk
- 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:

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

Project Flow:

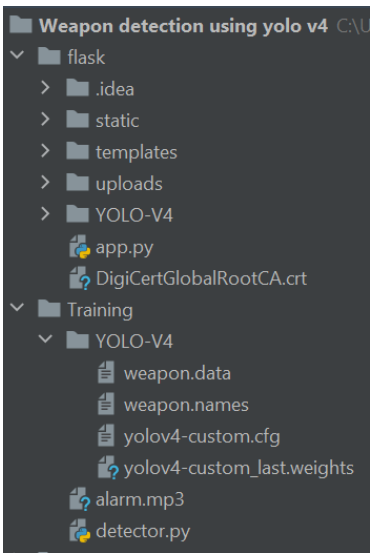
- The user interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once the model analyses the input the summary is showcased on the UI

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 and alarm

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 detector.py python file

Activity 1: Import the required libraries

We will be importing the necessary packages initially.

```
import cv2
from playsound import playsound
```

Activity 2: Load pre-trained weights

The weapon detection (Knife & Handgun) model is trained with Yolo-v4. The weights and configuration of the model are saved and provided to you. Now just load the pre-trained files with a unique variable. For the weights and cfg files, kindly reach your mentor.

```
# Load weight file
weight = r"YOLO-V4/yolov4-custom_last.weights"
# Load config file
config = r"YOLO-V4/yolov4-custom.cfg"
# Initialize class names
classes = ["Handgun", "Knife"]
```

Activity 3: Load pre-trained model with OpenCV

Read input video and pass pre-trained model weights to the OpenCV.

- OpenCV has a VideoCapture() method to load the video from python code.
- The weight and configuration of the Yolo model are passed to the deep neural network by cv2.dnn.readNet() to read files by OpenCV.
- Now, OpenCV detects the model with the help of the dnn_DetectionModel() method and for setting up the parameters setInputParams() is used.

```
# Input 1
cap = cv2.VideoCapture("input_1.mp4")
# Loading yolo with opencv
net = cv2.dnn.readNet(weight, config)
model = cv2.dnn_DetectionModel(net)
model.setInputParams(scale=1/255, size=(416, 416))
```

Activity 4: Weapon detection with bbox and alarm

When a weapon is detected, that weapon should be highlighted with a bounding box and it should play an alarm sound.

- The read() method is used to read the video input frame by frame and for reducing the video pixels resize() method from OpenCV is used. As a parameter, pass the video frame and required size.
- The output of the Yolo model will be in the form of id, scores, and bbox. So three variables are initialized to the detect() method. As a parameter, pass the video frame and threshold values.
- To draw the bounding box, OpenCV has the rectangle() method, and to visualize the detected class and accuracy above the bbox, the putText() method is used.
- With the help of conditions the alarm will be raised if accuracy is greater than 30%.

```
weapon = False
while True:
    # Capturing frames from video
    _, frame = cap.read()
    # Resizing frame
    frame = cv2.resize(frame, (640, 480))
    # Initialize variables for output of pre-trained model
    classID, scores, bboxes = model.detect(frame, nmsThreshold=0.4, confThreshold=0.5)
    for classID, scores, bboxes in zip(classID, scores, bboxes):
        x, y, w, h = bboxes
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
        cv2.putText(frame, (classes[classID]+' accuracy: '+(str(round((scores*100), 2)))+'%'), (x, y - 10),
                    cv2.FONT_HERSHEY_PLAIN, 1, (255, 0, 0), 2)
        cv2.putText(frame, (classes[classID]+' Detected'), (20,30), cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 255), 2)
        if scores >= 0.3:
            weapon = True
    cv2.imshow('video', frame)
    if weapon==True:
        playsound("alarm.mp3")
    # Terminate the video
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

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.

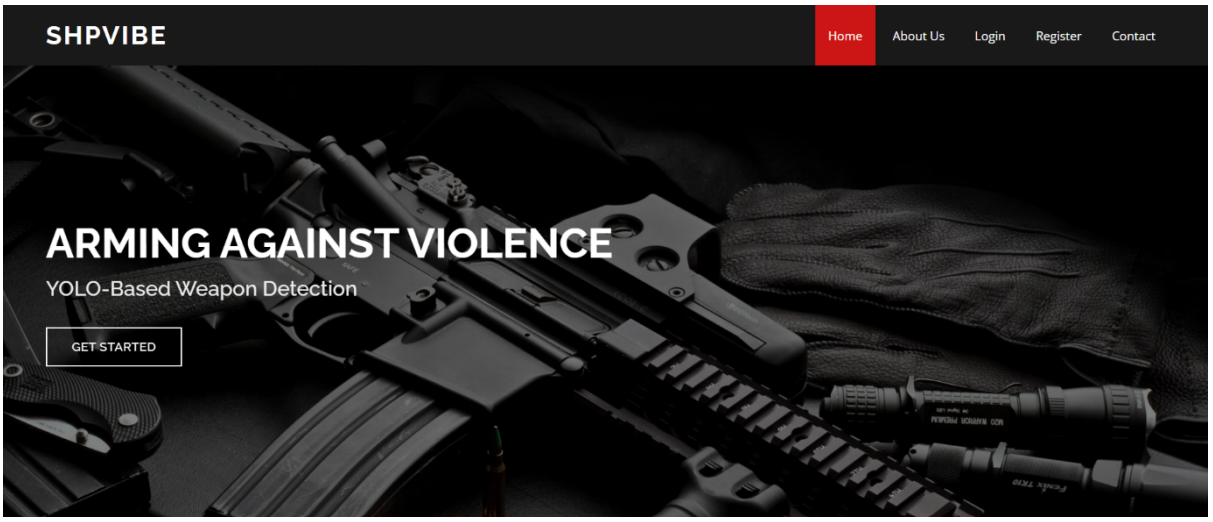
This section has the following tasks

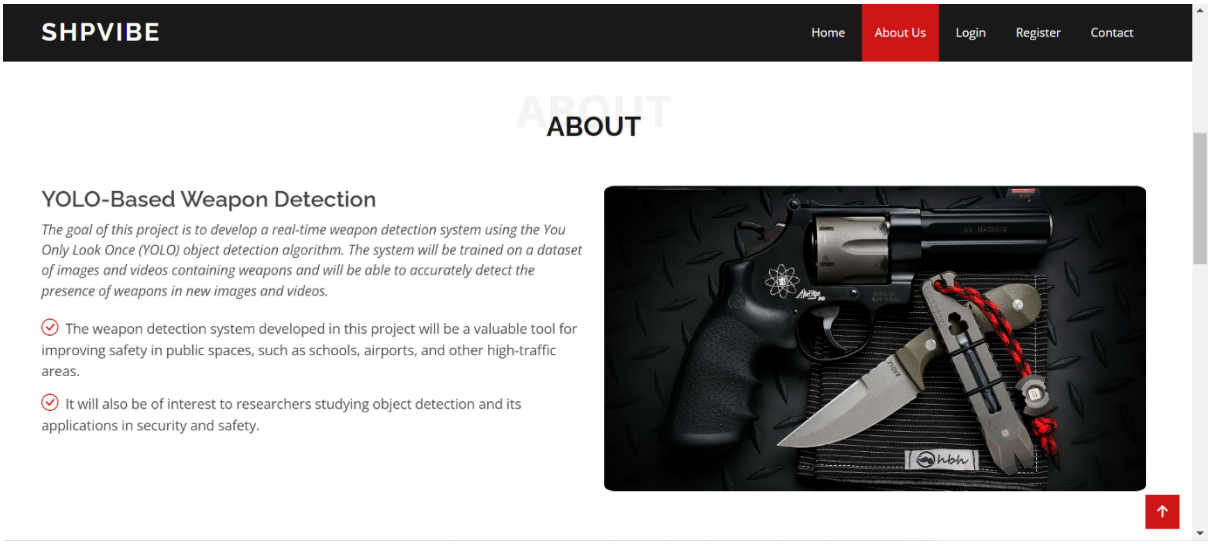
- Building HTML Pages
- Building server-side script

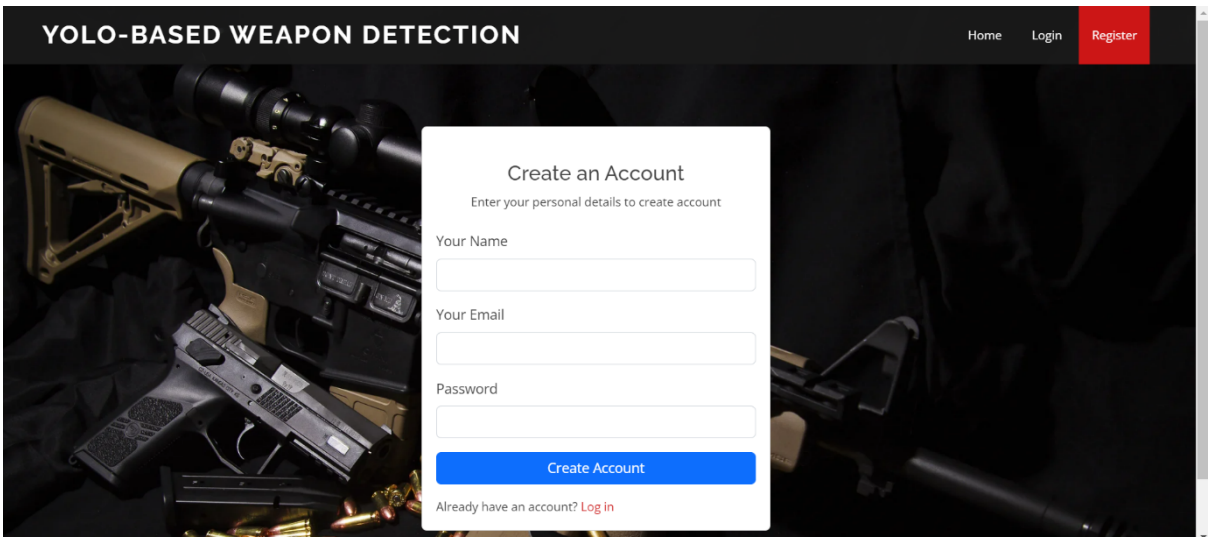
Activity1: Building Html Pages:

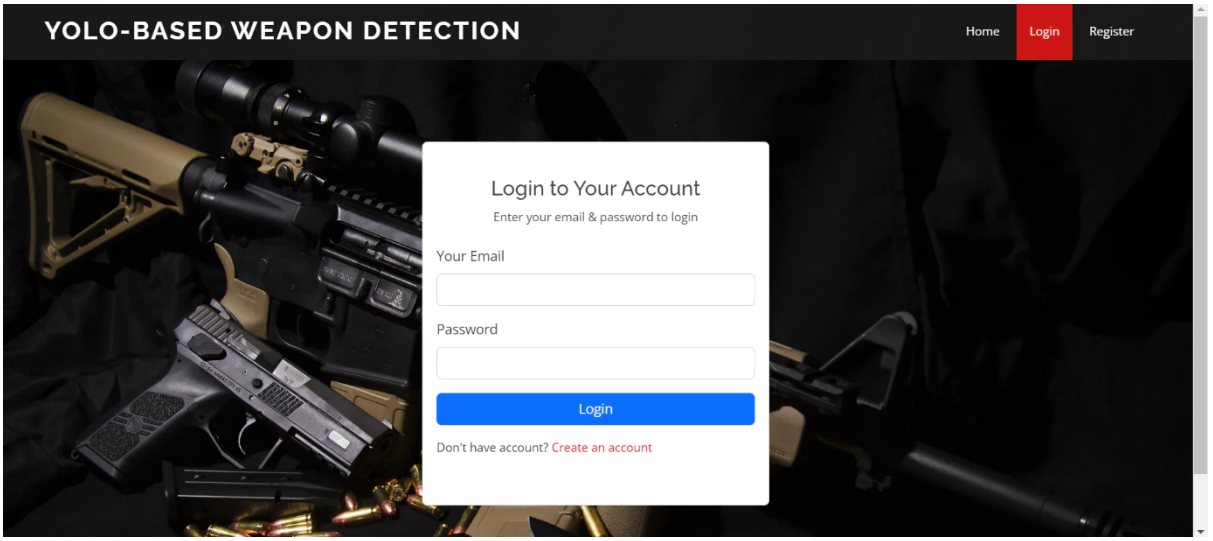
For this project we have created 7 HTML files and saved them in the templates folder.

Let’s see how those html pages looks like:

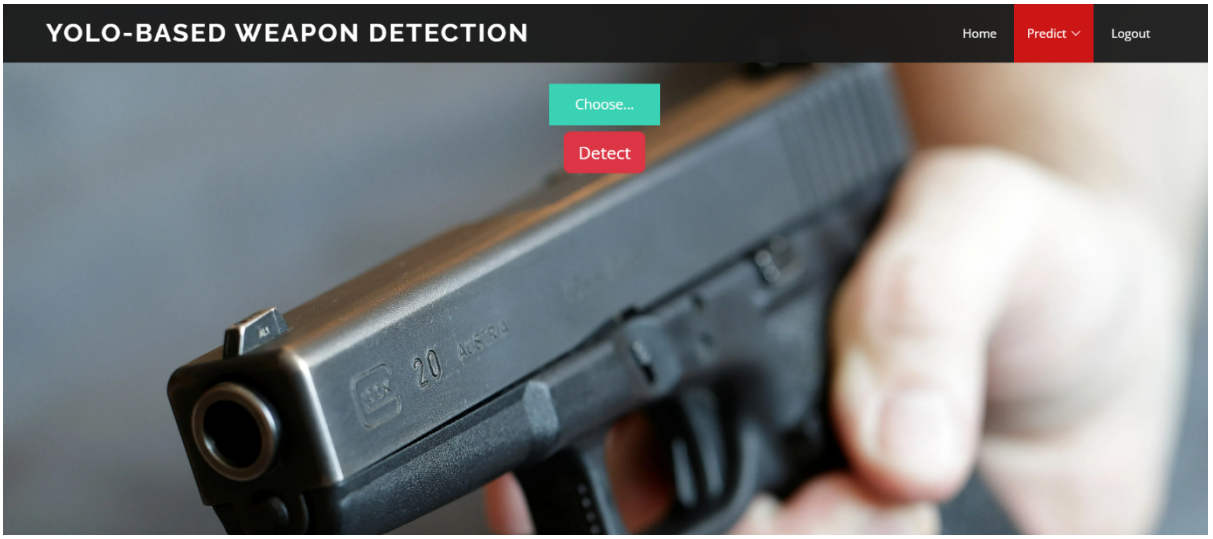


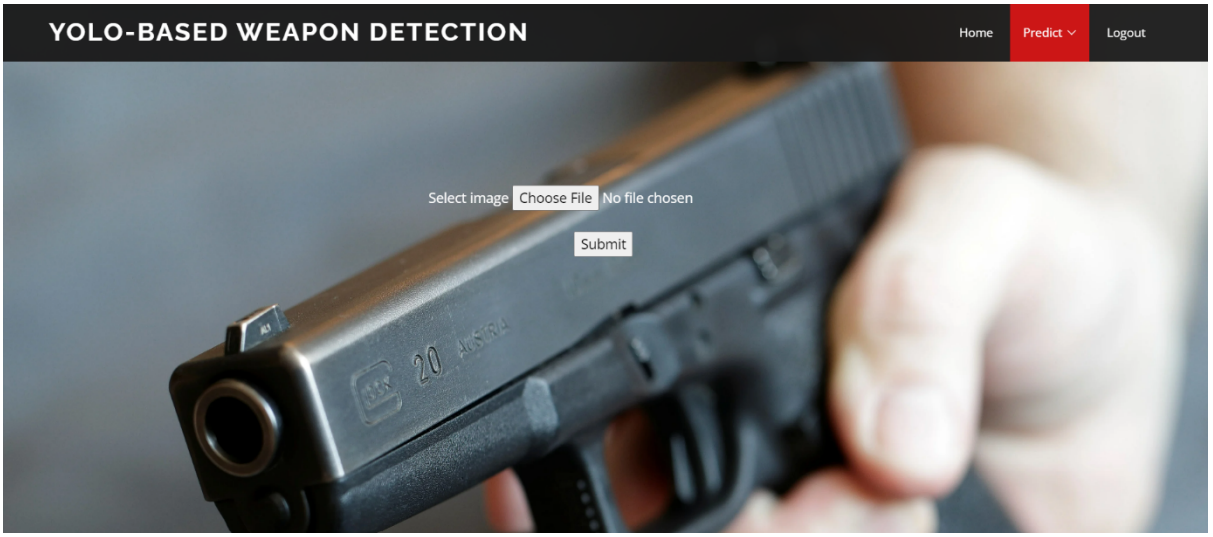


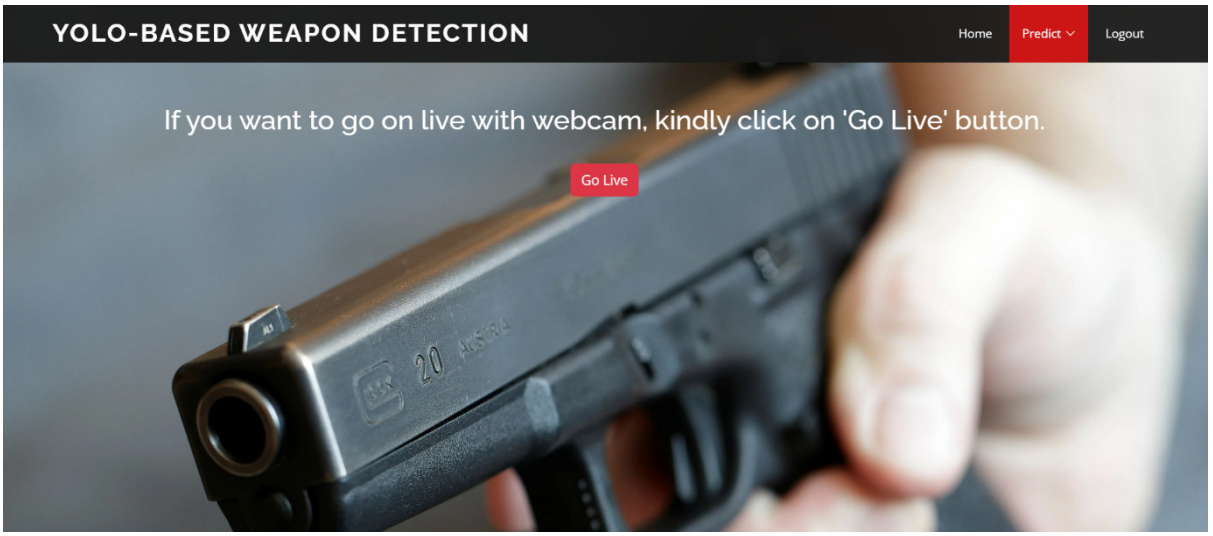

YOLO-BASED WEAPON DETECTION
Home
Login
Register
Create an Account
Enter your personal details to create account
Your Name
Your Email
Password
Create Account
Already have an account? Log in


YOLO-BASED WEAPON DETECTION
Home
Login
Register
Login to Your Account
Enter your email & password to login
Your Email
Password
Login
Don't have account? Create an account


YOLO-BASED WEAPON DETECTION
Demo
Predict
Logout
Image As Input
Video As Input
Live (WebCam)
DEMO
The below video is the demo output of the yolo project.When ever a gun or knife detected, an alert will be given in form of alarm.
clideo.com







Activity 2: Build Python code:

- Import the libraries

```
from flask import Flask, render_template, request, session
import cv2
import os
from playsound import playsound
import ibm_db
import re
```

- 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__)
```

- From ibm_db package, connect() method is used to connect DB service from IBM to Flask framework.
- To understand about IBM DB, refer the link https://www.ibm.com/docs/en/db2/11.5?topic=framework-application-development-db

```
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=2d46b6b
print("connected")
```

- 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.

```
# Load weight file
weight = r"YOLO-V4/yolov4-custom_last.weights"
# Load config file
config = r"YOLO-V4/yolov4-custom.cfg"
# Initialize class names
classes = ["Handgun", "Knife"]
# Loading yolo with opencv
net = cv2.dnn.readNet(weight, config)
model = cv2.dnn_DetectionModel(net)
model.setInputParams(scale=1/255, size=(416, 416))
```

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.

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


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


@app.route('/home')
def home1():
    return render_template('after_login.html')


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

- Fetching user name, email and password from web page and passing those values to the register table which we have created in IBM DB with SQL query.

```python
@app.route("/reg", methods=['POST', 'GET'])
def signup():
    msg = ''
    if request.method == 'POST':
        name = request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER_WEAPON WHERE name= ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            return render_template('login.html', error=True)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address!"
        else:
            insert_sql = "INSERT INTO REGISTER_WEAPON VALUES (?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            # this username & password is should be same as db-2 details & order also
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.execute(prep_stmt)
            msg = "You have successfully registered !"
    return render_template('login.html', msg=msg)
```

- Fetching email and password from login web page and matching with our register table. If email and password are matched, it'll render to next template. If its not matched we'll get an alert in login web page as Incorrect email/password.

```python
@app.route("/log", methods=['POST', 'GET'])
def login1():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER_WEAPON WHERE EMAIL=? AND PASSWORD=?"  # from db2 sql table
        stmt = ibm_db.prepare(conn, sql)
        # this username & password is should be same as db-2 details & order also
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['EMAIL']
            session['email'] = account['EMAIL']
            return render_template('after_login.html')
        else:
            msg = "Incorrect Email/password"
            return render_template('login.html', msg=msg)
    else:
        return render_template('login.html')
```

- 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.

```python
@app.route('/predict', methods=["GET", "POST"])
def img_pred():
    if request.method == 'POST':
        f = request.files['file']  # requesting the file
        basepath = os.path.dirname('__file__')  # storing the file directory
        filepath = os.path.join(basepath, "uploads", f.filename)  # storing the file in uploads folder
        f.save(filepath)  # saving the file

        img = cv2.imread(filepath)  # load and reshaping the image
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Initialize variables for output of pre-trained model
        classID, scores, bboxes = model.detect(img, nmsThreshold=0.4, confThreshold=0.3)
        for classID, scores, bboxes in zip(classID, scores, bboxes):
            x, y, w, h = bboxes
            if scores >= 0.7:
                cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 1)
                cv2.putText(img, (classes[classID] + ' accuracy: ' + (str(round((scores * 100), 2))) + '%'),
                            (x, y - 5),
                            cv2.FONT_HERSHEY_PLAIN, 1, (255, 0, 0), 1)
                cv2.putText(img, 'Weapon Detected', (20, 30), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
                # playsound("alarm.mp3")
                # weapon = True
        cv2.imshow('output', img)
        cv2.waitKey(0)
```

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

```python
@app.route('/predict_video', methods=["GET", "POST"])
def vid_pred():
    if request.method == 'POST':
        f = request.files['file']  # requesting the file
        basepath = os.path.dirname('__file__')  # storing the file directory
        filepath = os.path.join(basepath, "uploads", f.filename)  # storing the file in uploads folder
        print(filepath)
        f.save(filepath)  # saving the file

        # Input 2
        cap = cv2.VideoCapture(filepath)
        weapon = False
        while True:
            # Capturing frames from video
            _, frame = cap.read()
            # print(frame.shape)
            # Resizing frame
            frame = cv2.resize(frame, (640, 480))
            # Initialize variables for output of pre-trained model
            classID, scores, bboxes = model.detect(frame, nmsThreshold=0.4, confThreshold=0.3)
            for classID, scores, bboxes in zip(classID, scores, bboxes):
                x, y, w, h = bboxes
                if scores >= 0.7:
                    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 1)
                    cv2.putText(frame, (classes[classID] + ' accuracy: ' + (str(round((scores * 100), 2))) + '%'), (x, y - 5), cv2.FONT_HERSHEY_PLAIN, 1, (255, 0, 0), 1)
                    cv2.putText(frame, 'Weapon Detected', (20, 30), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
                    # playsound("alarm.mp3")
                    # weapon = True
            cv2.imshow('video', frame)
            if weapon==True:
                playsound("alarm.mp3")
            # Terminate the video
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

- If input live input, kindly use the code below which have already explained in milestone 1. Just change a value (instead of video path, go with ‘0’ parameter which triggers the webcam).

```python
@app.route('/predict_live')
def liv_pred():
    # Input 2
    capp = cv2.VideoCapture(0)
    weapon = False
    while True:
        # Capturing frames from video
        _, frame = capp.read()
        #print(frame.shape)
        # Resizing frame
        #frame = cv2.resize(frame, (640, 480))
        # Initialize variables for output of pre-trained model
        classID, scores, bboxes = model.detect(frame, nmsThreshold=0.4, confThreshold=0.3)
        for classID, scores, bboxes in zip(classID, scores, bboxes):
            x, y, w, h = bboxes
            if scores >= 0.7:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 1)
                cv2.putText(frame, (classes[classID] + ' accuracy: ' + (str(round((scores * 100), 2))) + '%'), (x, y - 5), cv2.FONT_HERSHEY_PLAIN, 1, (255, 0, 0), 1)
                cv2.putText(frame, 'Weapon Detected', (20, 30), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
                # playsound("alarm.mp3")
                # weapon = True
        cv2.imshow('video', frame)
        if weapon==True:
            playsound("alarm.mp3")
        # Terminate the video
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    capp.release()
    cv2.destroyAllWindows()
```

Main Function:

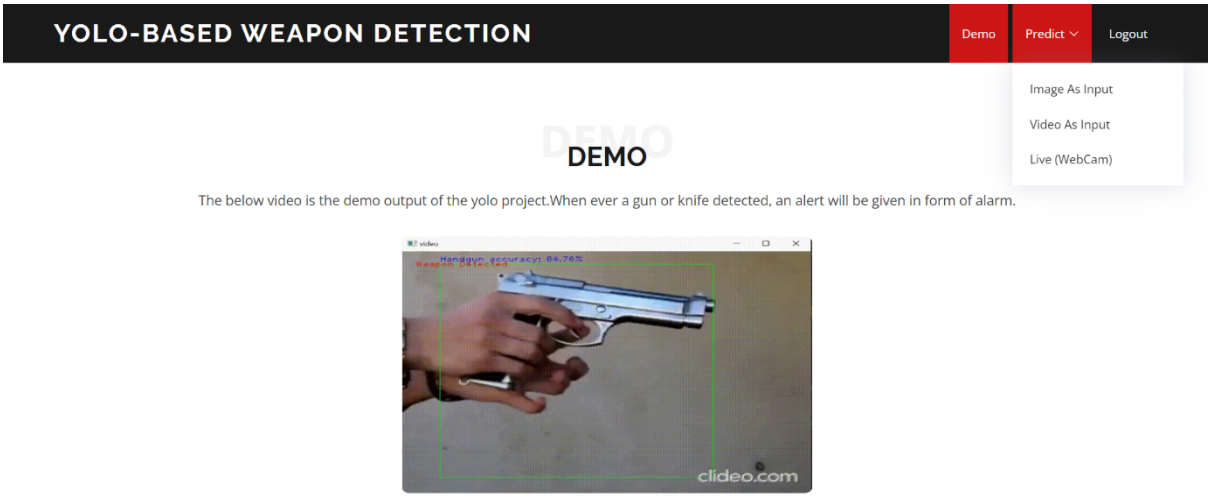
```
if __name__ == "__main__":
    app.run(debug=True)
```

Activity 3: Run the application

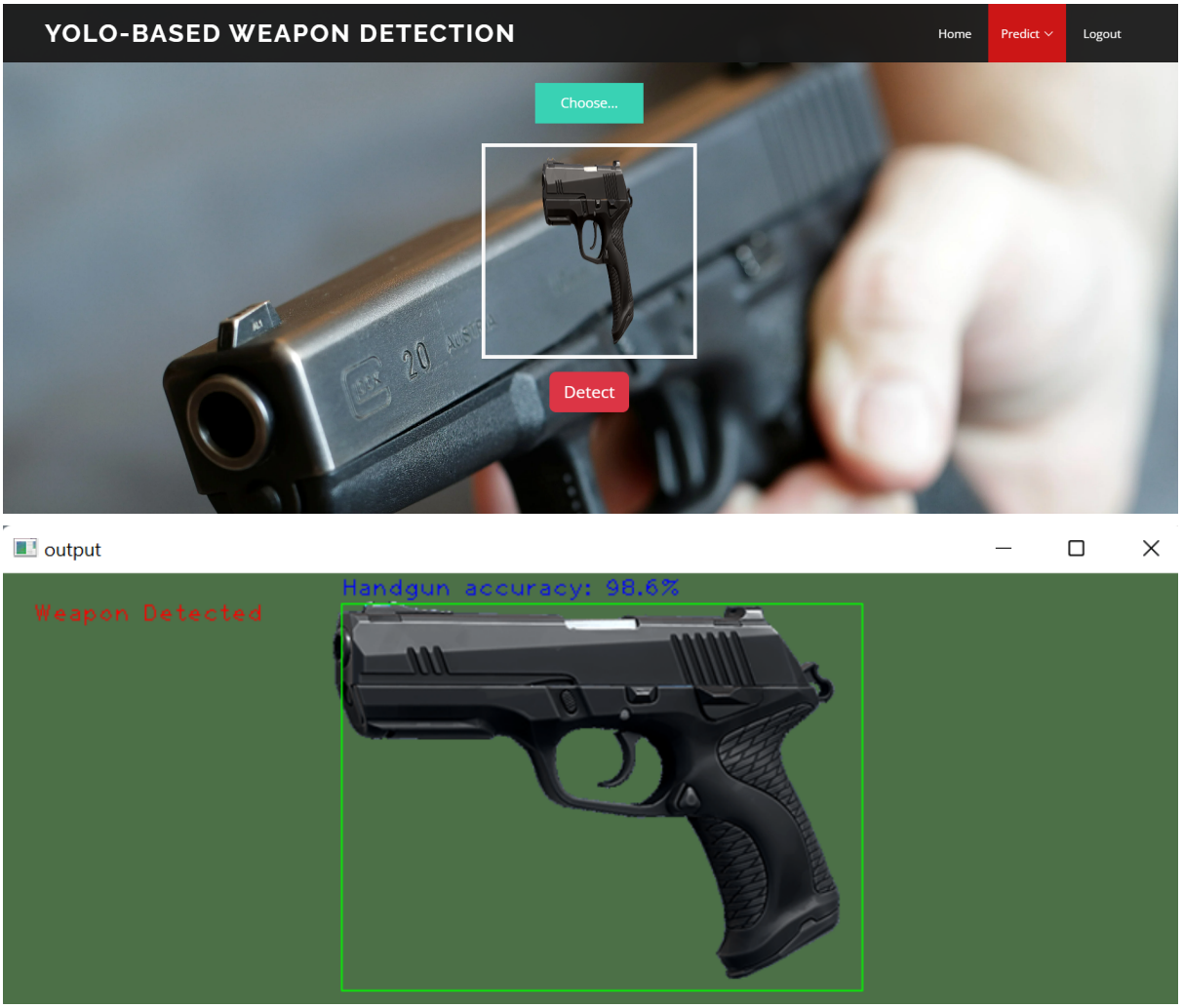
- Open the anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type the “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button,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

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 135-972-108
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

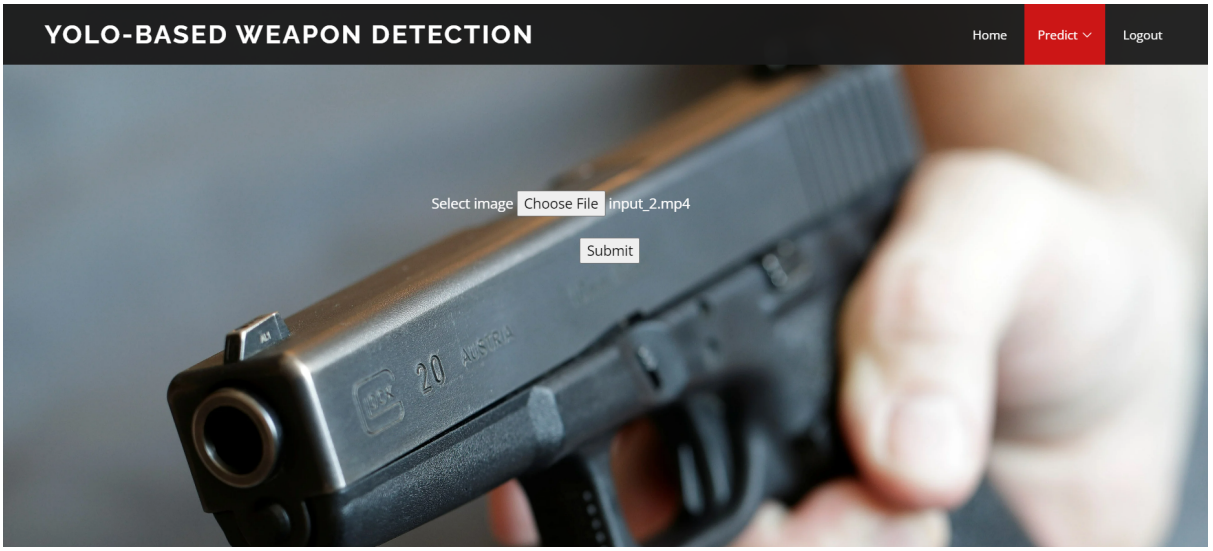
Output:



Output for Image as Input



Output for Video as Input



Live Input – Click on Go Live button, webcam will be triggered and weapon will be detected on live input.