YOLO Based Weapon Detection

Submitted by

Sai cheran TS

GUIDE:

1.Introduction:

1.1 Overview:

The YOLO (You Only Look Once) based weapon detection project is an innovative and cutting-edge computer vision application designed to automatically detect and identify weapons in images and videos in real-time. YOLO is a popular object detection algorithm known for its speed and accuracy, making it well-suited for real-time applications like weapon detection.

The primary objective of this project is to enhance public safety and security by enabling the automated identification of firearms and other dangerous weapons in various scenarios. It finds utility in diverse environments, including surveillance systems, airports, public events, schools, and other high-security areas where detecting weapons quickly and accurately is of paramount importance.

1.2 Purpose:

The use of YOLO-based weapon detection serves several important purposes in enhancing public safety and security, Preventing Armed Crimes, Enhancing Security at Public Events, Improving School Safety ,Strengthening Surveillance Systems, Assisting Law Enforcement Agencies ,Combatting, Minimizing Human Error, Saving Time and Resources, Deterring Criminal Activity

2.Literature Survey:

2.1.Existing problem:

While YOLO-based weapon detection is a powerful technology with numerous benefits, it also faces some challenges and existing problems that need to be addressed:

1. False Positives and False Negatives: YOLO-based models may occasionally produce false positives (incorrectly identifying non-weapons as weapons) or false negatives (failing to detect actual weapons). These errors can lead to unnecessary alerts or missed potential threats, impacting the system's reliability.

2. Variability in Weapon Types: Firearms and other weapons come in various shapes and sizes, and detecting them accurately requires a diverse training dataset. If the model is not trained on a wide range of weapon types, it may struggle to recognize less common or unconventional weapons.

3. Concealment Techniques: Criminals may try to conceal weapons using clothing or other objects, making detection more challenging. YOLO-based systems may have difficulty spotting hidden weapons, leading to potential security gaps.

2.2:Proposed Solution:

To address the existing problems in YOLO-based weapon detection, several solutions and strategies can be implemented:

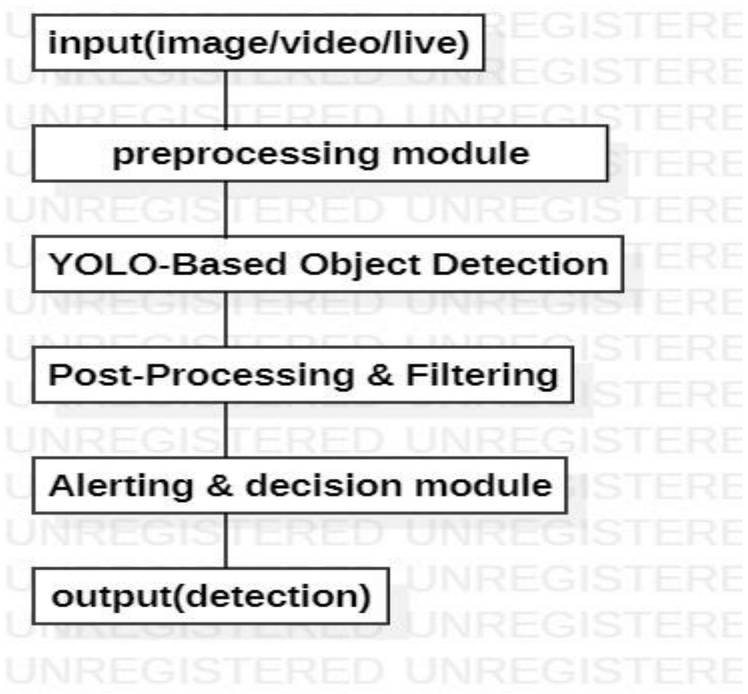
1.Improving Training Data: Creating a more diverse and comprehensive training dataset can help reduce false positives and improve detection accuracy. This dataset should include various weapon types, different angles, lighting conditions, and potential concealment scenarios.

2. Transfer Learning and Fine-tuning: Utilize transfer learning techniques to leverage pre-trained models on large datasets and then fine-tune them on the specific weapon detection task. This approach can help overcome data scarcity and expedite the model training process.

3. Incorporate Contextual Information: Enhance the weapon detection system by incorporating contextual information from the environment. For instance, integrating other surveillance technologies like facial recognition or behavioural analysis may aid in distinguishing potential threats from innocent behaviours.

3.Theoritical Analysis:

3.1:Block Diagram:


input(image/video/live)
preprocessing module
YOLO-Based Object Detection
Post-Processing & Filtering
Alerting & decision module
output(detection)

3.2.Hardware/Software Designing:

Flask version 2.0.2

Python version 3.9.13

Opencv version 4.8.0

Jupyter notebook

4.Experimental Investigation:

Experimental analysis for YOLO-based weapon detection typically involves the following steps:

1.Data Collection

2. Data Preprocessing

3. Model Selection

4. Model Training

5. Evaluation Metrics

6. Model Evaluation

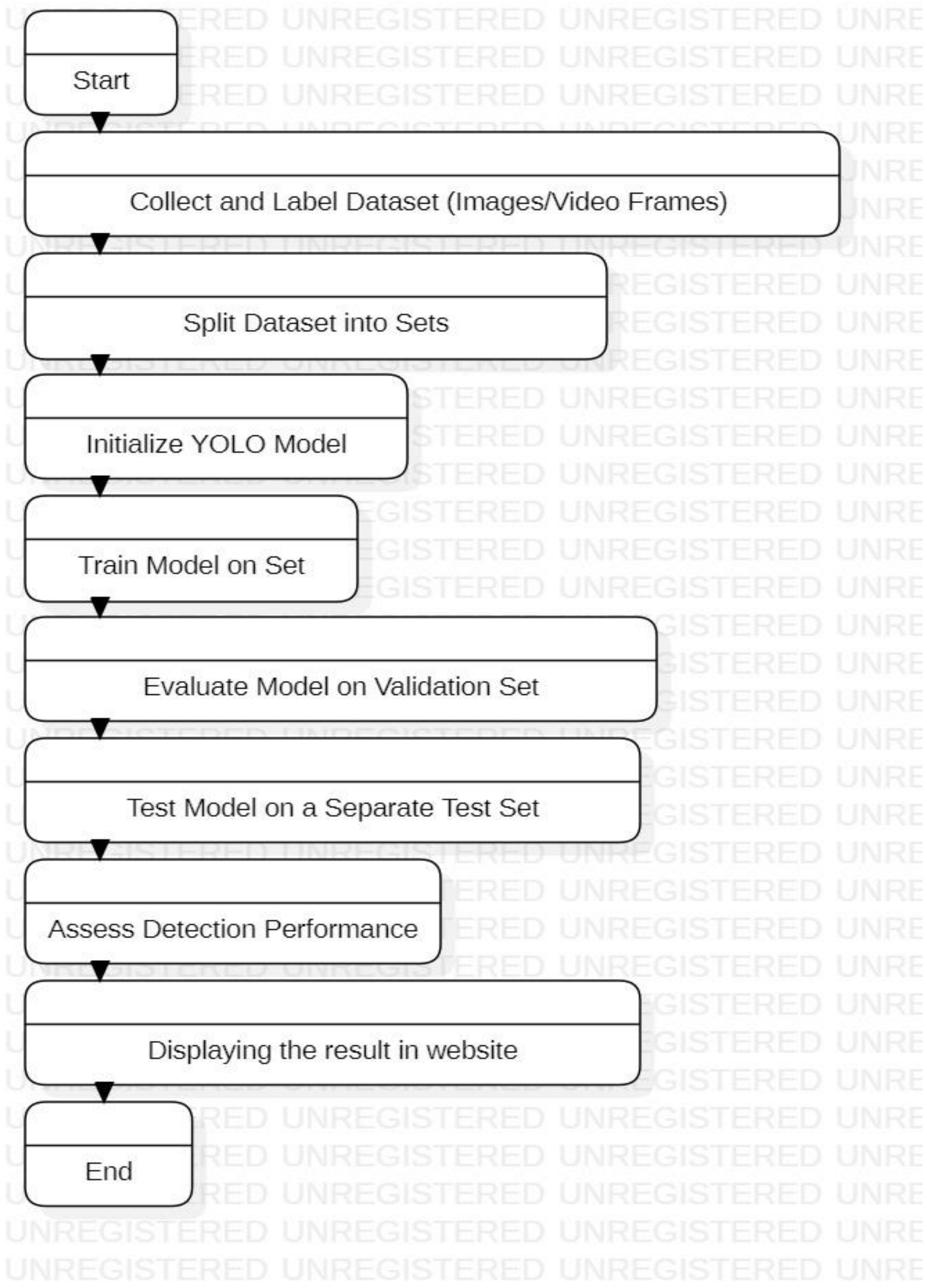
7. Test Set Evaluation

8. Comparison and Analysis

9. Discussion of Results

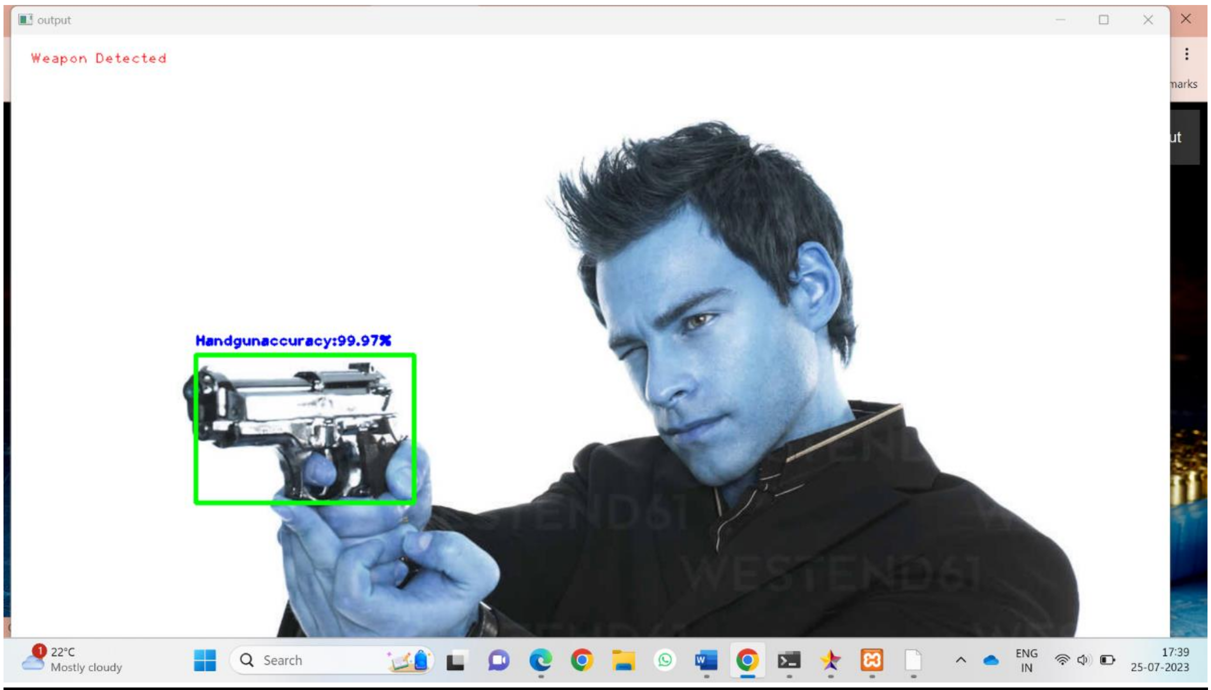
10. Ethical Considerations

5.Flowchart:

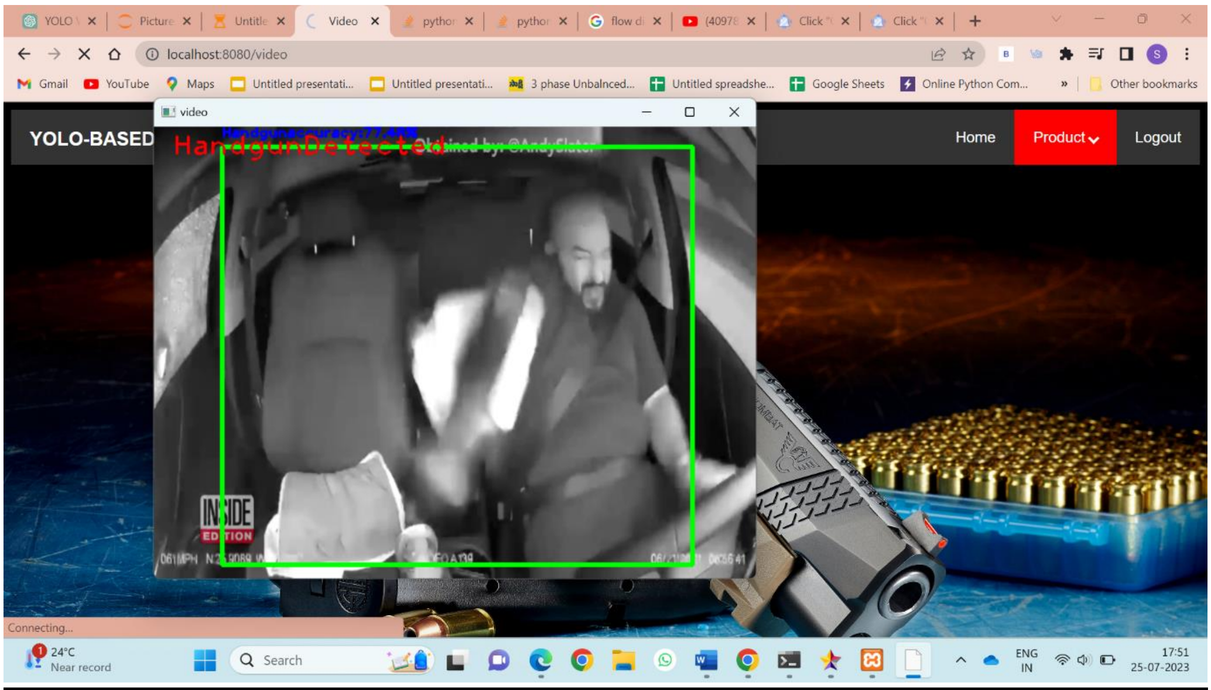

Start
Collect and Label Dataset (Images/Video Frames)
Split Dataset into Sets
Initialize YOLO Model
Train Model on Set
Evaluate Model on Validation Set
Test Model on a Separate Test Set
Assess Detection Performance
Displaying the result in website
End

6.Result:

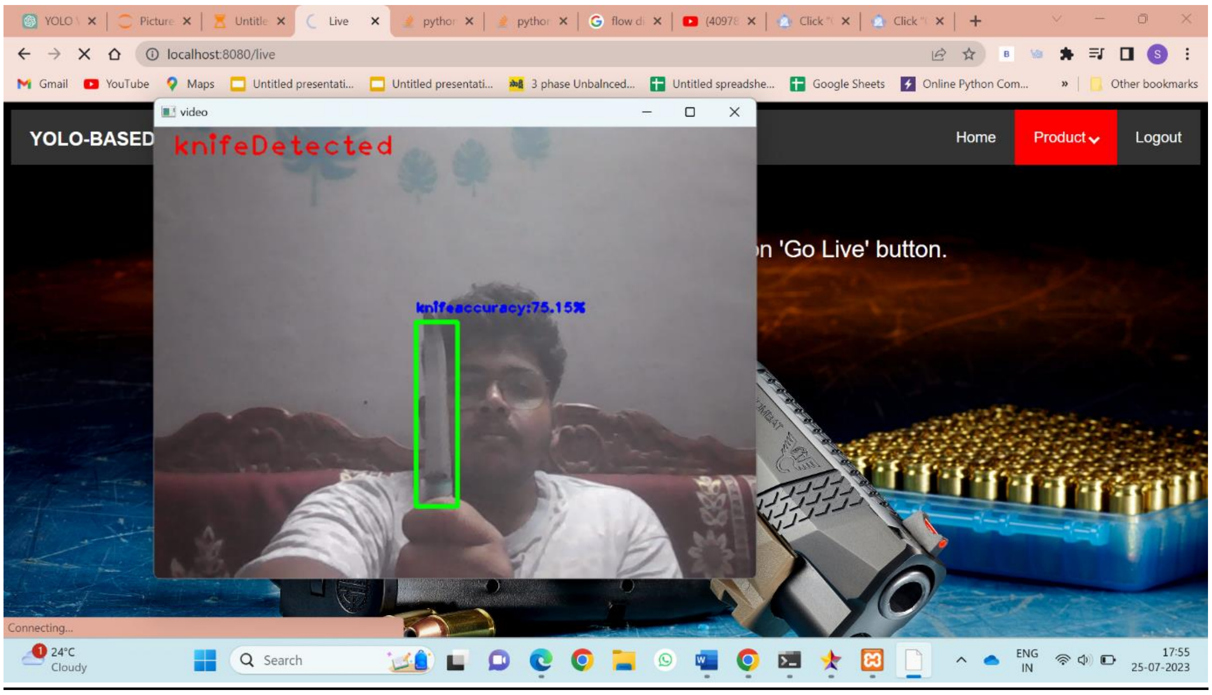
Image as input:



Video as input:



Live video:



7.Advantages and Disadvantages:

Advantages:

1. Real-Time Detection: YOLO is known for its speed and efficiency. It can process images or video frames in real-time, making it suitable for applications that require quick responses, such as video surveillance or security systems.

2. End-to-End Approach: YOLO performs object detection and classification in a single pass, providing both bounding boxes and class probabilities. This end-to-end approach simplifies the pipeline and makes it easier to implement and deploy.

3. High Accuracy: With the advancements in architecture (e.g., YOLOv4) and the availability of large-scale datasets for pre-training

(e.g., COCO), YOLO-based models can achieve high accuracy in detecting weapons.

4. Generalization: YOLO can generalize well to different object categories, including weapons, due to its ability to learn from diverse datasets.

Disadvantages:

1. Detection of Small Objects: YOLO may struggle with detecting very small objects, including small weapons, due to the down-sampling process in its architecture.

2. Localization Accuracy: While YOLO can provide bounding boxes for detected objects, the localization accuracy might not be as precise as some other algorithms, especially when objects are close together or partially occluded.

3. Training Data and Bias: The performance of YOLO heavily depends on the quality and diversity of the training data. Biases present in the training data can be learned by the model, leading to biased predictions.

4. Limited Context: YOLO processes images independently and may not effectively utilize contextual information when detecting objects, which could affect detection accuracy in complex scenes.

8.Applications:

YOLO (You Only Look Once) based weapon detection has several applications in various domains, particularly in enhancing security and safety measures. Some of the key applications include:

1. Public Safety and Security: Deploying YOLO-based weapon detection systems in public spaces such as airports, train stations,

shopping malls, and stadiums can enhance security by identifying and alerting security personnel to the presence of weapons in real-time.

2. Law Enforcement: Law enforcement agencies can use YOLO-based weapon detection in body-worn cameras or surveillance systems to assist officers in identifying potentially dangerous situations and to improve officer safety during patrols and operations.

3. School Safety: To prevent violent incidents in schools, YOLO-based weapon detection can be integrated with security cameras or access control systems to monitor and detect weapons on school premises.

4. Border Security: YOLO-based weapon detection can be employed at border crossings, checkpoints, and customs to detect illegal firearms and weapons, helping prevent smuggling and ensuring national security.

5. Prison Security: In correctional facilities, YOLO-based weapon detection can be used to monitor common areas, visitor checkpoints, and inmate movements, reducing the risk of weapon possession and violence inside the prison.

6. Private Security and VIP Protection: YOLO-based weapon detection can be implemented for private security firms or VIP protection services to enhance security measures during events, gatherings, or personal protection.

7. Surveillance Drones: Drones equipped with YOLO-based weapon detection can be used for remote monitoring of large areas, such as borders or critical infrastructure, to identify potential threats.

8. Automated Screening Systems: YOLO-based weapon detection can be integrated into security screening systems at airports, government buildings, or private establishments, automating the process of identifying dangerous objects in bags or belongings.

9.Conclusion:

YOLO (You Only Look Once) based weapon detection is a powerful and efficient technology that holds great potential in enhancing security and safety measures across various domains. Its real-time capabilities, high accuracy, and ability to detect multiple objects in a single pass make it a popular choice for object detection applications, including weapon detection. To ensure responsible and ethical use of YOLO-based weapon detection, it is crucial to address privacy concerns, implement strict data protection measures, and be mindful of potential biases in the training data. Furthermore, public awareness and transparency regarding the deployment of such technologies are essential to maintaining public trust. YOLO-based weapon detection holds great promise in bolstering security efforts and enhancing public safety, but its deployment should be done responsibly, ethically, and in a manner that respects individual rights and societal values. By striking a balance between the benefits and challenges, YOLO-based weapon detection can contribute to creating safer environments in our communities.

10.Future Scope:

1. Improved Accuracy: Researchers and developers will continue to work on improving the accuracy and robustness of YOLO-based models for weapon detection. This could involve fine-tuning architectures, exploring novel loss functions, and incorporating more diverse and specific training datasets.

2. Better Small Object Detection: Efforts will be made to enhance the detection of small and occluded weapons, as well as to handle complex scenarios where weapons might be partially hidden or disguised.

3. Real-Time Performance on Edge Devices: As computing power in edge devices improves, there will be a focus on optimizing YOLO

models for real-time performance on resource-constrained devices like surveillance cameras, drones, and embedded systems.

4. Multi-modal Integration: YOLO-based weapon detection could be integrated with other sensors, such as thermal cameras or acoustic sensors, to create multi-modal systems for improved detection accuracy and redundancy.

5. Transfer Learning and Domain Adaptation: Techniques for transfer learning and domain adaptation will be explored to make YOLO-based weapon detection more effective in diverse real-world scenarios and to reduce the need for extensive custom datasets.

6. Explainable AI: Efforts will be made to make YOLO-based weapon detection more interpretable and explainable, allowing users to understand how the model arrives at its predictions and building trust in its decisions.

7. Hardware Acceleration: With the rise of specialized hardware for deep learning tasks, there will be a focus on optimizing YOLO-based models for various accelerators, such as GPUs, TPUs, and dedicated neural processing units (NPUs).

8. Privacy-Preserving Techniques: As privacy concerns remain a critical consideration, research will be directed toward privacy-preserving approaches, such as federated learning or differential privacy, to ensure that sensitive data is protected during the model training process.

10.Bibilography:

1. YOLO: Real-Time Object Detection: The original YOLO paper by Joseph Redmon et al. introduced the YOLO algorithm and its real-time object detection capabilities.

Paper: https://arxiv.org/abs/1506.02640

2. YOLO9000: Better, Faster, Stronger: This follow-up paper extended YOLO to handle a large number of object categories and introduced the concept of YOLO9000.

Paper: https://arxiv.org/abs/1612.08242

3. YOLOv3: An Incremental Improvement: YOLOv3 is an improved version of YOLO that introduced various architectural changes and improved performance.

Paper: https://arxiv.org/abs/1804.02767

4. YOLOv4: Optimal Speed and Accuracy of Object Detection: YOLOv4 is an advanced version that achieved state-of-the-art performance in object detection tasks.

Paper: https://arxiv.org/abs/2004.10934

5. Darknet: Darknet is the open-source neural network framework used to train YOLO models. The official Darknet repository contains implementation details and pre-trained models.

Repository: https://github.com/pjreddie/darknet

6. COCO Dataset: The COCO (Common Objects in Context) dataset is widely used for training and evaluating YOLO models, including those used in weapon detection tasks.

Website: https://cocodataset.org/

7. Deep Learning: Although not specifically focused on YOLO, deep learning books cover the fundamental concepts, architectures, and techniques used in YOLO and similar algorithms.

- "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

- "Neural Networks and Deep Learning: A Textbook" by Charu C. Aggarwal

Appendix:

Source code:

```
import re
from flask import Flask,render_template,request,session
import cv2
import os
from playsound import playsound
import mysql.connector
import re
app=Flask(__name__)
conn = mysql.connector.connect(user='sai', password='Saicheran',
                        host='localhost',
                        database='proj')
mycursor=conn.cursor()
print("connected")
weight=r"../project/yolov4-custom_last.weights"
config=r"../project/yolov4-custom.cfg"
classes=["Handgun","knife"]
net=cv2.dnn.readNet(weight,config)
model=cv2.dnn_DetectionModel(net)
model.setInputParams(scale=1/255,size=(416,416))
app.secret_key = 'super secret key'
@app.route('/')
def project():
    return render_template('main.html')
@app.route('/home')
def home():
    return render_template('main.html')
@app.route('/About')
def home1():
    return render_template('about.html')
@app.route('/login')
def login1():
    return render_template('login.html')
@app.route('/login',methods=['POST'])
def login():
    msg1=''
    if request.method=='POST':
        email=request.form["email"]
        password=request.form["password"]
        value1=[]
        value1.append(email)
        value1.append(password)
        value1tup=tuple(value1)
        sql2="SELECT * FROM new WHERE email=%s and password=%s"
        mycursor.execute(sql2,value1tup)
        account=mycursor.fetchone()
        if account is not None:
            session['Loggedin']=True
            session['id']=account[1]
            session['email']=account[1]
            return render_template('demo.html')
        else:
            msg1="Incorrect Email/password"
            return render_template('login.html',msg1=msg1)

    return render_template('login.html')
@app.route('/demo', methods=['POST'])
def demo():
    if request.method == 'POST':
        return render_template('demo.html')
@app.route('/register')
def signup1():
    return render_template('register.html')
@app.route('/contact')
def con():
    return render_template('contact.html')
@app.route("/register" ,methods=['POST'])
def signup():
    msg=''
    if request.method=='POST':
        name=request.form["name"]
        email=request.form["email"]
        password=request.form["password"]
        sql = "SELECT * FROM new WHERE name=%s"
        mycursor.execute(sql,(name,))
        acc=mycursor.fetchone()
        if acc is not None:
            return render_template("login.html")
        else:
            value=[]
            value.append(name)
            value.append(email)
            value.append(password)
            valuetup=tuple(value)
            sql1 = "INSERT INTO new(name,email,password) VALUES(%s,%s,%s)"
          mycursor.execute(sql1,valuetup)
            conn.commit()
            msg="You have successfully registered!"
    return render_template('login.html',msg=msg)
@app.route('/image')
def img1():
    return render_template('image.html')
```

```
@app.route('/image',methods=["POST"])
def img():
    if request.method=='POST':
        f=request.files['file']
        basepath=os.path.dirname(os.path.abspath('application.py'))
        filepath=os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)
        img=cv2.imread(filepath)
        img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
     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),3)
              cv2.putText(img,(classes[classID]+'accuracy:'+(str(round((scores*100),2)))+'%'),(x,y-10),
cv2.FONT_HERSHEY_PLAIN,1,(255,0,0),2)
                cv2.putText(img,'Weapon Detected',(20,30),cv2.FONT_HERSHEY_PLAIN,1,(0,0,255),1)
        cv2.imshow('output',img)
        cv2.waitKey(0)
    cv2.destroyAllWindows()
    return render_template("image.html")
@app.route('/video')
def vid1():
     return render_template("video.html")
@app.route('/video',methods=['POST'])
def vid():
    if request.method=='POST':
        f=request.files['file']
        basepath=os.path.dirname(os.path.abspath('application.py'))
        filepath=os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)
        cap=cv2.VideoCapture(filepath)
        weapon=False
        while True:
            _, frame=cap.read()
            frame=cv2.resize(frame,(640,480))
            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>=0.7:
                weapon=True

            if weapon==True:
                cv2.imshow('video',frame)
            if cv2.waitKey(1) & 0xFF==ord('q'):
                break
    cv2.destroyAllWindows()
    return render_template("video.html")
@app.route('/live')
```

```
def live1():
    return render_template("live.html")
@app.route('/live',methods=['POST'])
def live():
    if request.method=='POST':
        capp=cv2.VideoCapture(0)
        weapon=False
        while True:
            _, frame=capp.read()
            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),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)
            cv2.imshow('video',frame)
            if cv2.waitKey(1) & 0xFF==ord('q'):
                break
    capp.release()
    cv2.destroyAllWindows()
    return render_template("live.html")
     if __name__ == '__main__':
    from werkzeug.serving import run_simple
    run_simple('localhost', 8080, app)
```