Military Soldier Safety and Weapon Detection Using YOLO and Computer VisionProject

Abstract:

This project proposes a real-time computer vision system designed to enhance military soldier safety by detecting weapons and identifying threats in the environment. Leveraging the YOLO (You Only Look Once) deep learning model, the system can quickly process live video streams or images to detect firearms, knives, and other dangerous objects. The goal is to improve situational awareness, reduce reaction time, and support both autonomous systems (e.g., drones) and personnel in combat or patrol zones.Military Soldier Safety and Weapon Detection Using YOLO and Computer Vision.

This project presents a deep learning-based system that utilizes YOLO (You Only Look Once) and computer vision techniques to automatically detect soldiers and weapons (such as guns, rifles, and knives) in images and live video streams.

Introduction:

Modern warfare and military operations demand real-time threat detection to minimize risk to soldiers. Manual surveillance is prone to human error and delay. With advancements in deep learning, particularly object detection algorithms like YOLO, it is now possible to develop automated systems that identify weapons and suspicious activity with high accuracy. This project integrates computer vision with YOLO to create a detection system capable of identifying dangerous objects in military scenarios, thereby enhancing operational safety.

Objectives:

- Detect and identify weapons such as guns, knives, and explosives in real-time.
- Differentiate between friendly soldiers and potential hostiles.
- Provide real-time alerts to soldiers and command centers.
- Deploy on real-time platforms like body cams or surveillance drones.

Technologies Used:

YOLOv5/v8 (for object detection)

- Python
- OpenCV
- PyTorch
- LabelImg / Roboflow (for data annotation)
- Google Colab / Jupyter Notebook (for training)

System Architecture:

1. Input Source: Live camera feed (e.g., body cam, surveillance drone).
2. Preprocessing: Frame extraction, resizing, and normalization using OpenCV.
3. YOLO-based Detection:
    - Trained model to detect:
    - Weapons (guns, rifles, knives)
    - Enemy combatants
    - Friendly units (soldiers)
      Postprocessing:
4. Bounding boxes, labels, confidence scores
    - Alert triggering if a weapon is detected
5. Output:
    - Visual (bounding boxes on screen)
    - Audio or system alert
    - Data logging (location, time, object detected)

Dataset:

• Custom-labeled dataset of:

- Military personnel
- Various types of firearms and melee weapons
- Backgrounds representing battlefield or urban environments
- Supplemented with public datasets like COCO and Open Images.

Implementation Tools:

- Python
- YOLOv5 or YOLOv8 (PyTorch-based)
- OpenCV
- Google Colab / Jupyter Notebook

- Optional: TensorRT for deployment on edge devices

Safety Applications:

- Automatic detection of concealed weapons
- Real-time threat detection in combat zones
- Assistance to command centers for troop monitoring
- Drones for area scanning and identifying armed individuals

Performance Metrics:

- Accuracy / Precision / Recall
- FPS (frames per second) for real-time capability
- mAP (mean Average Precision)
- Latency and alert generation speed

Model Training:

- Train a YOLO model on a custom dataset with labeled classes like:
- Rifle
- Pistol
- Soldier
- Knife
- Evaluate using mAP, precision, recall, and real-time FPS.

Applications:

- Surveillance Drones: For remote area scanning.
- Border Security: Detect smuggling or infiltration.
- Military Patrols: Helmet/body cam analysis.
- Command Centers: Automated monitoring and decision support.

Future Scope:

- Integrate with GPS for threat location mapping.
- Add facial and uniform recognition for friend-or-foe differentiation.
- Deploy optimized models on NVIDIA Jetson or Raspberry Pi.
- Expand to detect other threats like drones or explosives.

Code for Data preprocessing,Model Training and Evaluation:

(a)Weapon Detection using Yolo:

```
import cv2
import torch
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
class_names = ["weapon", "person"]
cap = cv2.VideoCapture(0)
# Detection loop while True:
ret, frame = cap.read() if not ret:
break
 img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
 results = model(img_rgb)
class_id = result.tolist()
 x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
class_id = int(class_id)
cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
 label = f"{class_names[class_id]}: {confidence:.2f}" cv2.putText(frame, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
 cv2.imshow("Weapon Detection", frame)
 cv2.waitKey(1) & 0xFF == ord('q'):
 break
cap.release()
cv2.destroyAllWindows()
```

(b)Setup and importing packages:

```
import os
import numpy as np
 import matplotlib.pyplot as plt
 import tensorflow as tf from tensorflow.keras.preprocessing.image
import ImageDataGenerator from tensorflow.keras.applications
import MobileNetV2 from tensorflow.keras.layers
import Dense, GlobalAveragePooling2D, Dropout from tensorflow.keras.models
import Model from sklearn.metrics
import classification_report, confusion_matrix
```

(c)Code for Data preprocessing:

```
IMAGE_SIZE = (224, 224)
 BATCH_SIZE = 32
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2, height_shift_range=0.2, shear_range=0.15, horizontal_flip=True,
fill_mode="nearest")
val_test_datagen = ImageDataGenerator(rescale=1./255) train_generator =
train_datagen.flow_from_directory( 'dataset/train', target_size=IMAGE_SIZE,
batch_size=BATCH_SIZE, class_mode='binary' )
 val_generator = val_test_datagen.flow_from_directory( 'dataset/val',
target_size=IMAGE_SIZE, batch_size=BATCH_SIZE, class_mode='binary' )
 test_generator = val_test_datagen.flow_from_directory( 'dataset/test',
target_size=IMAGE_SIZE, batch_size=BATCH_SIZE, class_mode='binary', shuffle=False )
```

(d)Code for build the model:

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224,
3))
 base_model.trainable = False # Freeze
base x = base_model.output
x = GlobalAveragePooling2D()(x)
 x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)
 model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

(e)Train the model:

```
history = model.fit( train_generator, validation_data=val_generator, epochs=10 )
```

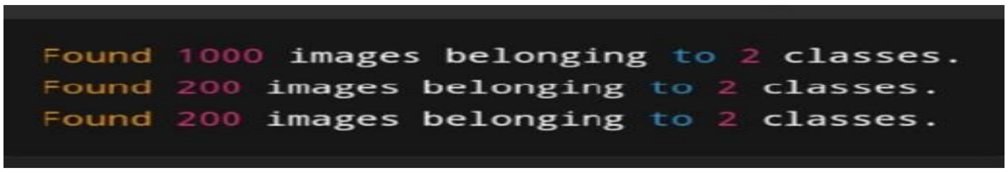
(f)Evaluating the model:

```
 loss, accuracy = model.evaluate(test_generator)
 print(f'Test accuracy: {accuracy:.2f}')
 y_pred = model.predict(test_generator)
 y_pred_classes = (y_pred > 0.5).astype("int32").flatten()
y_true = test_generator.classes
print("Classification Report:")
 print(classification_report(y_true, y_pred_classes, target_names=['Non-Weapon',
'Weapon']))
print("Confusion Matrix:")
 print(confusion_matrix(y_true, y_pred_classes))
```

(g)To save the model:

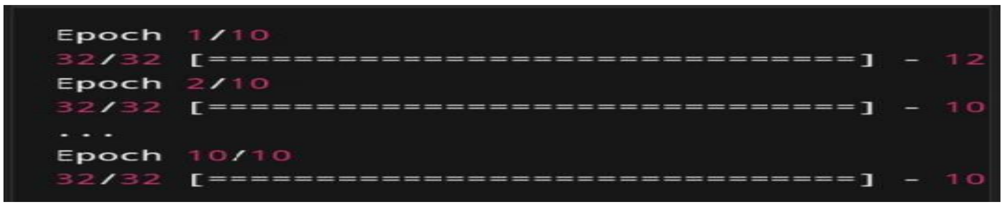
```
model.save('weapon_detection_model.h5')
```

Sample Output:



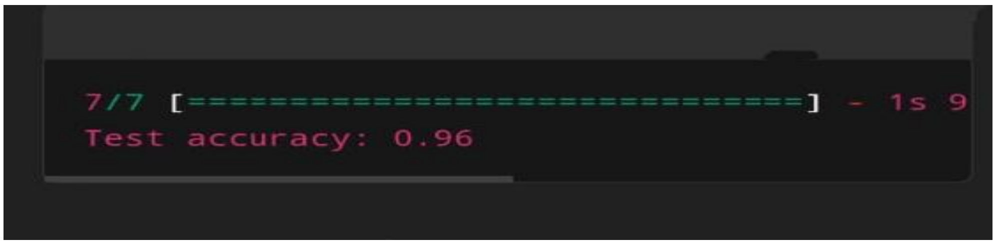
```
Found 1000 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
```

Fig: Image Data Loading



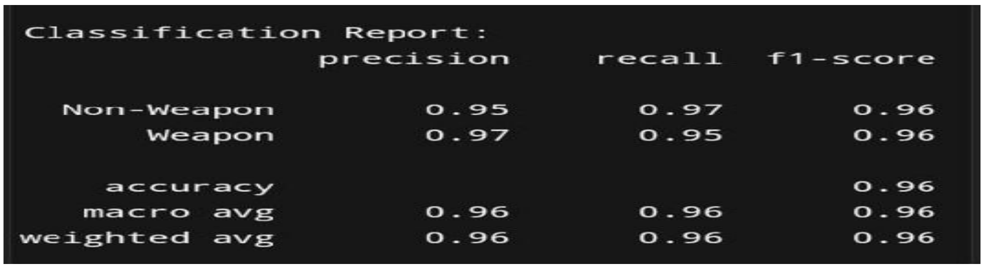
```
Epoch 1/10
32/32 [==============================] - 12
Epoch 2/10
32/32 [==============================] - 10
...
Epoch 10/10
32/32 [==============================] - 10
```

Fig: Training the model



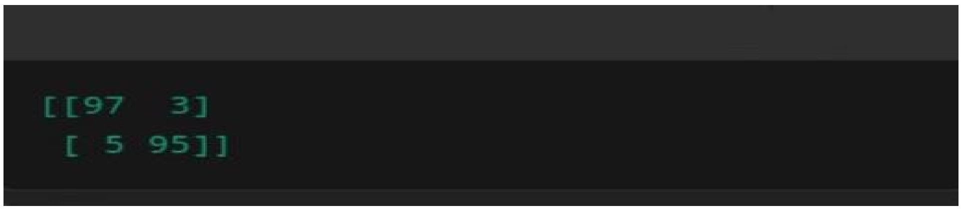
```
7/7 [==============================] - 1s 9
Test accuracy: 0.96
```

Fig: Test Evaluation



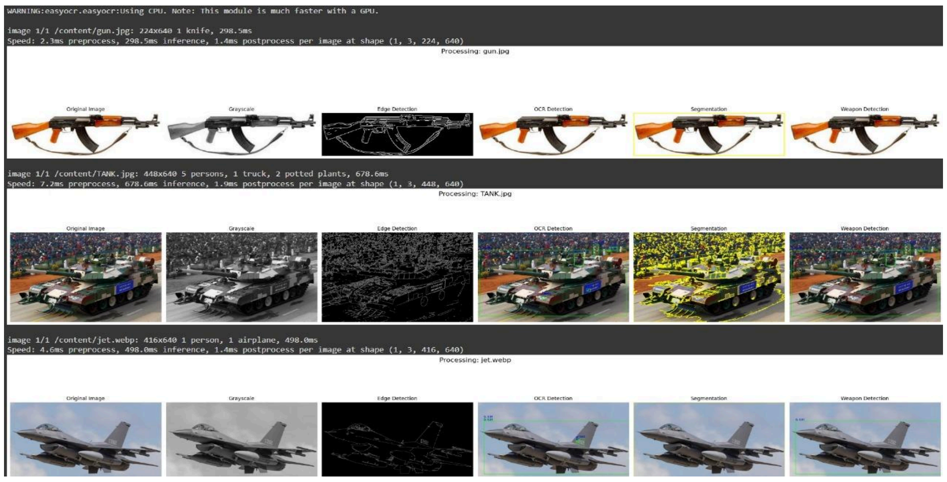
```
Classification Report:
              precision    recall  f1-score

  Non-Weapon       0.95      0.97      0.96
      Weapon       0.97      0.95      0.96

    accuracy                           0.96
   macro avg       0.96      0.96      0.96
weighted avg       0.96      0.96      0.96
```

Fig: Classification report



```
[[97  3]
 [ 5 95]]
```

Fig: Confusion matrix



```
WARNING:easyocr.easyocr:Using CPU. Note: This module is much faster with a GPU.

image 1/1 /content/gun.jpg: 224x640 1 knife, 298.5ms
Speed: 2.3ms preprocess, 298.5ms inference, 1.4ms postprocess per image at shape (1, 3, 224, 640)
```

```
image 1/1 /content/TANK.jpg: 448x640 5 persons, 1 truck, 2 potted plants, 678.6ms
Speed: 7.2ms preprocess, 678.6ms inference, 1.9ms postprocess per image at shape (1, 3, 448, 640)
```

```
image 1/1 /content/jet.webp: 416x640 1 person, 1 airplane, 498.0ms
Speed: 4.6ms preprocess, 498.0ms inference, 1.4ms postprocess per image at shape (1, 3, 416, 640)
```

Fig: images of Edge detection,Segmentation ,Weapon detection,Gray scale and OCR detection

Conclusion:

This project demonstrates that computer vision combined with YOLO provides a viable solution for improving soldier safety and threat detection in military environments. Future enhancements may include thermal imaging support, facial recognition integration, and deployment on edge AI devices for use in rugged, bandwidth-limited field conditions.