Military Soldier Safety and Weapon Detection using YOLO and Computer Vision

Aim:

- This project aims to address these challenges by leveraging computer vision and YOLO (You Only Look Once), a state-of-the-art object detection algorithm, to automate the process of detecting and classifying objects in real-time.
- The system will be trained on a dataset containing images of military and civilian scenarios, with annotations for objects.

Problem Statement:

- In modern military operations, ensuring the safety of soldiers and maintaining situational awareness in conflict zones is of paramount importance.
- One of the key challenges is the ability to quickly and accurately detect potential threats, such as weapons, enemy combatants, and unauthorized vehicles, while also distinguishing between friendly forces, civilians, and non-threatening entities.

Goal:

1.Detect Threats in Real-Time:

Identify weapons, enemy soldiers, and unauthorized vehicles that pose a threat to military personnel.Provide real-time alerts to soldiers or command centers, enabling quick response to potential dangers.

2.Enhance Situational Awareness:

Provide a comprehensive view of the battlefield by detecting and tracking objects such as military vehicles, trenches, and soldiers.

3.Operate in Diverse Environments:

Function effectively in various environments, including urban areas, forests, and deserts, where lighting conditions, occlusions, and background clutter may vary.

4.Improve Soldier Safety:

Reduce the risk of ambushes or surprise attacks by detecting hidden threats (e.g., camouflaged soldiers, concealed weapons).

Use Cases:

1.Military Surveillance and Threat Detection:

Monitor conflict zones to detect weapons, enemy soldiers, and unauthorized vehicles in real-time.

2. Soldier Safety and Alert Systems:

Provide real-time alerts to soldiers about nearby threats, such as weapons or enemy combatants.

3. Border Security and Intrusion Detection:

Identify unauthorized crossings of military or civilian vehicles at border checkpoints.

4. Disaster Response and Rescue Operations:

Distinguish between military personnel, civilians, and vehicles during disaster relief operations.

5. Training and Simulation:

Use the system in virtual training environments to simulate real-world scenarios for soldiers.

6. Combat Zone Analysis:

Analyze combat zones to identify the presence of trenches, military vehicles, and other strategic objects.

Approach:

1. Data Collection and Preparation:
   - Gather a dataset containing images of military and civilian scenarios with YOLO annotations.
   - Preprocess the dataset to ensure compatibility with the YOLO model.
2. Model Training:
   - Train a YOLO model to detect and classify multiple objects, including soldiers, weapons, vehicles, and trenches.
3. Real-Time Detection:
   - Deploy the trained model to detect objects in real-time video feeds or images.
4. Threat Classification:
   - Classify detected objects as threats (e.g., weapons, enemy soldiers) or non-threats (e.g., civilians, friendly soldiers).
5. Streamlit Integration:
   - Develop a user-friendly web interface for uploading images/videos and visualizing detection results.
6. Performance Evaluation:
   - Evaluate the model's performance using metrics like precision, recall, and mean average precision (mAP).

Exploratory Data Analysis (EDA):

1. Image Analysis:

a. Train-Test-Validation Distribution

Objective:

- Analyze the distribution of image dimensions (width and height) and resolutions.

Explanation:

- Check if all images have consistent dimensions or if they vary significantly.
- Identify outliers (e.g., extremely large or small images) that may affect model training.
- Resize images to a standard resolution (e.g., 640x640) to ensure uniformity during training.

b. Aspect Ratio

Objective:

- Examine the aspect ratio (width/height) of images.

Explanation:

- Determine if the dataset contains images with varying aspect ratios (e.g., square, rectangular).

- Decide whether to pad or crop images to maintain a consistent aspect ratio for YOLO training.

c. Image Quality

Objective:

- Assess the quality of images in the dataset.

Explanation:

- Check for blurry, overexposed, or underexposed images that may hinder object detection.
- Identify images with noise or artifacts that could affect model performance.

2. Annotation Analysis

a. Number of Annotations per Image

Objective:

- Analyze the distribution of annotations (bounding boxes) per image.

Explanation:

- Determine if some images have too few or too many annotations.
- Identify images with no annotations (if any) and decide whether to exclude them from training.

3.Dataset Splits

Objective:

- Analyze the distribution of images and annotations across train, test, and validation sets.

Explanation:

- Ensure that each split has a representative distribution of classes and object sizes.
- Check for data leakage (e.g., similar images in both train and test sets).

4.Data Augmentation Analysis

a. Augmentation Impact

Objective:

- Evaluate the impact of data augmentation techniques on the dataset.

Explanation:

- Test augmentations like rotation, flipping, scaling, and cropping to see how they affect object visibility and annotation accuracy.
- Ensure that augmentations do not distort small objects like weapons.

5. Challenges and Insights

a. Small Object Detection

Objective:

- Identify challenges in detecting small objects (e.g., weapons).

Explanation:

- Analyze the proportion of small bounding boxes in the dataset.

- Consider techniques like higher-resolution input or specialized anchor boxes to improve detection.

b. Class Imbalance

Objective:

- Address class imbalance in the dataset.

Explanation:

- If certain classes (e.g., trenches) are underrepresented, use techniques like oversampling or synthetic data generation.

c. Annotation Quality

Objective: Ensure high-quality annotations.

Explanation:

- Identify and correct mislabeled or missing annotations.
- Use tools like LabelImg or CVAT for annotation review and correction.a. Image Size and Resolution

Coding:

a)Setup and importing packages

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf from tensorflow.keras.preprocessing.image
importImageDataGeneratorfrom 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
```

b) Code for processing

```
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
)
```

c) 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'])
```

d) Train the model

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

e) Evaluating the model

```
loss, accuracy = model.evaluate(test_generator)
print(f'Test accuracy: {accuracy:.2f}')
# Predictions and confusion matrix
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))
```

f)To save the model

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

Sample Output:

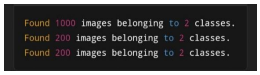


Fig: Image Data Loading

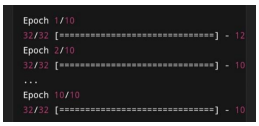


Fig: Training the model

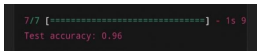
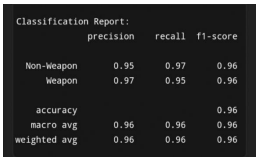


Fig: Test Evaluation



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

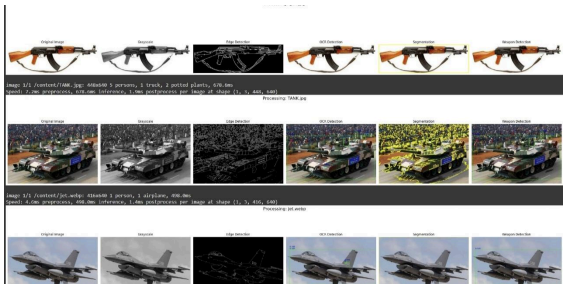


Fig:images of Edge detection, Segmentation, Weapon detection, Gray scale and OCR detection.

Result:

Thus the Military Soldier Safety and Weapon Detection using YOLO and Computer Vision output was verified successfully.