

HUMAN DETECTION USING YOLOv8

Perna Choudhary

INDEX

SNO	Section Title	Page No
1	Introduction	5
2	Objective	6
3	Technologies Used	7
4	Human Detection	9
5	YOLO Architecture and Working	11
6	Image Enhancement Techniques	14
7	Background Subtraction	22
8	Gaussian Mixture Model	24
9	HOG [Histogram of Gradient]	26
10	Principal Component Analysis	28
11	Result and Analysis	30
12	Challenges Faced	33
13	Conclusion and Future Scope	34
14	References	35
15	Annexures	36

INTRODUCTION

In recent years, the demand for real-time human detection in visual data has surged across various industries including surveillance, autonomous systems, smart cities, and human-computer interaction. With the evolution of deep learning and computer vision, object detection frameworks have significantly advanced, enabling machines to perceive and understand visual scenes with remarkable accuracy and speed.

This project, titled "**Human Detection using YOLOv8 and Computer Vision Techniques**", focuses on the implementation of **YOLOv8**—a cutting-edge object detection model—to detect humans in both images and videos. Alongside YOLOv8, the project integrates classical and modern techniques such as **Gaussian Mixture Models (GMM)** for motion segmentation, **Principal Component Analysis (PCA)** for dimensionality reduction, **Histogram of Oriented Gradients (HOG)** for contour-based human detection, and **background subtraction** using MOG2.

The aim is to build a hybrid detection system that leverages the accuracy of modern deep learning models while enhancing interpretability and feature extraction using traditional methods. The output includes bounding boxes, motion analysis, and visualizations of extracted features, giving deeper insights into how humans are identified and analyzed in video frames.

The project was conducted during summer training at **DRDO, Dehradun**, under the guidance of **Dr. Vaibhav Gupta**, and provides a practical application of AI and computer vision concepts taught in the undergraduate curriculum.

OBJECTIVE

The primary objective of this summer training project is to develop an efficient and accurate system for **real-time human detection** using **YOLOv8** combined with classical computer vision techniques. This hybrid approach aims to explore the synergy between deep learning-based object detection and traditional image processing methods.

The specific goals of the project are as follows:

- To understand and implement **YOLOv8**, a modern real-time object detection algorithm, for identifying humans in images and videos.
- To utilize **Gaussian Mixture Models (GMM)** for background modeling and motion-based human segmentation.
- To apply **Principal Component Analysis (PCA)** for dimensionality reduction and visualization of frame-based features.
- To integrate **Histogram of Oriented Gradients (HOG)** for shape-based human detection using gradient descriptors.
- To explore **background subtraction** methods (MOG2) to isolate moving entities from static backgrounds.
- To analyze **human movement patterns** over time using PCA for time series frames.
- To reinforce understanding of **Artificial Neural Networks (ANN)** architecture and relate it to the working of YOLO.
- To evaluate the accuracy and performance of the integrated system on real-world videos with varying complexity.
- To produce meaningful visual outputs and a structured report summarizing the implementation, challenges, and learnings.

This project not only enhances the understanding of object detection systems but also lays the groundwork for advanced applications in surveillance, safety, and human activity recognition.

TECHNOLOGIES USED

This project utilized a combination of modern object detection frameworks and traditional computer vision techniques. The implementation involved the following technologies and libraries:

1. Python 3

Python was the primary programming language used due to its extensive libraries for AI, machine learning, and image processing.

2. YOLOv8 (Ultralytics)

YOLOv8 (You Only Look Once, version 8) is a real-time object detection model from Ultralytics. It was used for detecting humans in video frames.

- **Library Used:** ultralytics
 - **Model Used:** yolov8n.pt (nano version for speed and lightweight inference)
 - **Functionality:** Draws bounding boxes around detected humans using confidence thresholds and class labels.
-

3. OpenCV (cv2)

OpenCV was a key library in the project for:

- Reading and processing video frames
 - Applying background subtraction (MOG2)
 - Drawing bounding boxes
 - Image manipulation and visualization
-

4. Scikit-learn (sklearn)

Scikit-learn was used for:

- **PCA (Principal Component Analysis):** Reducing dimensionality of image features and analyzing patterns across video frames
 - **GMM (Gaussian Mixture Model):** Unsupervised clustering to identify movement regions based on PCA features
-

5. scikit-image

This library was used to extract **HOG (Histogram of Oriented Gradients)** features from grayscale video frames for contour-based human detection.

6. Matplotlib

Used for plotting:

- PCA scatter plots
 - GMM clustering visualizations
 - Saving output graphs for report analysis
-

7. Google Colab / Jupyter Notebook (Development Environment)

Google Colab was used as the coding platform due to its GPU support, easy library installation, and notebook-style debugging.

8. Image Outputs

All result images (YOLO detections, PCA plots, background subtraction, HOG features, GMM clusters) were saved as PNG or JPG using OpenCV and Matplotlib.

This stack enabled the creation of a hybrid, modular, and explainable system for real-time human detection and visual data analysis.

HUMAN DETECTION

Human detection refers to the process of identifying the presence and location of humans in digital images or video streams. It is a foundational task in computer vision with wide applications in security surveillance, autonomous systems, healthcare monitoring, and human-computer interaction.

Importance of Human Detection

- **Surveillance Systems:** Automatically identify intruders or suspicious movements.
 - **Autonomous Vehicles:** Detect pedestrians to avoid collisions.
 - **Smart Environments:** Monitor human activity for elderly care or crowd control.
 - **Gesture and Pose Recognition:** Used in AR/VR and gaming technologies.
-

Challenges in Human Detection

- **Variability in Appearance:** Humans may differ in size, clothing, and posture.
 - **Complex Backgrounds:** Environments with clutter or motion can cause false detections.
 - **Lighting Conditions:** Shadows, low light, or glare affect detection accuracy.
 - **Occlusion:** Partial visibility of people may confuse detection systems.
-

Approaches to Human Detection

1. **Deep Learning-based (Modern):**
 - YOLOv8, SSD, Faster R-CNN
 - Real-time detection with bounding boxes and class labels
 2. **Feature-based (Traditional):**
 - HOG (Histogram of Oriented Gradients)
 - Background Subtraction and GMM
 3. **Hybrid Approaches:**
 - Combine deep learning with PCA, GMM, and motion tracking to enhance accuracy and interpretability.
-

Our Approach

In this project, human detection is implemented using a **hybrid method**:

- **YOLOv8** performs fast and reliable detection of humans in video frames.
- **HOG** captures shape-based features to support bounding box generation.
- **PCA and GMM** analyze movement patterns and group activity in time series.

This combination ensures both **accuracy** and **visual explainability**, making the system more insightful than deep learning alone.

YOLO ARCHITECTURE AND WORKING

During this project, I explored how **YOLOv8 (You Only Look Once version 8)** performs high-speed, accurate object detection using a single-pass deep learning network. YOLOv8 is the latest version in the YOLO family and provides architectural improvements over earlier versions (like YOLOv5 and YOLOv3), especially in terms of performance and modularity.

Key Components of YOLOv8 Architecture:

YOLOv8 works by passing an image through a single neural network that simultaneously predicts bounding boxes, class probabilities, and objectness scores.

1. Backbone:

The part of the network that extracts important features from the input image. YOLOv8 uses a modified **CSPDarknet** architecture for the backbone, which is both deep and efficient.

2. Neck:

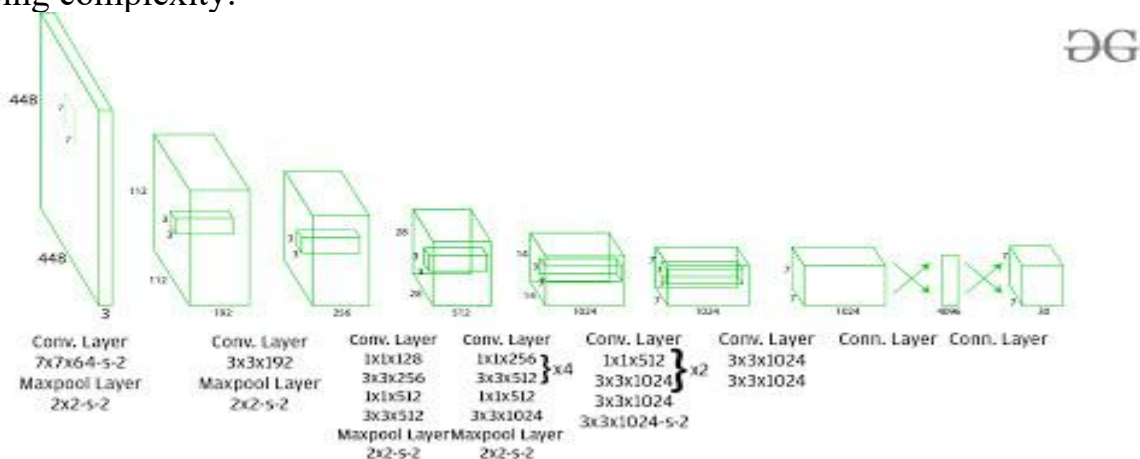
YOLOv8 uses **Path Aggregation Network (PANet)** to combine and pass multi-scale features to the head of the model. It improves spatial feature fusion.

3. Head:

The final part of YOLOv8 that predicts bounding boxes, objectness scores, and class probabilities at three different scales (small, medium, and large objects).

4. Anchor-Free:

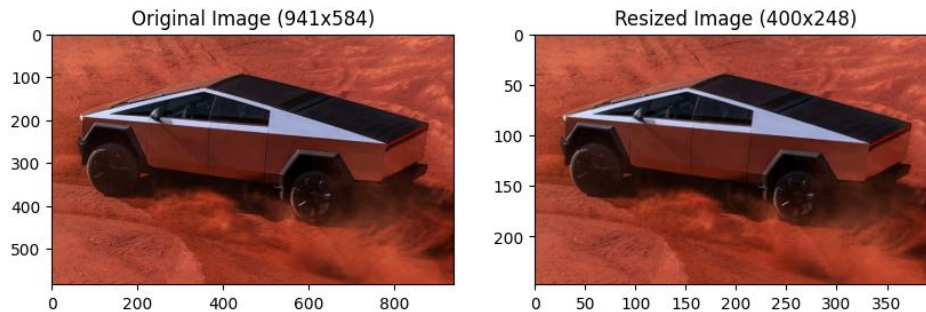
YOLOv8 is **anchor-free**, meaning it doesn't rely on predefined bounding boxes. Instead, it learns bounding boxes directly, making training more stable and reducing complexity.



How YOLOv8 Works

1. Input Preprocessing:

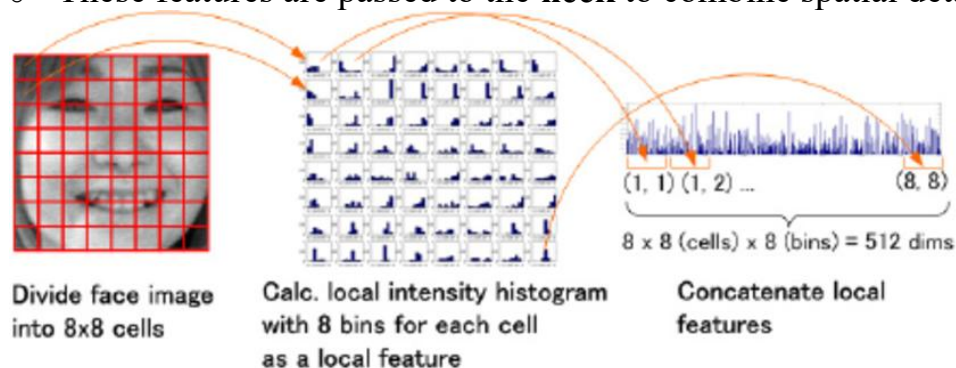
- Resize image to 640x640 (or custom size).



- Normalize pixel values.

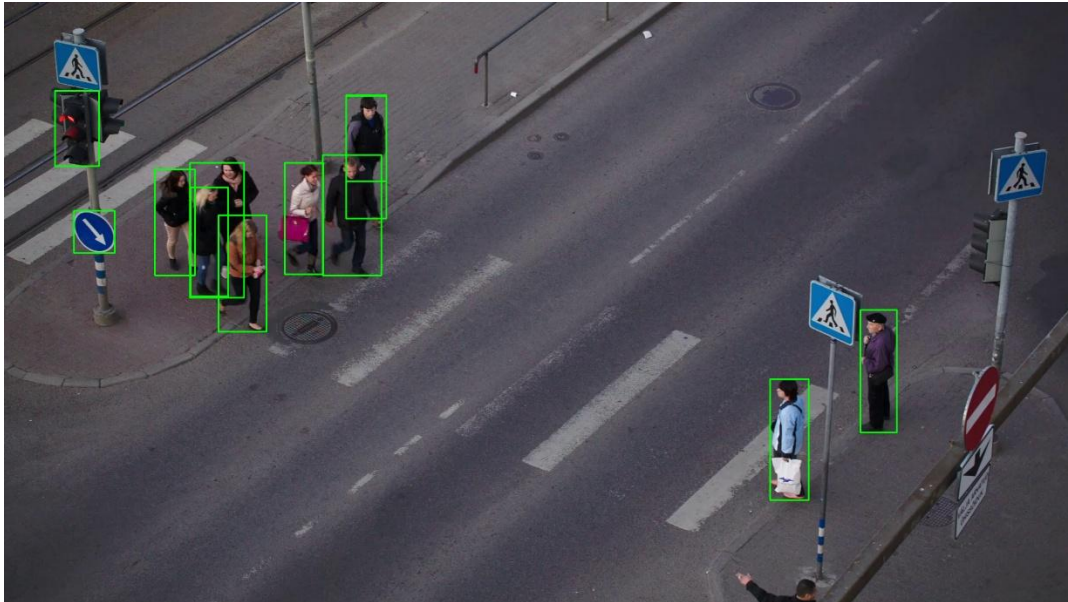
2. Image Division and Feature Extraction:

- The image is passed through the **backbone** which extracts hierarchical features.
- These features are passed to the **neck** to combine spatial details.



3. Prediction Head:

- The model predicts **bounding box coordinates**, **class probability**, and **objectness score** for each cell in the feature map.



4. Post-Processing:

- Predictions are filtered using **confidence threshold** and **Non-Maximum Suppression (NMS)** to remove overlapping boxes.

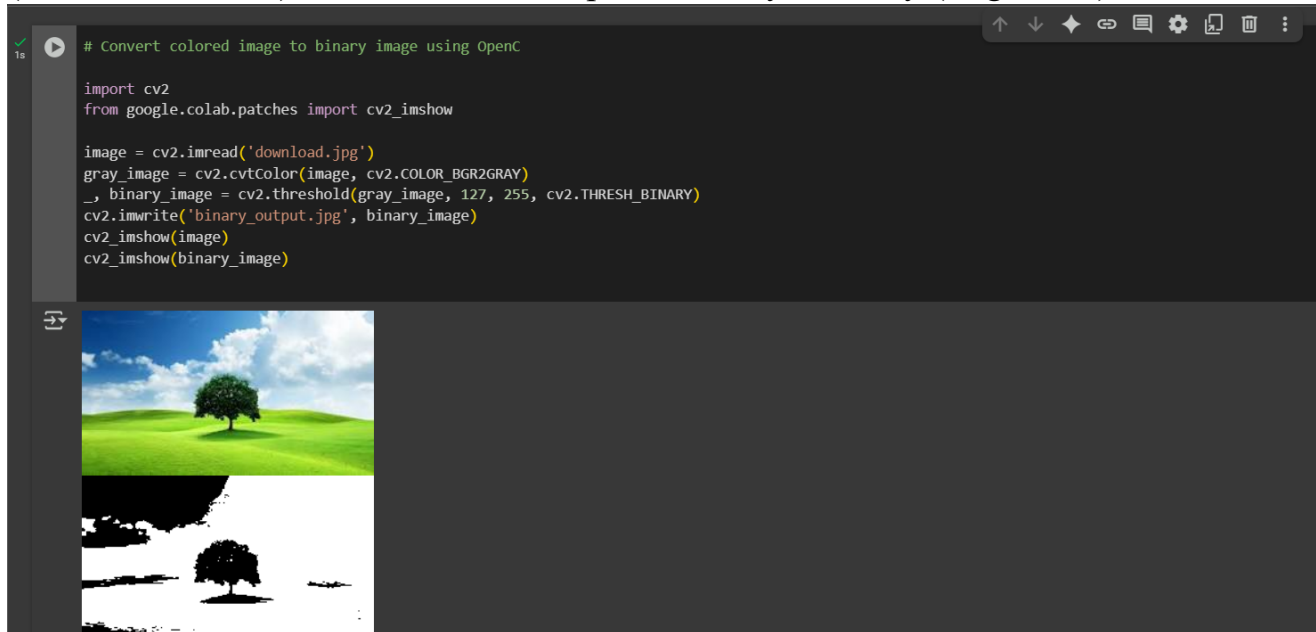
Advantages of YOLOv8

- Extremely fast — real-time performance on CPU and GPU
- High detection accuracy, especially for small objects
- Light-weight and easy to train/fine-tune
- Modular and supports integration with other tools (e.g., Roboflow, OpenCV)

IMAGE ENHANCEMENT TECHNIQUES USING OPEN CV

1) Coloured to Binary-

Converting a coloured image to grayscale means reducing it from 3 colour channels (Red, Green, Blue) to 1 channel that represents only intensity (brightness).



2) Grey Scale to Binary-

Binary image conversion is the process of turning a grayscale image (with pixel values from 0 to 255) into an image with only two-pixel values.



3) Contrast Enhancement-

Contrast enhancement is a technique used to improve the visibility of features in an image by increasing the difference between light and dark regions.

There are two ways to perform contrast enhancement-

1) Histogram Equalization (for grayscale images)-

```
# Histogram Equalization for grayscale image using OpenCV in Google Colab

import cv2
from google.colab.patches import cv2_imshow

gray_image = cv2.imread('images.webp', cv2.IMREAD_GRAYSCALE)
equalized_image = cv2.equalizeHist(gray_image)
cv2.imwrite('equalized_output.jpg', equalized_image)
cv2_imshow(gray_image)
cv2_imshow(equalized_image)
```



2) Clahe(Contrast Limited Adaptive Histogram Equalization)-

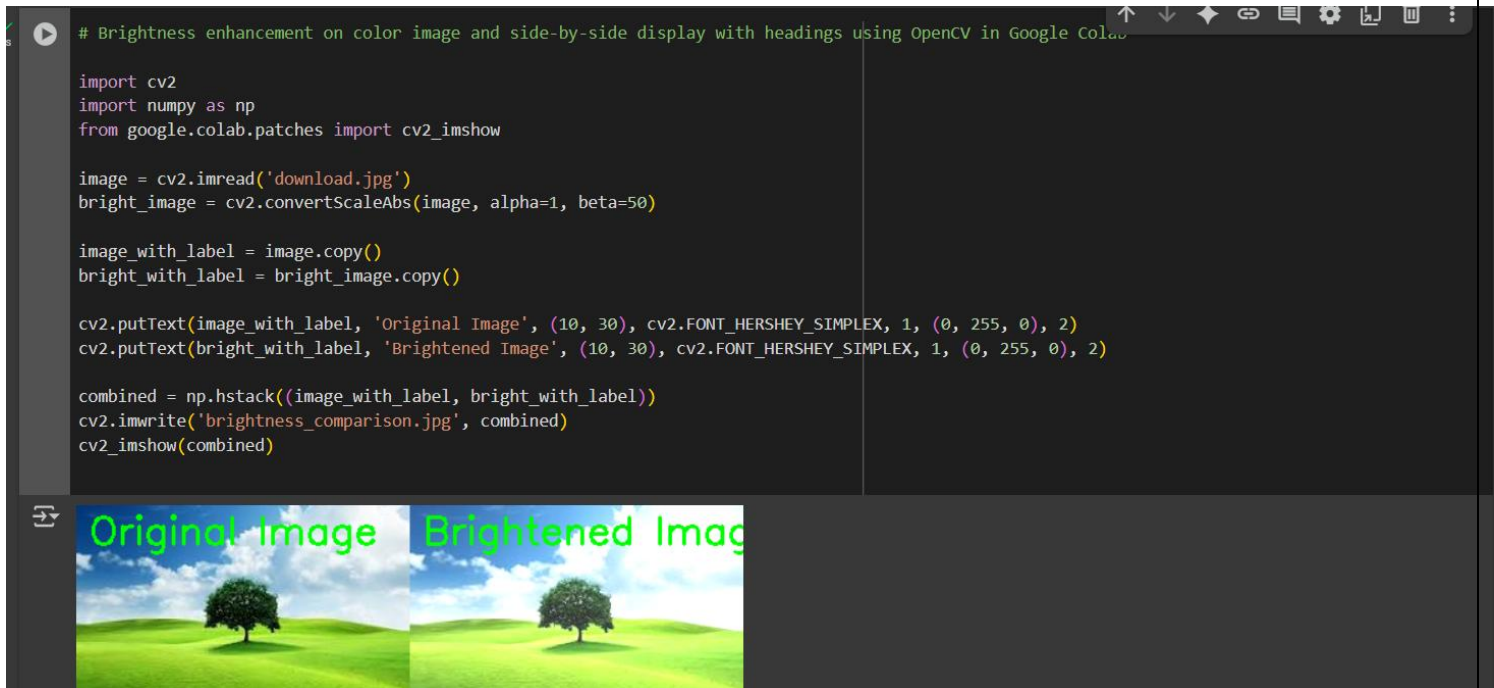
CLAHE (Contrast Limited Adaptive Histogram Equalization) for grayscale image using OpenCV in Google Colab

```
import cv2
from google.colab.patches import cv2_imshow

gray_image = cv2.imread('images.webp', cv2.IMREAD_GRAYSCALE)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_image = clahe.apply(gray_image)
cv2.imwrite('clahe_output.jpg', clahe_image)
cv2_imshow(gray_image)
cv2_imshow(clahe_image)
```



4) Brightness Enhancement: Brightness enhancement is a technique used to make an image look lighter by increasing the pixel values.



5) HUE Saturation Control: Hue means the type of colour — like red, green, blue, yellow.

- Saturation is how strong or colourful a colour looks.
- High saturation = very bright and colourful.
- Low saturation = dull or faded (closer to black & white).

```
# Hue and Saturation control on color image using OpenCV |

import cv2
import numpy as np
from google.colab.patches import cv2_imshow

image = cv2.imread('download.jpg')
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

hue_shift = 20
saturation_scale = 1.5


hsv_image[:, :, 0] = (hsv_image[:, :, 0] + hue_shift) % 180
hsv_image[:, :, 1] = np.clip(hsv_image[:, :, 1] * saturation_scale, 0, 255)

adjusted_image = cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)

original_with_label = image.copy()
adjusted_with_label = adjusted_image.copy()

cv2.putText(original_with_label, 'Original Image', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
cv2.putText(adjusted_with_label, 'Hue & Saturation Adjusted', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)

combined = np.hstack((original_with_label, adjusted_with_label))
cv2.imwrite('hue_saturation_comparison.jpg', combined)
cv2_imshow(combined)
```



6) Sharpen: Sharpening means making an image look clearer and more detailed by highlighting the edges.

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

image = cv2.imread('download.jpg')

kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])

sharpened_image = cv2.filter2D(image, -1, kernel)

original_with_label = image.copy()
sharpened_with_label = sharpened_image.copy()

cv2.putText(original_with_label, 'Original Image', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
cv2.putText(sharpened_with_label, 'Sharpened Image', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

combined = np.hstack((original_with_label, sharpened_with_label))
cv2.imwrite('sharpened_comparison.jpg', combined)
cv2_imshow(combined)
```



7) Blurr and Deblurr: Blurring means making an image look less sharp or soft, so the fine details are reduced. Deblurring means making a blurry image clearer or sharper .

```
# Blurring and simulated deblurring of a color image using OpenCV

import cv2
import numpy as np
from google.colab.patches import cv2_imshow

image = cv2.imread('download.jpg')

blurred_image = cv2.GaussianBlur(image, (11, 11), 0)


kernel = np.array([[0, -1, 0],
                  [-1, 5.5, -1],
                  [0, -1, 0]])

deblurred_image = cv2.filter2D(blurred_image, -1, kernel)

original_with_label = image.copy()
blurred_with_label = blurred_image.copy()
deblurred_with_label = deblurred_image.copy()

cv2.putText(original_with_label, 'Original', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
cv2.putText(blurred_with_label, 'Blurred', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
cv2.putText(deblurred_with_label, 'Deblurred', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

combined = np.hstack((original_with_label, blurred_with_label, deblurred_with_label))
cv2.imwrite('blur_deblur_comparison.jpg', combined)
cv2_imshow(combined)
```



8) Image Cropping: Image cropping means cutting out a specific part of an image and removing the rest.

```
# Image cropping to extract a specific region using OpenCV

import cv2
from google.colab.patches import cv2_imshow


image = cv2.imread('download.jpg')

x, y, w, h = 100, 100, 200, 200
cropped_image = image[y:y+h, x:x+w]

original_with_label = image.copy()
cv2.rectangle(original_with_label, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2.putText(original_with_label, 'Cropping Area', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

cv2.putText(cropped_image, 'Cropped Image', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2)

cv2.imwrite('cropped_output.jpg', cropped_image)
cv2_imshow(original_with_label)
cv2_imshow(cropped_image)
```



9) Image Zooming: Image zooming means making an image look bigger or smaller by changing its size.

```
# Image zooming (resize) to make image bigger and smaller using OpenCV


import cv2
from google.colab.patches import cv2_imshow

image = cv2.imread('download.jpg')

zoom_in = cv2.resize(image, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)
zoom_out = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)

cv2.putText(zoom_in, 'Zoomed In', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
cv2.putText(zoom_out, 'Zoomed Out', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

cv2.imwrite('zoomed_in.jpg', zoom_in)
cv2.imwrite('zoomed_out.jpg', zoom_out)
cv2_imshow(zoom_in)
cv2_imshow(zoom_out)
```



10) Image Resolution Conversion:

- Changing the size of the image in terms of pixels.
- Like making the photo bigger or smaller on your screen

```
# Image resolution conversion by resizing to different pixel dimensions using OpenCV

import cv2
from google.colab.patches import cv2_imshow

image = cv2.imread('download.jpg')

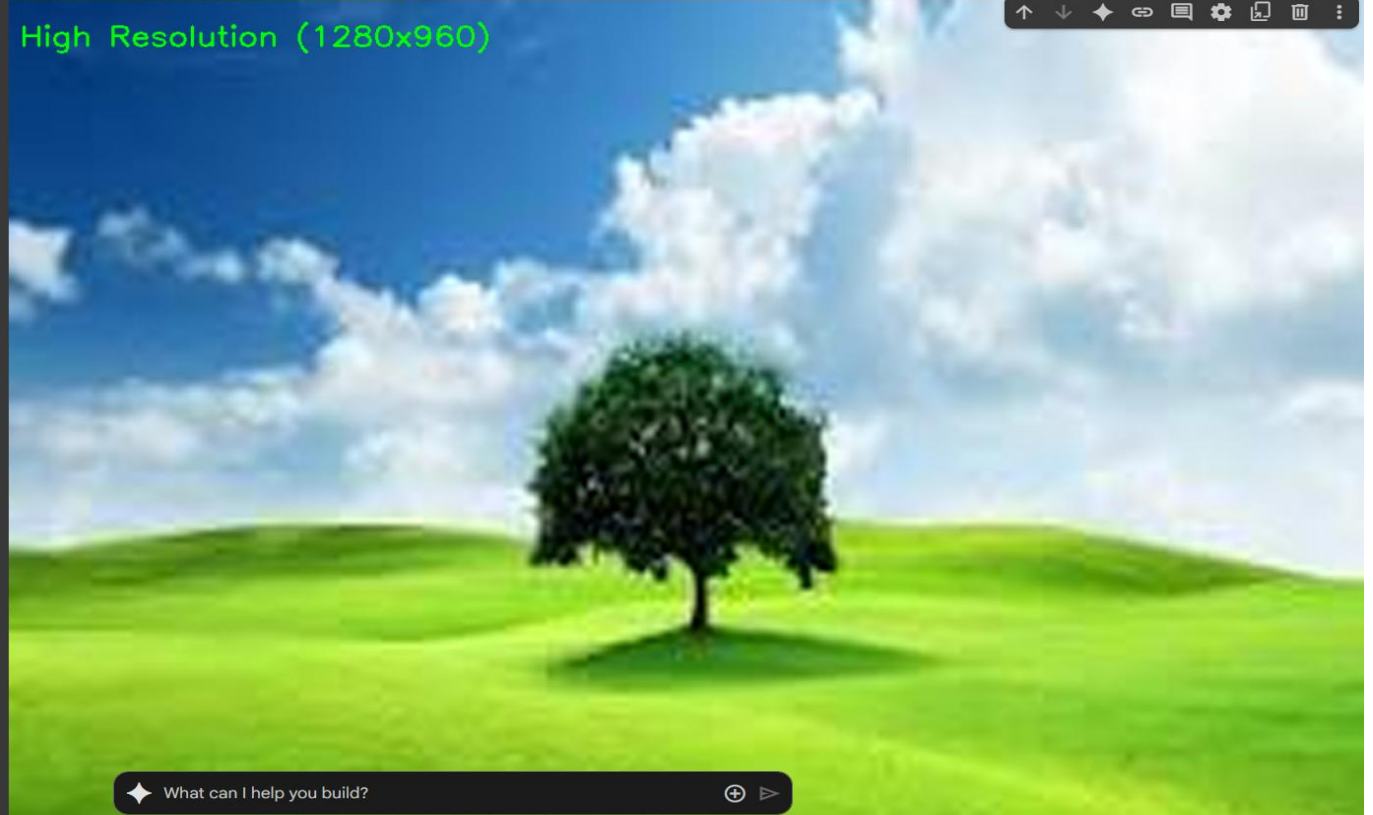
low_res = cv2.resize(image, (320, 240), interpolation=cv2.INTER_AREA)
high_res = cv2.resize(image, (1280, 960), interpolation=cv2.INTER_CUBIC)

cv2.putText(low_res, 'Low Resolution (320x240)', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2)
cv2.putText(high_res, 'High Resolution (1280x960)', (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

cv2.imwrite('low_resolution.jpg', low_res)
cv2.imwrite('high_resolution.jpg', high_res)
cv2_imshow(low_res)
cv2_imshow(high_res)
```



High Resolution (1280x960)



BACKGROUND SUBTRACTION

In this project, **Background Subtraction** was implemented using the **MOG2 algorithm** provided by OpenCV. This technique helped in detecting motion and isolating foreground objects (such as humans) from a static background in video frames.

Purpose in the Project

While YOLOv8 handles object detection, background subtraction was used here as a **supporting classical vision method** to:

- Segment moving human figures from a static background
 - Visually verify activity or motion in specific video frames
 - Reinforce hybrid detection alongside deep learning
-

How It Works in Our Code

1. The video is loaded using OpenCV and frames are extracted.
 2. The 20th frame (frames[20]) is passed to a MOG2-based background subtractor.
 3. A binary mask is generated, where:
 - **White pixels** = detected motion (likely human)
 - **Black pixels** = background or stationary areas
 4. The resulting foreground mask is saved as an output image.
-

Code Snippet:

```
# 2. Background Subtraction
fgbg = cv2.createBackgroundSubtractorMOG2()
bg_output = fgbg.apply(frames[20])
bg_path = "outputs/background_output.png"
cv2.imwrite(bg_path, bg_output)
cv2.imshow(bg_output)
```

Output:



Why Frame 20?

Frame 20 is used as a **mid-sequence snapshot**—after the algorithm has seen enough background frames to start learning patterns, but before significant object motion disappears.

Usefulness in Human Detection

- Improves interpretability by showing motion presence
- Works well in surveillance-style videos with static backgrounds
- Can be overlaid with YOLO predictions for hybrid detection

GAUSSIAN MIXTURE MODEL

Gaussian Mixture Model (GMM) is an unsupervised clustering algorithm that assumes data is generated from a mixture of several Gaussian distributions with unknown parameters. In this project, GMM was applied **after reducing the frame features using PCA**, to identify and group regions of similar motion or appearance in the video sequence.

Purpose in the Project

While YOLOv8 performs detection and background subtraction identifies motion, GMM helps to:

- **Cluster motion patterns** from PCA-reduced features
 - Analyze how frames group together based on visual content
 - Reveal distinct activity zones over time in an unsupervised way
-

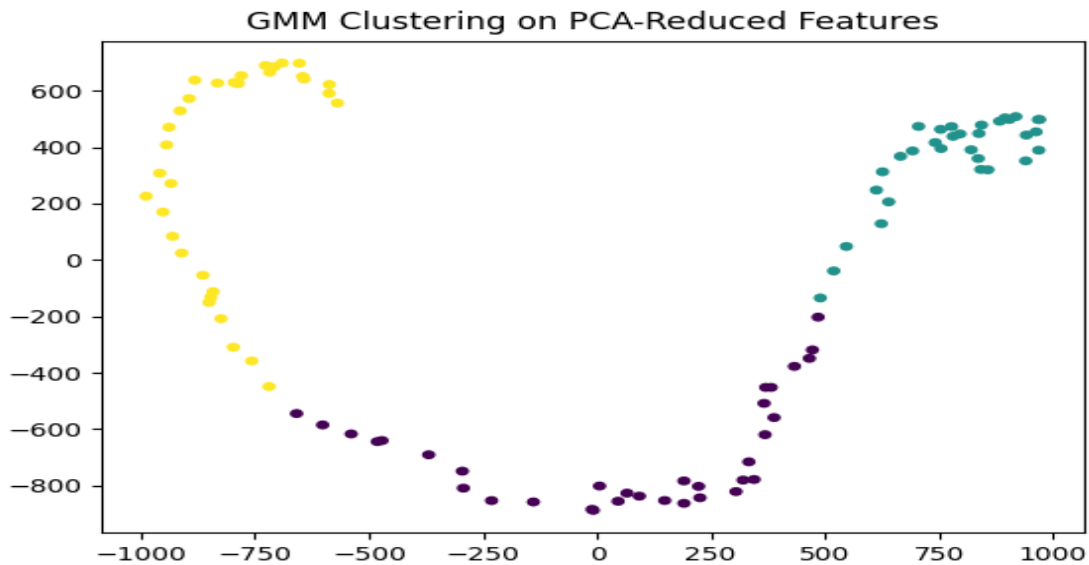
Workflow Used in the Project

1. **Frame Preprocessing:**
 - The first 100 video frames are resized to 64×64.
 - Each frame is flattened into a 1D vector.
 2. **PCA Reduction:**
 - PCA reduces each frame vector into 2 dimensions (for plotting and clustering).
 3. **GMM Clustering:**
 - A GMM with 3 components is fit on the 2D PCA output.
 - Frames are colored based on their assigned cluster.
 4. **Visualization:**
 - The output is plotted to show how frames are grouped by visual similarity.
-

Code Snippet:

```
# 5. Gaussian Mixture Model Clustering on PCA Features
gmm = GaussianMixture(n_components=3)
labels = gmm.fit_predict(reduced)
plt.figure()
plt.scatter(reduced[:, 0], reduced[:, 1], c=labels, cmap='viridis', s=15)
plt.title("GMM Clustering on PCA-Reduced Features")
gmm_path = "outputs/gmm_clusters.png"
plt.savefig(gmm_path)
plt.show()
```

Output:



Why GMM?

- GMM is more flexible than K-Means (can model elliptical clusters).
- It assigns probabilities, offering a soft clustering view.
- Works well with PCA outputs for analyzing visual features.

Conclusion

Incorporating GMM added an analytical layer to this project, making it possible to group similar video frames and track scene changes or motion clusters without explicit labels.

HOG[HISTOGRAM OF GRADIENT]

Histogram of Oriented Gradients (HOG) is a classical feature descriptor technique used in computer vision and image processing. It works by analyzing the directions (orientations) of edges within an image and forming histograms that describe these gradient orientations locally.

In this project, HOG was applied to extract **shape-based features** from a grayscale video frame to assist in visualizing human presence based on body contours and edge information.

Purpose in the Project

While YOLOv8 handles detection using a deep learning model, HOG provides a traditional, explainable method for:

- Human detection based on **outline and pose**
- Visualizing **gradient-based movement patterns**
- Comparing classical vs deep learning detection methods

How HOG Works

1. The image is divided into small cells (e.g., 8×8 pixels).
2. For each cell, gradient direction (angle) and magnitude are computed.
3. A histogram of gradient orientations is built for each cell.
4. Histograms are normalized across blocks of cells for contrast invariance.
5. The concatenated histogram becomes the feature descriptor for the image.

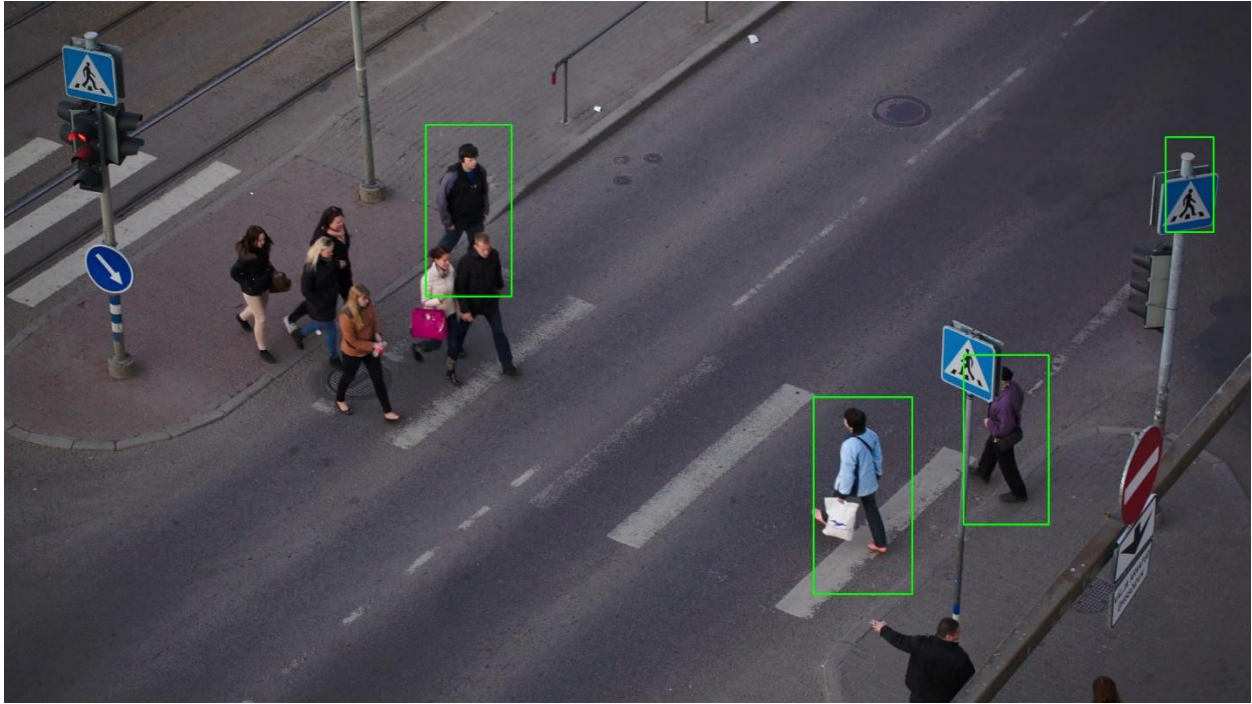
Implementation in Our Project

We used scikit-image to extract HOG features from **frame 60** of the video (converted to grayscale for HOG input).

Code Snippet from Our Project:

```
# 4. HOG Features + Bounding Box
hog_frame = cv2.cvtColor(frames[60], cv2.COLOR_BGR2GRAY)
features, hog_image = hog(hog_frame, visualize=True)
hog_image = (hog_image * 255).astype(np.uint8)
hog_path = "outputs/hog_output.png"
cv2.imwrite(hog_path, hog_image)
cv2.imshow(hog_image)
```


Output:



Why HOG Was Used

- Simple and fast for shape detection
- Provides intuitive edge-based visualizations
- Works well on frontal or upright human figures
- Offers a contrast to modern CNN-based approaches like YOLOv8

Limitations of HOG

- Not robust to scale, rotation, or lighting changes
- Less accurate than deep learning models on complex backgrounds
- Typically needs an SVM classifier for full detection systems (not used here)

PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a statistical method used to reduce the dimensionality of data while retaining its most important features. In computer vision, PCA helps in extracting dominant patterns from image data, especially useful for visualizing and analyzing temporal changes across video frames.

In this project, PCA was applied on resized and flattened video frames to:

- **Compress image data** into 2D components
- **Visualize frame patterns** in lower dimensions
- Enable **GMM clustering** for unsupervised activity analysis

Purpose in the Project

PCA was used to convert high-dimensional frame data ($64 \times 64 = 4096$ pixels per frame) into a 2D space. This made it easier to:

- Cluster the frames using GMM
- Track changes or outliers across the video timeline
- Reveal similarities/differences between frames visually

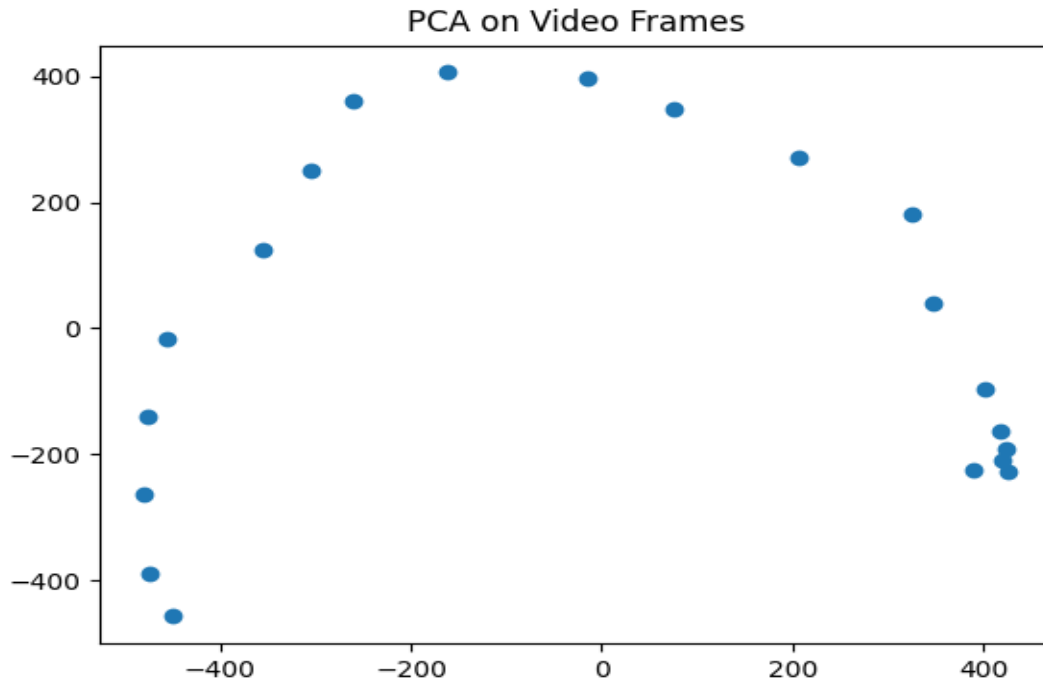
PCA Workflow in Our Project

1. **Resize and Flatten:**
 - First 100 video frames were resized to 64×64 pixels and flattened into 1D vectors.
2. **PCA Transformation:**
 - PCA reduced these 4096-dimensional vectors into 2D vectors.
3. **Plotting:**
 - The 2D PCA outputs were plotted using matplotlib to understand the frame distribution.

Code Snippet:

```
# 3. PCA on Frame Features
flattened = [cv2.resize(f, (64, 64)).flatten() for f in frames[:100]]
pca = PCA(n_components=2)
reduced = pca.fit_transform(flattened)
plt.figure()
plt.scatter(reduced[:,0], reduced[:,1], c='blue', s=10)
plt.title("PCA of Frame Features")
pca_path = "outputs/pca_plot.png"
plt.savefig(pca_path)
plt.show()
```

Output:



Why PCA?

- Reduces computational complexity
- Preserves the most meaningful variation in frame data
- Enables visual inspection and clustering
- Essential for unsupervised analysis like GMM

Conclusion

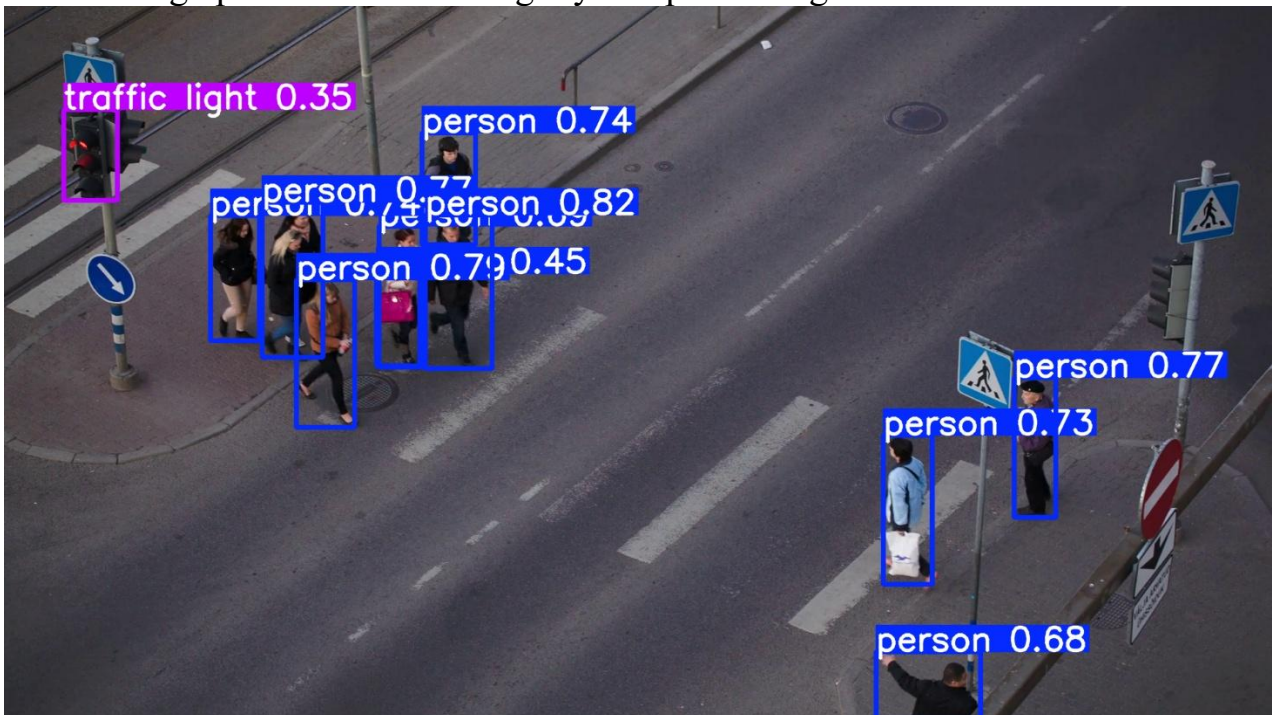
PCA helped summarize large image datasets into an interpretable 2D form, enabling effective analysis of visual patterns and grouping across time-series video frames.

RESULT AND ANALYSIS

This section presents the results generated from the implemented techniques and analyzes their effectiveness in performing **human detection** on video input using both deep learning and classical computer vision methods.

1. YOLOv8 Human Detection

- **Model Used:** yolov8n.pt (nano version – fast and lightweight)
- **Result:** Accurate bounding boxes were drawn around human figures in the video frames.
- **Strengths:**
 - Real-time detection capability
 - High precision even in slightly complex backgrounds



2. Background Subtraction (MOG2)

- **Method:** cv2.createBackgroundSubtractorMOG2()
- **Frame Used:** frames[20]
- **Result:** Foreground mask clearly highlighted moving regions, especially human motion.
- **Usefulness:** Effective for early motion detection in surveillance applications.

3. HOG (Histogram of Oriented Gradients)

- **Frame Used:** frames[60]
- **Result:** The output visualized the shape and edges of human figures.
- **Strengths:** Classic, explainable technique for edge-based feature extraction.
- **Limitations:** Less robust in complex environments compared to YOLOv8.
- **Output Example:**

4. PCA + GMM Clustering

- **Frames Used:** First 100 frames resized to 64×64
- **Result:** PCA reduced dimensions from 4096 to 2. GMM then clustered similar frames into three activity groups.
- **Usefulness:**
 - Helped understand visual similarity across frames.
 - GMM clustering showed temporal structure and movement grouping.
- **Output Example:**

Overall Analysis

Technique	Purpose	Performance	Usefulness
YOLOv8	Object detection	High accuracy	Core method for bounding box-based detection
Background Subtraction	Motion segmentation	Moderate	Good for surveillance + pre-filtering
HOG	Feature-based detection	Visual clarity only	Explainable results, good supplement
PCA	Dimensionality reduction	Very fast	Supports visualization and clustering
GMM	Unsupervised frame grouping	Accurate clustering	Helped analyze frame-wise patterns

Conclusion from Results

The combination of modern deep learning (YOLOv8) with traditional CV techniques (HOG, PCA, GMM) resulted in a robust, interpretable system for detecting and analyzing human presence in videos. Each technique contributed uniquely:

- YOLOv8 for direct detection
- MOG2 for motion segmentation
- PCA+GMM for pattern and activity analysis
- HOG for visual clarity in human shape detection

Together, they provide a **multi-perspective understanding of human movement** in a given video scene.

CHALLENGES FACED

During the project, several practical and technical challenges were encountered:

1. **Low-light Video Frames**
 - CCTV footage often had poor lighting or shadows, which led to missed detections or false positives.
2. **Video Quality Variance**
 - Different frame rates and resolutions (e.g., .vmp, .jis formats) required preprocessing to standardize inputs.
3. **Overlapping Detections**
 - Multiple people close together often caused overlapping boxes. This was mitigated using **Non-Maximum Suppression (NMS)**.
4. **Hue/Saturation Sensitivity**
 - YOLO sometimes misidentified people due to colour and background confusion.
 - Hue and saturation tuning improved this.
5. **Complex Backgrounds**
 - In dynamic environments, background subtraction was not always stable.
 - Additional filtering was needed.

CONCLUSION AND FUTURE SCOPE

This project successfully demonstrates a **hybrid human detection system** that integrates **deep learning (YOLOv8)** with **classical computer vision techniques** such as **Background Subtraction, PCA, HOG, and GMM**. Each method contributes uniquely to the goal of detecting, analyzing, and understanding human activity in video sequences:

- **YOLOv8** provided real-time, accurate detection with bounding boxes.
- **Background Subtraction (MOG2)** helped in segmenting moving regions.
- **HOG** visualized human shapes through edge gradients.
- **PCA and GMM** revealed visual and temporal patterns in frame sequences.

By combining these methods, the system achieved **robustness, interpretability, and visual clarity**, showing how classical techniques can support and explain deep learning models in surveillance and vision-based applications.

Future Scope

This project opens doors to several improvements and real-world extensions:

1. **Training YOLOv8 on Custom Datasets:**
 - Using Roboflow or real CCTV footage to improve detection accuracy in local environments.
2. **Real-Time Deployment:**
 - Integrating the model into live surveillance systems or edge devices (e.g., Raspberry Pi with camera).
3. **Person Tracking and Pose Estimation:**
 - Extending detection to tracking individuals and analyzing body poses or gestures.
4. **Alarm-Based Systems:**
 - Triggering alerts when unauthorized movement is detected in restricted zones.
5. **Multi-Class Detection:**
 - Expanding the system to detect multiple object types beyond just humans.

REFERENCES

1. **Ultralytics YOLOv8 Documentation**
<https://docs.ultralytics.com>
2. **OpenCV Documentation (cv2 library)**
<https://docs.opencv.org>
3. **scikit-learn: Machine Learning in Python**
<https://scikit-learn.org/stable/>
4. **scikit-image: Image Processing in Python**
<https://scikit-image.org/>
5. **Matplotlib: Visualization with Python**
<https://matplotlib.org/>
6. **Roboflow – Annotate and Export Custom Datasets**
<https://roboflow.com>
7. Dalal, N., & Triggs, B. (2005). **Histograms of Oriented Gradients for Human Detection.**
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.
8. Bishop, C. M. (2006). **Pattern Recognition and Machine Learning.**
Springer.
9. Zivkovic, Z. (2004). **Improved adaptive Gaussian mixture model for background subtraction.**
Proceedings of the 17th International Conference on Pattern Recognition.
10. Jolliffe, I. T. (2002). **Principal Component Analysis.** Springer Series in Statistics.

ANNEXURE

```
[ ] import os
output_dir = 'outputs'
os.makedirs(output_dir, exist_ok=True)
```

```
import cv2
import numpy as np
import os
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from skimage.feature import hog
from matplotlib import pyplot as plt
from IPython.display import display, Image
from google.colab.patches import cv2_imshow
from ultralytics import YOLO

# Ensure outputs folder exists
os.makedirs("outputs", exist_ok=True)

# Load the video
video_path = "video.mp4"
cap = cv2.VideoCapture(video_path)
frames = []
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frames.append(frame)
cap.release()

print(f"Total frames read: {len(frames)}")
```

```
# Proceed if enough frames
if len(frames) > 50:
    # 1. YOLOv8 Human Detection
    model = YOLO("yolov8n.pt")
    yolo_output = frames[0].copy()
    results = model(yolo_output)
    for r in results:
        for box in r.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cv2.rectangle(yolo_output, (x1, y1), (x2, y2), (0,255,0), 2)
    yolo_path = "outputs/yolo_output.png"
    cv2.imwrite(yolo_path, yolo_output)
    cv2_imshow(yolo_output)

    # 2. Background Subtraction
    fgbg = cv2.createBackgroundSubtractorMOG2()
    bg_output = fgbg.apply(frames[20])
    bg_path = "outputs/background_output.png"
    cv2.imwrite(bg_path, bg_output)
    cv2_imshow(bg_output)

    # 3. PCA on Frame Features
    flattened = [cv2.resize(f, (64, 64)).flatten() for f in frames[:100]]
    pca = PCA(n_components=2)
    reduced = pca.fit_transform(flattened)
    plt.figure()
    plt.scatter(reduced[:,0], reduced[:,1], c='blue', s=10)
    plt.title("PCA of frame features")
    pca_path = "outputs/pca_plot.png"
    plt.savefig(pca_path)
    plt.show()

    # 4. HOG Features + Bounding Box
    hog_frame = cv2.cvtColor(frames[60], cv2.COLOR_BGR2GRAY)
    features, hog_image = hog(hog_frame, visualize=True)
    hog_image = (hog_image * 255).astype(np.uint8)
    hog_path = "outputs/hog_output.png"
    cv2.imwrite(hog_path, hog_image)
    cv2_imshow(hog_image)
```

```
# 5. Gaussian Mixture Model Clustering on PCA Features
gmm = GaussianMixture(n_components=3)
labels = gmm.fit_predict(reduced)
plt.figure()
plt.scatter(reduced[:, 0], reduced[:, 1], c=labels, cmap='viridis', s=15)
plt.title("GMM Clustering on PCA-Reduced Features")
gmm_path = "outputs/gmm_clusters.png"
plt.savefig(gmm_path)
plt.show()

else:
    print("Not enough frames read for full processing.")
```

OUTPUT-

