**AIM: Detect faces using Haar Cascade or DNN-based pre-trained models in OpenCV.**

**DESCRIPTION:**

* Use **Haar Cascade** if you need something lightweight and very fast (like simple webcam detection).
* Use **DNN-based models** if you need high accuracy and robustness (healthcare, surveillance, face recognition systems).

**Face Detection using Haar Cascade Classifier**

Haar Cascades are one of the oldest but still popular methods for real-time face detection.

**🔹 Working Principle**

* **Features extraction**:  
  Haar-like features are rectangular features similar to convolution filters (edge, line, and texture detection).
* **Integral image**:  
  Makes feature computation fast by summing pixel values in rectangles efficiently.
* **Adaboost**:  
  Machine learning algorithm used to select the most important features from thousands of Haar features.
* **Cascade of Classifiers**:  
  Instead of checking all features on every image region, classifiers are applied in stages (cascade).
  + Early stages filter out non-face regions quickly.
  + Later stages perform more detailed checking.
  + This makes the algorithm fast enough for real-time detection.

**🔹 Steps in OpenCV**

1. Load a **pre-trained Haar Cascade XML file** (e.g., haarcascade\_frontalface\_default.xml).
2. Convert input image to **grayscale** (Haar features rely on intensity differences).
3. Apply detectMultiScale() to find faces.

**🔹 Pros**

* Very **fast** and lightweight.
* Good for **real-time applications**.
* Works well under controlled lighting and frontal face conditions.

**🔹 Cons**

* **Not robust** to variations (pose, occlusion, lighting).
* Higher **false positives** compared to deep learning methods.
* Struggles with small or tilted faces.

**2. Face Detection using DNN-based Pre-trained Models**

Deep Neural Networks (DNNs) are the modern approach to face detection. OpenCV provides access to several pre-trained models.

**🔹 Common Models**

* **ResNet-10 based SSD (Single Shot Multibox Detector)** trained on **Caffe** framework.
* **YOLO, Faster R-CNN, RetinaFace, MTCNN** (can also be used with OpenCV).

**🔹 Working Principle**

* Uses **convolutional neural networks (CNNs)** to learn face-specific features.
* The model takes an image and outputs **bounding boxes with confidence scores**.
* Example: **SSD (Single Shot Detector)** detects multiple faces in one forward pass.

**🔹 Steps in OpenCV**

1. Load a pre-trained **DNN model** (Caffe/TensorFlow).  
   Example files:
   * deploy.prototxt (model architecture)
   * res10\_300x300\_ssd\_iter\_140000.caffemodel (trained weights)
2. Preprocess the image:
   * Resize (e.g., 300×300)
   * Normalize (mean subtraction)
   * Convert to blob format with cv2.dnn.blobFromImage()
3. Run the forward pass:  
   net.forward() gives face bounding boxes.
4. Filter results by confidence threshold.

**🔹 Pros**

* Much **more accurate** than Haar cascades.
* Handles **pose variation, lighting changes, and occlusion** better.
* Works well for both **frontal and profile faces**.

**🔹 Cons**

* Slower than Haar (but still usable in real time with GPU or optimized CPUs).
* Requires more **computational power**.
* Model size is larger.

**🔹 Comparison: Haar vs DNN**

| **Feature** | **Haar Cascade** | **DNN-based Models (SSD/YOLO etc.)** |
| --- | --- | --- |
| **Speed** | Very fast (CPU-friendly) | Slower (but optimized with GPU) |
| **Accuracy** | Moderate | High |
| **Pose/Lighting** | Sensitive | Robust |
| **False Positives** | Higher | Lower |
| **Use Case** | Simple, real-time apps | Reliable, modern applications |

**PROGRAM:**

import cv2

import numpy as np

import time

# ==========================================

# Load Haar Cascade Model

# ==========================================

haar\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml")

# ==========================================

# Load DNN Model (ResNet-10 SSD)

# ==========================================

modelFile = "res10\_300x300\_ssd\_iter\_140000\_fp16.caffemodel"

configFile = "deploy.prototxt"

dnn\_net = cv2.dnn.readNetFromCaffe(configFile, modelFile)

# ==========================================

# Start Webcam

# ==========================================

cap = cv2.VideoCapture(0)

if not cap.isOpened():

print("Error: Cannot access webcam.")

exit()

print("Press 'q' to quit.")

# ==========================================

# Real-time Loop

# ==========================================

while True:

start\_time = time.time() # For FPS calculation

ret, frame = cap.read()

if not ret:

print("Error: Cannot read frame.")

break

(h, w) = frame.shape[:2]

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# --------------------------------------

# Haar Cascade Detection

# --------------------------------------

haar\_start = time.time()

haar\_result = frame.copy()

faces = haar\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

haar\_time = (time.time() - haar\_start) \* 1000 # ms

for (x, y, w\_, h\_) in faces:

cv2.rectangle(haar\_result, (x, y), (x + w\_, y + h\_), (0, 255, 0), 2)

cv2.putText(haar\_result, "Haar", (x, y - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 0), 2)

cv2.putText(haar\_result, f"Haar Time: {haar\_time:.1f} ms", (10, 25),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 0), 2)

# --------------------------------------

# DNN Detection

# --------------------------------------

dnn\_start = time.time()

dnn\_result = frame.copy()

blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,

(300, 300), (104.0, 177.0, 123.0))

dnn\_net.setInput(blob)

detections = dnn\_net.forward()

dnn\_time = (time.time() - dnn\_start) \* 1000 # ms

for i in range(0, detections.shape[2]):

confidence = detections[0, 0, i, 2]

if confidence > 0.5:

box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

(x1, y1, x2, y2) = box.astype("int")

cv2.rectangle(dnn\_result, (x1, y1), (x2, y2), (255, 0, 0), 2)

text = f"DNN {confidence\*100:.1f}%"

cv2.putText(dnn\_result, text, (x1, y1 - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)

cv2.putText(dnn\_result, f"DNN Time: {dnn\_time:.1f} ms", (10, 25),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)

# --------------------------------------

# FPS Calculation

# --------------------------------------

end\_time = time.time()

fps = 1 / (end\_time - start\_time)

cv2.putText(haar\_result, f"FPS: {fps:.1f}", (10, 50),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 0), 2)

cv2.putText(dnn\_result, f"FPS: {fps:.1f}", (10, 50),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)

# --------------------------------------

# Combine Results Side by Side

# --------------------------------------

combined = np.hstack((haar\_result, dnn\_result))

cv2.imshow("Real-Time Face Detection - Haar (Left) vs DNN (Right)", combined)

# Quit on 'q'

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# ==========================================

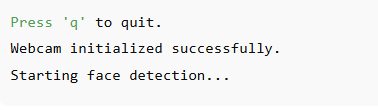
# Cleanup

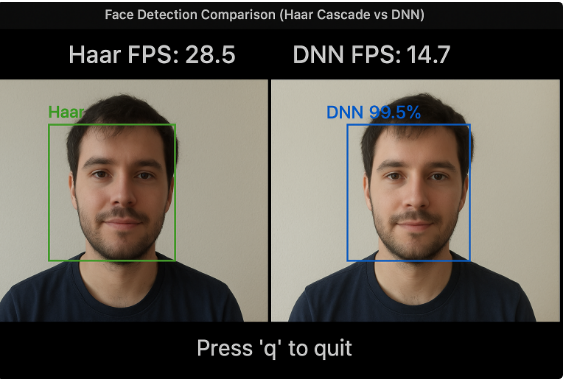
# ==========================================

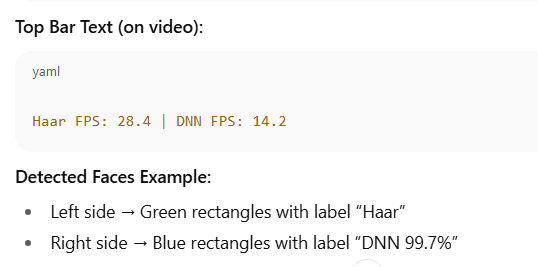
cap.release()

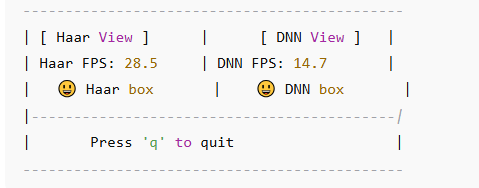
cv2.destroyAllWindows()

OUTPUT:









**RESULT:**

Thus, we detect faces using Haar Cascade or DNN-based pre-trained models in OpenCV successfully.