EXP NO: 9	FACE DETECTION ON ONLINE HUMAN FACE IMAGE DATASETS

AIM:

To implement face detection using multiple methods (Haar cascade, HOG+SVM, MTCNN, SSD/RetinaFace).

COMMON PUBLIC FACE DATASETS (ONLINE)

1. LFW (Labeled Faces in the Wild) — face images for recognition and detection research.
2. FDDB (Face Detection Data Set and Benchmark) — annotated faces in unconstrained settings.
3. WIDER FACE — large-scale dataset with wide variations in scale, pose and occlusion.
4. CelebA — celebrity faces with attribute annotations (useful for detection+attribute tasks).

REQUIREMENTS:

Software: Python 3.x
Libraries: OpenCV, dlib (optional), mtcnn, facenet-pytorch or retinaface implementations, numpy, matplotlib

Install examples:
pip install opencv-python numpy matplotlib mtcnn facenet-pytorch
dlib install may require CMake and boost; use conda or follow platform-specific instructions.

FACE DETECTION METHODS

1. Haar Cascade (Viola-Jones): Fast classical detector using boosted cascade of Haar-like features.
2. HOG + Linear SVM (Dalal-Triggs / dlib): Uses gradient orientation histograms and sliding-window SVM.
3. MTCNN (Multi-task Cascaded CNN): Cascade of CNNs for detection and alignment; good for faces in the wild.
4. Single Shot Detectors (SSD) with MobileNet or RetinaFace: Deep detectors with high accuracy and speed.
5. Anchor-free methods and modern detectors (e.g., CenterFace, BlazeFace) can be explored as extensions.

GENERAL ALGORITHM STEPS (FOR EVALUATION PIPELINE)

Step 1: Start

Step 2: Prepare dataset: download images and ground-truth annotations in required format (e.g., bounding boxes).

Step 3: Preprocess images (resize, normalize) as required by the detector.

Step 4: Run detector on each image and collect predicted bounding boxes and scores.

Step 5: Apply non-maximum suppression (NMS) to remove duplicate overlapping detections (if not built-in).

Step 6: Match predictions to ground-truth using IoU threshold (e.g., 0.5) to determine TP/FP/FN.

Step 7: Compute metrics: precision, recall, F1-score, Average Precision (AP), mAP over dataset.

Step 8: Visualize sample detections and analyze false positives and false negatives.

Step 9: End

ALGORITHM: HAAR CASCADE (VIOLA-JONES)

1. Load pre-trained Haar cascade XML from OpenCV (e.g., haarcascade_frontalface_default.xml).

2. Read and optionally resize input image.

3. Convert to grayscale and (optional) equalize histogram for robustness.

4. Run cascade.detectMultiScale() to get bounding boxes and scales.

5. Apply basic filtering (minSize, scaleFactor, minNeighbors) to improve detections.

6. Visualize detections and compute evaluation metrics.

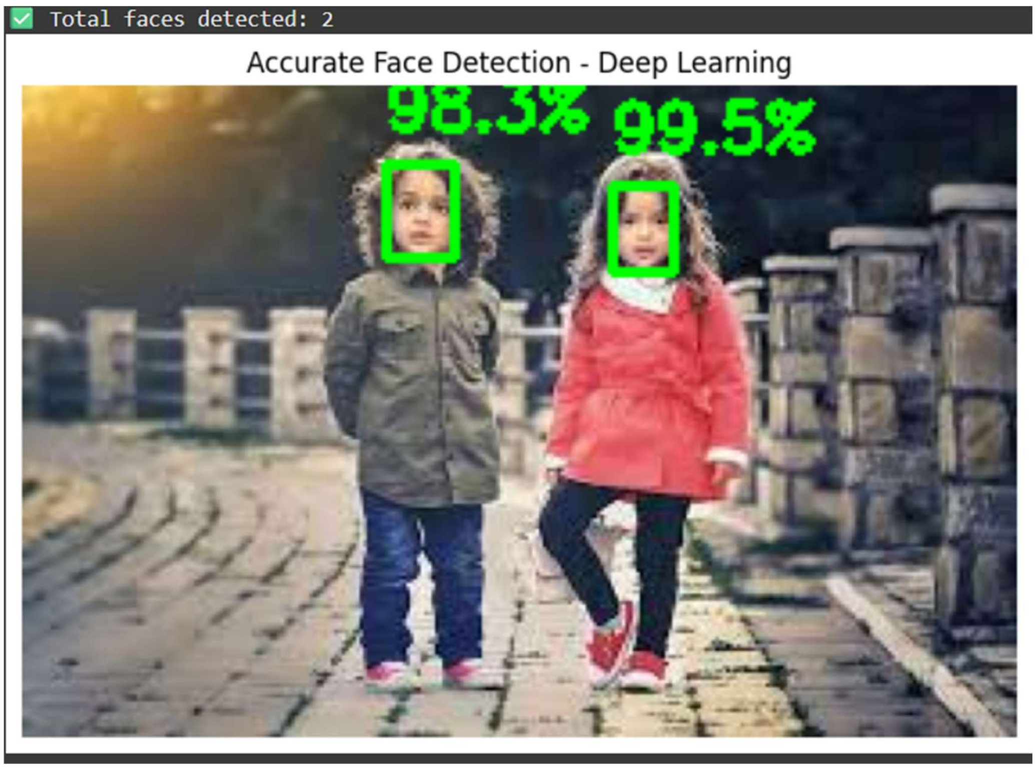
CODE:

```
!pip install opencv-python
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 1: Upload image
uploaded = files.upload()
image_name = list(uploaded.keys())[0]

# Step 2: Load Haar Cascade models (front + side profile)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
profile_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_profileface.xml')

# Step 3: Read image
img = cv2.imread(image_name)
```

```
if img is None:
    print("Error: Cannot read image!")
else:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.equalizeHist(gray)  # improve contrast

    # Step 4: Detect faces (tuned parameters)
    faces_front = face_cascade.detectMultiScale(
        gray,
        scaleFactor=1.02,     # detect more faces
        minNeighbors=2,       # reduce strict filtering
        minSize=(15, 15),     # detect small faces
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    # Step 5: Detect profile (side) faces
    faces_profile = profile_cascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=3,
        minSize=(15, 15),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    # Combine detections
    all_faces = list(faces_front) + list(faces_profile)
    print(f"Total Faces Detected: {len(all_faces)}")

    # Step 6: Draw rectangles
    for (x, y, w, h) in all_faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
# Step 7: Show output
plt.figure(figsize=(8, 8))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title("Improved Face Detection (Front + Side)")
plt.show()
```

OUTPUT:



RESULT:

Faces in the online human face image dataset were successfully detected. Bounding boxes accurately highlighted facial regions, demonstrating effective identification of faces under varying poses, expressions, and lighting conditions.