**Face Detection AI Project**

**Author: Vaka Srinu Vasa Reddy  
Institution: pace institute of technology & sciences  
Course: Python  
Submission Date: 2025**

**Project Overview**

**This project detects human faces in images or video streams using OpenCV, an open-source computer vision library. The system uses a pre-trained Haar Cascade Classifier to locate faces in real-time.**

**Objective**

**Detect human faces from images or live webcam feed.**

**Draw bounding boxes around detected faces.**

**(Optional) Extend the project for face recognition or mask detection.**

**Tools & Technologies**

|  |  |
| --- | --- |
| Component | Description |
| Python | Programming language used. |
| OpenCV | Library for image processing and vision. |
| Haar Cascade | Pre-trained model for face detection. |
| Matplotlib (opt) | For displaying images in notebooks. |

**Project Structure**

**face\_detection\_ai/**

**│**

**├── face\_detection.py # Main face detection script**

**├── sample\_image.jpg # Test image (optional)**

**├── README.md # Documentation**

**└── requirements.txt # Dependencies**

**1. Image-Based Detection**

**This script loads a single image, detects faces, and displays the result with bounding boxes.**

**File: face\_detection.py (Image Section)**

**Python**

**import cv2**

**from matplotlib import pyplot as plt**

**# Load pre-trained Haar Cascade model for face detection**

**face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')**

**# Read input image**

**img = cv2.imread('sample\_image.jpg')**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**# Detect faces:**

**# scaleFactor=1.3: compensates for faces that are closer or farther away**

**# minNeighbors=5: specifies how many neighbors each candidate rectangle should have to be considered a face**

**faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)**

**# Draw rectangle around each detected face**

**for (x, y, w, h) in faces:**

**cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2) # Color: Green (0, 255, 0), Thickness: 2**

**# Display the output (using Matplotlib for notebook/script compatibility)**

**img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**

**plt.imshow(img\_rgb)**

**plt.axis('off')**

**plt.title('Detected Faces')**

**plt.show()**

**2. Real-Time Webcam Detection**

**This script accesses the default webcam, processes frames in a loop, and displays the video feed with bounding boxes drawn around faces.**

#### Note: This code uses cv2.imshow() and requires a local environment with GUI support (it will not run directly in cloud notebooks like Google Colab).

**File: face\_detection.py (Webcam Section)**

**Python**

**import cv2**

**# Load pre-trained Haar Cascade model**

**face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')**

**# Initialize video capture (0 is typically the default webcam)**

**cap = cv2.VideoCapture(0)**

**while True:**

**# Read frame-by-frame**

**ret, frame = cap.read()**

**if not ret:**

**print("Error: Could not read frame from camera.")**

**break**

**# Preprocessing: Convert to grayscale for faster detection**

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

**# Detect faces**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 5)**

**# Draw bounding box**

**for (x, y, w, h) in faces:**

**cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)**

**# Display the resulting frame**

**cv2.imshow('Real-Time Face Detection', frame)**

**# Exit the loop on 'q' key press**

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

**break**

**# Release the capture and close all windows**

**cap.release()**

**cv2.destroyAllWindows()**

**How It Works**

The detection process is a simple pipeline:

1. **Capture Input:** Acquire the image or a continuous video stream frame.
2. **Preprocessing:** Convert the input from the standard BGR (Blue, Green, Red) color space to **grayscale**. This significantly reduces the computational load without losing the necessary features for face detection.
3. **Face Detection:** The **Haar Cascade Classifier** scans the grayscale image/frame at multiple scales. It uses simple rectangular features (Haar features) and a boosting algorithm to quickly determine if a region is likely a face.
4. **Visualization:** For every region identified as a face, the script draws a **bounding box** (a green rectangle in this case) on the original color image/frame.

**Possible Extensions**

| **Feature** | **Description** |
| --- | --- |
| **Face Recognition** | **Identify *known* faces by comparing the detected face to a database of people (e.g., using the face\_recognition library).** |
| **Emotion Detection** | **Integrate with deep learning models to classify and label the detected face's emotion (e.g., happy, sad, angry).** |
| **Mask Detection** | **Train a custom model to detect whether a person is correctly wearing a protective face mask.** |
| **Web Deployment** | **Build an interactive web application using frameworks like Streamlit or Flask to allow remote usage without a local environment setup.** |