University of the Punjab

Gujranwala Campus

Department of Information Technology

Project: Face Detection using Haar Cascade Classifier in OpenCV

Prepared by:

Sawera Shahid (BIT21202)
Aleeza Aftab (BIT21208)
Noor Ikram (BIT21230)
Sabiha Khan (BIT21233)
Ayesha Shehbaz (BIT21242)

Submitted to:

Miss Fouqia Zafeer

Table of Contents

Abstract

This project implements a face detection system using the Haar Cascade classifier provided by OpenCV. The system loads a pre-trained model to detect human faces in an image, draws bounding rectangles around detected regions, and displays the result. The documentation explains the code in detail and discusses the potential for further enhancements.

Acknowledgement

I would like to express my sincere gratitude to my respected teacher, Ms. Fouqia Zaheer, for her invaluable guidance, support, and encouragement throughout this project. Her expertise and insights have played a crucial role in shaping the direction of my work.

Introduction

Face detection is a fundamental task in computer vision with applications in surveillance, human-computer interaction, and biometric systems. One of the most efficient methods for detecting faces is using Haar Cascade classifiers, which have been widely adopted for their simplicity and performance. This project leverages OpenCV's built-in Haar Cascade for frontal face detection to process and analyse static images.

Objectives:

1. **Implement face detection:** Utilize OpenCV's Haar Cascade classifier to identify faces in images.
2. **Draw visual indicators:** Outline detected faces with rectangles for easy visual verification.
3. **Demonstrate fundamental techniques**: Show the process of loading models, image pre-processing, detection, and displaying results.
4. **Lay groundwork for further projects:** Provide a basis for more advanced face detection and recognition tasks.

Tools and Technologies

- **Programming Language:** Python
- **Library:** OpenCV (Open-Source Computer Vision Library)
- **Algorithm:** Haar Cascade Classifier for object detection
- **Development Environment:** Jupyter Notebook, PyCharm, or any other Python IDE

Methodology

1. Haar Cascade Overview

The Haar Cascade classifier is a machine learning-based approach where a cascade function is trained from many positive and negative images. OpenCV provides several pre-trained Haar Cascades for face detection that can be readily used in projects. The classifier works by scanning the image at multiple scales and positions to detect features that match the trained patterns.

2. Step-by-Step Code Explanation

a) Loading the Haar Cascade Classifier:

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

The pre-trained Haar Cascade XML file for frontal face detection is loaded from OpenCV’s data repository.

b) Reading the Input Image:

```
image = cv2.imread(image_path)
```

The image is read from the specified path. Ensure that the image path is valid.

c) Converting to Grayscale:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Since the Haar Cascade works best on grayscale images, the original image is converted from BGR to grayscale.

d) Detecting Faces:

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))
```

- **detectMultiScale** scans the image for features that resemble a face. Parameters:
- **scaleFactor:** Specifies how much the image size is reduced at each image scale.
- **minNeighbors:** Specifies how many neighbors each candidate rectangle should have to retain it.
- **minSize:** Minimum possible object size to detect.

e) Drawing Rectangles Around Detected Faces:

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

    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

For every detected face, a rectangle is drawn with a specified color (blue) and line thickness.

f) Displaying the Result:

```
cv2.imshow('Face Detection', image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

The resulting image with highlighted faces is displayed in a window. The window waits for a key press before closing.

g) Example Usage:

```
detect_faces('example.jpg')
```

The function call (presumably wrapped in a function named detect_faces) demonstrates how to apply the face detection on an example image.

Results and Discussion

- **Detection Accuracy:** The algorithm performs well on frontal face images; however, detection accuracy may vary with different lighting conditions or face orientations.
- **Performance:** Haar Cascades are computationally efficient and suitable for real-time detection in simple applications.
- **Limitations:** The algorithm may not perform as well on rotated or partially occluded faces. Future work might include exploring more robust methods such as deep learning-based detectors (e.g., using DNN modules in OpenCV).

Conclusion and Future Work

Conclusion

This project successfully demonstrates face detection using OpenCV's Haar Cascade classifier. The code is simple yet effective for detecting faces in static images and provides a foundation for more complex computer vision applications.

Future Work

1. **Video Stream Detection:** Extend the implementation to detect faces in live video feeds.
2. **Enhanced Preprocessing:** Incorporate techniques like histogram equalization to improve detection under varied lighting.
3. **Alternative Algorithms:** Experiment with deep learning-based methods for improved accuracy and robustness.
4. **Real-Time Performance:** Optimize and deploy the application for real-time face detection in practical applications.

References

1. OpenCV Documentation
2. Haar Cascade Classifier Overview
3. Face Detection using Haar Cascades

Appendix:

Full Code Listing

```
import cv2


def detect_faces(image_path):

    # Load the Haar cascade classifier for face detection

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')



    # Read the image from the given path

    image = cv2.imread(image_path)

    if image is None:

        print("Error: Image not found or unable to load.")

        return



    # Convert the image to grayscale

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)



    # Detect faces in the grayscale image

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))



    # Draw rectangles around detected faces

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

        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0),
2)
```

```
    # Display the result in a window
    cv2.imshow('Face Detection', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
detect_faces('example.jpg')
```