Mini Project Report on

Face Detection Using Python

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

ANIRUDH RANA **2118225**

Under the Mentorship of
Dr.Vikrant Sharma

Department of Computer Science and Engineering
GRAPHIC ERA HILL UNIVERSITY DEHRADUN

FEB, 2023

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the project report entitled **"Face Detection using Python"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the under the mentorship of **Dr Vikrant Sharma,** Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun.

Anirudh Rana University Roll No: 2118225

Table of Contents

Chapter No.	Description	Page No.
Chapter 1	Introduction	**1-3**
Chapter 2	Literature Survey	**4-5**
Chapter 3	Methodology	**6-7**
Chapter 4	Result and Discussion	**8**
Chapter 5	Conclusion and Future Work	**9**
	References	**10**

Problem Statement-Face detection using Python.

Chapter 1

Introduction

In the following sections, a brief introduction and the problem statement for the work has been included.

1.1 Introduction

Face detection is a computer technology that is used to identify or verify a person from a digital image or video frame. One popular technique for face detection is using a deep learning model called a Multi-task Cascaded Convolutional Networks (MTCNN). MTCNN is trained to detect faces in an image and also to identify facial landmarks such as the eyes, nose, and mouth.

In Python, there are several libraries available for face detection such as OpenCV, dlib, and MTCNN. These libraries provide pre-trained models that can be easily integrated into a Python script. For example, using OpenCV, you can detect faces in an image using the Haar Cascade classifier, which is a machine learning based algorithm for object detection. Similarly, using dlib you can use the pre-trained shape predictor model to detect facial landmarks.

The overall process of face detection using Python typically involves the following steps:

1. Importing necessary libraries
2. Loading an image or video frame
3. Pre-processing the image or video frame (resizing, grayscale, etc.)
4. Feeding the image or video frame into the face detection model
5. Post-processing the model's output (bounding box, facial landmarks, etc.)
6. Displaying the result on the screen.

The above Python script uses several tools to detect faces in an image. These include:

1.1.1 OpenCV library: OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. It has a wide range of functionality, including image processing, video capture and analysis, object detection, and machine learning. The script uses OpenCV to read and display the image, convert the image to grayscale, and draw rectangles around the detected faces.

1.1.2 Haar cascade classifier: A Haar cascade classifier is a machine learning-based algorithm for object detection. The script uses a pre-trained Haar cascade classifier to detect faces in the image. The classifier is trained on thousands of images of faces and non-faces and it learns to recognize the patterns that are commonly found in faces.

1.1.3 imread() and imshow() functions: These are OpenCV functions that are used to read and display the image, respectively. The imread() function reads an image from a file and returns an array of pixels representing the image. The imshow() function displays the image on the screen.

1.1.4 cvtColor() function: This is an OpenCV function that is used to convert an image from one color space to another. In this script, it is used to convert the image from RGB color space to grayscale.

1.1.5 detectMultiScale() method: This is a method of the Haar cascade classifier that is used to detect objects in the image. The method takes in the grayscale image and several parameters such as scaleFactor and minNeighbors. The scaleFactor parameter controls the image scale at which the classifier is applied and minNeighbors parameter controls the number of rectangles to group together to detect a face.

1.1.6 rectangle() function: This is an OpenCV function that is used to draw a rectangle on an image. The function takes in the image, the coordinates of the top-left corner of the rectangle,

the width and height of the rectangle, the color of the rectangle, and the thickness of the rectangle.

All of the above tools are available as part of the OpenCV library, which can be installed via pip or other package managers.

Chapter 2

Literature Survey

2.1 History

The history of face detection dates back to the 1960s, with the earliest research in the field being focused on developing methods to automatically identify human faces in photographs. One of the first published papers on the topic was "The Use of the Information in Picture Signals" by Woodrow W. Bledsoe in 1964, which proposed a system that used edge detection and template matching to identify faces in images.

In the 1970s and 1980s, researchers began to explore other approaches for face detection, such as using eigenfaces, a method developed by Matthew Turk and Alex Pentland in 1991. Eigenfaces is a technique that uses principal component analysis (PCA) to represent faces as a set of eigenvectors, which can then be used to identify faces in images. This method became popular and was used in a number of face recognition systems.

In the 1990s, Viola and Jones proposed a new method for face detection, which is based on the concept of "Haar-like features". This method uses simple features, such as edges and lines, to represent a face and a machine learning-based algorithm to detect faces in images. This method, known as the Viola-Jones algorithm, became widely used in face detection systems and is still used today.

In recent years, with the rise of deep learning, new techniques for face detection have been developed. The Multi-task Cascaded Convolutional Networks (MTCNN) is an example of deep learning-based face detection method that has been widely used and showed good performance. The deep learning models such as MTCNN are trained on large datasets of images and can detect faces in images and videos with high accuracy.

2.2 Research and Articles

"**Real-time face detection using OpenCV and Python**" by Adriana Romero, published in the Journal of Physics: Conference Series in 2016. This article describes a method for real-time face detection using the OpenCV library and Python programming language. The method uses a Haar cascade classifier to detect faces in video frames and also gives an idea of how to optimize the detection process.

"**Deep Learning-Based Object Detection: A Comprehensive Review**" by S. Ali and R. Jafri, published in the Journal of Information and Communication Technology in 2019. This article provides a comprehensive review of object detection techniques based on deep learning, including face detection. It covers popular deep learning architectures such as R-CNN, Fast R-CNN, Faster R-CNN, and RetinaNet, and discusses their pros and cons for face detection.

"**A Review of Face Detection Techniques**" by M. A. Raza, S. A. Raza, and A. Shafait, published in the International Journal of Computer Applications in 2012. This article provides an overview of various face detection techniques, including traditional methods such as Haar cascades and PCA, as well as more recent methods such as deep learning-based models. The article also provides a comparison of the accuracy and performance of these methods.

All of these articles demonstrate that face detection using Python is a well-researched topic and there are many techniques available for detecting faces in images and videos, ranging from traditional methods like Haar cascades to more recent deep learning-based methods like MTCNN. The choice of method depends on the specific requirements of the application and the available resources.

Chapter 3

Methodology

3.1 Explanation

Here I have described the procedure that I used to create this chat application.

3.1.1 Loading the necessary libraries and dependencies: The first step is to import the necessary libraries and dependencies such as OpenCV or dlib, and any other required libraries such as numpy, pandas, etc.

3.1.2 Loading the image or video: The next step is to load the image or video that you want to process. The image or video can be loaded using OpenCV's imread() function or other appropriate functions depending on the library used.

3.1.3 Pre-processing the image or video: Once the image or video is loaded, it is typically pre-processed to prepare it for face detection. This may involve resizing the image, converting it to grayscale, or performing other types of image processing.

3.1.4 Loading the face detection model: Next, the face detection model is loaded. This can be done using pre-trained models available in the libraries or using a custom trained model

3.1.5 Running the face detection model on the image or video: The pre-processed image or video is then passed through the face detection model, which will detect the faces in the image or video.

3.1.6 Post-processing the model's output: The output of the face detection model is typically a list of bounding boxes, which indicate the location of the faces in the image or

video. Additional post-processing steps may be required to refine the detection results, such as removing false positives, or finding facial landmarks.

3.1.7 Displaying the results: The final step is to display the results of the face detection process on the screen. This can be done using OpenCV's imshow() function or other appropriate functions depending on the library used.

It is worth noting that the above steps may vary slightly depending on the specific library and face detection method used.

Chapter 4

Result and Discussion

When using OpenCV and a Haar cascade classifier for face detection in Python, the results can be highly dependent on the quality of the input image and the parameters used for detection. The Haar cascade classifier is a machine learning-based algorithm that is trained to recognize patterns commonly found in faces. It uses simple features such as edges and lines to represent a face and detect it in an image.

One of the main advantages of using a Haar cascade classifier is that it is relatively fast and can be used for real-time face detection. It can also be easily integrated into a Python script using the OpenCV library.

One of the main limitations of the Haar cascade classifier is that it is based on simple features and can only detect faces that are similar to the ones it was trained on.

In terms of results, the Haar cascade classifier typically returns a list of bounding boxes that indicate the location of the detected faces in the image. It can also be used to detect facial landmarks such as the eyes, nose, and mouth. The results can be further improved by using post-processing techniques such as removing false positives or finding facial landmarks.

Chapter 5

Conclusion and Future Work

In conclusion, face detection is an active area of research and development, and there are several techniques available for detecting faces in images and videos using Python. The choice of technique depends on the specific requirements of the application and the available resources.

Traditional methods such as Haar cascades, which are based on simple features, are relatively fast and can be used for real-time face detection, but they may not always provide the best results. On the other hand, deep learning-based methods such as Multi-task Cascaded Convolutional Networks (MTCNN) have been shown to be more accurate and robust in detecting faces in images and videos with different poses, lighting, and facial expressions.

In terms of future work, there are several directions that face detection research could take. One area of focus could be to develop more accurate and robust methods for face detection, especially for images and videos with poor lighting, low resolution, or non-frontal poses. Another area of focus could be to improve the speed of face detection methods, so that they can be used in real-time applications.

Overall, face detection using Python is an active area of research and development, and there are many opportunities for future work to improve the accuracy and speed of face detection methods and make them more suitable for different types of applications.

References

"**Real-time Face Recognition: An End-to-End Project**" by Adrian Rosebrock on PyimageSearch

"**Face Detection using Python and OpenCV**" by Harisson on Real Python (https://realpython.com/face-detection-with-python/)

"**A beginner's guide to training your own object detector with OpenCV**" by Adrian Rosebrock on PyimageSearch

"**Face Detection with Python using OpenCV**" by Venkatesh Potluri on Medium.

"**Face Detection with OpenCV and Deep Learning**" by Rajeev Ratan on LearnOpenCV (https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/)