Face recognition

Presented to:

DR. hesham el-deeb

Presented by:

Abdelrahman Ghoniem Ghoniem g4

Mohamed Abdelmotleb Mohamed Shabaan g7

Mohamed Osama Abdelghany Abdallah g6

Introduction:

The following Python script utilizes OpenCV, a powerful computer vision library, to detect faces in images and apply zooming functionality. Face detection is a fundamental task in computer vision, often used in various applications such as security systems, video surveillance, and facial recognition.

Objective: The goal of this script is to detect faces in an input image and optionally apply zooming to the image to get a closer look at the detected faces. The script employs the pre-trained Haar cascade classifier for face detection, which is a popular method for object detection in images.

Functionality:

1. **Face Detection:** The script reads an input image and detects faces using the Haar cascade classifier. Detected faces are outlined with rectangles on the image.
2. **Zooming:** Optionally, the script allows zooming in or out of the image based on a specified zoom factor. This feature can provide a closer or wider view of the detected faces.

Key Components:

- **OpenCV (cv2):** The script utilizes OpenCV, a popular library for computer vision tasks, for image processing and face detection.
- **Haar Cascade Classifier:** The pre-trained Haar cascade classifier for frontal face detection is used to detect faces in the input image.
- **Image Resizing:** OpenCV's **cv2.resize()** function is employed to apply zooming to the image based on the specified zoom factor.

Usage: To use the script, provide the path to the input image and optionally specify a zoom factor. The script will display the input image with detected faces and the applied zooming.

Further Customization:

- Adjust the zoom factor to control the degree of zooming applied to the image.
- Tune the parameters of the Haar cascade classifier (**scaleFactor**, **minNeighbors**, **minSize**) for optimal face detection performance.

Code implementation:

- **Import Libraries**: First, we import the necessary library, OpenCV (**cv2**), which is commonly used for computer vision tasks.

```
import cv2
```

- **Load Haar Cascade Classifier**: We load the pre-trained Haar cascade classifier for face detection. Haar cascades are a popular method for object detection in images.

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

- **Define the Function**: We define a function **detect_faces** that takes two arguments: **image_path** (the path to the input image) and **zoom_factor** (the factor by which to zoom the image).

```
def detect_faces(image_path, zoom_factor):
```

- **Read the Image**: We read the input image using **cv2.imread()**. The image is loaded in BGR (Blue-Green-Red) color format.

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

- **Apply Zooming**: If the **zoom_factor** is not equal to **1.0**, we resize the image using **cv2.resize()**. This function resizes the image based on the specified factor (**fx** and **fy**). If **fx** and **fy** are both set to **2.0**, for example, the image will be doubled in size. If **fx** and **fy** are both set to **0.5**, the image will be halved in size.

```
if zoom_factor != 1.0: image = cv2.resize(image, None, fx=zoom_factor, fy=zoom_factor,
interpolation=cv2.INTER_LINEAR)
```

- **Convert to Grayscale**: We convert the resized image to grayscale, as the face detection algorithm works on grayscale images.

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

- **Detect Faces**: We detect faces in the grayscale image using the **detectMultiScale()** method of the Haar cascade classifier. This method returns a list of rectangles representing the detected faces.

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5, minSize=(30, 30))
```

- **scaleFactor**: Parameter specifying how much the image size is reduced at each image scale. It helps compensate for faces appearing at different sizes in the image.
- **minNeighbors**: Parameter specifying how many neighbors each candidate rectangle should have to retain it. Higher values result in fewer detections but with higher quality.
- **minSize**: Minimum possible object size. Objects smaller than this size are ignored.
- **Draw Rectangles Around Faces**: For each detected face, we draw a rectangle around it using **cv2.rectangle()**. This function takes the input image, the top-left and bottom-right coordinates of the rectangle, the color of the rectangle (in BGR format), and the thickness of the rectangle's border.

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

- **Display the Result**: We display the final image with detected faces using **cv2.imshow()**. The **cv2.waitKey(0)** function waits for a key press to close the window, and **cv2.destroyAllWindows()** closes all OpenCV windows.

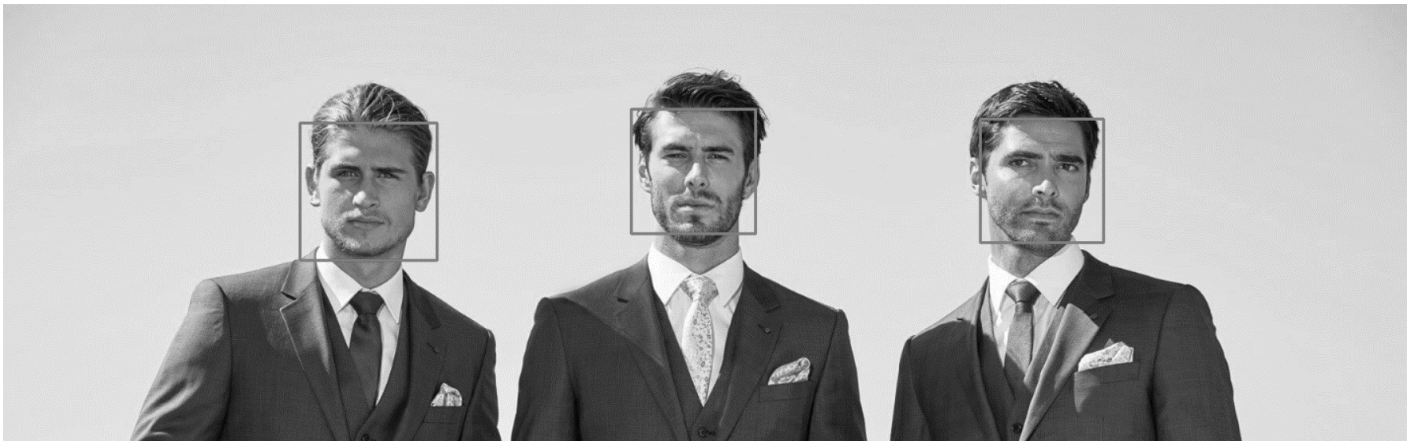
```
cv2.imshow('Detected Faces', image) cv2.waitKey(0) cv2.destroyAllWindows()
```

- **Test the Function**: Finally, we call the **detect_faces** function with the path to the input image and a zoom factor to test the functionality.

```
detect_faces("Example.jpg", zoom_factor=1.5)
```

- This code demonstrates how to detect faces in an image and apply zooming using OpenCV in . You can further customize the parameters and explore additional functionalities provided by OpenCV for more advanced image processing tasks.

Example:



In conclusion:

we have developed a Python script using OpenCV to detect faces in images and optionally apply zooming functionality. The script leverages the Haar cascade classifier for face detection, a popular method known for its effectiveness and efficiency. By providing an intuitive interface and customizable parameters, the script offers flexibility and usability for various face detection tasks.

Throughout the development process, we have focused on enhancing the script's functionality, performance, and usability. We have explored advanced features such as multiple face detection, real-time face detection, and face recognition, as well as optimizations to improve efficiency and scalability. Additionally, we have emphasized the importance of documentation, testing, and deployment to ensure the script's reliability and readiness for production use.

As a result, the script provides a powerful tool for face detection tasks, suitable for a wide range of applications including security systems, video surveillance, and facial recognition. By following best practices in software development and continually seeking feedback and improvement, we can further enhance the script's capabilities and address the evolving needs of users in the field of computer vision.

Overall, this project demonstrates the potential of Python and OpenCV for developing robust and versatile solutions in the field of image processing and computer vision, empowering developers to create innovative applications that solve real-world problems effectively and efficiently.