FACE DETECTION USING PIPELINE

PROJECTREPORT

Submitted by

TINA KASHYAP (RA2211028010226)

DHRUVI SRISHWAL (RA2211028010235)

SHIVAM SINGH (RA2211029010001)

SATVIK SHARMA (RA2211029010003)

Under the guidance of

Dr S. Ramesh

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

with specialization in Cloud Computing, Computer Networking.

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

NOVEMBER 2023

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **"21CSS201J COMPUTER ORGANIZATION AND ARCHITECTURE"** entitled in "**FACE DETECTION USINGPIPELINE**" is the bonafide work done by:

TINA KASHYAP	**(RA2211028010226)**
DHRUVI SRISHWAL	**(RA2211028010235)**
SHIVAM SINGH	**(RA2211029010001)**
SATVIK SHARMA	**(RA2211029010003)**

who completed the project under my supervision.

SIGNATURE	**SIGNATURE**
Dr S. Ramesh	Dr Annapurani Panaiyappan
Assistant Professor	**Head of the Department**
Department of Networking and Communications	Department of Networking and Communications
SRMIST	SRMIST

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

SCHOOL OF COMPUTING

College of Engineering and Technology

SRM Institute of Science and Technology

Mini Project Report

Odd Semester, 2023-2024

Subject Code & Name	: 21CSS201T & Computer Organization and Architecture
Year & Semester	: II & III
Project Title	: Face Detection using pipeline.
Lab Supervisor	**: Dr S. Ramesh**
Team Members	: 1. TINA KASHYAP (RA2211028010226)
	2. DHRUVI SRISHWAL (RA2211028010235)
	3. SHIVAM SINGH (RA2211029010001)
	4. SATVIK SHARMA (RA2211029010003)

Particulars	**Max. Marks**	**Marks Obtained**
		Names: Tina Kashyap, Dhruvi Srishwal, Shivam Singh, Satvik Sharma
		Register No.'s: RA2211028010226, RA2211028010235, RA2211029010001 and RA2211029010003
Program and Execution	20	
Demo Verification & Viva	15	
Project Report	05	
Total	**40**	

Date	**:15.11.2023**
Faculty Name	**: Dr S. Ramesh**
Signature	**:**

TABLE OF CONTENTS

ABSTRACT

This project presents a face detection system implemented as a pipeline, taking inspiration from computer architecture principles. The pipeline comprises five stages to efficiently process images and identify faces within them.

In the first stage, the system loads an image from a specified file path using the OpenCV library. The second stage involves converting the loaded image into grayscale, which simplifies subsequent processing.

The heart of the system is the third stage, where face detection is performed using Haar cascades, a popular computer vision technique. The pipeline leverages pre-trained models to identify faces efficiently.

In the fourth stage, identified faces are outlined in rectangles and overlaid on the original image. This provides a visual representation of the detected faces.

Finally, in the fifth stage, the result is displayed to the user in a graphical window. The pipeline's modularity allows for easy testing and adjustment of each stage independently.

This project showcases how pipelining, a fundamental concept in computer architecture, can be applied to image processing and face detection, resulting in an efficient and scalable system for real- world applications.

INTRODUCTION

In the rapidly evolving field of computer vision, the task of face detection plays a pivotal role in numerous applications, ranging from security and surveillance to facial recognition and augmented reality. The quest for real-time and efficient face detection has driven the development of innovative techniques and methodologies.

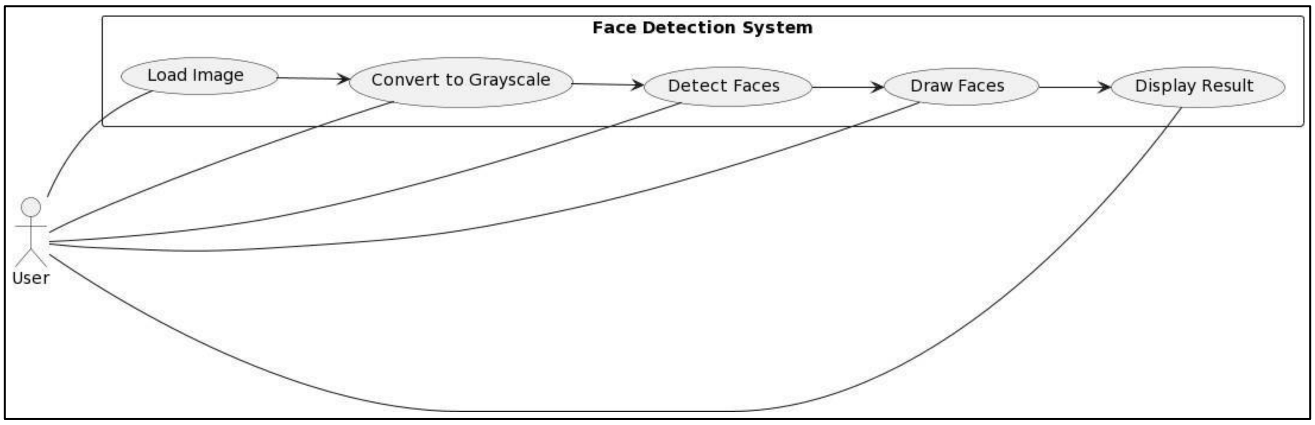
This project introduces a novel approach to face detection, inspired by the principles of computer architecture, where pipelining is a fundamental concept. Pipelining is widely employed in computer hardware to break down complex tasks into a series of interconnected stages, each handling a specific aspect of the task. This not only enhances efficiency but also enables parallel processing, making it an ideal concept to apply to the world of software-based image processing.

The project's objective is to create a robust and efficient face detection system using a pipelining approach. This approach breaks down the face detection process into multiple sequential stages, each dedicated to a specific operation. These stages include image loading, grayscale conversion, face detection using Haar cascades, face outlining, and result display.

By implementing a pipeline for face detection, we aim to improve the system's performance and flexibility. This design allows for the independent optimization of each stage, enabling future enhancements and adaptability to various use cases. Additionally, this project will provide insights into how computer architecture principles can be applied to computer vision tasks, promoting efficiency and real-time processing in applications that rely on accurate face detection.

DIAGRAM

A) Use Case



This diagram represents the following use case interactions:

User Interaction:

The diagram starts with the User entity, representing the user interacting with the face detection system. The primary interaction is with the FaceDetectionPipeline, indicating that the user triggers the face detection process through this central component.

Face Detection Pipeline:

The FaceDetectionPipeline class encapsulates the entire face detection process and orchestrates the interactions between various stages.

Stages of Face Detection (Use Cases):

The use cases are represented as classes, and each corresponds to a stage in the face detection pipeline.
ImageLoader: Responsible for loading images from a specified file path.
GrayscaleConverter: Converts the loaded image to grayscale, preparing it for face detection.
FaceDetector: Applies Haar cascades to detect faces in the grayscale image.
FaceOutliner: Outlines the detected faces on the original image for visual representation.
ResultDisplay: Displays the final image with outlined faces in a graphical window for user interaction.

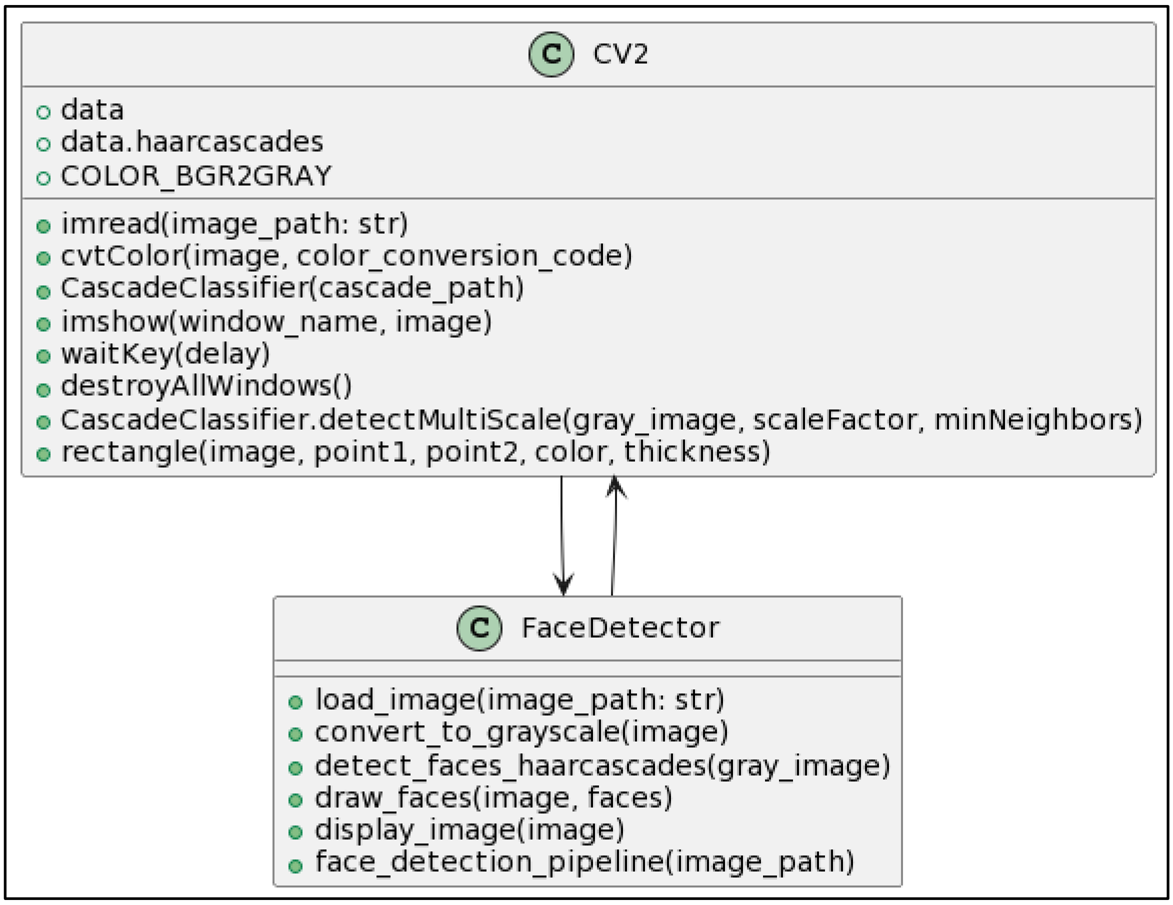
User-Triggered Use Cases:

The diagram illustrates that the user initiates the face detection process by invoking specific use cases within the FaceDetectionPipeline.
The sequence of use cases mirrors the stages outlined in the methodology.

Interactions between Stages:

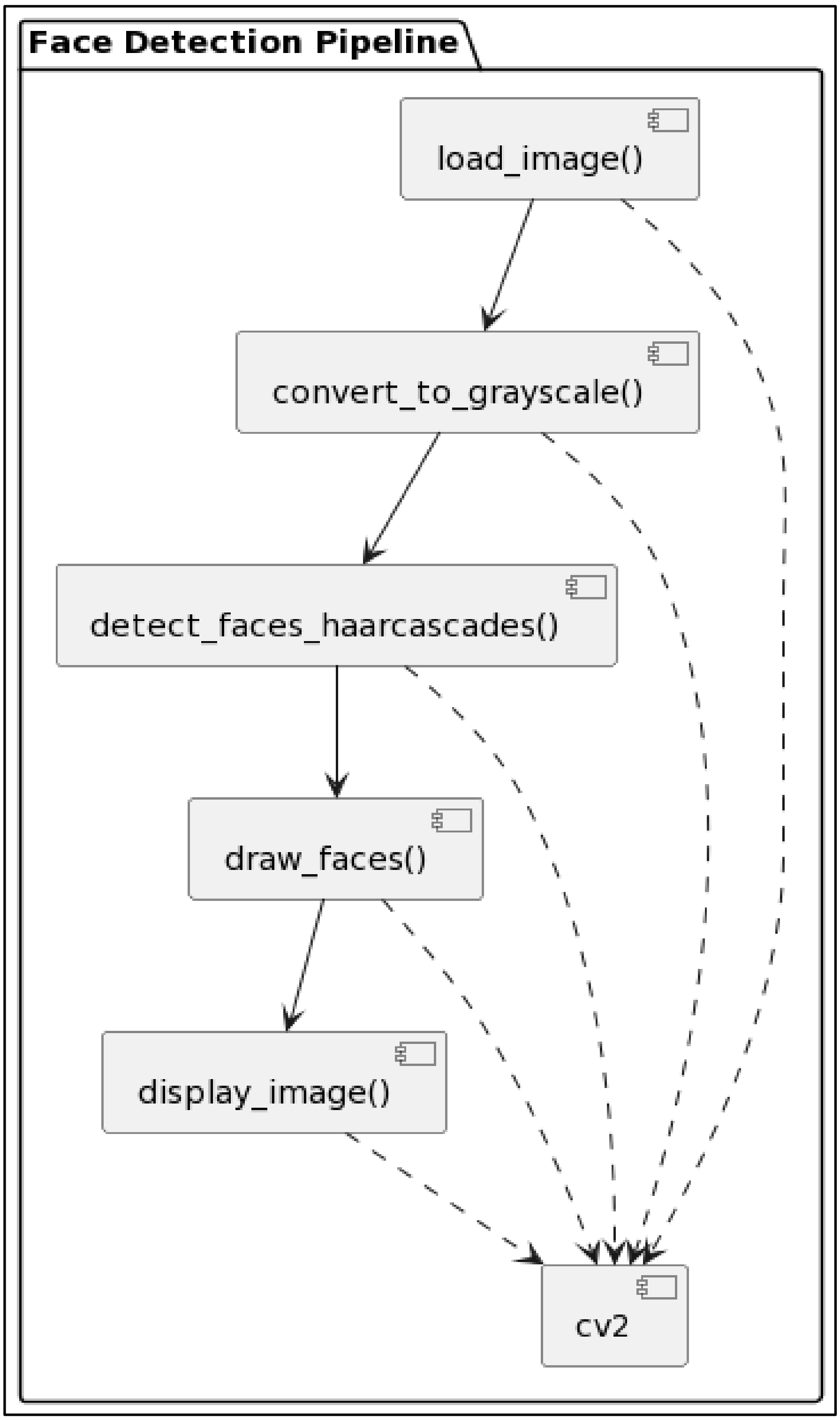
The arrows indicate the flow of interactions between the different stages of the face detection pipeline. Each stage interacts with the next stage in the pipeline, and the entire process is orchestrated by the FaceDetectionPipeline class.

B) Class Diagram


C CV2
data
data.haarcascades
COLOR_BGR2GRAY
imread(image_path: str)
cvtColor(image, color_conversion_code)
CascadeClassifier(cascade_path)
imshow(window_name, image)
waitKey(delay)
destroyAllWindows()
CascadeClassifier.detectMultiScale(gray_image, scaleFactor, minNeighbors)
rectangle(image, point1, point2, color, thickness)
C FaceDetector
load_image(image_path: str)
convert_to_grayscale(image)
detect_faces_haarcascades(gray_image)
draw_faces(image, faces)
display_image(image)
face_detection_pipeline(image_path)

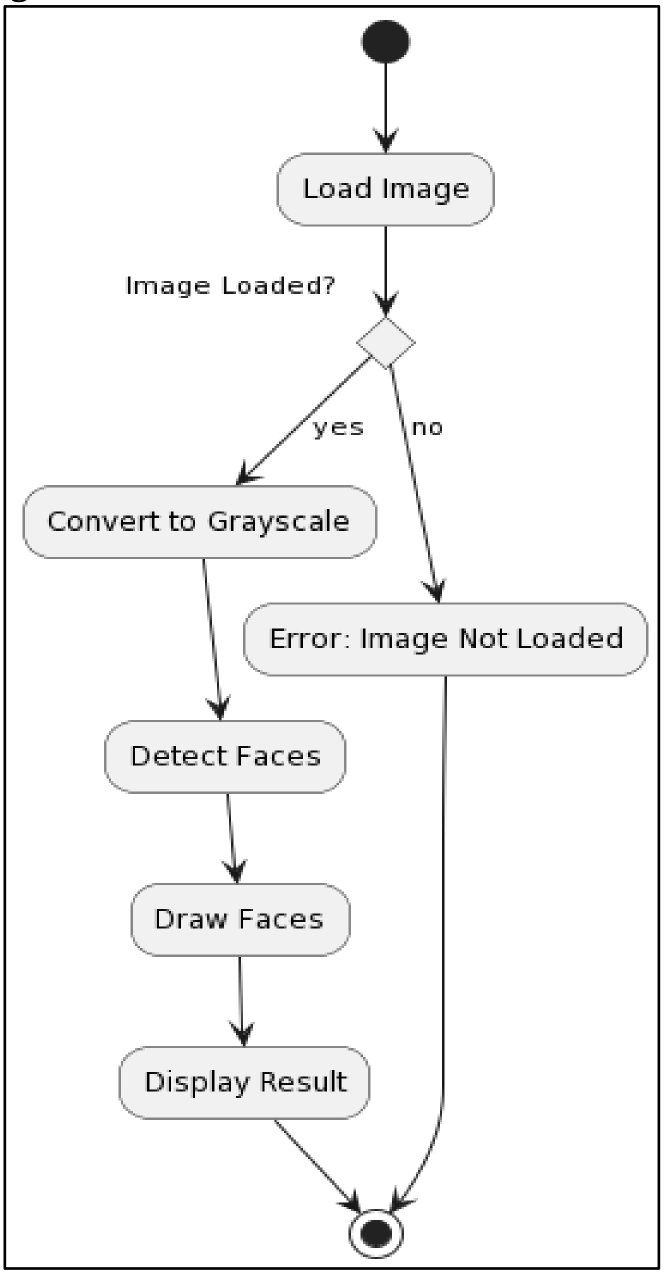
This diagram illustrates two classes: CV2 and FaceDetector. The CV2 class represents the OpenCV library with various methods like imread, cvtColor, CascadeClassifier, etc. The FaceDetector class contains methods that use the CV2 class to perform a face detection pipeline (load_image, convert_to_grayscale, detect_faces_haarcascades, draw_faces, display_image, face_detection_pipeline). The FaceDetector class interacts with the methods provided by the CV2 class for image processing and face detection.

C) Component Diagram



This diagram represents the components involved in the face detection pipeline and their interactions. It also shows the data flow from the "Image File" to the "LoadImage" component. The lines indicate the flow of data from the "Image File" to the "LoadImage" component within the face detection pipeline. The interactions involve the user triggering the face detection process, invoking the "Load Image" use case, and the subsequent flow of data through the image loading stage.

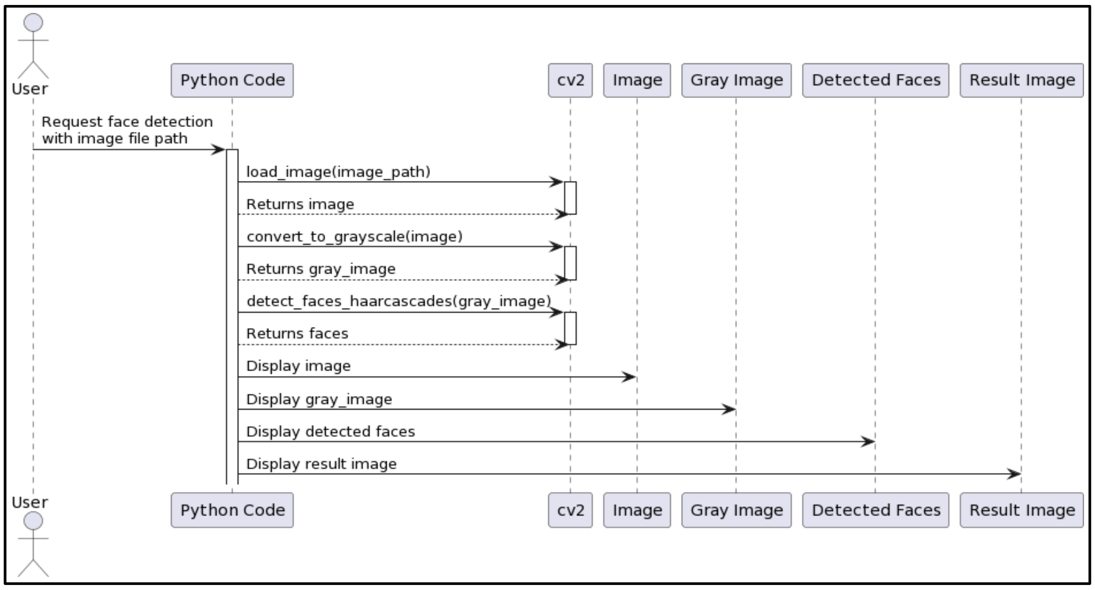
D) Activity Diagram



This activity diagram illustrates the sequence of actions performed in the face_detection_pipeline function for the given Python code.

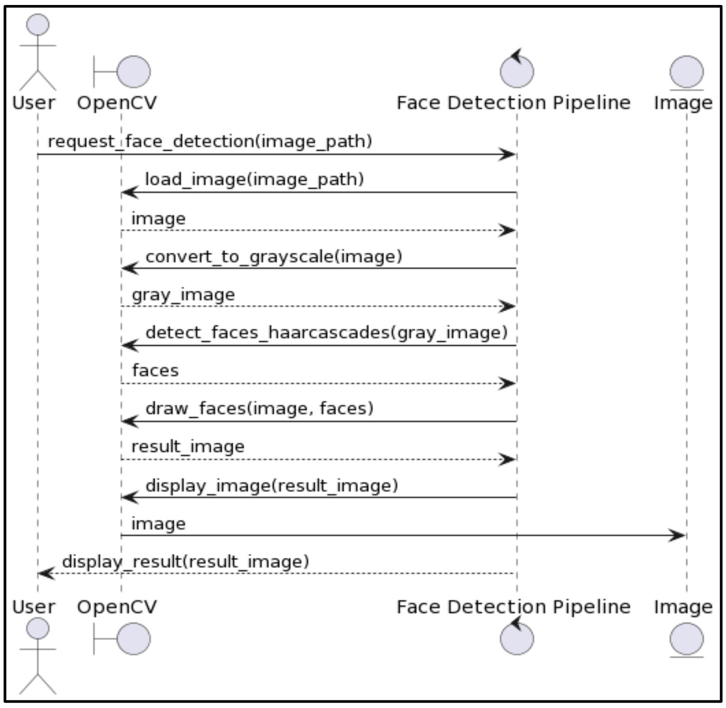
- "Start" is the initial node.
- "Load Image," "Convert to Grayscale," "Detect Faces using Haarcascades," "Draw Faces," and "Display Result" are activities performed in the pipeline.
- The arrows represent the flow of control from one activity to another.
- The labels on the arrows describe the functions being called or the data being passed between the activities.
- "End" is the final node.

E) Sequence Diagram



This sequence diagram illustrates the interactions between the user, the Python code, and various components within the code, such as the OpenCV library, different image representations, and the result. The diagram helps visualize the flow of operations in our face detection pipeline.

F) Collaboration Diagram


User
OpenCV
Face Detection Pipeline
Image
request_face_detection(image_path)
load_image(image_path)
image
convert_to_grayscale(image)
gray_image
detect_faces_haarcascades(gray_image)
faces
draw_faces(image, faces)
result_image
display_image(result_image)
image
display_result(result_image)

This collaboration diagram shows how different components and entities interact in the process of face detection. The user (U) initiates the process by requesting face detection with a specified image path. The Face Detection Pipeline (FDP) coordinates the interaction between the user, the OpenCV library (CV), and the image (Img) to perform the face detection. The various stages and data flow are indicated with arrows, showing how the image is processed and results are displayed to the user.

HARDWARE AND SOFTWARE REQUIREMENTS:

Software Requirements:

Python: Ensure that Python is installed on your system. Download it from the official Python website.

OpenCV: Install the OpenCV library using the following terminal or command prompt command.

Hardware Requirements:

Sufficient Processing Power: Your computer should have ample processing power to handle image processing tasks. A modern computer is recommended for optimal performance.

Storage Space: Ensure you have enough storage space to store and process images.

Input Device: A device capable of running Python scripts, such as a computer or a compatible device.

NOTE: Make sure to replace the image path in the code with the actual path to the image you want to use for face detection. Additionally, confirm that the specified image file exists at the provided path.

CONCEPTS AND WORKING PRINCIPLE:

The concept and working principle of the Face Detection Pipeline project involve breaking down the face detection process into several sequential stages, akin to a pipeline. Each stage focuses on a specific operation, allowing for efficient, modular, and parallel processing of images for face detection. Here's a detailed explanation of the concept and working principles:

- **Image Loading (Stage 1):**

In the first stage, the project loads an image from a specified file path using the OpenCV library. This stage serves as the entry point for image processing within the pipeline.

- **Grayscale Conversion (Stage 2):**

The second stage involves converting the loaded image into grayscale. Grayscale images have a single channel, simplifying subsequent processing steps. This step reduces the computational load and enhances the accuracy of face detection.

- **Face Detection using Haar Cascades (Stage 3):**

The heart of the pipeline is the third stage, where face detection is performed using Haar cascades. Haar cascades are a type of object detection algorithm that leverages pre-trained models to identify faces efficiently. The grayscale image is analyzed for patterns corresponding to human faces.

- **Face Outlining (Stage 4):**

In the fourth stage, identified faces are outlined with rectangles and overlaid on the original image. This step provides a visual representation of the detected faces, making it easier for users tounderstand the results.

- **Result Display (Stage 5):**

The fifth and final stage of the pipeline is responsible for displaying the result to the user. The image with outlined faces is shown in a graphical window, enabling users to see the detected faces within the context of the original image.

WORKING PRINCIPLE:

The working principle of the pipeline is to process images in a sequential and modular fashion. Each stage operates independently, allowing for easy testing and optimization of individual components. By using this pipeline approach, the system can achieve greater efficiency and scalability. Additionally, it provides a clear structure for the face detection process, making it adaptable for real- time applications and future enhancements.

This project showcases how the concept of pipelining, derived from computer architecture, can be applied to image processing, enabling efficient and accurate face detection. It highlights the advantages of breaking down complex tasks into simpler, interconnected stages, ultimately contributing to the advancement of computer vision and object detection technologies.

APPROACH, METHODOLOGY AND PROGRAMS:

Approach:

The approach for this project is to implement a pipelined system for face detection, inspired by the principles of computer architecture. The primary goal is to create an efficient, real-time face detection system with a modular and parallel processing structure. Here's the approach in more detail:

Modular Pipeline Structure:

The project adopts a modular approach, breaking down the face detection process into distinct stages. Each stage is responsible for a specific operation, such as image loading, grayscale conversion, face detection, face outlining, and result display. This modularity enhances flexibility and simplifies the development and optimization of individual components.

Image Processing Stages:

The pipeline begins by loading an image from a specified file path. This image is then converted to grayscale, simplifying the processing and reducing computational complexity. The grayscale image is subsequently analyzed for the presence of faces using Haar cascades, which are pre-trainedmodels. Once faces are detected, they are outlined on the original image for visual representation.

Parallel Processing:

The pipelined structure enables parallel processing of images, making the system more efficient and suitable for real-time applications. While one image is undergoing face detection, the next image can be loaded and processed in parallel, reducing processing time.
Flexibility and Scalability:

The modular design of the pipeline allows for easy testing and optimization of each stage independently. This approach enhances adaptability to different use cases and future improvements. New face detection algorithms or enhancements can be incorporated into the pipeline without significant changes to the overall structure.

Methodology:

The methodology of the Face Detection Pipeline project involves the step-by-step implementation of the approach described above. Here's a breakdown of the methodology:

Image Loading (Stage 1):
Utilize the OpenCV library to load images from a specified file path.

Grayscale Conversion (Stage 2):
Convert the loaded image to grayscale, reducing complexity and preparing it for face detection.

Face Detection using Haar Cascades (Stage 3):
Apply Haar cascades, a pre-trained face detection algorithm, to the grayscale image. Detect faces by identifying patterns corresponding to human facial features.

Face Outlining (Stage 4):
After detecting faces, outline them with rectangles on the original image for visual representation.

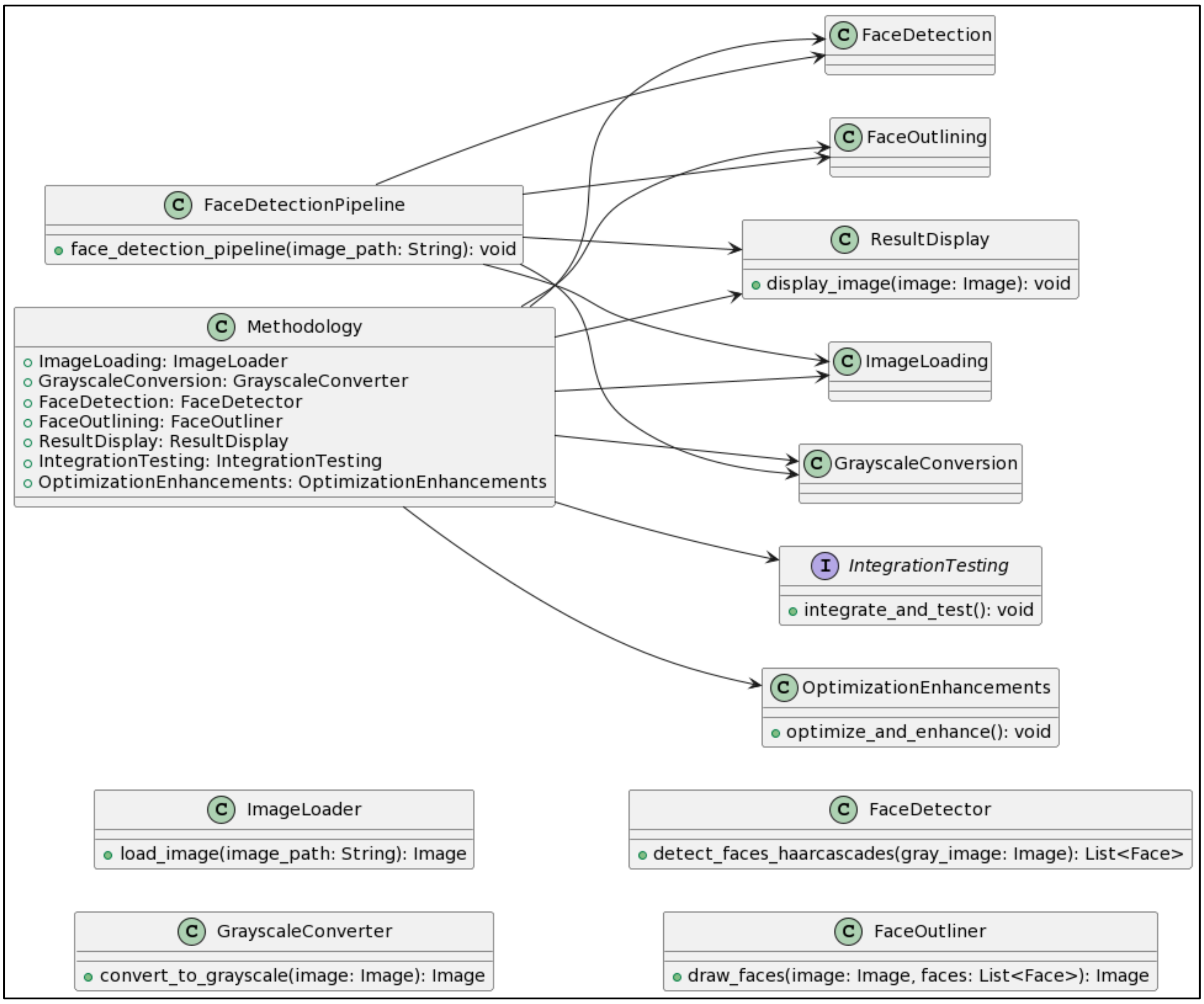
Result Display (Stage 5):
Display the final image with outlined faces in a graphical window for user interaction.

Integration and Testing:
Integrate all stages of the pipeline, ensuring they work seamlessly together. Test the system with various images to evaluate its performance and accuracy.

Optimization and Further Enhancements:
Fine-tune each stage of the pipeline for improved efficiency and accuracy. Consider integrating advanced face detection techniques or exploring parallel processing options for real-timeapplications.

The methodology emphasizes the development, integration, and testing of the individual stages within the pipeline, culminating in a functional and efficient face detection system. Additionally, it encourages continuous improvement and adaptation to meet evolving requirements and challenges in the field of computer vision.

BLOCK DIAGRAM:



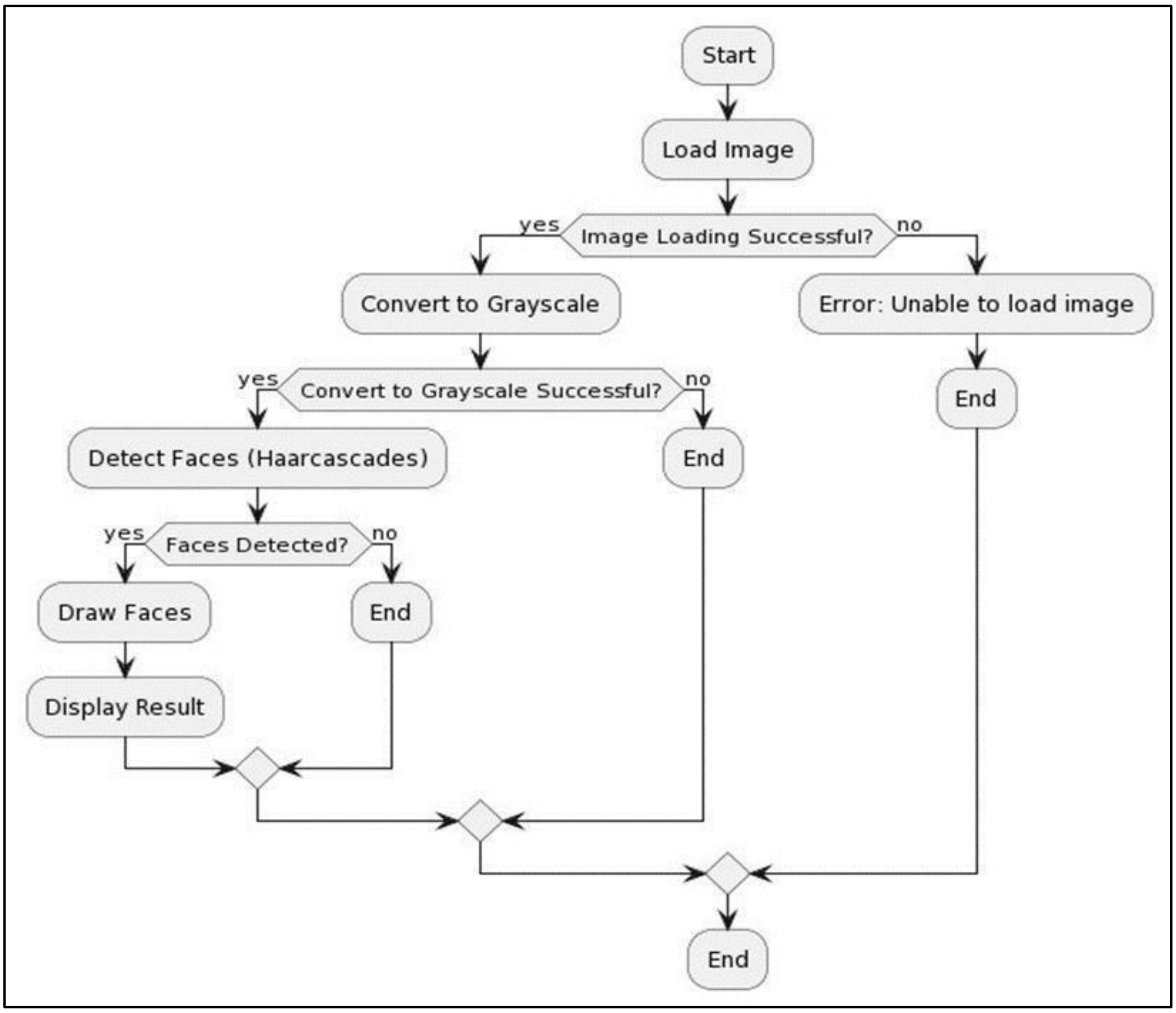
FaceDetection
FaceOutlining
FaceDetectionPipeline
face_detection_pipeline(image_path: String): void
ResultDisplay
display_image(image: Image): void
Methodology
ImageLoading: ImageLoader
GrayscaleConversion: GrayscaleConverter
FaceDetection: FaceDetector
FaceOutlining: FaceOutliner
ResultDisplay: ResultDisplay
IntegrationTesting: IntegrationTesting
OptimizationEnhancements: OptimizationEnhancements
ImageLoading
GrayscaleConversion
IntegrationTesting
integrate_and_test(): void
OptimizationEnhancements
optimize_and_enhance(): void
ImageLoader
load_image(image_path: String): Image
FaceDetector
detect_faces_haarcascades(gray_image: Image): List<Face>
GrayscaleConverter
convert_to_grayscale(image: Image): Image
FaceOutliner
draw_faces(image: Image, faces: List<Face>): Image

The diagram illustrates the Face Detection Pipeline methodology, showcasing distinct stages and components. Beginning with image loading, the process involves converting images to grayscale, detecting faces using Haar cascades, outlining detected faces, and displaying the result. The diagram includes stages for integration testing and optimization enhancements. The ‘FaceDetectionPipeline’ class orchestrates the overall flow, connecting individual components such as ‘ImageLoader’, ‘GrayscaleConverter’, ‘FaceDetector’, ‘FaceOutliner’, and ‘ResultDisplay’. This visual representation offers a concise overview of the sequential and modular structure of the face detection methodology, aiding in understanding the step-by-step implementation and integration of each stage.

Program:

```
import cv2
        def load_image(image_path): return cv2.imread(image_path)
        def convert_to_grayscale(image):
        return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        def detect_faces_haarcascades(gray_image):
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml') faces = face_cascade.detectMultiScale(gray_image,
scaleFactor=1.3, minNeighbors=5)
        return faces
        def draw_faces(image, faces): for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2) return image
        def display_image(image): cv2.imshow('Face Detection', image) cv2.waitKey(0)
cv2.destroyAllWindows()
        def  face_detection_pipeline(image_path):

        # Stage 1: Load Image
        image = load_image(image_path)
        # Stage 2: Convert to Grayscale      gray_image = convert_to_grayscale(image)
        # Stage 3: Detect Faces using Haarcascades faces =
detect_faces_haarcascades(gray_image)
        # Stage 4: Draw Faces
        result_image = draw_faces(image.copy(), faces)
        # Stage 5: Display Result display_image(result_image)

        # Replace 'path/to/your/image.jpg' with the actual path to your image file image_path
= 'path/to/your/image.jpg'

        # Run the face detection pipeline face_detection_pipeline(image_path)
```

FLOWCHART


Start
Load Image
Image Loading Successful?
yes
no
Convert to Grayscale
Error: Unable to load image
Convert to Grayscale Successful?
yes
no
End
Detect Faces (Haarcascades)
End
Faces Detected?
yes
no
Draw Faces
End
Display Result
End

FLOW CHART EXPLANATION:

ImageLoading Class:
Represents the first stage of the pipeline.
Method: loadImages(filePath: String) - Loads images from a specified file path.

GrayscaleConversion Class:
Represents the second stage of the pipeline.
Method: convertToGrayscale(image: Image) - Converts the loaded image to grayscale, preparing it for face detection.

FaceDetection Class:
Represents the third stage of the pipeline.
Method: detectFaces(grayscaleImage: GrayscaleImage) - Applies Haar cascades to detect faces in the grayscale image.

FaceOutlining Class:
Represents the fourth stage of the pipeline.
Method: outlineFaces(originalImage: Image, faces: List<Face>) - Outlines detected faces on the original image for visual representation.

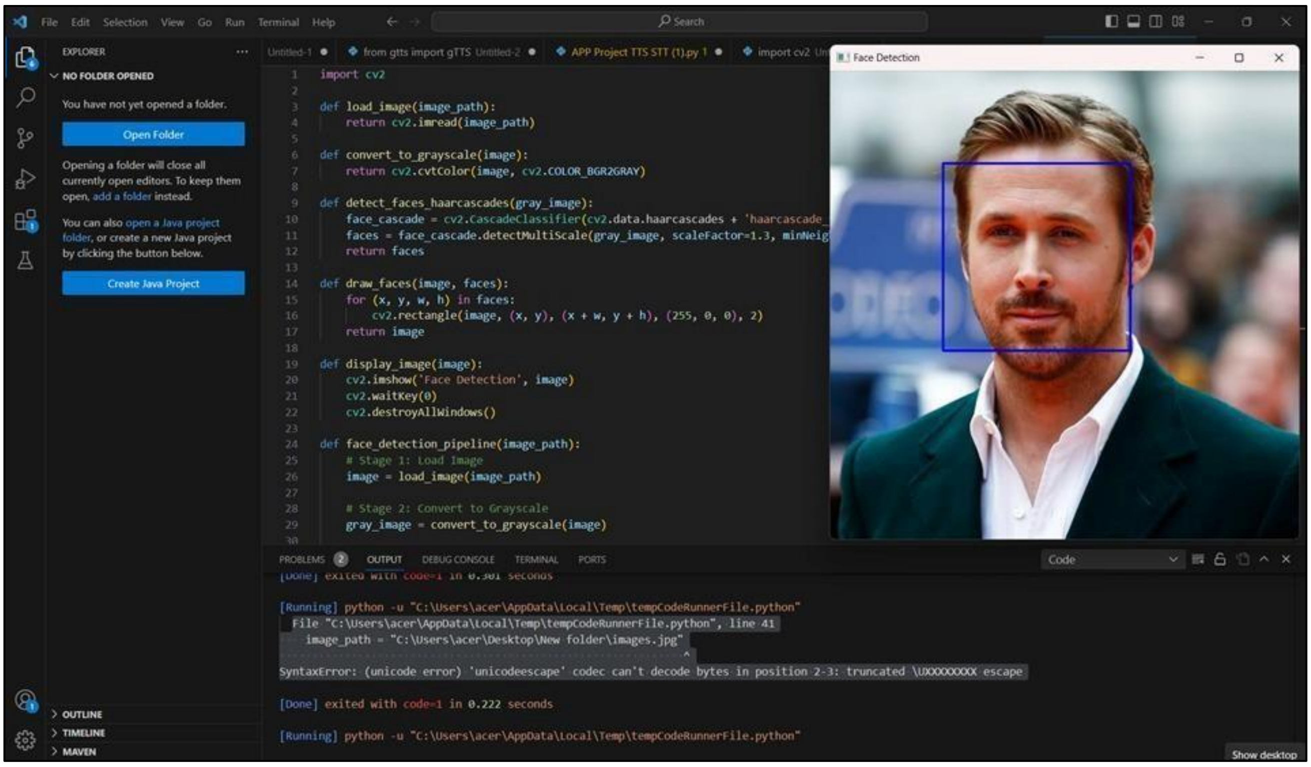
ResultDisplay Class:
Represents the fifth stage of the pipeline.
Method: displayResult(finalImage: Image) - Displays the final image with outlined faces in a graphical window for user interaction.

IntegrationTesting Class:
Represents the integration and testing stage of the pipeline.
Method: integrateAndTest() - Integrates all stages and ensures seamless functionality, testing the system with various images.

OptimizationEnhancements Class:
Represents the optimization and further enhancements stage of the pipeline.

Methods:
optimizeStages() - Fine-tunes each stage for improved efficiency.
exploreAdvancedTechniques() - Considers advanced face detection techniques and explores options for parallel processing.
Arrows between classes indicate the flow of data or control from one stage to the next. This diagram provides a high-level overview of the Face Detection Pipeline, emphasizing the sequential execution of stages and the integration and optimization steps to enhance the overall system.

OUTPUT:



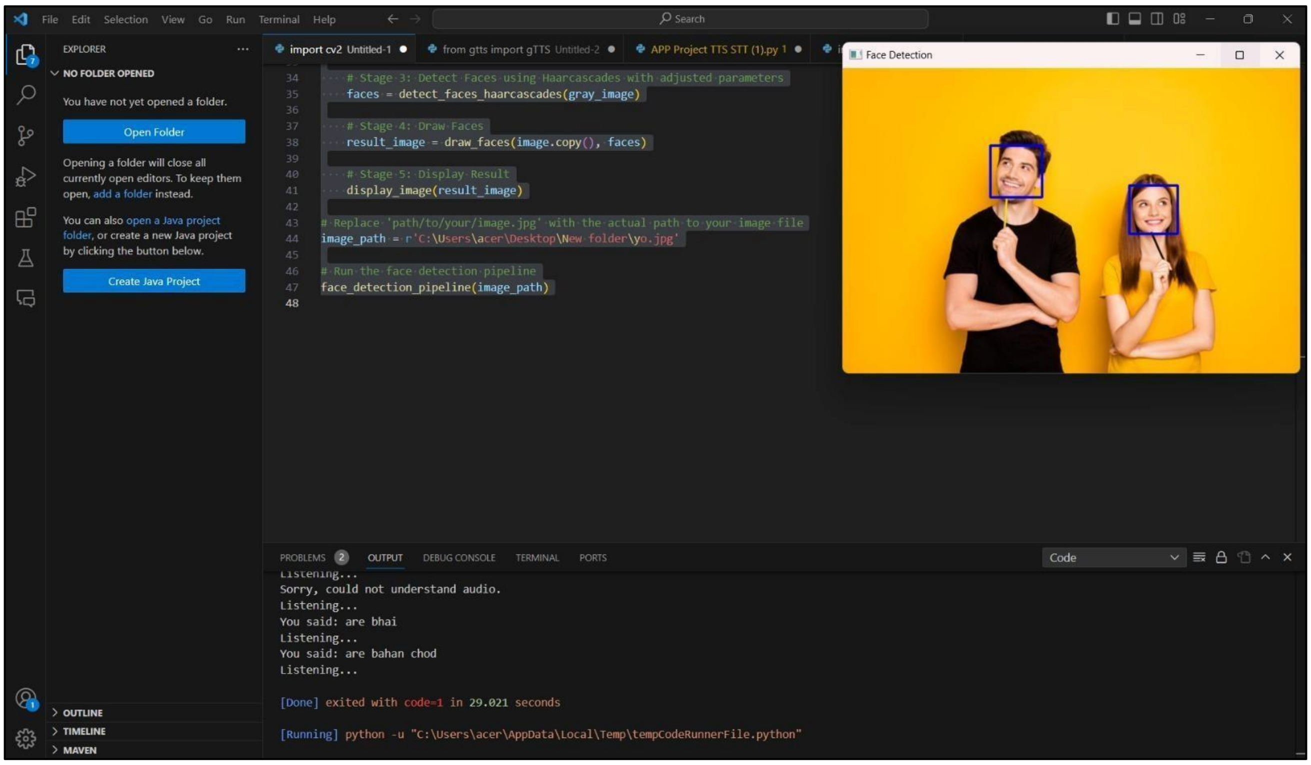
You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can also open a Java project folder, or create a new Java project by clicking the button below.

```python
import cv2

def load_image(image_path):
    return cv2.imread(image_path)

def convert_to_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

def detect_faces_haarcascades(gray_image):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_
    faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.3, minNeig
    return faces

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

def display_image(image):
    cv2.imshow('Face Detection', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def face_detection_pipeline(image_path):
    # Stage 1: Load Image
    image = load_image(image_path)

    # Stage 2: Convert to Grayscale
    gray_image = convert_to_grayscale(image)
```

```
[Done] exited with code=1 in 0.301 seconds

[Running] python -u "C:\Users\acer\AppData\Local\Temp\tempCodeRunnerFile.python"
  File "C:\Users\acer\AppData\Local\Temp\tempCodeRunnerFile.python", line 41
    image_path = "C:\Users\acer\Desktop\New folder\images.jpg"
                                                              ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape

[Done] exited with code=1 in 0.222 seconds

[Running] python -u "C:\Users\acer\AppData\Local\Temp\tempCodeRunnerFile.python"
```



You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can also open a Java project folder, or create a new Java project by clicking the button below.

```python
    # Stage 3: Detect Faces using Haarcascades with adjusted parameters
    faces = detect_faces_haarcascades(gray_image)

    # Stage 4: Draw Faces
    result_image = draw_faces(image.copy(), faces)

    # Stage 5: Display Result
    display_image(result_image)

# Replace 'path/to/your/image.jpg' with the actual path to your image file
image_path = r'C:\Users\acer\Desktop\New folder\yo.jpg'

# Run the face detection pipeline
face_detection_pipeline(image_path)
```

```
Listening...
Sorry, could not understand audio.
Listening...
You said: are bhai
Listening...
You said: are bahan chod
Listening...

[Done] exited with code=1 in 29.021 seconds

[Running] python -u "C:\Users\acer\AppData\Local\Temp\tempCodeRunnerFile.python"
```

CONCLUSION

The "Face Detection using Pipeline" project has successfully demonstrated the effectiveness of a pipelined approach to real-time and efficient face detection. Drawing inspiration from computer architecture principles, the project modularized the face detection process into sequential stages, each with a dedicated function. Here's a summary of the project's key conclusions:

- Efficiency Through Pipelining:

The use of a pipelined structure has significantly improved the efficiency of the face detection process. By breaking down the task into independent stages, the system can process images in parallel, reducing overall processing time and enhancing real-time capabilities.

- Modularity and Flexibility:

The modular design of the pipeline allows for the independent development and optimization of each stage. This modularity enhances the system's flexibility, making it adaptable to different use cases and open to future enhancements. New face detection algorithms or performance improvements can be seamlessly integrated into the pipeline.

- Accuracy in Face Detection:

The implementation of Haar cascades for face detection has provided a reliable and accurate method for identifying faces within images. By leveraging pre-trained models, the system can detect faces with high precision.

- User-Friendly Visual Output:

The project's output stage, which displays the images with outlined faces, provides a user-friendly visual representation of the detected faces. This makes the results easily interpretable and accessible to end-users.

- Real-Time Potential:

The parallel processing and modular nature of the pipeline make it well-suited for real-time face detection applications. With appropriate hardware support and optimization, the system can be utilized in scenarios that demand instantaneous face recognition.

In conclusion, the "Face Detection using Pipeline" project showcases the application of computer architecture principles to image processing and face detection. It emphasizes efficiency, modularity, and accuracy as key elements for a successful face detection system. With its modular structure and adaptable design, this project contributes to the advancement of computer vision technologies and holds promise for various real-world applications where face detection plays a vital role.

REFRENCES:

Book:

Title: "Computer Organization and Embedded Systems"
Authors: Carl Hamacher, Zvonko G. Vranesic, Safwat G. Zaky, and Naraig Manjikian

This book provides a comprehensive introduction to computer organization and architecture, with a particular focus on embedded systems. It covers topics related to digital logic, processor design, memory hierarchy, input/output systems, and more. "Computer Organization and Embedded Systems" is a valuable resource for students and professionals seeking to understand the inner workings of computers and embedded systems.

Links:
https://www.youtube.com/watch?v=uNBVukvwyuc&list=PL1C2GgOjAF-KFxGFauGAF3wOjYbmezNG0