A Project Report

on

FACE DETECTION IN A VIDEO USING BIG DATA TOOLS AND COMPUTER VISION TECHNIQUES

Submitted for partial fulfillment of the award of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering

by

N. SUNEETHA	**-**	**20BQ1A05H0**
P. V. L GAYATHRI	**-**	**20BQ1A05H2**
N. KOTESWARA RAO	**-**	**20BQ1A05F9**
SK. KAHAR MOULA	**-**	**21BQ5A0520**

Under the Guidance of
Mrs. G. RamaDevi
Assistant Professor

(Autonomous)

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY
An Autonomous Institute
Approved by AICTE, Permanently Affiliated to JNTU, Kakinada
Accredited by NAAC with 'A' Grade - ISO 9001:2008 Certified
Nambur (V), Peda Kakani (M), Guntur Dt. - 522508

April, 2024.

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY
(Autonomous)
Permanently Affiliated to JNTU, Kakinada, Approved by AICTE
Accredited by NAAC with 'A' Grade, ISO 9001:2015 Certified
Nambur, Pedakakani (M), Guntur (Dt) - 522508
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
B.Tech Programme Accredited by NBA

CERTIFICATE

This is to certify that the project report titled "**FACE DETECTION IN VIDEO USING BIG DATA TOOLS AND COMPUTER VISION TECHNIQUES**" is being submitted by **Ms. N. Suneetha, Ms. P. V. L Gayathri, Mr. N. KoteswaraRao, Mr. Sk. Kahar Moula** bearing Regd No's **20BQ1A05H0, 20BQ1A05H2, 20BQ1A05F9, 21BQ5A0520** respectively in IV B.Tech II semester *Computer Science & Engineering* is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

Project Guide

(Mrs. G. RamaDevi, Assistant Professor)

Head of Department

(Dr. V. Ramachandran, Professor)

Submitted for Viva-voce held on _____

Internal Examiner

External Examiner

DECLARATION FORMAT

We NEERUMALLA SUNEETHA, PABBISETTY VENKATA LAKSHMI GAYATHRI, NADIKOTI KOTESWARA RAO, SHAIK KAHAR MOULA hereby declare that the Project Report entitled “FACE DETECTION VIDEO USING BIG DATA TOOLS AND COMPUTER VISION TECHNIQUES” done by us under the guidance of Ms. G. RAMA DEVI at Vasireddy Venkatadri Institute of Technology is submitted in partial fulfillment of the requirements for the award of degree in Computer Science and Engineering

DATE :

PLACE :

SIGNATURE OF THE CANDIDATE(s)

1. N. Suneetha - 20BQ1A05H0
2. P. V. L Gayathri - 20BQ1A05H2
3. N. Koteswara Rao - 20BQ1A05F9
4. Sk. Kahar Moula - 21BQ5A0520

.

ACKNOWLEDGEMENT

We take this opportunity to express my deepest gratitude and appreciation to all those people who made this project work easier with words of encouragement, motivation, discipline, and faith by offering different places to look to expand our ideas and helped us towards the successful completion of this project work.

First and foremost, we express our deep gratitude to **Mr. Vasireddy Vidya Sagar**, Chairman, Vasireddy Venkatadri Institute of Technology for providing necessary facilities throughout the B Tech program.

We express our sincere thanks to **Dr. Y. Mallikarjuna Reddy**, Principal, Vasireddy Venkatadri Institute of Technology for his constant support and cooperation throughout the B. Tech program.

We express our sincere gratitude to **Dr. V. Ramachandran**, Professor & HOD, Computer Science & Engineering, Vasireddy Venkatadri Institute of Technology for his constant encouragement, motivation and faith in offering different places to look to expand our ideas.

We would like to express our sincere gratefulness to our Guide, **Mrs. G. RamaDevi** Assistant Professor, CSE for his insightful advice, motivating suggestions, invaluable guidance, help and support in the successful completion of this project.

We would like to express our sincere heartfelt thanks to our Project Coordinator **Dr. N. Sri Hari**, Associate Professor, CSE for his valuable advices, motivating suggestions, moral support, help and coordination among us in successful completion of this project. We would like to take this opportunity to express our thanks to the Teaching and Non-Teaching Staff in the Department of Computer Science & Engineering, VVIT for their invaluable help and support.

Name (s) of Students:

1. **N. Suneetha - 20BQ1A05H0**
2. **P.V.L Gayathri - 20BQ1A05H2**
3. **N. Koteswara Rao - 20BQ1A05F9**
4. **Sk. Kahar Moula - 21BQ5A0520**

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Face detection within video content has long been challenged by precision limitations. However, advancements in technology, particularly the utilization of big data techniques, offer promising solutions to enhance both speed and efficiency in this task. By leveraging Apache Spark, a distributed computing framework capable of running across multiple nodes simultaneously, the conversion of individual video frames into an analyzable format becomes more efficient. The integration of the YOLO Face Detection model, based on deep learning principles, further enhances the accuracy of face detection within each frame. Additionally, Deep Face, a powerful tool for facial recognition, is employed to pinpoint target faces and extract instances across multiple frames. This integrated approach not only addresses the precision limitations traditionally associated with face detection but also opens up possibilities for extensive monitoring and analysis within video content.

CHAPTER 1

INTRODUCTION

1.1 Overview

The application of face detection in video, leveraging big data tools and computer vision techniques, is a fascinating intersection of intensive handling of data and advanced image analysis. Face detection, a crucial task in computer vision, involves the identification and localization of human faces in still or moving images or video frames. This process plays a vital role in various applications, such as security monitoring, emotion analysis, and facial recognition.

To achieve face detection in videos, algorithms analyses the visual elements of an image or video. Given the considerable volume of data involved in video streams, using big data solutions becomes imperative. Apache Spark, a notable example, proves invaluable due to is scalability, fault tolerance, and distributed processing capabilities.

Furthermore, the efficiency of face detection algorithms is critical in real-world scenarios. They must handle dynamic changes in lighting conditions, varying facial expressions, and diverse backgrounds. Robust algorithms contribute to the accuracy and reliability of applications relying on face detection.

OpenCV, an abbreviation for the Computer Vision Open-Source Library, stands as an open-source software library for computer vision and machine learning tasks. It was initially developed by Intel in 1999 and has since become one of the most widely used libraries for real-time computer vision applications. Here's a breakdown of its key features and capabilities. OpenCV provides a vast array of tools and algorithms for image processing tasks such as filtering, edge detection, morphological operations, and color space manipulation. As face detection applications continue to expand, the integration of real-time processing becomes crucial. Real-time face detection is essential in scenarios such as surveillance, where quick and accurate identification is paramount. This necessitates not only efficient algorithms but also the utilization of parallel processing and optimized hardware for rapid decision making. Moreover, privacy concerns have become increasingly relevant in the deployment of face detection systems.

1.2 Applications

Face detection in video has numerous applications across various industries and domains. Some of the key applications include:

Security and Surveillance: Face detection is widely used in security systems for identifying individuals in real-time. It helps in monitoring crowded areas, airports, public transportation, and other sensitive locations for security purposes. It can also be integrated with access control systems to grant or deny entry based on recognized faces.

Automated Attendance Systems: In educational institutions and workplaces, face detection in videos can automate attendance marking. By analyzing video feeds, attendance can be recorded accurately without manual intervention, saving time and reducing errors.

Marketing and Advertising: Face detection in video enables marketers to gather data on viewer demographics, such as age, gender, and emotional responses to advertisements. This data can be used to tailor marketing campaigns more effectively and personalize content to target audiences.

Customer Insights: In retail environments, face detection can be used to analyze customer behavior, such as dwell time in front of products and overall foot traffic patterns. This information helps retailers optimize store layouts, product placements, and marketing strategies.

Emotion Analysis and User Experience: Face detection can be used to gauge emotional responses of users interacting with software applications or digital content. This information is valuable for improving user experience in areas such as gaming, virtual reality, and customer service interactions.

Healthcare: In healthcare settings, face detection in video can assist in patient monitoring and analysis. For example, it can be used to detect signs of pain, distress, or fatigue in patients during consultations or in hospital environments. It can also aid in identifying individuals for medical records and patient tracking.

Automotive Safety: In advanced driver-assistance systems (ADAS) and autonomous vehicles, face detection is used to monitor driver attentiveness and detect signs of drowsiness or distraction. This helps enhance safety by alerting drivers or triggering automated interventions when necessary.

Entertainment and Content Creation: Face detection technology is employed in various entertainment applications, including virtual try-on experiences for cosmetics and fashion, augmented reality filters in social media platforms, and special effects in movies and video games.

Law Enforcement and Forensics: Face detection assists law enforcement agencies in identifying suspects from surveillance footage and tracking their movements across different locations. It also plays a crucial role in forensic investigations by matching faces to criminal databases and reconstructing events.

Accessibility: Face detection technology can improve accessibility features for individuals with disabilities. For instance, it can be used to enable hands-free interaction with devices and software applications, allowing users to control interfaces through facial gestures and expressions.

Overall, the applications of face detection in video are diverse and continue to expand as the technology advances, offering innovative solutions across various sectors.

CHAPTER 2

AIM AND SCOPE

The aim of this project is to improve the precision and efficiency of face detection within video content by leveraging big data techniques. Specifically, the project seeks to employ Apache Spark for parallel processing across multiple nodes to convert individual video frames into an analyzable format. Additionally, the project aims to utilize the YOLO Face Detection model for robust face detection within each frame, followed by Deep Face for precise identification and tracking of target faces across multiple frames.

2.1 Existing System

Face detection in real-world scenarios presents challenges due to variations in expressions, lighting, and occlusions. Deep learning techniques have emerged as effective tools in addressing these challenges. Various methodologies and frameworks focus on improving face detection accuracy and efficiency.

Deep learning models, such as the deep cascaded multi-task framework, exploit correlations between tasks to enhance face detection and landmark localization. These approaches leverage complex strategies to improve detection accuracy across diverse scenarios. Studies propose correlation-based approaches that utilize response maps from CNN models to improve face detection in video sequences. By aligning face features more accurately, these methods aim to provide consistent and interpretable results.

Researchers leverage specialized datasets to train and evaluate face detection models. The integration of face mesh technology with deep neural networks demonstrates advancements in accurately identifying faces across varying conditions.

The application of these advancements extends to practical security and surveillance systems. Prototypes utilizing technologies like Haar Cascade for face detection, integrated with microcomputers such as Raspberry Pi, offer promising solutions for enhancing access control and surveillance capabilities in various industries.

The existing literature highlights the use of deep learning techniques and advanced methodologies to enhance face detection precision. By addressing challenges such as variations in expressions and lighting conditions, these approaches offer improved accuracy and efficiency in detecting faces within video content.

The landscape of face detection and recognition systems is evolving rapidly, driven by advancements in deep learning techniques and their applications across various domains. Traditionally, face detection in real-world scenarios has been challenged by factors such as variations in expressions, lighting conditions, and occlusions. However, deep learning techniques have demonstrated remarkable success in addressing these challenges, leveraging complex strategies to achieve higher precision and efficiency.

One notable approach proposed in recent research is the deep cascaded multi-task framework, which exploits correlations between tasks to enhance performance in both face detection and landmark localization. This framework capitalizes on deep learning's ability to learn intricate patterns and relationships within data, resulting in improved accuracy and robustness in detecting faces across different conditions.

Additionally, advancements in face recognition technology have been propelled by the robustness of deep learning models, enabling significant improvements in recognition accuracy. The proposed AB-FR model, for instance, integrates convolutional neural networks (CNNs) with attention mechanisms to enhance feature extraction from facial images. By considering temporal and geometric features across different images of the same individual, this model addresses the limitations of traditional deep learning-based methods, resulting in enhanced recognition performance and robustness.

Furthermore, the integration of face mesh technology with deep neural networks presents a promising approach for robust face detection and recognition, offering versatility across varying conditions such as illumination changes and non-frontal poses. Leveraging datasets like the Labeled Wild Face (LWF), researchers have achieved commendable accuracy rates in face recognition tasks, showcasing practical solutions for access control and identification systems.

Moreover, comprehensive surveys and studies have provided insights into the historical evolution of face detection and recognition algorithms, highlighting the increasing role of neural networks in driving advancements in this field. The comparative evaluations among different algorithms have shed light on their strengths and limitations, emphasizing the need for continuous improvement to mitigate false positives and unlock the full potential of face detection technology in critical domains.

In summary, the existing systems and methodologies discussed in recent research underscore the ongoing efforts to enhance the accuracy, efficiency, and applicability of face detection and recognition systems. Leveraging deep learning techniques and insights gained from experimentation and analysis, researchers and developers continue to push the boundaries of what is achievable in this field, paving the way for innovative solutions with diverse applications in security, surveillance, education, and beyond.

2.2 Proposed System

The proposed system aims to utilize advanced deep learning techniques for accurate face detection and verification in videos. It combines the YOLOv8 model for face detection and the Deep Face model for face verification. Additionally, Apache Spark is employed for parallel processing of video frames to enhance speed and efficiency.

The training process involves initializing the YOLOv8 model with pre-trained weights and setting up data loaders to load the training dataset, which comprises face images. The model is then trained on the dataset for multiple epochs to learn features for face detection tasks. Once trained, the model is validated on a separate dataset to evaluate its performance. The validation module is initialized with the trained model and validation dataset. Various metrics, including confusion matrix-based metrics for object detection tasks, are computed to assess the model's performance. The evaluation results are recorded for analysis.

The pretrained Deep Face model is loaded to perform face verification. Detected faces from the video frames are passed as input to the Deep Face model along with the target image for verification. The algorithm extracts features from detected faces and matches them against known identities to verify the presence of the target face in the video frames. The frames containing matched faces from the Deep Face module are combined using the Moviepy module. The matched faces across frames are extracted and compiled into a single video output containing instances of the target image within the video.

Apache Spark is employed for parallel processing of video frames to enhance efficiency. The input image and video are processed, and the frames of the video are distributed to processors via Apache Spark for parallel processing. Apache Spark distributes the frames across multiple nodes for simultaneous processing. This distributed approach enables faster processing of video frames, reducing the overall processing time. After processing, Apache Spark returns the frames that match with the target image, facilitating efficient face detection and verification in the video content.

CHAPTER 3

CONCEPTS AND METHODS

3.1 Problem Description

The problem addressed in this project focuses on automating face detection and verification within video content using advanced deep learning techniques. Traditional face detection methods often encounter challenges in accurately identifying faces in video streams, particularly in real-time scenarios where precision and efficiency are crucial. Additionally, manual verification of detected faces can be time-consuming and subjective, requiring automated systems to improve accuracy and efficiency.

To tackle these challenges, this project leverages state-of-the-art deep learning models, specifically YOLOv8 for face detection and Deep Face for face verification. YOLOv8 is a robust and efficient model known for its real-time capabilities and high accuracy in detecting objects, including faces, within images and videos. Deep Face, on the other hand, specializes in facial recognition tasks, allowing for the verification of detected faces against known identities.

Furthermore, to enhance processing speed and efficiency, Apache Spark is utilized for parallel processing of video frames. Apache Spark's distributed computing framework enables simultaneous processing of multiple video frames across multiple nodes, significantly reducing processing time and enabling real-time face detection and verification. The project aims to overcome the limitations of traditional face detection methods by leveraging advanced deep learning techniques and parallel processing capabilities.

3.2 Proposed Solution

3.2.1 Dataset

The "face-detection-mik1i" dataset, hosted on Roboflow, emerges as a valuable resource tailored for machine learning tasks, particularly in the domain of face detection. Curated explicitly for this purpose, the dataset contains

a diverse array of images showcasing human faces, ready to be detected and localized by machine learning models. With varying resolutions, lighting conditions, backgrounds, and facial orientations, the dataset mirrors real-world scenarios, providing a comprehensive and representative collection for training and evaluation purposes. Each image within the dataset is meticulously annotated with bounding boxes, delineating the precise locations of faces, a fundamental aspect for supervised learning tasks.

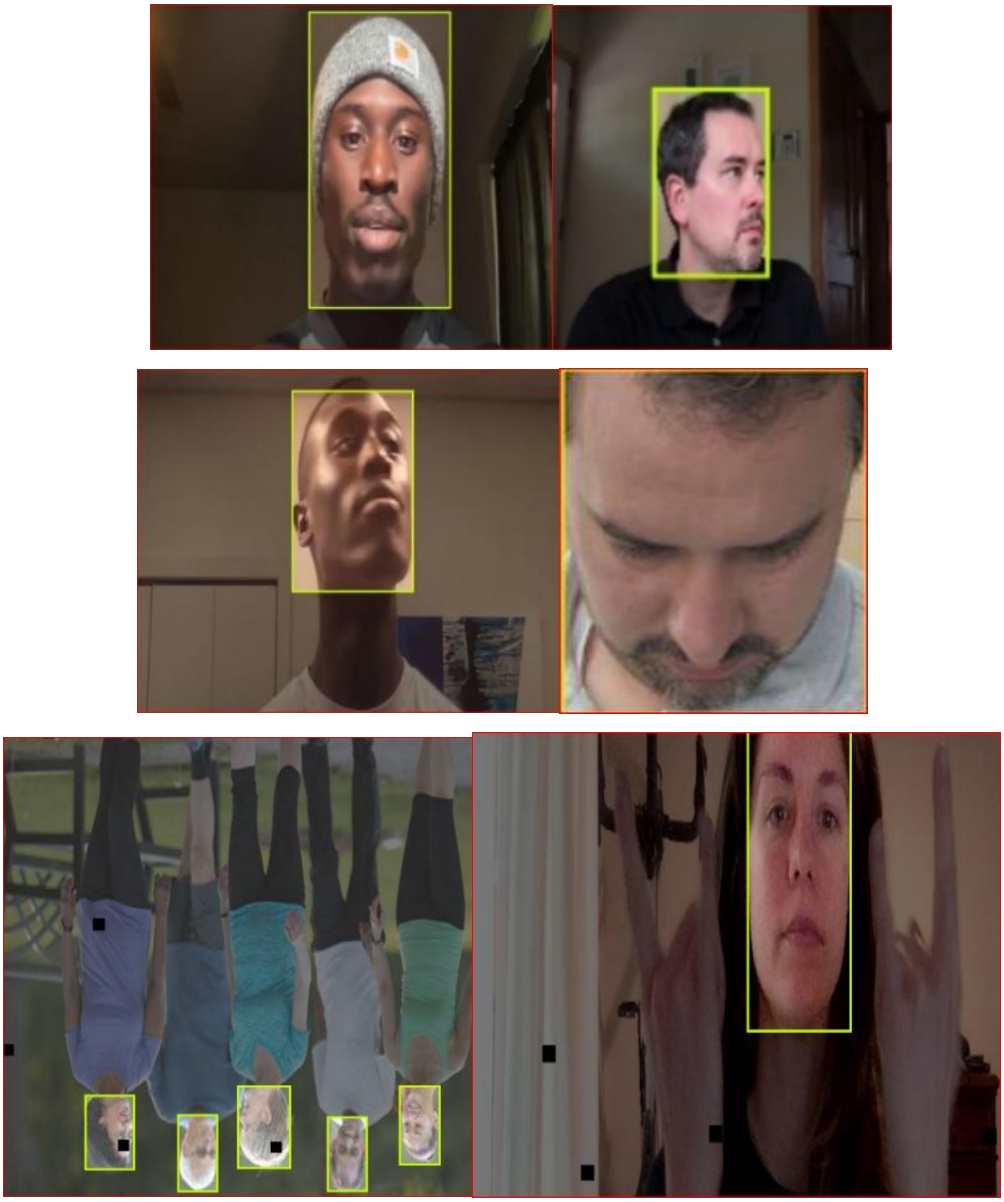

Fig3.1 Dataset Images

3.2.2 Dataset Preparation

The dataset preparation involves the organization and partitioning of images containing faces, each annotated with bounding boxes indicating the precise location of the faces. This dataset, obtained from Robo-flow, serves as a foundational resource for training and evaluating face detection models.

The dataset is divided into three subsets: the train set, the validation set, and the test set. The train set comprises 87% of the total images, totaling 2871 images, while the validation set consists of 8%, totaling 267 images. Finally, the test set constitutes 4% of the dataset, containing 145 images. This division facilitates robust evaluation and validation of face detection algorithms across different stages of development.

Every image within the dataset is annotated with bounding boxes, meticulously indicating the locations of faces present in the image. These annotations are crucial for supervised learning tasks, enabling machine learning models to learn and accurately detect faces within images. By providing annotated bounding boxes, the dataset offers valuable ground truth information necessary for training and evaluating the performance of face detection models.

Overall, the dataset preparation ensures the availability of a well-organized and annotated dataset suitable for training, validating, and testing face detection algorithms. This dataset serves as a foundational resource for researchers, developers, and practitioners seeking to advance the field of computer vision, particularly in the domain of face detection.

3.2.3 Dataset Preprocessing

The preprocessing of the dataset involves several essential steps to ensure uniformity, standardization, and suitability for face detection tasks. In this case, the following preprocessing techniques have been applied:

The Auto-Orient operation is applied to adjust the orientation of images automatically. This step ensures that all images are correctly oriented, eliminating any potential inconsistencies or misalignments that may affect the accuracy of face detection algorithms. By aligning the orientation of images, the preprocessing step enhances the reliability and consistency of subsequent processing stages.

The resizing operation involves adjusting the dimensions of images to a standardized size of 640x640 pixels. Resizing images to a consistent resolution is crucial for ensuring uniformity across the dataset. By stretching images to the specified dimensions, variations in image sizes are minimized, facilitating easier processing and analysis. Additionally, resizing images to a larger dimension may help preserve important details and features, enhancing the performance of face detection algorithms.

Overall, the preprocessing of the dataset involves these essential steps to standardize and optimize the images for face detection tasks. By applying Auto-Orient to correct image orientations and resizing images to a consistent resolution of 640x640 pixels, the dataset is prepared in a manner conducive to accurate and efficient face detection algorithm training and evaluation.

3.2.4 Data Augmentation

Data augmentation is a crucial preprocessing technique employed to increase the diversity and robustness of a dataset, particularly in computer vision tasks such as face detection. The provided parameters specify various transformations applied to the images to create augmented versions. Here's how each parameter contributes to data augmentation:

90deg Rotate: Images are rotated clockwise, counter-clockwise, or upside down by 90 degrees. This augmentation introduces variations in the orientation of faces, enhancing the model's ability to detect faces regardless of their orientation in the input images.

Flip: Horizontal and vertical flips are applied to images. Flipping horizontally or vertically creates mirrored versions of the original images, introducing variations in facial features' positions and orientations.

Crop: Images are randomly cropped with a minimum zoom of 0% and a maximum zoom of 25%. Cropping focuses on specific regions of the images, simulating different scales and viewpoints, which can help the model generalize better to faces of varying sizes and positions.

Gray Scale: Gray scale transformation is applied to 5% of the images. Converting images to grayscale removes color information, making the model more robust to variations in lighting conditions and color distributions.

Hue, Saturation, Brightness, and Exposure: Random adjustments are made to the hue, saturation, brightness, and exposure levels of images within specified ranges. These adjustments simulate changes in lighting conditions and color tones, improving the model's adaptability to diverse environments.

Blur: Gaussian blur with a maximum kernel size of 1 pixel is applied to images. Blurring introduces smoothness and reduces high-frequency noise, potentially enhancing the model's ability to focus on essential facial features while ignoring irrelevant details.

Cutout: Five rectangular regions, each with a size of 2% of the image's area, are randomly removed from images. This cutout augmentation helps the model learn robust features by forcing it to focus on other parts of the image, thus preventing overfitting and improving generalization.

By applying these data augmentation techniques, the dataset is augmented with a diverse range of variations, such as different orientations, scales, colors, and distortions. This augmented dataset provides a richer training

environment for face detection models, enabling them to learn more robust and generalizable representations of facial features.

Parameters	Value
90deg Rotate	Clockwise, Counter-Clockwise, Upside Down
Flip	Horizontal, Vertical
Crop	0% Minimum Zoom, 25% Maximum Zoom
Gray Scale	Apply to 5% of images
Hue	Between -20° and +20°
Saturation	Between -10% and +10%
Brightness	Between -10% and +10%
Exposure	Between -10% and +10%
Blur	Up to 1px
Cutout	5boxes with 2% size each

Table 3.1 Data Preprocessing

3.2.5 Model Architectures

1. YOLOV8 Model

YOLOv8, or You Only Look Once version 8, is an object detection algorithm that belongs to the YOLO (You Only Look Once) family of models. YOLO models are known for their real-time object detection capabilities, meaning they can quickly and accurately identify objects within an image or a video frame.

Backbone Network: YOLOv8 uses a backbone network, often based on CSPDarknet53, which is an extended version of Darknet, a deep neural network designed for object detection. The backbone extracts features from the input image.

Feature Pyramid Network (FPN): YOLOv8 employs a Feature Pyramid Network to capture features at different scales. This helps in detecting objects of varying sizes within an image.

Detection Head: The detection head in YOLOv8 is tasked with predicting both bounding boxes and the probabilities of various classes. It uses a set of convolutional layers to process the feature maps from the backbone and FPN, producing predictions for each grid cell.

Anchor Boxes: YOLOv8 utilizes anchor boxes to improve bounding box prediction accuracy. These anchor boxes are predefined sizes that help the model better adapt to the scale of different objects in the image.
Output Format: The final output of YOLOv8 is a set of bounding boxes, each associated with a specific class and its confidence score. The bounding boxes are predicted for different grid cells and anchor boxes at multiple scales.

Training: YOLOv8 is trained using a combination of labeled images and their corresponding bounding box annotations. The model is optimized using techniques like stochastic gradient descent with backpropagation.

Variants: YOLOv8 has different variants, including YOLOv8-S, YOLOv8-M, YOLOv8-L, and YOLOv8-X, with varying model sizes and computational demands. Users can choose a variant based on their specific requirements regarding speed and accuracy.

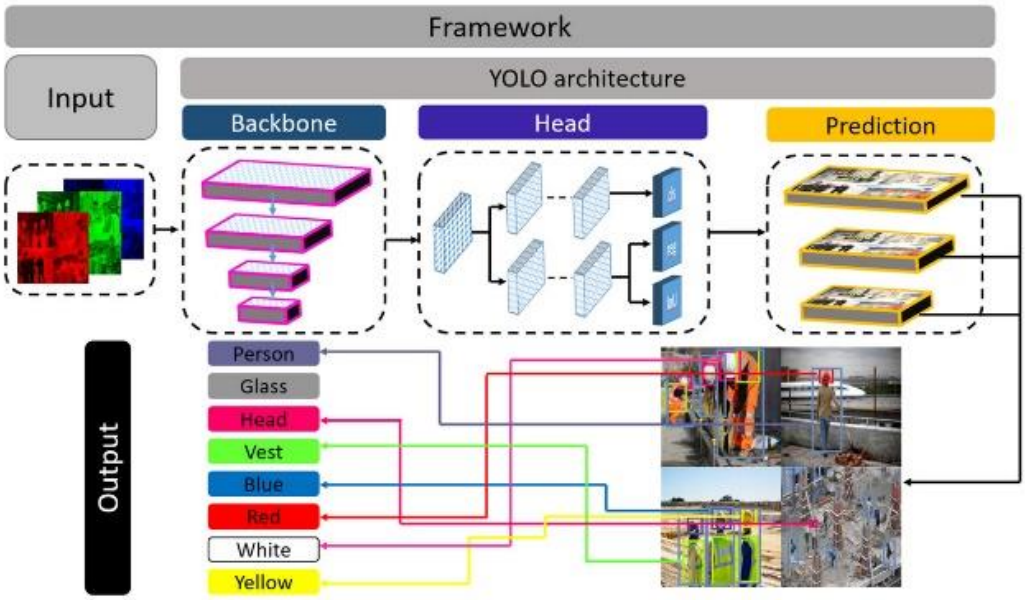

Fig3.2: YOLOv8 Architecture

2. Deep Face Model

"Deep Face" refers to a deep learning-based face recognition system developed by Facebook's AI Research (FAIR) lab. It gained significant attention when it was introduced in 2014 due to its high accuracy in identifying faces across large datasets. Here's an overview:
Architecture: Deep Face utilizes a deep convolutional neural network (CNN) architecture, which is trained on a massive dataset of labeled face images. The network consists of multiple layers that learn hierarchical representations of facial features, enabling it to recognize faces with high accuracy.
Face Verification: Deep Face is primarily designed for face verification tasks, which involve determining whether two face images belong to the same person or not. It accomplishes this by encoding facial images into a compact representation (embedding) in a high-dimensional space, where the similarity between two embeddings can be measured using techniques like cosine similarity or Euclidean distance.

Training Data: Deep Face was trained on a large-scale dataset called the "Social Face Classification" dataset, which contains millions of labeled face images belonging to thousands of individuals. This extensive training dataset helped Deep Face learn robust representations of facial features, enabling it to generalize well to unseen faces.

Performance: Deep Face achieved state-of-the-art performance on various face recognition benchmarks, surpassing previous methods in terms of accuracy and scalability. In particular, it demonstrated high accuracy even in challenging conditions such as variations in pose, lighting, and facial expression.

Overall, Deep Face represents a significant advancement in the field of face recognition, demonstrating the power of deep learning techniques for accurately identifying and verifying individuals in images. However, it also underscores the importance of addressing privacy and ethical concerns associated with the widespread adoption of facial recognition technology.

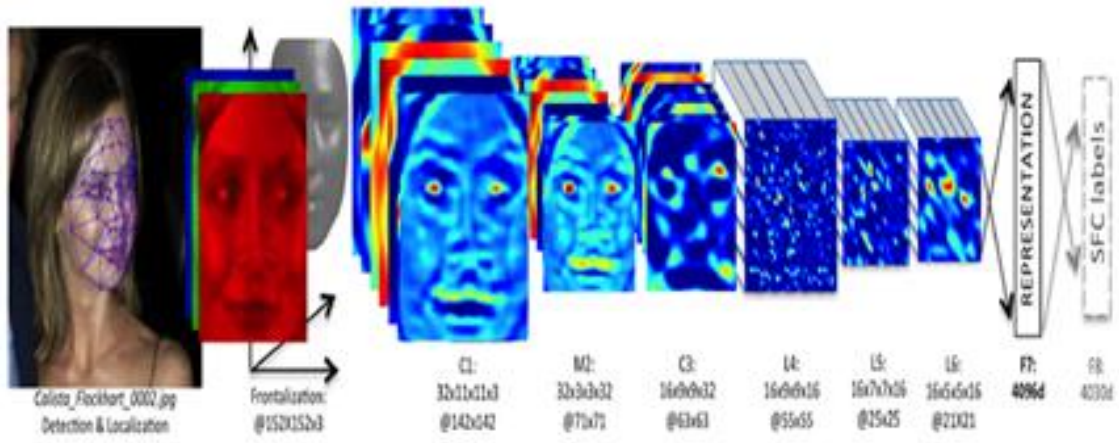

Fig3.3: Deep face Architecture

3.Apache Spark

Fast and versatile, Apache Spark is a cluster computing platform made. for distributed data processing. It offers a programming interface with implicit data parallelism and fault tolerance for entire clusters. Spark was created in Scala and is compatible with many programming languages, such as java, Python and R, so a broad spectrum of developers can use it.

There are lot of features in spark. When compared to other conventional data processing technologies, Spark is renowned for its extraordinary speed. By using in-memory processing, it lessens the requirement for a lot of disk input/output. Faster processing speeds are achieved by optimizing job execution through the usage of the Directed Acyclic Graph (DAG) execution.

A high-level API in Java, Scala, Python and R is provided by Spark, which streamlines the development process. Additionally, it offers interactive shell functionality so that code snippets can be examined and tested before being integrated into more complex workflows.

There is high adaptability in Apache Spark. Resilient distributed datasets and lineage information are two ways that spark provides fault tolerance. Spark can recompute an RDD using lineage information in the event that a partition is lost, guaranteeing data integrity and dependability.

Apache Spark's architecture is built to handle large-scale data processing tasks efficiently through distributed computing. At the heart of this architecture lies the Driver Program, which serves as the main control process. The Driver Program splits the user's application into tasks and coordinates their execution across the cluster. It maintains critical information about the application state, including the Directed Acyclic Graph (DAG) of operations and the lineage of Resilient Distributed Datasets (RDDs), which ensures fault tolerance and data recovery in case of failures.

Apache Spark can run on various cluster managers, such as Apache YARN, Apache Mesos, or its standalone cluster manager. These cluster managers allocate resources across applications running on the cluster and oversee their execution. Executors, the worker nodes in the Spark cluster, are responsible for executing tasks. Each executor can handle one or more tasks concurrently and manages data partitions in memory or disk. Executors communicate with the Driver Program and other executors for task coordination and data exchange.

Resilient Distributed Dataset (RDD) serves as the fundamental data abstraction in Apache Spark. RDDs represent distributed collections of immutable objects that are partitioned across the cluster and can be operated on in parallel. Spark transformations, such as map, filter, and reduce, are applied to RDDs to perform data processing tasks. RDD lineage ensures fault tolerance by tracking the sequence of transformations applied to the original data, enabling data recovery in case of failures.

The Directed Acyclic Graph (DAG) scheduler plays a crucial role in optimizing the execution plan of the Spark application. It converts the user's application into a logical execution plan represented as a DAG of stages. The scheduler identifies opportunities for optimization, such as pipelining and task co-location, to improve performance. Tasks are scheduled for execution by the Task Scheduler, which handles task distribution, fault tolerance, and data locality optimization. Tasks are scheduled to run on nodes where the data they require is already present, reducing data transfer overhead.

Apache Spark provides various storage levels to control how RDDs are stored in memory or on disk. Cached RDDs can be stored in memory across iterations or stages, improving performance by avoiding recomputation. Overall, Apache Spark's architecture is designed for scalability, fault tolerance, and efficient distributed data processing, making it well-suited for handling large-scale datasets and complex data processing workflows effectively.

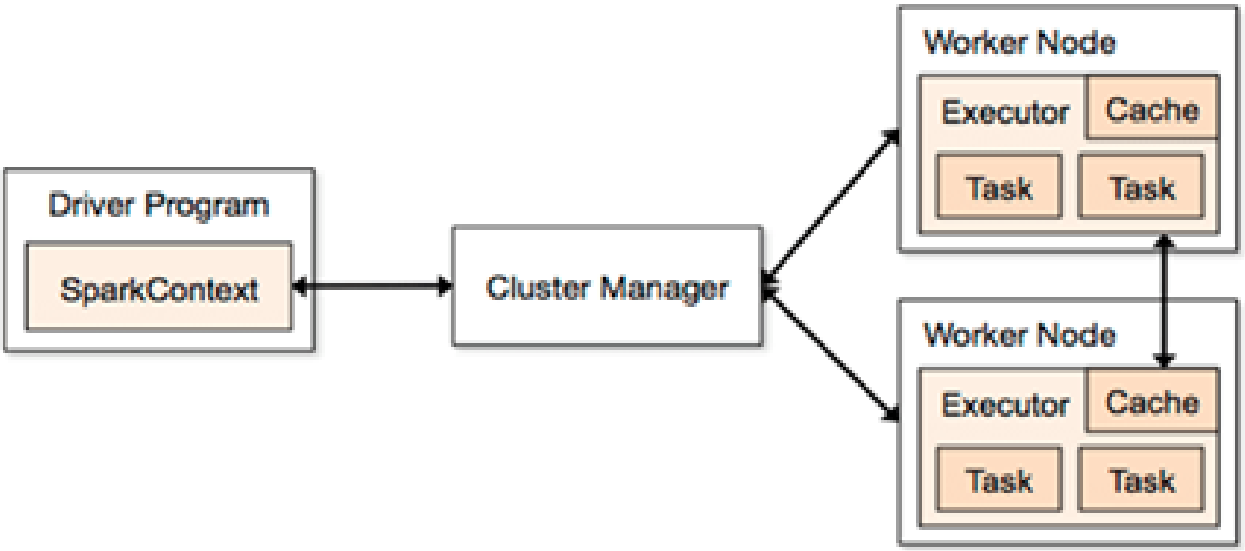

Fig3.4: Apache Spark Architecture

3.3 System Analysis

3.3.1 Requirement Specification

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

3.3.2 Functional Requirements

Functional requirements define the internal workings of the software: that is, the technical details, data manipulation and processing and other specific functionality that show how the use cases are to be satisfied. They are supported by non-functional requirements, which impose constraints on the design or implementation.

Hardware Requirements

The hardware requirements serve as the basis for the implementation of the system. They are used by software engineers as the starting point for system design. It shows what the system does and not how it should be implemented.

Processor: Intel core i5

Processor speed: 1.30GHZ or above

RAM: 8GB or above

Software Requirements

The software requirements document is the software specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. It is useful in estimating the cost, planning team activities, performing tasks, and tracking the team's progress throughout the development activity.

Operating system: Windows 11

Programming Language: Python

IDE: Google Colab

3.3.3 Non -Functional Requirements

Non-functional requirements are requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that specify specific behavior or functions. Typical non- functional requirements often called the utilities of a system. Other terms for non-functional requirements are "constraints", "quality attributes" and "quality of service requirements".

Reliability: If any exceptions occur during the execution of the software it should be caught and thereby prevent the system from crashing.

Scalability: The system should be developed in such a way that new modules and functionalities can be added, thereby facilitating system evolution.

Cost: The cost should be low because a free availability of software package.

3.4 Block Diagrams

Block diagrams, flowcharts, and use case diagrams are all tools used in different contexts to represent processes, systems, or interactions. Here's a description of each:

Block Diagrams:

Block diagrams are a type of diagram that represents a system or process as a series of interconnected blocks, each representing a specific component or function. These blocks are typically geometric shapes, such as rectangles or squares, with labels describing their function or purpose. Arrows or lines connecting the blocks indicate the flow of information or signals between them. Block diagrams are often used in engineering, electronics, and control systems to illustrate the structure and interactions within a system at a high level. They help visualize the overall architecture and components of a system, making it easier to understand and analyze complex systems.

Flowcharts:

Flowcharts are diagrams that depict the flow of control or data within a system or process. They use a variety of symbols and shapes, such as rectangles, diamonds, circles, and arrows, to represent different steps, decisions, inputs, and outputs in a process. Each symbol has a specific meaning, such as a rectangle representing a process step, a diamond representing a decision point, and arrows indicating the flow of control or data between steps. Flowcharts are widely used in various fields, including software development, business process management, and project management, to document, analyze, and communicate processes and workflows. They help visualize the sequence of steps in a process, identify potential bottlenecks or inefficiencies, and facilitate communication and collaboration among stakeholders.

Use Case Diagrams:

Use case diagrams are a type of behavioral diagram in the Unified Modeling Language (UML) that depict the interactions between a system and its users or other systems in terms of use cases and actors. Use cases represent the specific functionalities or features that a system provides to its users, while actors represent the users or external systems interacting with the system. Use case diagrams use oval-shaped symbols to represent use cases, stick figures or other symbols to represent actors, and lines to indicate the relationships or interactions between them. Use case diagrams are commonly used in software development to capture and communicate the functional requirements of a system from the perspective of its users. They help identify the different ways users can interact with the system, clarify the system's scope and boundaries, and serve as a basis for further elaboration and refinement of requirements during the software development process.

3.4.1 Flow Chart

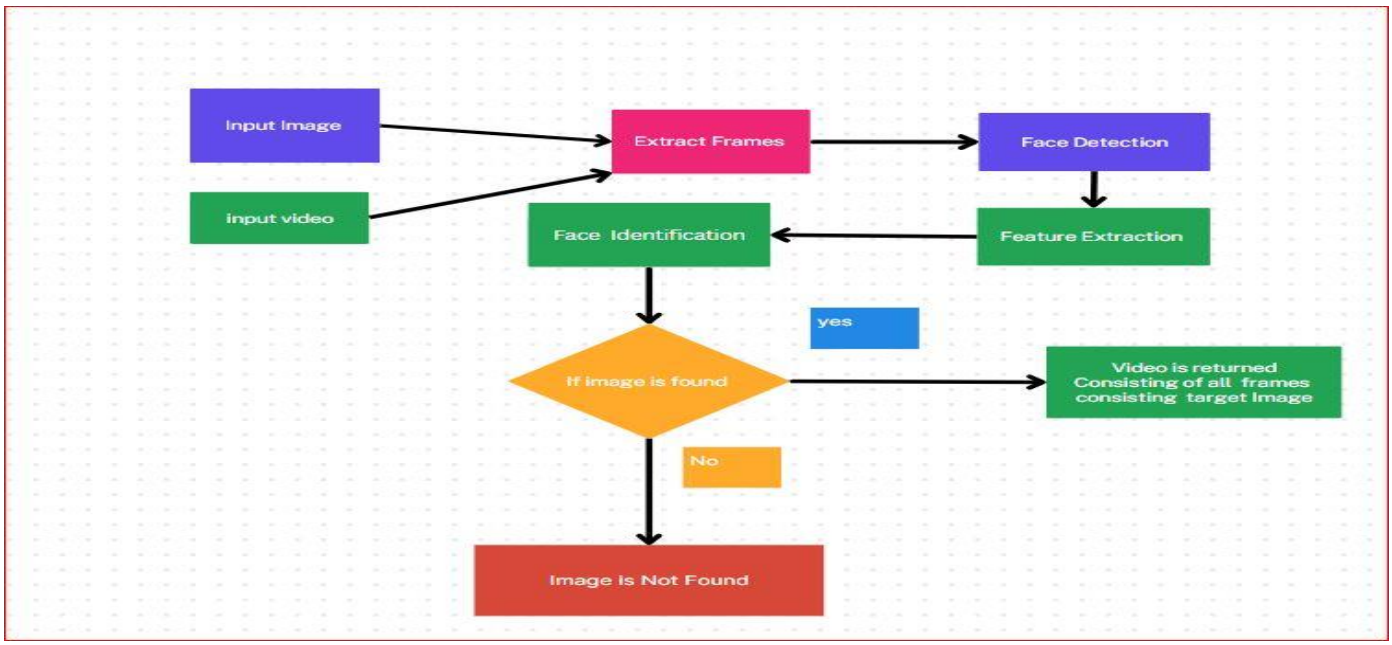

Fig3.5: Flow chart Diagram of Face Detection in Video

3.4.2 Use Case Diagram

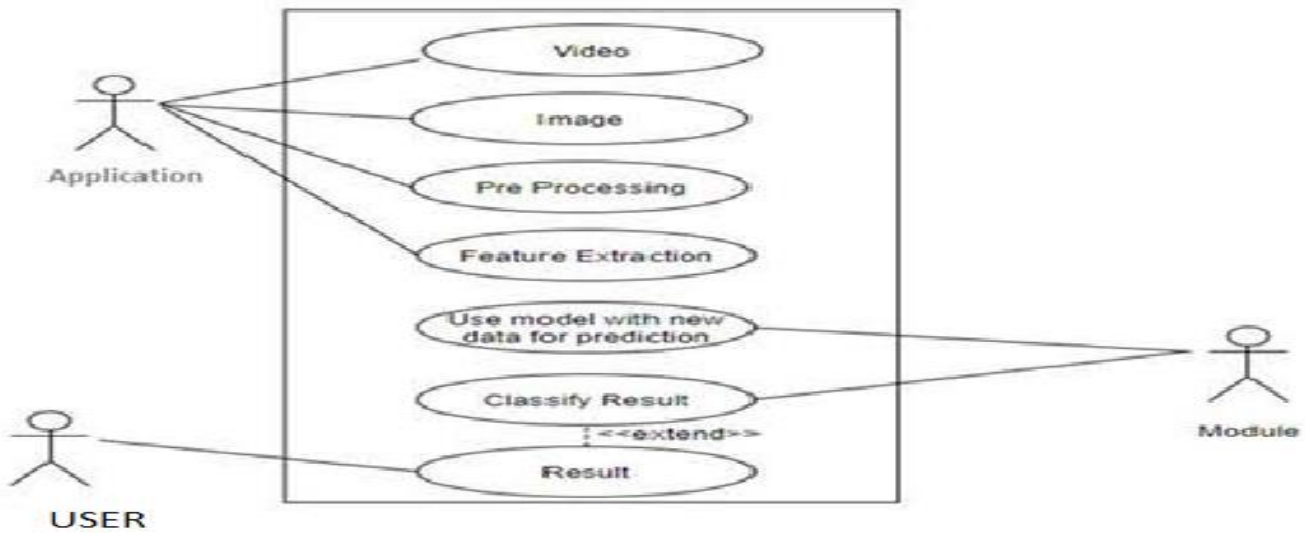

Fig 3.6 Use case Diagram of Face Detection in Video

CHAPTER 4

IMPLEMENTATION

4.1 TOOLS USED

In the implementation of the "Face detection in a video using big data tools and computer vision techniques" system, various tools and technologies are utilized to develop and train. These tools facilitate different aspects of the project, including data preprocessing, model training. Here are the key tools used:

4.1.1 Python

Python, a versatile and widely-used programming language, plays a central role in our face detection project. Renowned for its simplicity, readability, and extensive libraries, Python provides a robust foundation for implementing complex computer vision algorithms and techniques. With its intuitive syntax and dynamic typing, Python facilitates rapid development and experimentation, enabling us to iterate quickly on our face detection system.

Python's ecosystem boasts a plethora of libraries and frameworks tailored for various tasks, making it an ideal choice for our project. One of the key libraries we leverage is OpenCV (Open Source Computer Vision Library), which offers a comprehensive suite of tools for image and video processing. Through OpenCV's Python bindings, we gain access to functions for tasks such as face detection, feature extraction, and object tracking, empowering us to build a robust face detection pipeline.

Moreover, Python's compatibility with cloud-based platforms like Google Colab enhances our project's scalability and accessibility. By harnessing the computational resources provided by Google Colab, including GPU acceleration, we expedite our model training and experimentation processes. This collaboration between Python and Google Colab enables us to tackle large-scale video datasets and optimize our face detection algorithms efficiently.

Python's versatility extends to its integration with libraries like MoviePy, which facilitates video editing and manipulation. Leveraging MoviePy's capabilities, we can enhance our video analysis by applying transformations, annotations, and overlays to visualize the results of our face detection algorithm dynamically. This integration enriches our understanding of the face detection process and facilitates the interpretation of our findings.

In summary, Python serves as the linchpin of our face detection project, providing a flexible and efficient programming environment. Through Python's extensive libraries, compatibility with cloud platforms like Google Colab, and seamless integration with tools like OpenCV and MoviePy, we construct a powerful face detection pipeline capable of handling diverse video datasets and delivering accurate results.

4.1.2 Google Colab

Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that revolutionizes the way we develop and execute Python code. It offers a collaborative environment for writing, running, and sharing Jupyter notebooks directly in the browser, eliminating the need for local installations or setup. Google Colab is particularly popular among data scientists, machine learning engineers, and researchers due to its seamless integration with Google Drive, extensive computational resources, and support for popular libraries and frameworks.

One of the key features of Google Colab is its provision of free access to GPU (Graphics Processing Unit) and TPU (Tensor Processing Unit) resources. These powerful hardware accelerators significantly enhance the performance of computation-intensive tasks such as training deep learning models. By leveraging GPU and TPU acceleration, users can speed up their model training and experimentation processes, reducing the time required to iterate on their projects.

Google Colab simplifies the process of setting up and managing dependencies by providing a pre-configured runtime environment with commonly used libraries pre-installed, including TensorFlow, PyTorch, scikit-learn, and more. Additionally, users have the flexibility to install additional packages using pip or conda directly within the notebook environment. This convenience streamlines the development process, allowing users to focus on building and experimenting with their models rather than worrying about software configurations.

Collaboration is a core aspect of Google Colab, with features that enable real-time collaboration and sharing of notebooks with colleagues or collaborators. Multiple users can work simultaneously on the same notebook, making it an ideal platform for team projects or collaborative research efforts. Furthermore, users can share their notebooks publicly or with specific individuals via a shareable link, facilitating knowledge sharing and collaboration within the community.

Google Colab seamlessly integrates with other Google services, particularly Google Drive, allowing users to import and export data directly from their Google Drive storage. This integration enhances data accessibility and facilitates the integration of external datasets into Colab notebooks. Additionally, users can access and manipulate data stored in Google Sheets, Google Cloud Storage, and other Google services within their Colab environment, further enhancing the platform's versatility.

In summary, Google Colab is a powerful and user-friendly platform that democratizes access to high-performance computing resources and facilitates collaborative development and experimentation with Python code. By providing free GPU and TPU access, pre-configured runtime environments, and seamless integration with Google Drive and other Google services, Google Colab empowers users to tackle complex projects, including machine learning, data analysis, and research, with ease and efficiency.

4.1.3 Ultralytics Library

Ultralytics is a versatile computer vision and deep learning library that prioritizes tasks like object detection, segmentation, and classification. It stands out for its implementation of YOLOv5, a highly efficient object detection model renowned for its speed and accuracy. YOLO (You Only Look Once) models are well-regarded for their ability to detect objects in real-time applications, making Ultralytics a valuable tool for developers and researchers in fields like autonomous vehicles, surveillance, and robotics.

One of the strengths of Ultralytics lies in its user-friendly interface, designed to simplify the process of training and deploying deep learning models. Whether you're a beginner or an experienced practitioner, Ultralytics' intuitive API allows for seamless navigation through tasks such as dataset preparation, model training, evaluation, and deployment. This accessibility lowers the barrier to entry for those new to computer vision, while still providing advanced capabilities for seasoned professionals.

The library offers support for various datasets commonly used in computer vision research, including COCO, Pascal VOC, and more. This flexibility enables users to work with diverse data sources, catering to a wide range of applications and research interests. Additionally, Ultralytics comes equipped with pre-trained models, allowing users to kickstart their projects without the need to train models from scratch. These pre-trained models serve as strong baselines for further customization and fine-tuning to specific tasks or datasets.

Ultralytics' modular design facilitates easy customization and extension of its functionality. Users can tailor the library to their specific requirements, integrating additional components or modifying existing ones as needed. This flexibility empowers researchers and developers to adapt Ultralytics to their unique use cases, fostering innovation and experimentation within the computer vision community.

As an integral part of the PyTorch ecosystem, Ultralytics seamlessly integrates with other PyTorch-based libraries and tools. This interoperability enables users to leverage the broader PyTorch ecosystem for deep learning tasks, tapping into a wealth of resources, frameworks, and pre-trained models. Furthermore, Ultralytics benefits from active development and a supportive community of users and developers. Regular updates and improvements ensure that the library remains at the forefront of computer vision research, while community engagement fosters collaboration and knowledge sharing among peers.

In summary, Ultralytics is a powerful and accessible library for computer vision tasks, particularly excelling in object detection, segmentation, and classification. Its implementation of YOLOv5, coupled with its user-friendly interface, extensive dataset support, modularity, and integration with the PyTorch ecosystem, makes it a valuable asset for researchers, developers, and enthusiasts alike.

4.1.4 Moviepy Module

MoviePy is a powerful Python library designed for video editing and manipulation tasks, offering a versatile set of features accessible through a simple and intuitive API. At its core, MoviePy facilitates a range of video editing operations, including cutting, concatenating, resizing, and rotating videos. These basic functionalities allow users to trim clips, merge multiple segments together, adjust dimensions, and correct orientations with ease. Whether you're a novice or an experienced video editor, MoviePy's straightforward interface makes these tasks accessible to users of all levels of expertise.

Beyond basic editing capabilities, MoviePy empowers users to apply a diverse array of effects to their videos, enhancing visual appeal and storytelling potential. From color adjustments to transitions, overlays, and text annotations, the library provides tools to transform raw footage into polished productions. Users can tweak brightness, contrast, and saturation, seamlessly transition between scenes, overlay images or text, and create dynamic text animations, all within the familiar Python environment.

Moreover, MoviePy enables the dynamic generation of videos through mathematical functions or image sequences, unlocking the potential for animations, visualizations, and simulations. By programmatically generating frames and assembling them into video clips, users can create custom content tailored to their specific needs or artistic vision. This functionality opens up new avenues for experimentation and creativity, allowing users to explore the intersection of code and visual storytelling.

In addition to video manipulation, MoviePy supports audio editing operations, further enriching the multimedia editing experience. Users can extract audio from video clips, merge audio files, adjust volume levels, and apply various audio effects, such as fades and equalization. This integration of audio editing capabilities within the same framework streamlines workflows and enhances the overall coherence of multimedia projects.

MoviePy's cross-platform compatibility ensures that users can harness its capabilities across different operating systems, including Windows, macOS, and Linux. Leveraging the power of FFmpeg for video encoding and decoding, the library ensures broad compatibility with various video formats and codecs, facilitating seamless integration into existing workflows.

Overall, MoviePy stands as a versatile and user-friendly tool for video editing and manipulation, catering to a wide range of users, from filmmakers and content creators to educators and researchers. With its accessible API, comprehensive documentation, and extensive feature set, MoviePy empowers users to realize their creative visions and bring their video projects to life, all within the familiar confines of the Python programming environment.

4.1.5 Java

Java is a versatile and widely-used programming language known for its portability, reliability, and robustness across various platforms and applications. Developed by Sun Microsystems in the mid-1990s, Java has since become one of the most popular programming languages in the world, powering a wide range of software applications, web services, mobile apps, and enterprise systems.

One of Java's key strengths is its platform independence, achieved through its "write once, run anywhere" philosophy. Java programs are compiled into bytecode, which can be executed on any platform with a Java Virtual Machine (JVM), making Java applications highly portable. This cross-platform compatibility has made Java a preferred choice for developing applications that need to run seamlessly across different operating systems and devices.

Java's strong emphasis on object-oriented programming (OOP) principles, such as encapsulation, inheritance, and polymorphism, facilitates the development of modular, reusable, and maintainable code. The language's rich standard library provides developers with a wide range of built-in classes and APIs for common tasks, including file I/O, networking, concurrency, and data manipulation, reducing development time and effort.

Additionally, Java's memory management model, based on automatic garbage collection, helps developers avoid memory leaks and other common pitfalls associated with manual memory management. This feature simplifies

memory management tasks and enhances the stability and reliability of Java applications, particularly in large-scale and long-running systems.

Java's ecosystem is vast and vibrant, with a wealth of third-party libraries, frameworks, and tools available to developers. Popular frameworks like Spring, Hibernate, and Apache Struts provide powerful abstractions and utilities for building enterprise-grade applications, while tools like Maven and Gradle streamline the build and dependency management process. The Java community is active and supportive, with numerous online resources, forums, and communities available for learning, collaboration, and knowledge sharing.

Moreover, Java's versatility extends to various application domains, from desktop software and web development to mobile app development (Android) and enterprise solutions. Its scalability, performance, and security features make it well-suited for developing mission-critical systems and high-performance applications, particularly in industries like finance, healthcare, and e-commerce.

In recent years, Java has continued to evolve and adapt to changing technology trends, with regular updates and new features introduced in successive versions of the language. Java's commitment to backward compatibility ensures that existing codebases remain relevant and maintainable, while ongoing innovations, such as Project Loom (concurrency improvements) and Project Panama (native interoperability), promise to further enhance Java's capabilities for future generations of developers.

Overall, Java's combination of portability, reliability, performance, and community support makes it a compelling choice for a wide range of software development projects, cementing its position as one of the most influential and enduring programming languages in the industry.

4.1.6 NumPy

NumPy is a fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. It forms the basis for many other libraries and tools in the Python scientific ecosystem. NumPy's primary data structure is the ndarray (N-dimensional array), which represents a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. These arrays enable efficient manipulation of large datasets, making NumPy indispensable for tasks ranging from numerical computations to data analysis and machine learning.

One of the key features of NumPy is its ability to perform array-oriented computing. This means that operations can be applied to entire arrays or large subsets of arrays without the need for explicit looping constructs. This approach not only simplifies code but also significantly improves performance, as many operations in NumPy are implemented in compiled C and Fortran code, resulting in faster execution compared to equivalent Python loops.

NumPy provides a rich set of functions for performing mathematical, logical, and statistical operations on arrays. These include basic arithmetic operations (addition, subtraction, multiplication, division), trigonometric functions, exponentiation, logarithms, and more. Additionally, NumPy offers functions for array manipulation, such as reshaping, concatenation, splitting, and indexing. These operations are optimized for speed and memory efficiency, making them ideal for handling large datasets efficiently.

Another crucial aspect of NumPy is its support for broadcasting, which allows arithmetic operations to be performed between arrays of different shapes. When operating on arrays of different shapes, NumPy automatically broadcasts the smaller array to match the shape of the larger array, thus eliminating the need for explicit looping or manual array manipulation.

NumPy also integrates seamlessly with other Python libraries and tools used in scientific computing and data analysis. For example, it serves as the foundation for libraries like SciPy, pandas, and scikit-learn, providing the underlying data structures and computational capabilities necessary for these libraries to function effectively. Moreover, NumPy arrays can be easily converted to and from other data structures commonly used in Python, such as lists and pandas DataFrames, facilitating interoperability with a wide range of tools and workflows.

In summary, NumPy is a powerful library that enables efficient manipulation of large arrays and matrices in Python. Its array-oriented computing paradigm, extensive collection of mathematical functions, support for broadcasting, and seamless integration with other libraries make it an essential tool for scientific computing, data analysis, and machine learning tasks in Python.

4.2 CODE

Code to store the current location in google colab home

```
import os
HOME=os.getcwd()
print(HOME)
```

To download the ultralytics library for yolo

```
! pip install ultralytics==8.0.196
```

Download the dataset from roboflow and import into our file

```
%cd {HOME}
!pip install roboflow --quiet
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="Vl5nUUAKOyul9My9ODQI")
project = rf.workspace("mohamed-traore-2ekkp").project("face-detection-mik1i")
dataset = project. version(18). download("yolov8")
```

To train the dataset with yolo

```
%cd {HOME}
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=65 imgsz=800
```

To validate the dataset

```
%cd {HOME}
!yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml
```

To do prediction on the required video to detect faces using yolov8

```
!yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source=/demo.mp4
--save_txt | tee output.txt
```

To install the required libraries for face verification and for parallel processing

```
!pip install deepface
!pip install moviepy
!apt-get update
!apt-get install openjdk-8-jdk-helpless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz
!tar -xvzf spark-3.5.1-bin-hadoop3.tgz
!pip install -q findspark
!pip install pyspark
```

To set the environment path

```
os.environ["JAVA_HOME"]="/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"]="/content/spark-3.5.1-bin-hadoop3"
```

Creating the Spark Application and Getting it

```
sc = SparkContext("local", "FaceDetectionWithSpark")
spark = SparkSession.builder.appName("FaceDetectionWithSpark").config("spark.executor.memory",
"8g").getOrCreate()
```

Verify the Target image with the frame in a video using deepface

```
result = DeepFace.verify(img1_path="/img1.jpg", img2_path="/home/store/cropped_image.jpg",
enforce_detection=False)
```

Combining all the frames with the target image to create a video

```
from moviepy.editor import ImageSequenceClip, VideoClip, TextClip
import os

# Path to the folder containing images
image_folder = '/home/store/cropped_images'

# Get a list of image files in the folder
image_files = [img for img in os.listdir(image_folder) if img.endswith(".jpg")]

# Sort the image files based on the numeric part of their names
image_files.sort(key=lambda x: int(".join(filter(str.isdigit, x))))

# Create a list of image paths
image_paths = [os.path.join(image_folder, img) for img in image_files]

# Set duration for each image frame (e.g., 0.5 seconds per frame)
image_duration = 1.5

# Set pause duration between frames (e.g., 1 second)
pause_duration = 0.0

# Create a list of clips for images and pauses
clips = []
for image_path in image_paths:
    image_clip = ImageSequenceClip([image_path], fps=24, durations=[image_duration])
    pause_clip = VideoClip(make_frame=lambda t: np.zeros((1, 1, 3)), duration=pause_duration)  # Black screen
as a pause
    clips.extend([image_clip, pause_clip])
```

```
# Concatenate clips into the final video
final_clip = concatenate_videoclips(clips, method="compose")

# Save the video file
output_path = '/face_detection2.mp4'
final_clip.write_videofile(output_path, codec="libx264", audio_codec="aac")
```

4.3 Performance Metric

Mean Average Precision (mAP) is a widely used performance metric in the field of object detection and information retrieval, particularly in the context of evaluating the accuracy of computer vision models. It is a comprehensive measure that takes into account both the precision and recall of a model across multiple classes or categories.

At its core, Average Precision (AP) measures the area under the precision-recall curve for a specific class or category. Precision represents the proportion of true positive detections among all positive detections, while recall measures the proportion of true positive detections among all ground truth instances. The precision-recall curve illustrates the trade-off between precision and recall at different thresholds for detecting objects.

To calculate AP for a single class, precision and recall values are computed at various confidence thresholds, and the area under the precision-recall curve is determined using methods such as the trapezoidal rule. The AP score ranges from 0 to 1, where higher values indicate better performance in terms of both precision and recall.

Mean Average Precision (mAP) extends AP to evaluate the overall performance of a model across multiple classes or categories. It computes the AP for each class individually and then takes the mean of these AP scores to obtain a single value representing the average performance across all classes. This approach provides a comprehensive assessment of the model's ability to detect objects across a diverse range of categories, accounting for variations in class frequencies and difficulty levels.

mAP is commonly used as a benchmark metric for comparing the performance of different object detection models and algorithms. It provides a holistic view of a model's performance, considering both the ability to accurately detect objects within each class and the ability to generalize across multiple classes. Higher mAP scores indicate better overall performance in object detection tasks, reflecting a balance between precision and recall across diverse categories. In this approach face detection using yolov8 is used and input is given as video along with target image and model is trained by using face dataset this gives a value of 90.1 mAp (Mean Average precision) value

CHAPTER 5

RESULTS AND SCREENSHOTS

In this approach face detection using yolov8 is used and input is given as video along with target image and model is trained by using face dataset this gives a value of 90.1 mAp (Mean Average precision) value and it is used as Train Set: 87% (2871 images), Valid Set: 8% (267 images), Test Set: 4% (145 images) and it is also preprocessed with auto-orient is applied along with resized to make stretch to 640x640. For face verification we used the deep face to match faces from face detection output with target input image and output frames are collected and merged into video using Moviepy module and frames processing is done using Apache spark. Output will be video consisting of target image. Incorporating Big data technology like Apache Spark for parallel processing of frames. Input image and video is processed and the frames of the video will be passed to Apache spark and it will be distributing the frames to processors and will be processing parallelly and finally returning the frames which match with the image.

Input Image

Fig5.1: Sample Input Image

Output Screenshots

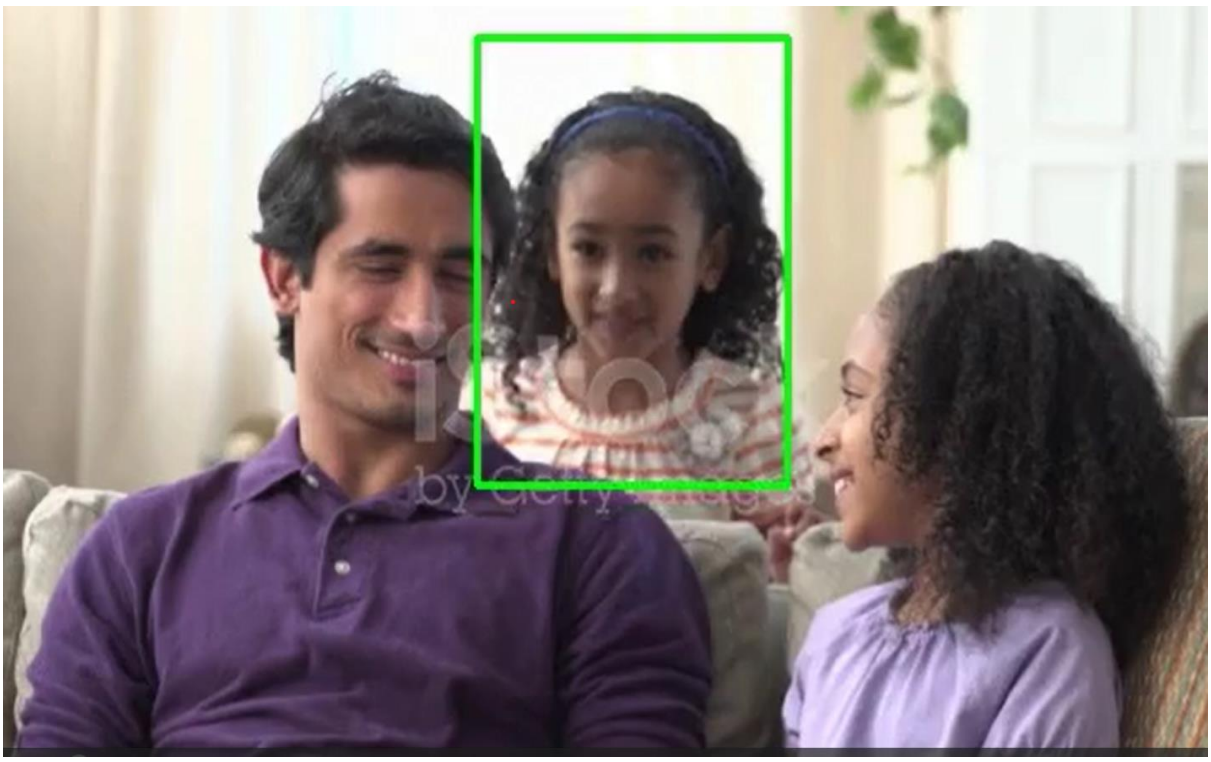

Fig5.2: Output screenshot-1 from output video

Fig5.3: Output screenshot-2 from output video

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In conclusion, our project on face detection in videos using big data tools and computer vision techniques highlights the transformative impact of leveraging advanced technologies to address real-world challenges. Computer vision, as a subfield of Artificial Intelligence, plays a crucial role in enabling machines to interpret and understand visual data, with applications ranging from security and surveillance to healthcare and entertainment. Throughout our investigation, we have emphasized the pivotal role of cutting-edge methodologies such as YOLOv8 for efficient face detection and DeepFace for robust face verification, showcasing the potential of these techniques in enhancing accuracy and scalability. Furthermore, our assessment and comparison of recent literature reviews utilizing OpenCV for human face detection underscore the versatility and accessibility of this widely-used library in addressing fundamental image processing tasks. By combining big data tools with state-of-the-art computer vision techniques, our project demonstrates the power of interdisciplinary collaboration in advancing the capabilities of intelligent systems and driving innovation in diverse domains. Looking ahead, the integration of emerging technologies and methodologies holds promise for further advancements in the field of computer vision, paving the way for more sophisticated and impactful applications in the future.

6.2 FUTURE WORK

In this approach we used to detect faces from video and to recognize the target image with detected faces in these there may be chance that it may take twins as same and detect while matching the images. In our approach, we employed face detection techniques to identify faces within video streams, followed by facial identification to match detected faces with target images. However, a challenge we encountered was the potential failure to recognize target images due to occlusions or partial obstructions of the faces. To address this limitation, future work will involve implementing image matching algorithms capable of handling occlusions, such as matching images with corresponding masks or dealing with partial facial features. Additionally, to enhance the efficiency of our system by leveraging more processors. By distributing computational tasks across multiple processors, we aim to achieve faster execution times, thereby improving the overall performance and scalability of our face detection and recognition system. This optimization strategy will be instrumental in meeting the demands of real-time applications and large-scale video processing tasks.

REFERENCES

[1] Peng, B., & Gopalakrishnan, A. K. (2019). A Face Detection Framework Based on Deep Cascaded Full Convolutional Neural Networks. 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS).

[2] Hao, Z., Feng, Q., & Kaidong, L. (2018). An Optimized Face Detection Based on Adaboost Algorithm. 2018 International Conference on Information Systems and Computer Aided Education (ICISCAE).

[3] Wati, D. A. R., & Abadianto, D. (2017). Design of face detection and recognition system for smart home security application. 2017 2nd International Conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE).

[4] Hsu, H.-W., Wu, T.-Y., Wong, W. H., & Lee, C.-Y. (2018). Correlation-Based Face Detection for Recognizing Faces in Videos. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[5] Rong Xie ,Qingyu Zhang, Enyuan Yang, Qiang Zhu(2019).A method of small Face Detection Based on CNN.2019 4th international Conference on Computational Intelligence and Applications(ICCIA).

[6] Shivalila Hangaragi, Tripty Singh, NeelimaN(2023), Face Detection and Recognition Using Face Mesh and Deep Neural Network. International Conference on Machine Learning and Deep Neural Network.

[7] Shahina Anwarul,Susheela Dahiya(2020). A Comprehensive Review on Face Recognition Methods and Factors Affecting Facial Recognition Accuracy.

[8] Pang, L., Ming, Y., & Chao, L. (2018). F-DR Net:Face detection and recognition in One Net. 2018 14th IEEE International Conference on Signal Processing (ICSP).

[9] Hao Yang and Xiaofeng Han .Face Recognization Attendence System Based on Real time Video Processing (2020).

[10] HtetAung ,Alexander V.Bobkov,Nyan LinTun. Face detection in real time video using Yolo algorithm based on Vgg16 ConvolutionalNeural Network. 2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM).

[11] Caifeng Shan(2010) Face Recognition and Retrieval in Video.

[12] Davis, M., Popa, S., & Surlea, C. (2010). Real-Time Face Recognition from Surveillance Video. In Intelligent Video Event Analysis and Understanding (1st ed., pp. 155-194). (Studies in Computational Intelligence; Vol. 332). Springer

[13] Ma, M., & Wang, J. (2018). Multi-View Face Detection and Landmark Localization Based on MTCNN. 2018 Chinese Automation Congress (CAC).

[14] Cuimei, L., Zhiliang, Q., Nan, J., & Jianhua, W. (2017). Human face detection algorithm via Haar cascade classifier combined with three additional classifiers. 2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI).

[15] Payal Bose1, Samir K Bandhyopadhyay1 and Vishal Goyal(2020) Human Different Head Poses and Facial Expression Analysis using Principal Component Analysis. 2020 Xeno Journal of Biomedical Sciences.