A

Mini Project Synopsis Report

On

Face Liveliness: Detection

For

partial fulfillment of award of the

B. Tech Degree in Information Technology

Under the Supervision of

(Mr. Arun Kumar Takuli)

Submitted by:

Faaeiz Khan	**2201920130065**
Gagan Chauhan	**2201920130066**
Jonty Gupta	**2201920130079**

Session: 2024-2025

G. L. Bajaj Institute of Technology and Management,

Greater Noida

GL BAJAJ
Institute of Technology & Management

Department of Information Technology

Declaration

We herewith declare that the project work conferred during this report entitled “Trusty Tours”, in partial fulfillment of the necessity for the award of the degree of Bachelor of Technology in Information Technology, submitted to A.P.J. Abdul Kalam Pradesh Technical University, Uttar Pradesh, is an authentic record of our own work distributed in Department of Information Technology & Engineering, G.L. Bajaj Institute of Technology & Management, Greater Noida. It contains no material antecedently printed or written by another person except wherever due acknowledgement has been created within the text. The project work reported during this report has not been submitted by us for award of the other degree or certification.

Signature:

Name: Mayank Gaur

Roll No : 2201920130099

Signature:

Name: Mohd. Laraib

Roll No : 2201920130107

Signature:

Name: Mayank Yadav

Roll No : 2201920130100

Signature:

Name: Mohd. Anas

Roll No: 2201920130103

Date:

Place: Greater Noida

GL BAJAJ
Institute of Technology & Management

Department of Information Technology

Certificate

This is to certify that Project Report entitled "Trusty Tours" that is submitted by Mohd. Laraib, Mohd. Anas, Mayank Gaur and Mayank Yadav in partial fulfillment of the necessity for the award of degree B. Tech. in Department of Information Technology of Abdul Kalam Technical University, are record of the candidate own work distributed by him below our oversight. The matter embodied during this thesis is original and has not been submitted for the award of the other degree.

Date:

Mr. Arun Kumar Takuli **Dr. P C Vashist**

(Assistant Professor) Head of Department

GL BAJAJ
Institute of Technology & Management

Department of Information Technology

Acknowledgement

We would like to express our sincere thanks to our project supervisor Mr. Arun Kumar Takuli and our Head of Department Dr. P.C Vashist for their invaluable guidance and suggestions. This project helped to us to understand the concept of MERN Stack. This project enriches our knowledge and experience of working in a team and a live project. Also, we would like to express gratitude to Mr. Arun Kumar Takuli for his help in preparation and overview of our project.

Lastly, we would like to thank all the faculties for providing their valuable time whenever needed for helping us carry on with our project.

TABLE OF CONTENTS

1. INTRODUCTION

Face liveness detection is an essential aspect of modern biometric systems, designed to ensure the authenticity of individuals attempting to access secure systems. With the rapid proliferation of facial recognition technology, the need for effective countermeasures against spoofing attacks has never been greater. Spoofing attacks, where an individual attempts to deceive a facial recognition system using photos, videos, or 3D masks, pose significant threats to the security of these systems. To address these vulnerabilities, face liveness detection systems have emerged, designed to differentiate between real, live faces and artificial representations. This project focuses on building a robust face liveness detection model, capable of identifying and preventing such attacks.

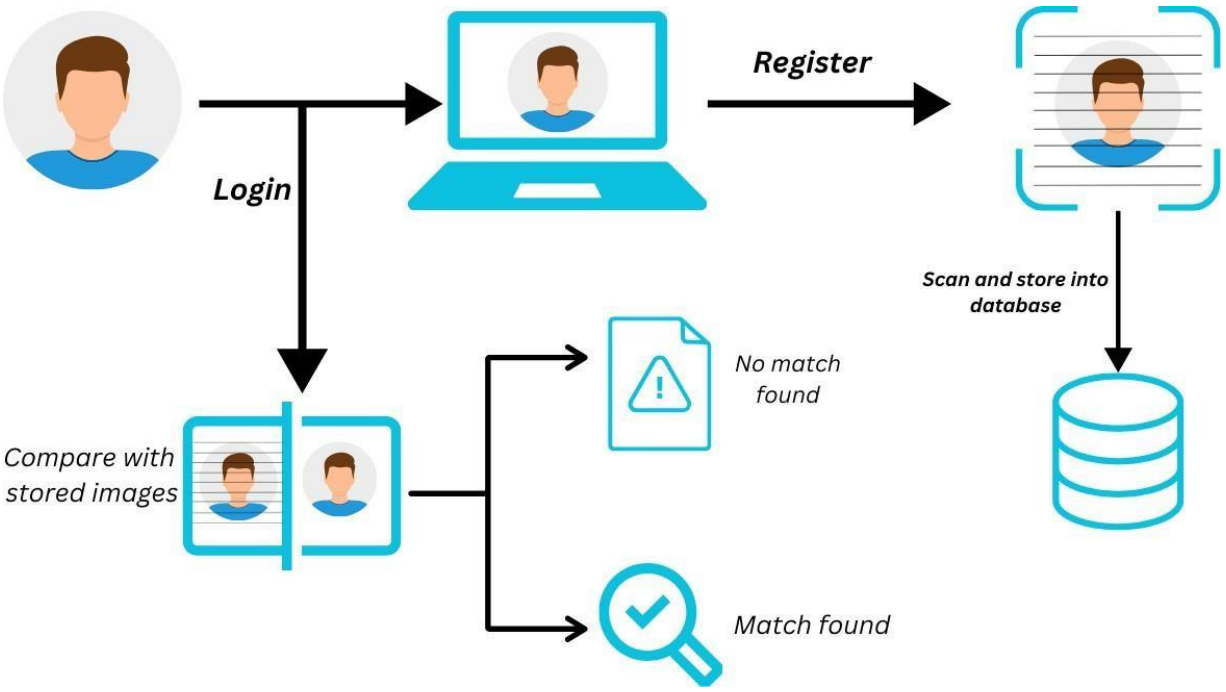

Fig 1. Flowchart showing the process of user registration and login.

1.1 Overview of Face Liveness Detection

Face liveness detection refers to the process of distinguishing between a real, live human face and an artificial object or media that mimics it. In a typical facial recognition system, liveness detection ensures that the individual presenting their face for recognition is physically present, rather than using a photograph, video, or a 3D model. This is especially important in applications such as secure device unlocking, banking, border control, and identity verification, where the integrity of the recognition system is paramount. The technology leverages subtle characteristics of the human face, such as texture, motion, and reaction to stimuli, to make this distinction.

While traditional face recognition systems primarily focus on identifying a person's identity, face liveness detection adds an additional layer of security by ensuring the validity of the individual attempting to authenticate. This is particularly crucial in areas like mobile devices, where facial recognition is used to unlock phones, and in border security systems where face recognition is used for passport control and immigration checks.

1.2 Motivation for the Project

As face recognition technology becomes more ubiquitous, the threats posed by spoofing attacks have become more sophisticated. With the advent of high-quality images, videos, and deepfake technology, the ability to deceive face recognition systems has become easier for attackers. Such attacks undermine the security of face-based authentication systems and can lead to unauthorized access to sensitive information or areas.

The motivation behind this project is to develop a robust solution for face liveness detection that can effectively combat these spoofing attempts. With a focus on real-time detection and leveraging pre-trained AI models, the goal is to create a system that can accurately differentiate between live faces and spoofed representations, thus enhancing the security of facial recognition systems in practical applications.

1.3 Project Objectives

The primary objective of this project is to design and implement a face liveness detection system that integrates a pre-trained AI model with custom input data to improve accuracy and

performance. This involves utilizing the power of machine learning to train a model capable of recognizing subtle facial features and movements that indicate liveness.

Some of the key objectives of this project include:

1. **Real-time Face Liveness Detection**: Building a system that can process facial data in real-time to verify the authenticity of the user.
2. **Accuracy Improvement**: Achieving high detection accuracy to minimize false positives and false negatives.
3. **Utilizing Pre-Trained Models**: Leveraging existing AI models that have been pre-trained on large datasets to speed up the development process and enhance the system's performance.
4. **Deployment in Real-World Applications**: Ensuring the system is practical and can be integrated into real-world applications such as mobile devices, banking systems, and secure access points.

1.4 Need for Visual Representation

While the introduction is primarily textual, it can be beneficial to include a visual aid such as a diagram or flowchart in this section to give a clear, immediate understanding of the process. For example:

- **Diagram of the Face Liveness Detection Process**: A simple flowchart showing how face liveness detection works, from capturing an image to distinguishing live faces from spoofed ones, would be useful. This provides a high-level overview of the system to complement the textual explanation.
- **Spoofing vs. Real Face Illustration**: A visual comparing a real human face and an artificial representation (like a photo or 3D model) can help readers understand the challenges of detecting spoofing attack.

1.5 Historical Incidents Underscoring the Need

Several high-profile incidents have underscored the urgent need for face liveness detection:

1. **Deepfake CEO SCAM (2019)**

   A chilling example of the power of deepfakes occurred in 2019 when cybercriminals us AI to mimic the voice and face of a company CEO. This deepfake was used to trick employees into transferring large sums of money, resulting in a financial loss of over €220,000.

2. **Political and Social Manipulation**

   Deepfake videos of political leaders making fake statements have been used to spread misinformation and create social unrest. One notable example includes a deepfake of Facebook CEO Mark Zuckerberg boasting about control over user data, which was created to demonstrate how easily misinformation could be spread.

2. BACKGROUND

Face liveness detection has emerged as a critical component in biometric security systems due to the increasing sophistication of identity spoofing techniques. To understand its development and necessity, it is essential to explore the existing solutions for facial recognition, their evolution, and the challenges they face.

Existing Biometric Solutions

Biometric authentication systems leverage unique physiological or behavioural characteristics, such as fingerprints, iris patterns, voice, and facial features, to verify identity. Among these, facial recognition has gained widespread adoption due to its non-intrusive nature and ease of use. Some of the most notable applications include:

1. **Smart Devices**: Face unlock features in smartphones (e.g., Apple's Face ID, Android Face Unlock) offer a quick and convenient way to secure personal devices.
2. **Banking and Finance**: Many financial institutions employ facial recognition to enhance security for online transactions and account access.
3. **Government and Law Enforcement**: Governments use facial recognition for border security, surveillance, and identifying individuals in criminal investigations.

Despite its advantages, facial recognition systems have inherent vulnerabilities that can be exploited by attackers.

Early Spoofing Techniques and Initial Countermeasures

Facial recognition systems initially struggled with simple spoofing attacks involving photographs, videos, or even 3D masks of legitimate users. To counter these threats, developers introduced basic liveness detection techniques:

1. **Blink Detection**: Systems were programmed to detect natural blinking as a sign of a live user.
2. **Head Movement Detection**: Users were asked to perform specific actions, such as turning their head or smiling, to prove their presence.

3. **Passive Liveness Detection**: Leveraging cues like light reflection, shadow patterns, or skin texture to differentiate between real faces and static images.

While these methods offered some level of security, they had several shortcomings. For example, blinking and head movements could be easily replicated using high-quality videos or animated deepfakes. Similarly, passive techniques struggled to detect sophisticated 3D mask attacks or high-resolution synthetic images.

The Emergence of Deepfakes: A New Threat

Deepfake technology has revolutionized identity spoofing by enabling the creation of hyper-realistic synthetic media. Unlike traditional spoofing techniques, deepfakes can generate dynamic content, simulating a person's face and movements in real-time. This poses a significant threat to biometric systems, as deepfakes can bypass traditional liveness detection mechanisms.

Real-World Impact of Deepfakes:

- **Identity Fraud**: Attackers can use deepfake videos to impersonate individuals and gain unauthorized access to secure systems.
- **Social Engineering**: Deepfakes can be used to manipulate individuals into divulging sensitive information or performing harmful actions.
- **Misinformation and Defamation**: Deepfake videos of public figures making false statements can damage reputations and spread misinformation.

Shortcomings of Existing Liveness Detection Systems

Despite advancements in face liveness detection, current systems face several limitations:

1. **Reliance on Static Features**: Many systems focus on analyzing static features like texture or light reflection, which deepfakes can replicate with high accuracy.
2. **Limited Response to Sophisticated Attacks**: Advanced 3D masks and real-time deepfake videos can bypass traditional liveness checks.
3. **High False Acceptance and Rejection Rates**: Existing systems often struggle to balance security and usability. Overly strict measures can lead to false rejections, frustrating legitimate users, while lenient systems may accept spoofed inputs.

4. **Lack of Adaptability**: As deepfake technology evolves, many liveness detection systems fail to adapt quickly enough, leaving them vulnerable to emerging threats.
5. **Performance and Scalability Issues**: High-security liveness detection methods, such as 3D depth sensing, require specialized hardware, which can be costly and impractical for widespread deployment.

3. LITERATURE REVIEW

The domain of **face recognition** and **anti-spoofing** has garnered significant attention in recent years, driven by the widespread adoption of facial biometrics for security applications, including attendance systems, mobile phone unlocking, and access control systems. While face recognition technologies have made significant advancements, **spoofing attacks** continue to present a major security challenge. The effectiveness of face recognition systems can be severely compromised by spoofing techniques such as presenting photographs, video playback, or 3D models in front of the camera, which can deceive traditional recognition systems into granting access. As a result, **anti-spoofing** technologies have become a crucial aspect of securing face recognition systems.

Spoofing Attacks in Face Recognition

Spoofing refers to the use of fraudulent data or techniques to trick biometric systems. In the context of face recognition, **spoofing attacks** can take various forms, including:

1. **Printed photographs** – The simplest form of spoofing, where an attacker presents a printed image of a legitimate user's face to the camera.
2. **Video replay attacks** – Where a recorded video of a person's face is played in front of the camera to impersonate the legitimate user.
3. **3D mask attacks** – More sophisticated spoofing, where 3D masks designed to resemble the target person's face are used to deceive the system.

Face recognition systems rely on capturing the facial features of users and comparing them with stored templates. However, the reliance on **static 2D images** in traditional systems makes them vulnerable to such attacks, highlighting the need for **liveness detection** to distinguish between real and spoofed faces.

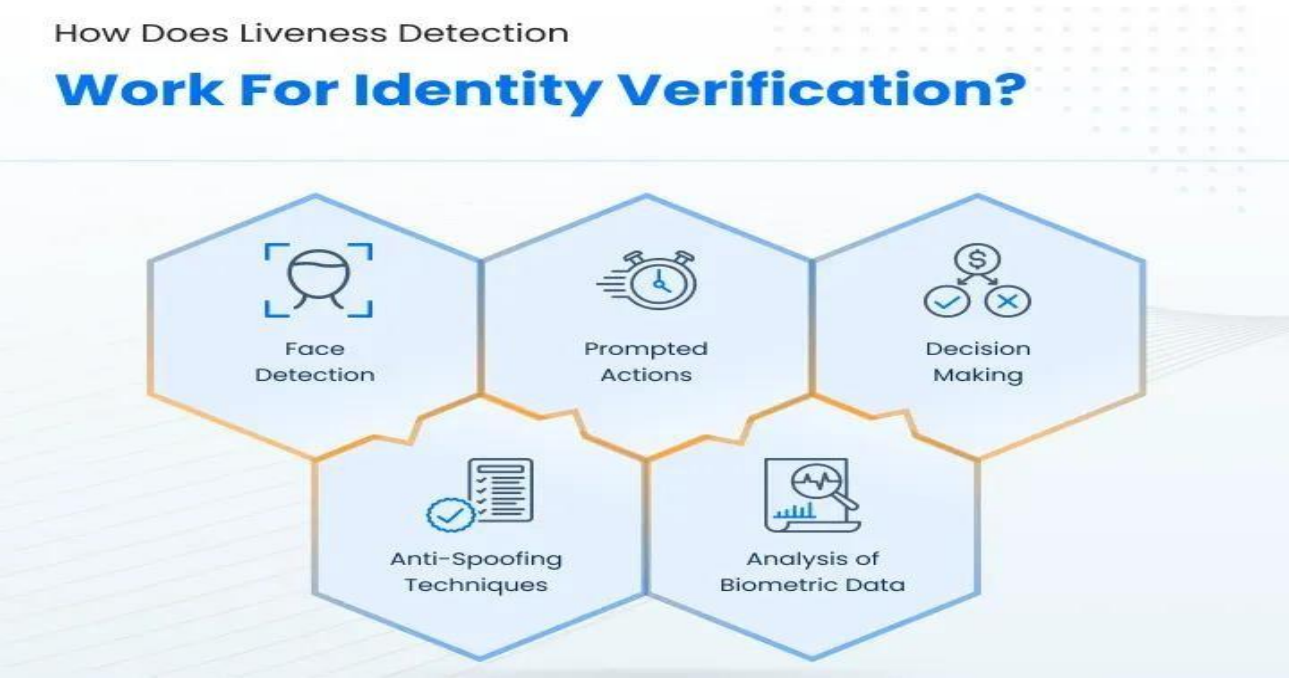

Fig 2. Overview of liveliness detection

Existing Techniques in Spoof Detection

Numerous methods have been developed over the years to improve the resilience of face recognition systems against spoofing attacks. These methods generally fall into two categories: **software-based approaches** and **hardware-based approaches**.

1. **Texture-Based Detection**:
   Early techniques focused on analyzing the texture of the skin to detect inconsistencies between live faces and spoofed images. For instance, **liveness detection** methods rely on features such as **skin reflectance** or the **presence of a 3D structure** to differentiate between a real face and a flat 2D image or video.
2. **Motion-Based Detection**:
   Another widely researched approach is based on detecting motion cues that are present in live face video but absent in static images. This includes detecting **blinking** or **head movements**. Techniques like **pulsatile light reflection** and **dynamic texture analysis** are used to evaluate subtle motion in the skin or face, which can provide an indication of whether the face is real or spoofed.

3. **Depth and IR-Based Detection**:

   More advanced systems use **infrared (IR) sensors** or **depth cameras** to capture more than just the visible light spectrum. These systems are capable of capturing **depth information** of the face, which makes it much harder to spoof with a 2D image. Such technologies are often more effective in detecting spoofed faces using photographs or videos, as they can assess the 3D structure of the face.

4. **Machine Learning Approaches**:

   Recently, machine learning techniques, particularly **deep learning** algorithms, have been employed to enhance spoof detection. Convolutional Neural Networks (CNNs) and **Recurrent Neural Networks (RNNs)** have been used to train models that can automatically detect fake faces based on a variety of facial features. These models learn to identify subtle differences in **skin texture, depth**, and **dynamic characteristics** that indicate whether a face is real or a spoof.

   - **Silent Face Anti-Spoofing**:

     One such method is **silent face anti-spoofing**, which utilizes deep learning models to detect subtle differences in facial features that can distinguish a live face from a spoofed one. This method often works by analyzing small details such as **micro-textures**, skin reflectance patterns, and **non-rigid motions** like blinking and lip movement. The primary advantage of silent face anti-spoofing is that it does not require additional hardware like infrared sensors or depth cameras, making it cost-effective and suitable for integration into existing face recognition systems.

Face Recognition and Attendance Systems

While anti-spoofing techniques are crucial for security, face recognition itself plays a critical role in automated systems. Face recognition systems have been widely adopted for applications such as **attendance management**, **security verification**, and **user authentication**. These systems work by capturing images of a person's face, extracting unique features (such as the distance between eyes, nose shape, and jawline), and comparing them with stored data to verify identity.

The implementation of **face attendance systems** leverages traditional face recognition technology to automate processes like tracking employee attendance, verifying students' presence, and even for biometric authentication in secure environments. These systems offer a significant advantage over traditional methods, such as passwords or RFID cards, as they provide a more seamless, contactless, and secure solution.

However, despite the widespread adoption of face recognition in attendance systems, **security concerns** remain. Face recognition systems are vulnerable to **image-based spoofing attacks**, where a person presents a photo or video of a registered user to gain unauthorized access. This is especially problematic in environments where security is paramount, such as **workplaces, schools**, and **government buildings**.

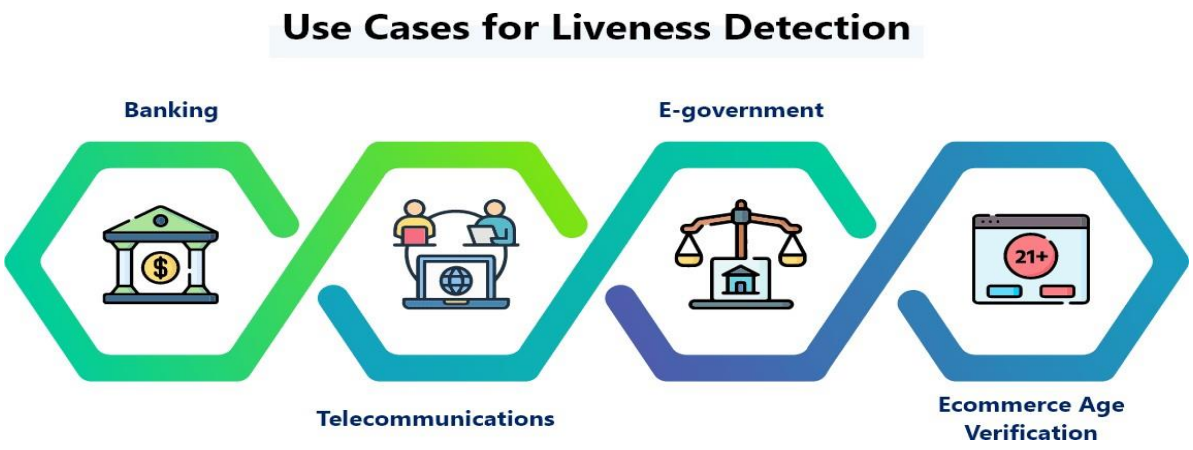

Fig 3. Use cases for liveliness detection

Spoof Detection Libraries and Resources

Several software libraries and tools have been developed to address the issue of spoofing in face recognition systems. Some of the notable libraries include:

1. **Face Recognition Libraries**:
   Libraries like **OpenCV**, **Dlib**, and **DeepFace** are popular for building face recognition systems. They offer pre-trained models for detecting and recognizing faces, and they can be easily integrated with various frameworks and platforms. However, these libraries do not inherently include spoof detection mechanisms, which is why additional anti-spoofing algorithms are needed.

2. **Silent Face Anti-Spoofing**:

   As mentioned earlier, **silent face anti-spoofing** techniques have become a significant advancement in the fight against spoofing. By leveraging deep learning models trained on large datasets of real and spoofed faces, these systems can effectively detect whether a face is live or fake, even without relying on expensive hardware.

3. **Spoof Detection Integration**:

   Many face attendance systems and face recognition libraries now integrate spoof detection models, such as the **silent face anti-spoofing** model. These systems are designed to be lightweight and can be deployed on standard devices (such as laptops, smartphones, or webcams) without requiring additional specialized hardware. The integration of spoof detection directly into face recognition systems is essential for improving security, especially in critical applications like attendance tracking.

Challenges and Future Directions

While many advancements have been made, there are still challenges in the field of face recognition and anti-spoofing. Some of the challenges include:

- **Accuracy**:

  Spoof detection systems must achieve a high level of accuracy to prevent false positives and false negatives. Even minor errors in spoof detection can result in unauthorized access or legitimate users being denied access.

- **Real-timeProcessing**:

  For face attendance systems, spoof detection needs to work in real-time to ensure that users can authenticate quickly without delays. This can be computationally expensive, especially when dealing with deep learning models.

- **Dataset Diversity**:

  Anti-spoofing models are often trained on specific datasets, which can limit their ability to generalize to different environmental conditions, lighting, or angles. There is a need for larger, more diverse datasets to improve the robustness of spoof detection systems.

- **Integration with Existing Systems**:
  Many face recognition systems are already deployed in various applications, and integrating spoof detection into these existing systems without significant hardware upgrades or software modifications is a major challenge.

4. METHODOLOGY

The methodology section describes the approach and steps taken to develop and implement the **Face Attendance System** with a focus on **liveness detection** (anti-spoofing). This system uses a combination of image capture, face recognition, and spoof detection techniques, ensuring only real users are authenticated.

1. System Architecture Overview

The face attendance system leverages a **web-based application** with integrated **real-time face recognition** and **anti-spoofing** techniques. The architecture is based on the following components:

- **Webcam Integration**: Capturing live images from users for both registration and login.
- **Face Recognition Model**: Identifying users by comparing captured images with stored ones.
- **Spoof Detection**: Determining whether the captured face is a real person or a spoofed image (e.g., a photo or video).
- **Backend Processing**: Using a Python API that interacts with the face recognition model and handles user data securely.
- **Frontend Interface**: A React-based user interface (UI) that allows users to register, log in, and view notifications.

2. Data Collection and Preprocessing

- **Image Capture for Registration**: When a new user registers, the system prompts them to **smile at the camera**. A **selfie** is taken and stored locally. The user must also provide a **username** that is associated with the captured image.
- **Preprocessing**: Images are preprocessed for clarity and uniformity. This includes resizing, face detection, and normalization to ensure consistent input for the face recognition system.

3. Face Recognition Process

- **Face Detection**: Using **OpenCV** or **Dlib**, the system first detects the presence of a face in the image or video stream. The **Haar Cascade Classifier** or **CNN-based** face detectors are employed to locate the face.

- **Feature Extraction**: Once the face is detected, facial features are extracted using pre-trained models such as **FaceNet** or **DeepFace**. These models convert the face into a unique embedding—a numerical representation of the face.
- **Comparison**: The captured face embedding is compared with the stored embeddings of registered users. If a match is found, the user is authenticated and allowed to log in.

4. Spoof Detection (Anti-Spoofing)

- **Spoof Detection Method**: The system includes an anti-spoofing mechanism, referred to as **silent face anti-spoofing**, which detects whether the face being presented is a real person or a spoofed image. This is achieved through:
    - **Texture Analysis**: Analyzing the texture and lighting of the image to detect inconsistencies between a real face and a printed photo or video.
    - **Motion Detection**: Observing subtle movements like blinking or head-turning. A real person naturally exhibits small, involuntary movements, which can be detected by the system using frame-by-frame analysis.
    - **IR or Depth-based Sensing**: For more advanced spoof detection, depth-based cameras or infrared sensors can be used to detect the 3D structure of the face, making it difficult to spoof with a 2D image.

5. Spoofing Testing and Evaluation

- **Testing with Fake Images**: Various types of spoofing attempts, such as holding up a printed photograph of the user or playing a video on the screen, were tested against the system to evaluate its ability to detect spoofing.
- **Spoof Detection Metrics**: The accuracy of spoof detection is measured using standard metrics like **False Acceptance Rate (FAR)** and **False Rejection Rate (FRR)**. A low FAR indicates that the system rarely accepts spoofed faces as genuine, and a low FRR indicates that real faces are accurately authenticated.

6. User Interface Design

- **Login and Registration Windows**: The front-end interface allows users to register by smiling at the camera and logging in by presenting their face for comparison with the stored image.

- **Error Handling**: The system provides feedback to the user if authentication fails (e.g., "unknown user" message), and guides them to either retry or register a new account.

7. Backend and Frontend Implementation

- **Backend**: The backend is developed using **Python** and involves creating a RESTful API to handle user authentication, image storage, and interaction with the face recognition model.
- **Frontend**: The frontend is implemented using **React**, providing a responsive UI for user interaction. The frontend communicates with the backend via HTTP requests to manage user sessions and display the results.

8. Testing and Validation

- **Test Scenarios**: The system was tested using both **real and fake user images** to check its ability to differentiate between a live face and a spoofed one. The goal was to ensure that the spoof detection feature could identify fake attempts accurately.
- **Validation**: The system was tested on various hardware configurations to ensure compatibility and reliability.

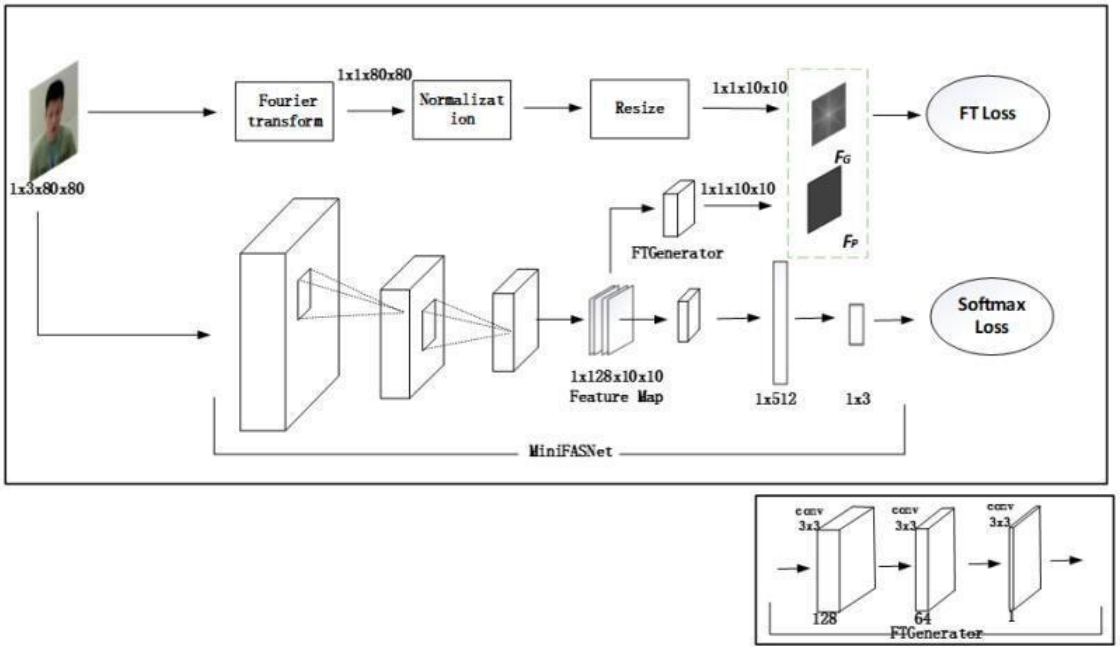

Fig 4. Working face detection ML Model

Tech Stack Overview

The tech stack used for the face attendance system with spoof detection includes **Python**, **OpenCV**, **HTML**, **CSS**, and **JavaScript**. Each of these technologies plays a vital role in ensuring the functionality, performance, and user experience of the application.

1. Python (Backend)

Python is the backbone of the system, handling the backend logic and operations. It is widely used for its simplicity, readability, and large ecosystem of libraries. Python is well-suited for tasks like data processing, image recognition, and machine learning, making it ideal for this project where face recognition and spoof detection are core components.

- **Why Python is Useful:**
  - Python has a vast collection of libraries and frameworks, such as **OpenCV** for computer vision tasks and **Flask** or **Django** for building web applications.
  - It is known for rapid development, making it perfect for building and prototyping the backend of the system quickly.
  - Python's ease of use and readability allow for efficient coding and maintenance of complex algorithms, such as face recognition.

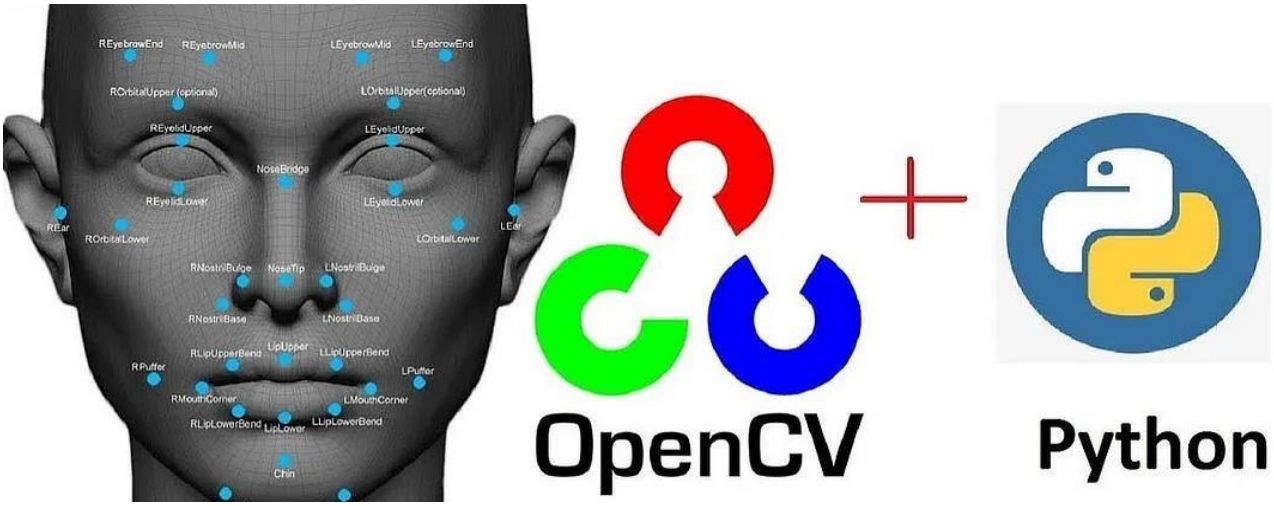

Fig 5. Technology used Images

2. OpenCV (Face Recognition and Spoof Detection)

OpenCV (Open Source Computer Vision Library) is a powerful library used for real-time computer vision applications. It is used in this project for the face recognition process, enabling the system to detect, identify, and verify faces based on the captured images.

- **Why OpenCV is Useful:**
  - OpenCV provides a large number of pre-built algorithms for face detection, which simplifies the process of integrating face recognition into the system.
  - It allows for fast processing of images and video streams, making it suitable for real-time applications like face attendance systems.
  - OpenCV also supports deep learning models, which are useful for spoof detection and distinguishing between real faces and spoofed images (e.g., printed photos).

3. HTML (Frontend Structure)

HTML (HyperText Markup Language) is the standard language used to create and design the structure of web pages. It provides the skeleton for the user interface, enabling users to interact with the system, such as logging in, registering, and viewing success or failure messages.

- **Why HTML is Useful:**
  - HTML is essential for building web applications by defining the structure and content of web pages.
  - It integrates seamlessly with other frontend technologies like **CSS** and **JavaScript** to create a dynamic and interactive user interface.

4. CSS (Styling)

CSS (Cascading Style Sheets) is used to control the appearance and layout of the web application. It allows for designing the user interface by styling elements such as buttons, text fields, forms, and images, making the application visually appealing and easy to use.

- **Why CSS is Useful:**
    - CSS ensures that the application is user-friendly, with a clean, organized, and visually consistent design.
    - It enables responsiveness, meaning the application can adapt to different screen sizes (such as mobile or desktop), providing an optimal user experience.

5. JavaScript (Frontend Interactivity)

JavaScript is a programming language that adds interactivity to web pages. In this system, JavaScript is used to handle dynamic actions, such as capturing images via the webcam, sending requests to the server for authentication, and updating the UI based on responses (e.g., successful login or spoof detection).

- **Why JavaScript is Useful:**
    - JavaScript enables real-time interaction between the user and the system without reloading the page, making the experience smoother and faster.
    - It provides essential features like handling user inputs, validating forms, and managing asynchronous requests, all of which are critical for creating a responsive, interactive web application.

Why This Tech Stack is Useful for the Face Attendance System

- **Scalability and Flexibility**: The combination of **Python**, **OpenCV**, and web technologies like **HTML**, **CSS**, and **JavaScript** makes the system highly scalable and flexible. Python can handle complex algorithms for face recognition and spoof detection, while the frontend technologies enable easy user interaction and responsiveness.
- **Ease of Development**: Python's simplicity, combined with the powerful features of **OpenCV**, allows for faster development, particularly for complex tasks such as facial feature extraction and comparison.
- **Cross-platform Compatibility**: With **HTML**, **CSS**, and **JavaScript**, the application can run across various platforms, ensuring that users on different devices can easily use the system.
- **Real-Time Performance**: **OpenCV** allows for efficient image processing and real-time performance, which is essential for applications involving live face recognition and spoof detection.

5. Conclusion and Future Scope

The **Face Attendance System with Liveness Detection** has been developed as a robust solution to address spoofing issues in face recognition systems. This section summarizes the project's achievements, highlights its importance, and explores potential enhancements for future development.

Conclusion

The project successfully demonstrates a working prototype of a face attendance system with integrated spoof detection. Key accomplishments include:

1. **Secure Authentication**:
   The system ensures that only genuine users can log in by leveraging **real-time face recognition** and **anti-spoofing mechanisms**. Spoofing attempts, such as using printed photos or videos, were effectively detected.
2. **Ease of Use**:
   The user-friendly interface allows for seamless registration and login. Both **new user registration** and **authentication** processes are simplified through webcam integration.
3. **Real-Time Performance**:
   The system achieves real-time processing for both face recognition and spoof detection, ensuring minimal delay during login and registration operations.
4. **Cross-Platform Compatibility**:
   The system’s backend is designed using Python, and the frontend utilizes React, making it deployable on various platforms, including cloud services like **AWS EC2**.
5. **Testing and Validation**:
   Extensive testing with both real and spoofed images confirmed the reliability and accuracy of the system. This reinforces the project's potential to be deployed in real-world applications.

Future Scope

Despite its current success, the project has room for improvement and expansion. Below are potential areas for future work:

1. **Enhanced Spoof Detection Techniques**:
    - Incorporate **3D depth sensing** or **infrared cameras** to improve spoof detection accuracy.
    - Explore **machine learning models** trained on diverse spoofing datasets for enhanced performance against complex attacks like 3D masks or high-quality videos.
2. **Scalability**:
    - Extend the system to support **larger databases** for enterprise-level applications.
    - Optimize algorithms for handling thousands of users while maintaining quick response times.
3. **Mobile Integration**:
    - Develop a mobile application version to provide face attendance functionality on the go.
    - Use smartphone cameras with built-in depth sensors for improved liveness detection.
4. **Multi-Factor Authentication (MFA)**:
    - Introduce additional security layers such as **OTP verification** or **voice recognition** for high-security environments.
5. **Performance Improvements**:
    - Optimize backend code to reduce processing time and improve system efficiency.
    - Investigate using **edge computing** to process data locally, reducing dependency on cloud servers.
6. **User Feedback and Adaptation**:
    - Implement a feedback mechanism for users to report issues or suggest improvements.

6. CODES

Main.py:

```
import os
import datetime
import pickle
import subprocess
import tkinter as tk
from tkinter import messagebox
import cv2
from PIL import Image, ImageTk
import face_recognition
import util  # Assuming util module is implemented and working

class App:
    def__init_(self):
        self.main_window = tk.Tk()
        self.main_window.title("Face Recognition App")
        self.main_window.geometry("1200x520+350+100")

        # Buttons
        self.login_button_main_window = util.get_button(self.main_window, 'Login', 'green',
self.login)
        self.login_button_main_window.place(x=750, y=200)

        self.logout_button_main_window = util.get_button(self.main_window, 'Logout', 'red',
self.logout)
        self.logout_button_main_window.place(x=750, y=300)

        self.register_new_user_button_main_window  =  util.get_button(
            self.main_window, 'Register New User', 'gray', self.register_new_user, fg='black'
        )
        self.register_new_user_button_main_window.place(x=750, y=400)
```

```
        # Webcam Label
        self.webcam_label = util.get_img_label(self.main_window)
        self.webcam_label.place(x=10, y=0, width=700, height=500)

        # Webcam Initialization
        self.cap = cv2.VideoCapture(0) # Use index 0 for the default camera
        if not self.cap.isOpened():
            messagebox.showerror("Error", "Webcam could not be initialized. Check the camera connection.")
            self.main_window.destroy()
            return

        self.process_webcam()

        # Directory Setup
        self.db_dir = './db'
        if not os.path.exists(self.db_dir):
            os.mkdir(self.db_dir)

        self.log_path = './log.txt'

    def process_webcam(self):
        ret, frame = self.cap.read()
        if ret:
            self.most_recent_capture_arr = frame
            img_ = cv2.cvtColor(self.most_recent_capture_arr, cv2.COLOR_BGR2RGB)
            self.most_recent_capture_pil = Image.fromarray(img_)
            imgtk = ImageTk.PhotoImage(image=self.most_recent_capture_pil)
            self.webcam_label.imgtk = imgtk
            self.webcam_label.configure(image=imgtk)
        else:
            messagebox.showerror("Error", "Failed to read from webcam. Restart the application.")
```

```
 self.webcam_label.after(20, self.process_webcam)
def login(self):
        if not hasattr(self, 'most_recent_capture_arr'):
            messagebox.showerror("Error", "No webcam feed detected. Ensure the camera is
functioning.")
            return

        unknown_img_path = './.tmp.jpg'
        cv2.imwrite(unknown_img_path, self.most_recent_capture_arr)

        try:
            # Use face_recognition CLI to check for a match
            output = subprocess.check_output(['face_recognition', self.db_dir, unknown_img_path],
text=True).strip()
            print("Face Recognition Output:", output)  # Debugging line

            # Extract name from the output
            name = output.split(',')[1][:-3]

            # Handle unknown cases
            if name in ['unknown_person', 'no_persons_found']:
                util.msg_box('Error', 'Unknown user. Please register as a new user or try again.')
            else:
                util.msg_box('Welcome back!', f'Welcome, {name}.')
                with open(self.log_path, 'a') as f:
                    f.write(f'{name},{datetime.datetime.now()},in\n')

        except Exception as e:
            messagebox.showerror("Error", f"Face recognition failed: {str(e)}")
        finally:
            os.remove(unknown_img_path)
```

```
def logout(self):
        util.msg_box('Logged Out', 'You have been successfully logged out.')
        self.main_window.destroy()

def register_new_user(self):
        self.register_new_user_window = tk.Toplevel(self.main_window)
        self.register_new_user_window.title("Register New User")
        self.register_new_user_window.geometry("1200x520+370+120")

        self.capture_label = util.get_img_label(self.register_new_user_window)
        self.capture_label.place(x=10, y=0, width=700, height=500)

        self.add_img_to_label(self.capture_label)

        self.entry_text_register_new_user = util.get_entry_text(self.register_new_user_window)
        self.entry_text_register_new_user.place(x=750, y=150)

        self.text_label_register_new_user = util.get_text_label(
            self.register_new_user_window, 'Please, \ninput username:'
        )
        self.text_label_register_new_user.place(x=750, y=70)

        self.accept_button = util.get_button(self.register_new_user_window, 'Accept', 'green',
self.accept_register_new_user)
        self.accept_button.place(x=750, y=300)

        self.try_again_button = util.get_button(self.register_new_user_window, 'Try Again', 'red',
self.try_again_register_new_user)
        self.try_again_button.place(x=750, y=400)

    def try_again_register_new_user(self):
        self.register_new_user_window.destroy()

    def add_img_to_label(self, label):
```

```
            imgtk = ImageTk.PhotoImage(image=self.most_recent_capture_pil)
        label.imgtk = imgtk
        label.configure(image=imgtk)

    def accept_register_new_user(self):
        name = self.entry_text_register_new_user.get(1.0, "end-1c").strip()
        if not name:
            messagebox.showerror("Error", "Username cannot be empty.")
            return

        embeddings = face_recognition.face_encodings(self.most_recent_capture_arr)
        if not embeddings:
            messagebox.showerror("Error", "No face detected. Try again.")
            return

        with open(os.path.join(self.db_dir, f'{name}.pickle'), 'wb') as file:
            pickle.dump(embeddings[0], file)

        util.msg_box('Success!', 'User was registered successfully!')
        self.register_new_user_window.destroy()

    def start(self):
        self.main_window.mainloop()

    def __del__(self):
        if hasattr(self, 'cap') and self.cap.isOpened():
            self.cap.release()


if __name__ == "__main__":
    app = App()
    app.start()
```

Util.py:

```
import os
import pickle

import tkinter as tk
from tkinter import messagebox

import face_recognition
def get_button(window, text, color, command, fg='white'):
    button = tk.Button(
                        window,
                        text=text,
                        activebackground="black",
                        activeforeground="white",
                        fg=fg,
                        bg=color,
                        command=command,
                        height=2,
                        width=20,
                        font=('Helvetica bold', 20)
                    )

    return button

def get_img_label(window):
    label = tk.Label(window)
    label.grid(row=0, column=0)
    return label



def get_text_label(window, text):
    label = tk.Label(window, text=text)
    label.config(font=("sans-serif", 21), justify="left")
    return label
```

```
def get_entry_text(window):
    inputtxt = tk.Text(window,
                       height=2,
                       width=15, font=("Arial", 32))
    return inputtxt


def msg_box(title, description):
    messagebox.showinfo(title, description)


def recognize(img, db_path):
    # it is assumed there will be at most 1 match in the db

    embeddings_unknown = face_recognition.face_encodings(img)
    if len(embeddings_unknown) == 0:
        return 'no_persons_found'
    else:
        embeddings_unknown = embeddings_unknown[0]

    db_dir = sorted(os.listdir(db_path))

    match = False
    j = 0
    while not match and j < len(db_dir):
        path_ = os.path.join(db_path, db_dir[j])

        file = open(path_, 'rb')
        embeddings = pickle.load(file)

        match = face_recognition.compare_faces([embeddings], embeddings_unknown)[0]
        j += 1
```

```
 if match:
        return db_dir[j - 1][:-7]
else:
        return 'unknown_person'
```

Requirements.txt:

```
click==8.1.7
dlib==19.24.6
face-recognition==1.3.0
face_recognition_models==0.3.0
numpy==1.26.4
opencv-python==4.6.0.66
Pillow==9.2.0
setuptools==75.6.0
```

7. REFERENCES

1. GitHub Repositories

a. Face Attendance System : https://shorturl.at/7tst6

b. Face Recognition Library: https://shorturl.at/YZteO

c. SilentFace Anti-Spoofing : https://shorturl.at/KczpK

2. Books & Research Papers

a. FaceRecognition: From Theory to Applications: Zhang, Z., & Zhang, X. (2019). Face Recognition: From Theory to Applications.

b. A Survey of Face Recognition Techniques: S. Zhao, S. Zhang, & X. Zhang (2018). A Survey of Face Recognition Techniques.

3. Online Resources

a. OpenCV Documentation

b. Python Face Recognition Tutorials

4. Other Websites

a. Medium Articles on Face Anti-Spoofing: https://medium.com/