PRESENTATION CONTENT

SLIDE 1 : TITLE SLIDE

Good [morning/afternoon], sir.
My name is Sankrita Patel, and today I'm here to present my **project** titled **Face Detection in a Video**, which is developed using **Django, React.js and OpenCV library**.
This project is focused on detecting human faces in a user-uploaded video files through a web-based application.

SLIDE 5: PROBLEM FORMULATION AND DEFINITION

In the digital era, face detection plays a vital role in areas like security surveillance, social media, biometric systems, and emotion analysis.
The main **problem** is to detect faces accurately in varying conditions such as poor lighting, occlusions, motion blur, and different angles in video content.
So, my project attempts to **solve this by building a system** that processes videos, detects faces frame by frame, and returns a video with highlighted faces.

SLIDE 6: MOTIVATION

The **motivation** behind this project comes from the increasing need for automatic video analysis.
With huge volumes of user-generated video content, there is a demand for systems that can detect and analyse faces – for monitoring, tagging, or anonymizing purposes.
I also wanted to explore how **modern web technologies** like React and Django could be integrated with **AI-based image processing.**

SLIDE 7: CONTRIBUTION

I contribute in this project by following ways:

- First, I built a **user-friendly web interface** that allows users to upload video files.
- Second, I developed a **backend logic** that detects faces frame by frame using **Haar Cascades** from OpenCV.
- Then, I implemented **video reconstruction**, where processed frames are stitched into a final output.
- Lastly, I ensured **seamless integration** between the frontend and backend for smooth video upload and download."

SLIDE 8: METHODOLOGY

Now coming to the **methodology**, I used the **Haar Cascade Classifier**, which is a machine learning-based approach available in OpenCV.
Here's the step-by-step process:

1. The user uploads a video from the frontend.
2. Frames are extracted using OpenCV.
3. Each frame is passed through the Haar classifier to detect faces.
4. Bounding boxes are drawn on detected faces.
5. The processed frames are then reassembled into a video.
6. The final video is returned to the user for download.

SLIDE 9: FACE DETECTION PIPELINE

This diagram illustrates the complete face detection pipeline followed in my project.

1. The process starts when the user uploads a video via the frontend.
2. The Django backend extracts each frame from the video using OpenCV.
3. Then, face detection is performed on every frame using Haar Cascade Classifier.
4. Once faces are detected, rectangles are drawn around them – this is the annotation phase.
5. After all frames are processed, they are compiled back into a video.
6. Finally, the output video is made available for the user to download.

SLIDE 10: WORKFLOW1 - USER AUTHENTICATION FLOW

This diagram illustrates the **User Authentication Flow** in the application.
The flow begins when a user visits the site. If the user is new, they are prompted to enter their details and verify their email via OTP before their data is stored in the database.
If the user already has an account, they enter their credentials.

- If valid, a token is generated and the user is logged in.
- If the user forgets their password, they can reset it by entering their email, receiving an OTP, and setting a new password.

This ensures a secure and smooth login experience.

Slide 11: Workflow 2 – Face Detection Flow

This second workflow shows the core functionality – face detection in videos.
Once a user is logged in, they arrive at the dashboard where they can:

- Upload a video,
- Or view previous detection history.

If they choose to upload a video, the video is processed, and frames are scanned for faces.

- If faces are detected, the results are shown to the user.
- Otherwise, a message is shown indicating no faces were found.

Alternatively, in the Detection History section:

- Users can view previous detections,
- Remove any record,
- Or export data to CSV.

This structured workflow makes the app user-friendly and functional.

Slide 12: Results and Discussion

I tested the system on various videos of different lengths and qualities.
In terms of accuracy:

- It works well for **frontal, clear faces** under good lighting.
- Detection drops slightly for side faces or dim lighting.
- False positives were rare but seen in some backgrounds.

In terms of performance:

- Short videos processed under 10 seconds.
- Longer videos took around 30–45 seconds.
- Performance improved with optimizations like threading and memory management.

Limitations include no real-time support and limited detection on non-frontal faces."

Slide 13-17: UI Screenshots / UI Pages

This slide shows some sample pages from the application:

- User Registration Page
- Login Page
- Password changing page
- The video upload page
- A screenshot of the processed video with rectangles

Slide 18: Conclusion and Future Scope

To conclude,
I was able to successfully **build a face detection system** that integrates **computer vision** with a **web-based interface**.
It's capable of processing offline videos and detecting faces frame by frame.
Future enhancements can include:

- Real-time webcam detection
- Using more advanced models like **YOLO or MTCNN**
- Cloud deployment for scalability
- And improving UI/UX further with modern designs

Slide 19: References / Bibliography

I've referred to official documentation and libraries of OpenCV, Django and React.js.

Slide 20: Closing

Thank you for your attention.
I'm happy to take any questions.