 A light bulb with a logo

Description automatically generated

**Workshop Project Report**

**Year of Submission: - 2023-24**

**Submit By:-**

Mahesh A V\_1VK21IS022\_ML23115

Lakshmi Prasad V\_1VK21IS020\_ML23126

Abhishek G\_1VK21IS001\_ML23112

Vishal V Nayak\_1VK21IS056\_ ML23188

Bharath s\_1VK21IS007\_ML23181

**Department of Information Science Engineering**

**VKIT, Gudimavu**

**Project Report: Face Detection with Python**

**Introduction:**

One of the applications of computer vision is face detection, which is the process of finding and recognizing human faces in digital images or videos. Several Python libraries can help with this task, such as OpenCV and face\_recognition. These libraries provide various functions and methods to perform face detection effectively and accurately.

**Dataset:**

Face detection is a task that does not require labels for the data, unlike face recognition, which aims to identify specific people. Any set of images or live video streams can be used as input for face detection algorithms, which will locate and mark the faces in the scene. This is a useful technique for applications such as security, surveillance, biometrics, and social media.

**Methodology:-**

To implement a face recognition system in Python, we can follow these steps:

1. Library Selection: Choose suitable libraries like OpenCV or face\_recognition. OpenCV is a popular computer vision library that provides various functions for image processing and manipulation. Face\_recognition is a library that wraps around OpenCV and simplifies the face recognition tasks.
2. Image Loading: Load images or access video frames using OpenCV. We can use the `cv2.imread` function to load an image from a file, or the `cv2.VideoCapture` class to capture frames from a webcam or a video file.
3. Face Encoding: Use `face\_recognition.face\_encodings` to generate encodings of known faces. This function takes an image and returns a list of 128-dimensional vectors that represent the faces in the image. We can store these encodings in a dictionary along with the names of the corresponding persons.
4. FaceDetection: Employ `face\_recognition.face\_locations` to identify face locations in images or frames. This function returns a list of tuples that contain the coordinates of the top, right, bottom, and left edges of each face in the image.
5. Comparison: Utilize `face\_recognition.compare\_faces` to compare known face encodings with detected faces. This function takes a list of known face encodings and a list of detected face encodings and returns a list of boolean values indicating whether each detected face matches any of the known faces.
6. Visualization: Draw rectangles around detected faces and label them with names or as "Unknown." We can use the `cv2.rectangle` and `cv2.putText` functions to draw shapes and text on the image. We can also use the `cv2.imshow` and `cv2.waitKey` functions to display the image and wait for a key press to exit.

**Model Building and Evaluation:-**

To detect faces, we usually use pre-trained models from libraries like face\_recognition. We need to evaluate these models to see how well they can detect faces in different situations, such as different lighting, angles, and facial expressions. The evaluation measures the accuracy and performance of the face detection algorithm.

**Comparative Study:-**

Face detection is a common task in computer vision that involves locating and identifying human faces in images or videos. Many algorithms or techniques can perform face detection, but they may differ in various aspects such as accuracy, speed, robustness, and resource efficiency. A comparative study is a way to evaluate and compare the performance of different face detection methods using some metrics or criteria. For example, a comparative study could use Python libraries such as OpenCV, Dlib, MTCNN, or Face Recognition to implement and test different face detection models or algorithms on a dataset of images or videos. The results of the comparative study could help to identify the strengths and weaknesses of each method and to choose the best one for a specific application or scenario.

**Results and Observations:**

1. Accuracy: Measure the accuracy of face detection on your dataset or video frames.
2. Performance: Note the speed and efficiency of face detection in real-time scenarios.
3. Challenges: Observe and discuss challenges faced, such as varying lighting conditions, occlusions, or face orientations affecting detection accuracy.
4. Improvements: Identify areas where the system could be improved, such as handling multiple faces, better handling of noise, etc.