---

1. Project Overview

The goal of this project was to build a **small face-recognition model** capable of identifying individuals from a gallery of known faces.

- The model takes an **image of a face** as input.
- It outputs the **predicted actor name** along with a **confidence score**.
- For this project, three actors were chosen: **Vijay, SK, and Dhanush**, with 5 images each.

---

2. Dataset

- **Custom dataset:** 3 actors × 5 images = 15 images.
- **Folder structure:**

```
dataset/
  vijay/
    img1.jpg
    img2.jpg
    ...
  sk/
    img1.jpg
    ...
  dhanush/
    img1.jpg
    ...
```

- Images were collected online. Some images were not very clear, demonstrating model performance on real-world data.

---

3. Tools & Libraries Used

- **Python 3.10**
- **PyTorch & torchvision** → for deep learning and pretrained models.

- **facenet-pytorch** → provides pretrained FaceNet (InceptionResnetV1) and MTCNN for face detection.
- **OpenCV** → image handling and preprocessing.
- **NumPy & matplotlib** → numerical operations and visualization.
- **scikit-learn** → Logistic Regression classifier and label encoding.

---

4. Model Architecture

1. **Face Detection & Alignment:**
    - Used **MTCNN** to detect faces and align them for consistent input.
    - Ensures that the embeddings are robust to minor rotations or misalignment.
2. **Face Embeddings:**
    - Used **FaceNet (InceptionResnetV1 pretrained on VGGFace2)**.
    - Generates **512-dimensional embeddings** for each detected face.
    - Reason: Provides a compact and discriminative feature representation.
3. **Classifier:**
    - **Logistic Regression** was used to classify embeddings into actor names.
    - Reason: Simple, efficient for a small dataset, and fast during inference.

---

5. Preprocessing & Data Handling

- Images resized and normalized to match FaceNet input requirements.
- Face detection and alignment performed using MTCNN.
- Minimal data augmentation applied (flips, rotations) due to small dataset size.

---

6. Training Steps

1. Iterate over all images in the dataset.
2. Detect and align faces with MTCNN.
3. Extract embeddings using FaceNet.
4. Encode actor labels using **LabelEncoder**.

5. Train **Logistic Regression** on embeddings.
6. Save trained model and label encoder to models/ for inference.

---

7. Inference Demo

- Load a test image (e.g., dataset/vijay/img1.jpg).
- Detect face using MTCNN.
- Extract embedding using FaceNet.
- Predict actor name using the trained classifier.
- Output example:

Predicted Actor: vijay, Confidence: 63.5%

- Multiple images were tested to check confidence scores and correct predictions.

---

8. Results & Interpretation

- **Accuracy:** Model successfully recognized actors from test images.
- **Confidence Examples:**
  - Vijay – 63.5%
  - SK – 92.1%
  - Dhanush – 95.0%

Strengths:

- Works well on frontal, clear faces.
- Quick inference using precomputed embeddings.

Limitations:

- Lower confidence on blurry or side-view images.
- Small dataset limits generalization.
- Future improvements: more images per actor, additional data augmentation, or advanced classifiers.

---

9. Why This Architecture Was Chosen

- **FaceNet embeddings:** Pretrained, robust, and compact feature representation.
- **Logistic Regression classifier:** Simple, interpretable, and suitable for small datasets.
- **MTCNN:** Reliable face detection and alignment.

This combination ensures **fast, simple, and reasonably accurate face recognition** with minimal training data.

---

10. Folder Structure

```
face_recognition_project/
|
├─ dataset/          <- Actor images
├─ models/           <- Trained model & label encoder
├─ train.py          <- Training script
├─ inference.py      <- Prediction script
└─ README.md         <- Setup instructions & project details
```