Face Verification App

Overview

This project is a **Face Verification Tool** built using Python's DeepFace framework and Streamlit for the UI. It enables:

- **Face Detection** using various backends (MTCNN, RetinaFace, OpenCV).
- **Face Recognition** using state-of-the-art models like ArcFace, VGG-Face, Facenet, etc.
- **Verification Workflow**: Matches a user-uploaded or camera-captured image with a database of known reference images.

Tech Stack

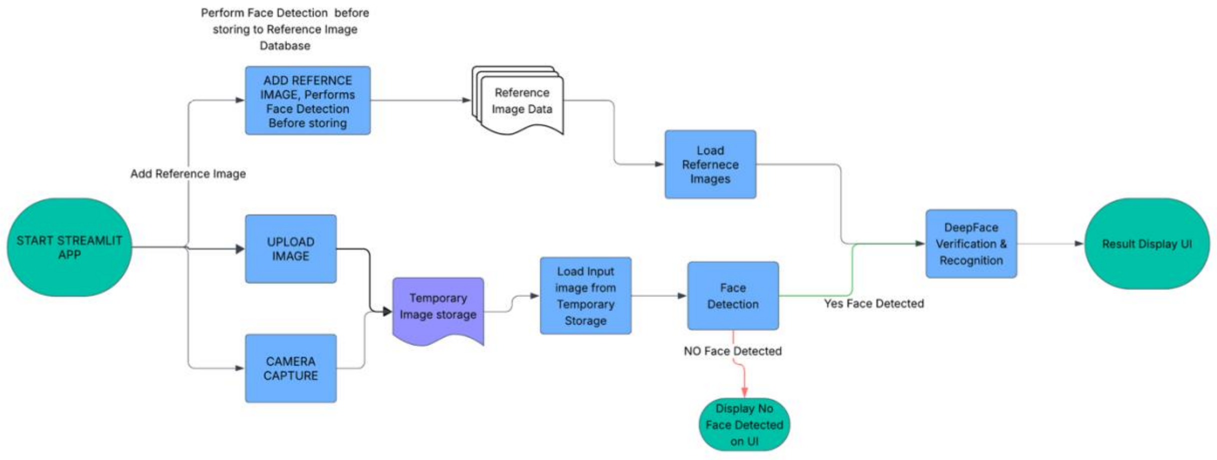
- **Frontend**: Streamlit
- **Backend**: DeepFace (ArcFace + RetinaFace)
- **Image Processing**: Pillow, OpenCV

Architecture

Modular Layout:

```
/
├── app.py                        # Entry-point Streamlit application
├── config.py                     # Configuration for models,paths,thresholds
├── src/
│   ├── face_recognizer.py        # Core logic for face verification
│   └── storage.py                # Loading reference images
├── ui/
│   └── components.py             # UI components: upload, camera, result dis
├── data/reference_faces/         # Folder containing reference images
├── outputs/                      # Stores logs, temporary results
```

Flow Diagram


Perform Face Detection before storing to Reference Image Database
ADD REFERNCE IMAGE, Performs Face Detection Before storing
Reference Image Data
Load Refernece Images
Add Reference Image
START STREAMLIT APP
UPLOAD IMAGE
CAMERA CAPTURE
Temporary Image storage
Load Input image from Temporary Storage
Face Detection
Yes Face Detected
NO Face Detected
Display No Face Detected on UI
DeepFace Verification & Recognition
Result Display UI

Core Logic

Detection:

Uses `DeepFace.extract_faces()` with `RetinaFace` backend to ensure face is present in input/reference image.

Retina Face:

RetinaFace is a high-precision face detection model released in May 2019, developed by the *Imperial College London* in collaboration with *InsightFace*, well-known for its face recognition library.

The model computes the bounding boxes of faces as well as keypoints for eyes and mouth. It also works flawlessly on high-resolution images without resizing and performs hierarchical detection processes, allowing for the robust detection of faces within the image.

Retina Face Architecture:

Architecture- *RetinaFace* enables the detection of small faces through hierarchical processing using a feature pyramid. It uses *ResNet50* as its backbone, supplying feature vectors from multiple layers of *ResNet50* to the detection stage.

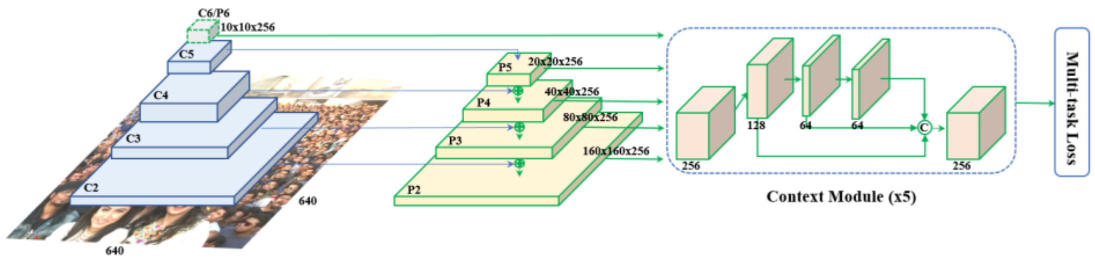
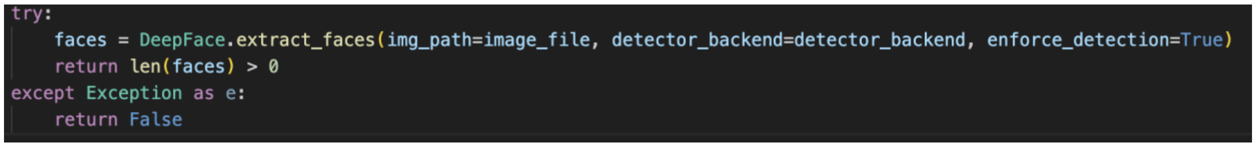


Figure 2. An overview of the proposed single-stage dense face localisation approach. RetinaFace is designed based on the feature pyramids with independent context modules. Following the context modules, we calculate a multi-task loss for each anchor.



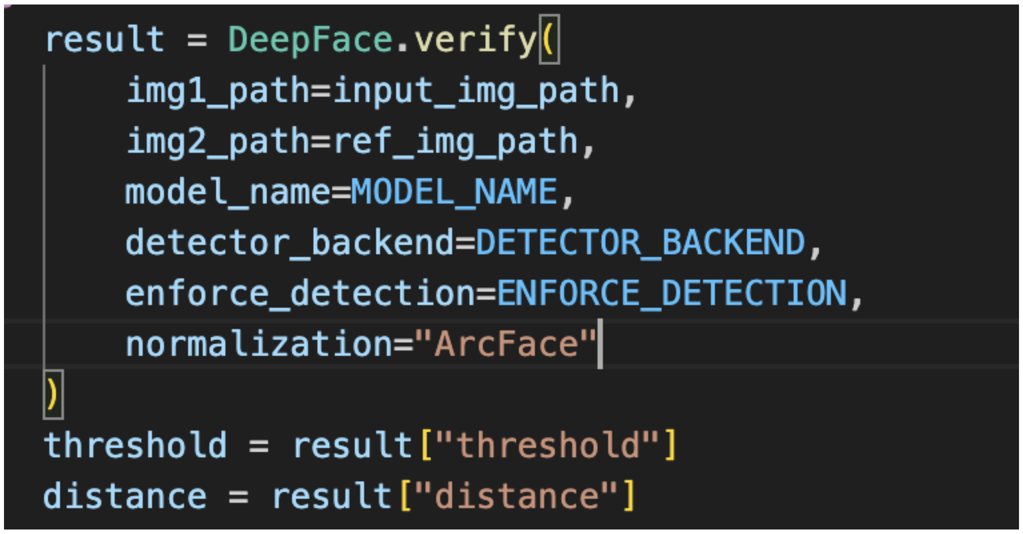
```
try:
    faces = DeepFace.extract_faces(img_path=image_file, detector_backend=detector_backend, enforce_detection=True)
    return len(faces) > 0
except Exception as e:
    return False
```

ReferenceArticle: https://medium.com/axinc-ai/retinaface-a-face-detection-model-designed-for-high-resolution-6c3900771a01

Verification:

Matches an uploaded image with all reference images using `DeepFace.verify()`.

```
DeepFace.verify(img1_path, img2_path, model_name="ArcFace",
detector_backend="retinaface")
```



```
result = DeepFace.verify(
    img1_path=input_img_path,
    img2_path=ref_img_path,
    model_name=MODEL_NAME,
    detector_backend=DETECTOR_BACKEND,
    enforce_detection=ENFORCE_DETECTION,
    normalization="ArcFace"
)
threshold = result["threshold"]
distance = result["distance"]
```

Scoring:

The Euclidean/cosine distance is converted into a score:

```
score = 100 * (1 - distance / max_dist_map[model])
```

```
the max_dist_map[model] – This value is the largest distance that is returned
as an output by the model for an input image, this helps in mapping the raw
score from the model to the percentage range from 0 to 100%

This is just for scaling the output score , it does not alter how the Facial
Recognition is done by the model.
```

Model & Preprocessing (ArcFace + RetinaFace)

ArcFace:

ArcFace is a machine learning model that takes two face images as input and outputs the distance between them to see how likely they are to be the same person. It can be used for face recognition and face search. It uses a *similarity learning* mechanism that allows *distance metric learning* to be solved in the classification task by introducing *Angular Margin Loss* to replace *Softmax Loss*.

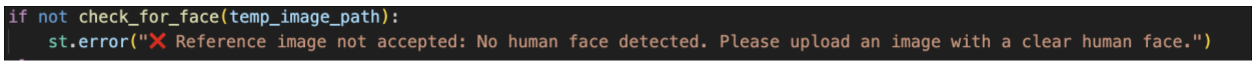
- **Architecture**: ResNet-based backbone + Additive Angular Margin Loss
- **Preprocessing**: Aligned, cropped face images
- **Training Dataset**: MS1M, refined VGGFace2
- **Input**: RGB aligned 112x112
- **Output**: 512-d feature embeddings
- **Distance Metric**: Cosine distance

RetinaFace (Detection)

- **Architecture**: MobileNet/ResNet backbone + context modules
- **Output**: Bounding boxes and facial landmarks
- **Strength**: Robust in uncontrolled environments

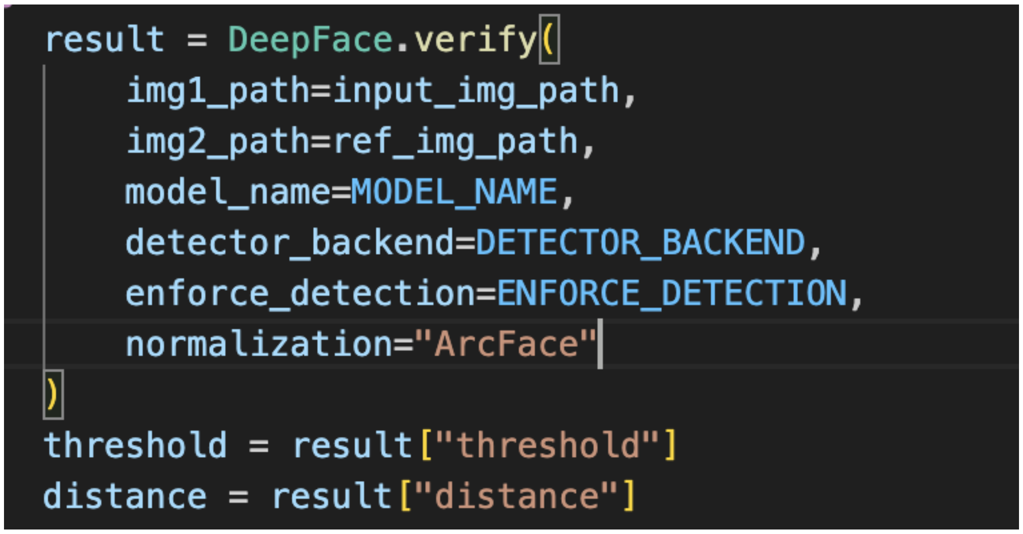
Approach Summary

1. **Face Detection First**: Ensures input images (reference or new) have detectable human faces.



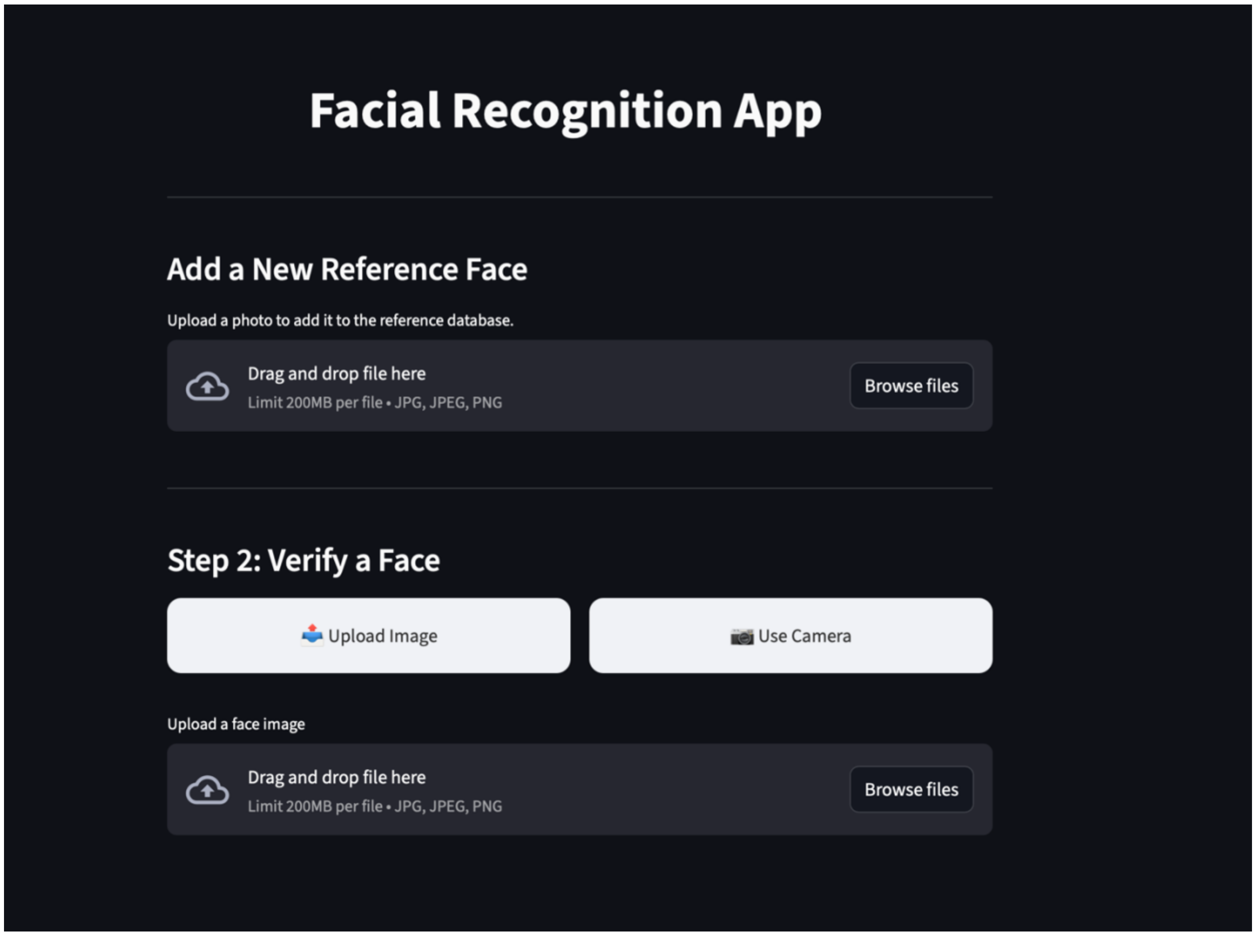
```
if not check_for_face(temp_image_path):
    st.error("❌ Reference image not accepted: No human face detected. Please upload an image with a clear human face.")
```

2. **Face Verification**: Compares face embeddings of input image with each reference image.



```
result = DeepFace.verify(
    img1_path=input_img_path,
    img2_path=ref_img_path,
    model_name=MODEL_NAME,
    detector_backend=DETECTOR_BACKEND,
    enforce_detection=ENFORCE_DETECTION,
    normalization="ArcFace"
)
threshold = result["threshold"]
distance = result["distance"]
```

3. **Scoring**: Computes normalized similarity score.
4. **UI Flow**: User uploads/captures face → Image validated → Compared against all reference faces → Result displayed with bounding boxes, confidence score.



Facial Recognition App

Add a New Reference Face

Upload a photo to add it to the reference database.

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

Step 2: Verify a Face

Upload Image

Use Camera

Upload a face image

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

Strengths of Approach

- **High Accuracy** using ArcFace (SOTA model)
- **Reliable Detection** using RetinaFace
- **User Friendly Interface** with Streamlit
- **Dynamic Addition** of Reference Faces

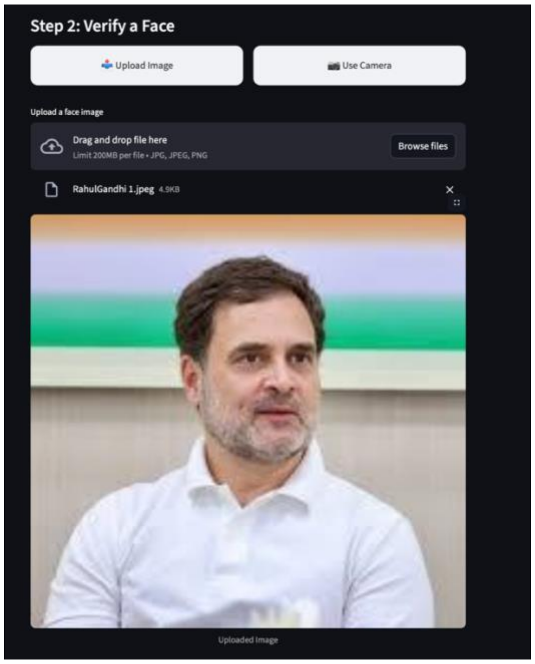
How to Run

```
pip install -r requirements.txt
streamlit run app.py
```

Configuration (config.py)

```
MODEL_NAME = "ArcFace"
DETECTOR_BACKEND = "retinaface"
ENFORCE_DETECTION = True
REFERENCE_IMAGE_DIR = "data/reference_faces"
TEMP_IMAGE_PATH = "outputs/temp_uploaded.jpg"
MATCH_THRESHOLD = 65  # Percent
```

Output Example1: Input Image (Slightly Tilted)

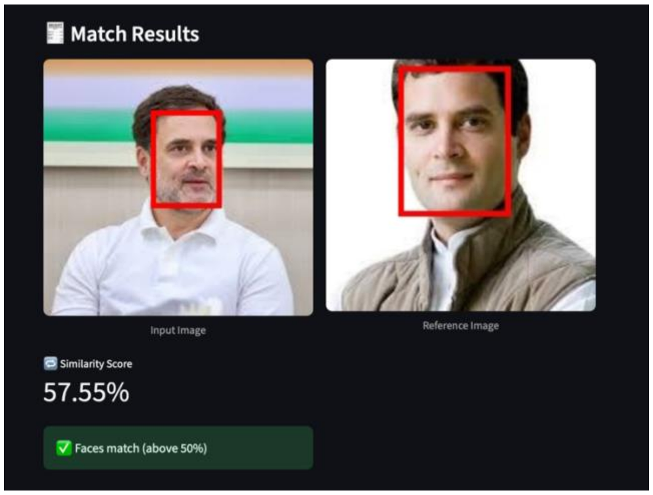

Step 2: Verify a Face

Upload a face image

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

RahulGandhi 1.jpeg 4.9KB

Match Result

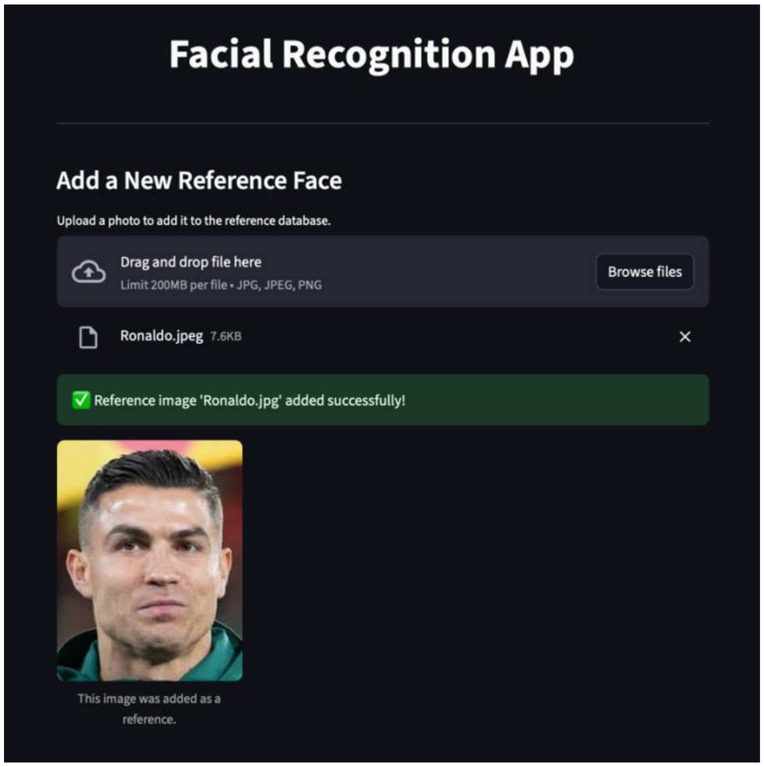

Match Results

Similarity Score
57.55%

✅ Faces match (above 50%)

Output Example2:

Adding the image as Reference


Facial Recognition App

Add a New Reference Face

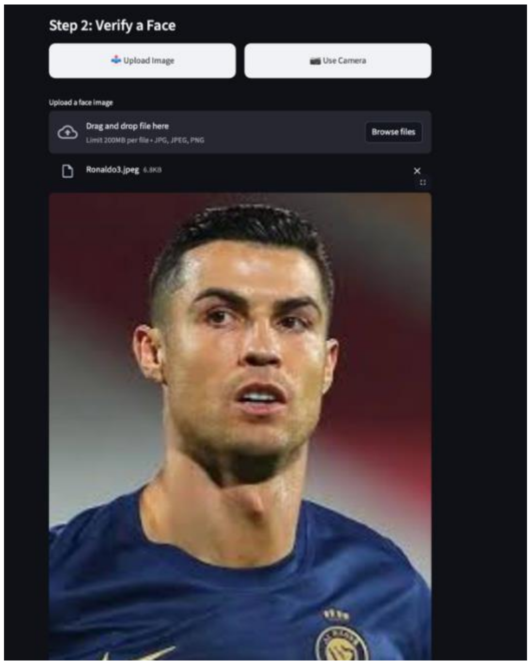
Upload a photo to add it to the reference database.

Ronaldo.jpeg 7.6KB

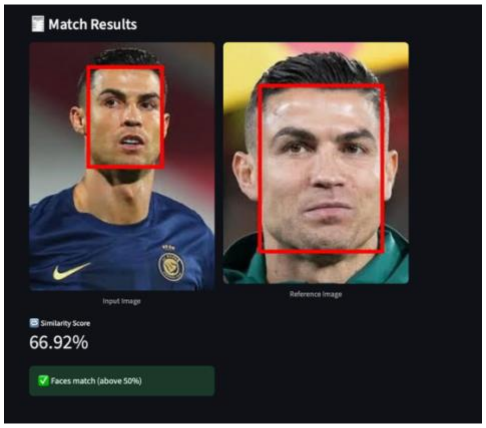
✅ Reference image 'Ronaldo.jpg' added successfully!

This image was added as a reference.

Uploading the Input image of same Person



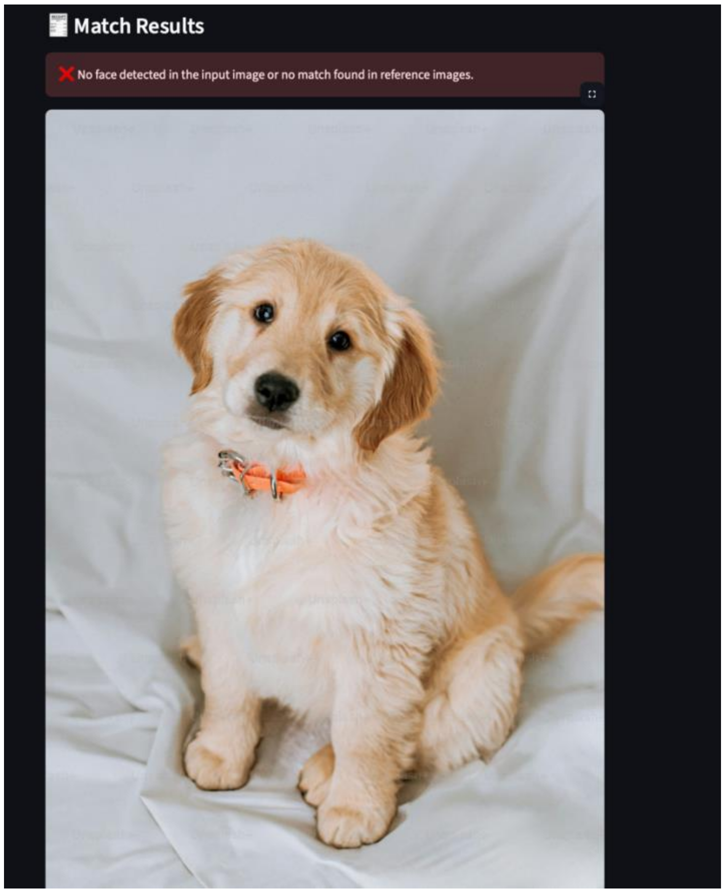
Match Result



66.92%

✅ Faces match (above 50%)

Example 3: Image without the Human


Match Results

❌ No face detected in the input image or no match found in reference images.

DEMO: Demo videos have been shared

Future Enhancements

- Please follow the attached presentation / pdf for Scalable Application System Design and Future Enhancement

Thankyou
Antim
+918791449508