**Face Verification System Documentation**

**Overview**

The face verification system was developed to enhance financial inclusion by verifying the identity of individuals using facial recognition technology. The system leverages a Django application with an API endpoint to verify users by comparing facial features extracted from images. The solution uses MTCNN for face detection, FaceNet embeddings for feature extraction, and an SVM model for classification.

**Architecture**

1. **Django Application Interface**:
   * A simple Django app was developed with a user-friendly interface.
   * The app takes the image path as input to verify the identity of the person in the image.
   * The image path is sent to an endpoint (verify\_image) which processes the image and returns the name of the person detected.
2. **Face Detection and Embedding**:
   * **MTCNN (Multi-task Cascaded Convolutional Networks)** was used for face detection.
     + MTCNN detects faces in images, returning bounding box coordinates for each detected face.
   * **FaceNet Embeddings**:
     + Once faces are detected, **FaceNet** is used to extract facial embeddings (feature vectors) from each face.
     + FaceNet is a deep learning model that generates a fixed-length vector representation for each face, which captures the unique features of the individual.
3. **Model Training**:
   * **Support Vector Machine (SVM)** was used as the classification model to predict the identity of the person in the image.
   * Each image was labeled with the name of the person it contains.
   * To encode the labels (names), **Label Encoding** was used.
     + This transforms the names into integer labels suitable for model training.
     + After prediction, the integer labels were **decoded back to names** to provide the correct identity.
   * **Accuracy** was chosen as the evaluation metric to assess the performance of the SVM model. It measures the percentage of correct predictions out of the total predictions.
4. **Model Saving**:
   * After training, the **SVM model, Label encoder model** and **FaceNet model** were saved for future use.
   * The trained models were serialized and stored so they could be loaded in the endpoint for real-time image verification.
5. **Django Endpoint**:
   * The verify\_image endpoint was developed to process incoming image paths.
   * The endpoint:
     + Loads the trained models.
     + Processes the image using MTCNN to detect faces.
     + Extracts face embeddings using FaceNet.
     + Classifies the embedding using the trained SVM model.
     + Returns the predicted name of the person in the image.

**Model Evaluation**

* The **SVM model** was evaluated using accuracy, which provides an indication of how well the model performs in predicting the correct identity.
* The training dataset consisted of labeled images of individuals, and the model was tested on new images to assess its generalization capability.

**Guide to Running the Pipeline**

1. Create a virtual environment to isolate dependencies for the project:

python3 -m venv venv

1. Activate the virtual environment:

venv\Scripts\activate

1. Install the required Python:

pip install -r requirements.txt

1. Run the Django development server:

python manage.py runserver

1. Once the server is running, you can access the web application via your browser. By default, the application will be accessible at <http://127.0.0.1:8000>.
2. On the homepage, you will see an input box asking for the **image path**.
3. Enter the path to the image you wish to verify (e.g., uploads/sample\_image.jpg).
4. Click the **"Verify"** button to send the image path to the backend.

**Conclusion**

This face verification system offers an efficient and scalable solution for verifying identities using facial recognition. It leverages state-of-the-art deep learning models and is deployed as a web service to be used for real-time verification.