**TangoEye CV Assessment Documentation**

1. **Overview**

This project is designed as a candidate assessment for a computer vision (CV) task. The objective is to demonstrate the ability to preprocess a dataset, extract useful features, perform clustering for person identification, and implement a search-and-retrieve mechanism. In addition, the candidate is required to implement attribute extraction and data visualization to enhance cluster analysis.

**Primary Objectives:**

* **Clustering:** Group all instances of the same person across a diverse collection of images.
* **Search & Retrieve:** Transform the clustering process into a practical image retrieval scenario using a query image to return all images that belong to the same individual.

**Secondary Objectives:**

* **Attribute Extraction:** Extract person-level attributes (e.g., dominant clothing color) which can provide additional insights into the clusters.
* **Data Visualization:** Display the dataset in a reduced-dimensional space (2D or 3D) to visually inspect the distribution of clusters

1. **Project Structure**

The solution is organized into several Python modules, each responsible for a distinct part of the pipeline:

* 1. **data\_process.py**

**Purpose:** Manages image ingestion and preprocessing.

**Key Functions:**

* **fetch\_and\_resize\_images(root\_dir, target\_size=(224, 224)):**  
  Recursively searches for image files, reads, and resizes them.
* **detect\_face(tensor\_image):**  
  Uses MTCNN (via facenet-pytorch) to detect faces in images.
* **write\_images\_to\_disk(image\_list, destination\_dir):**  
  Writes processed images into a designated directory.
  1. **feature\_extraction.py**

**Purpose:** Extract 512-dimensional facial embeddings from preprocessed images using a pre-trained FaceNet model (InceptionResnetV1)..

**Key Functions:**

* **get\_face\_embedding(image\_path):**  
  Reads, preprocesses, and converts an image to its embedding. The embeddings are then stored as a dictionary mapping image paths to vectors and saved as a pickle file.
  1. **data\_process.py**

**Purpose:** Clusters images based on facial embeddings using the HDBSCAN algorithm.

**Key Functions:**

* **load\_embedding\_data(path\_to\_embeddings):**  
  Loads the precomputed embeddings.
* **run\_hdbscan\_clustering(embedding\_dict, min\_cluster\_size=3):**  
  Applies HDBSCAN to generate cluster labels
* **group\_images\_by\_cluster(labels, paths):\_**  
  Organizes image paths by cluster labels
* **save\_cluster\_mapping(clusters\_dict, output\_file):**  
  Saves the resulting cluster mapping as a pickle file.
  1. **attribute\_extraction.py**

**Purpose:** Extract person-level attributes from images, with an initial focus on dominant clothing color.

.

**Key Functions:**

* **get\_nearest\_color\_name(color\_bgr):**  
  Determines the closest matching basic color from predefined references.
* **find\_main\_color(image, num\_colors=1):**  
  Uses k-means clustering to identify the dominant color in an image.
* **detect\_clothing\_color(image\_file\_path):**  
  Extracts the dominant color from a targeted region of the image.
* **analyze\_cluster\_colors(cluster\_data\_path):**  
  Iterates over clusters to determine the most frequent dominant clothing color for each
  1. **search\_retrive.py**

**Purpose:** Implements a search-and-retrieve mechanism based on the clustering results.

**Key Functions:**

* **extract\_embedding(image\_path):**  
  Extracts an embedding from a query image.
* **build\_faiss\_index(embeddings\_dict):**  
  Constructs a FAISS index for rapid similarity search.
* **find\_nearest\_image(query\_embedding, embeddings\_dict):**  
  Identifies the closest matching image using FAISS.
* **retrieve\_cluster\_images(query\_image\_path):**  
  Determines the cluster for the query image and returns all images within that cluster.
  1. **visualize\_cluster.py**

**Purpose:** Visualizes clustering results by reducing facial embeddings to 2D space and plotting them.

**Key Functions:**

* **prepare\_data():**  
  Aggregates embeddings and cluster mapping data for visualization.
* **plot\_clusters\_2d(X, labels, title="2D UMAP Clusters", save\_path=None):**  
  Uses UMAP for dimensionality reduction and displays a scatter plot of the clusters.

1. **Execution Workflow**

Follow these steps to execute the entire pipeline:

1. **Data Processing:**
   * Run data\_process.py to load, resize, and save images from the raw dataset to the processed folder.
2. **Feature Extraction:**
   * Execute feature\_extraction.py to compute facial embeddings from the processed images. These embeddings are saved in a pickle file.
3. **Clustering:**
   * Run clustering.py to perform HDBSCAN clustering on the embeddings and generate a cluster mapping file.
4. **Attribute Extraction:**
   * Execute attribute\_extraction.py to analyze each cluster and extract attributes (e.g., dominant clothing color).
5. **Search and Retrieval:**
   * Use search\_retrive.py from the command line with a query image to retrieve all images in the corresponding cluster.
6. **Visualization:**
   * Run visualize\_cluster.py to produce a 2D scatter plot of the clusters using UMAP.

Each module is self-contained, allowing independent testing and development before integrating them into the complete workflow.

1. **References**

* **Libraries and Frameworks:**
* **Facenet-Pytorch:** [**github.com/timesler/facenet-pytorch**](https://github.com/timesler/facenet-pytorch)
* **HDBSCAN:** [**hdbscan.readthedocs.io**](https://hdbscan.readthedocs.io/)
* **FAISS:** [**github.com/facebookresearch/faiss**](https://github.com/facebookresearch/faiss)
* **UMAP:** [**umap-learn.readthedocs.io**](https://umap-learn.readthedocs.io/)