**Project Report: Image Clustering with HDBSCAN**

**Student name: Mbiydzenyuy Stephen Junior**

Contents

[1. Introduction 2](#_Toc186278915)

[2. Background 2](#_Toc186278916)

[What is Clustering? 2](#_Toc186278917)

[What is HDBSCAN? 2](#_Toc186278918)

[Why Use PCA? 2](#_Toc186278919)

[3. Step-by-Step Process 3](#_Toc186278920)

[Step 1: Load Images 3](#_Toc186278921)

[Step 2: Visualize Images 4](#_Toc186278922)

[Step 3: Preprocess Images 8](#_Toc186278923)

[Step 4: Dimensionality Reduction 10](#_Toc186278924)

[Step 5: Apply HDBSCAN Clustering 12](#_Toc186278925)

[Step 6: Visualize and Analyze Results 14](#_Toc186278926)

[Step 7: Test on New Data 16](#_Toc186278927)

[4. Conclusion 18](#_Toc186278928)

[5. References 18](#_Toc186278929)

[List of Figures 18](#_Toc186278930)

# 1. Introduction

This project aims to cluster images using the HDBSCAN algorithm. Clustering is a technique used to group similar items together. In this case, we are grouping similar images. The project involves several steps, including loading images, preprocessing them, reducing their dimensions, clustering them, and evaluating the results.

# 2. Background

## What is Clustering?

Clustering is a type of unsupervised learning that involves grouping data points into clusters based on their similarities. It is widely used in various fields such as image processing, market segmentation, and bioinformatics.

## What is HDBSCAN?

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is an advanced clustering algorithm that extends DBSCAN (Density-Based Spatial Clustering of Applications with Noise). HDBSCAN can find clusters of varying densities and is robust to noise.

## Why Use PCA?

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while retaining most of the variance. This helps in reducing computational complexity and improving clustering performance.

# 3. Step-by-Step Process

## Step 1: Load Images

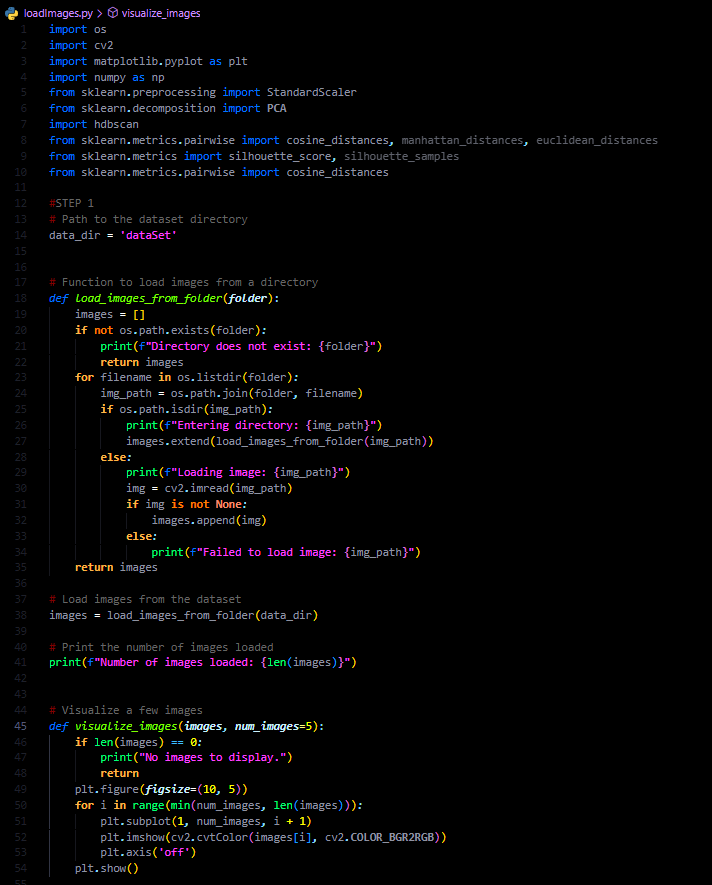
First, we load images from a specified directory. The images are resized to a consistent shape to ensure they can be processed uniformly.

Figure 1 Script to load the images from the data set and print them

**Purpose**

The code is designed to load and prepare images stored in a folder. It resizes the images to a uniform shape, making them ready for further processing or analysis. This is essential when dealing with a collection of images that may have different sizes.

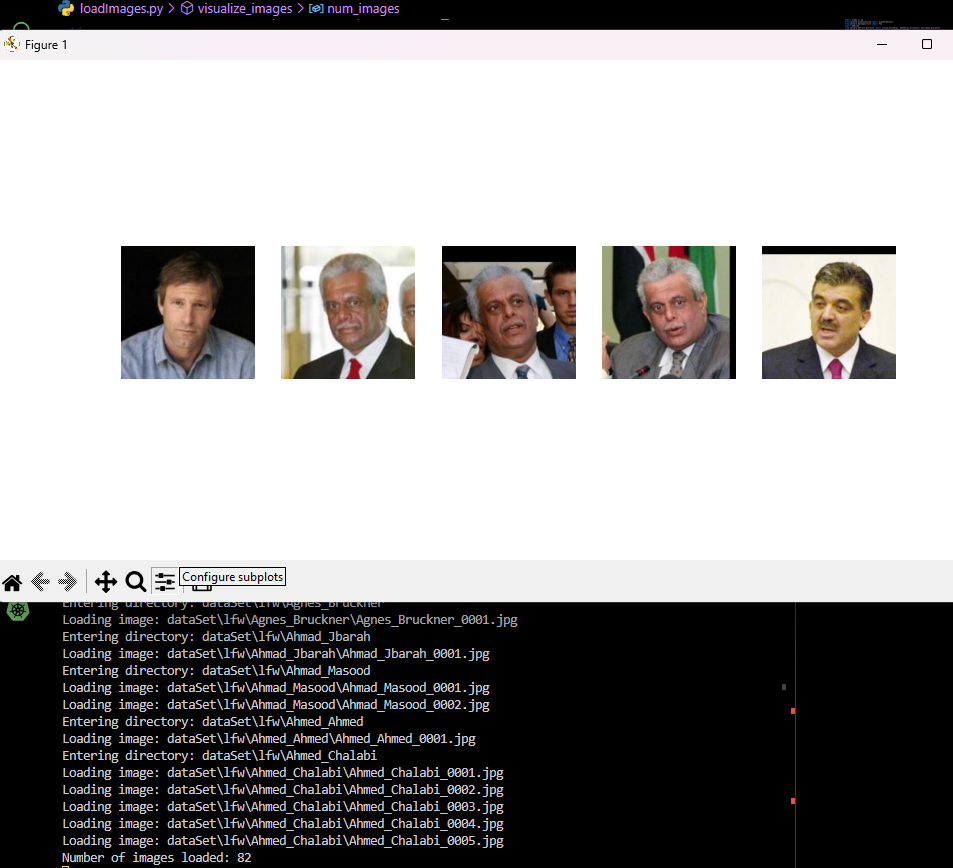
****

Figure 2 Results from step one ( 82 images loaded and 5 images visualized)

## Step 2: Visualize Images

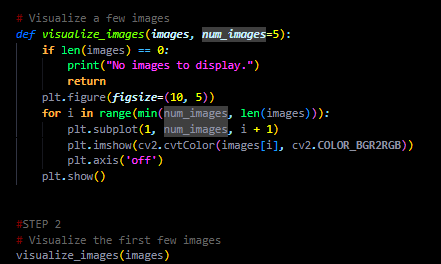
We visualize a few images to ensure they are loaded correctly.

Figure 3 Script to visualize the images

1. **Function Definition**:
   * We define a function visualize\_images that takes two parameters: images (a list of images) and num\_images (the number of images to display, defaulting to 5).
2. **Check for Empty List**:
   * The function first checks if the images list is empty. If it is, it prints "No images to display." and returns without doing anything further. This is a safeguard to handle cases where no images are loaded.
3. **Create a Figure**:
   * If there are images to display, the function creates a new figure using [plt.figure(figsize=(10, 5))](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "). The figsize parameter specifies the size of the figure in inches. Here, we set the width to 10 inches and the height to 5 inches.
4. **Loop Through Images**:
   * The function then loops through the first num\_images images in the images list. The min(num\_images, len(images)) ensures that we do not attempt to display more images than are available in the list.
5. **Display Each Image**:
   * For each image, the function creates a subplot using [plt.subplot(1, num\_images, i + 1)](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "). This specifies a grid of 1 row and num\_images columns, and places the current image in the i + 1-th position.
   * The image is displayed using [plt.imshow(cv2.cvtColor(images[i], cv2.COLOR\_BGR2RGB))](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "). The [cv2.cvtColor](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) function converts the image from BGR color space (used by OpenCV) to RGB color space (used by Matplotlib).
   * The axis is turned off using [plt.axis('off')](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") to hide the axis ticks and labels, making the image display cleaner.
6. **Show the Figure**:
   * Finally, the function calls [plt.show()](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") to display the figure with the images.

**Visual Example**

Let's assume we have a dataset of images of different animals. After loading and resizing the images, we can use the visualize\_images function to display the first few images. The output might look something like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Image 1 | Image 2 | Image 3 | Image 4 | Image 5 |

Each cell in the grid represents an image from the dataset. This visualization helps us quickly verify that the images are loaded correctly and are of the expected size and quality.

**Importance of Visualization**

Visualizing the images at this stage is crucial for several reasons:

* **Verification**: It allows us to verify that the images are loaded correctly and resized to the desired dimensions.
* **Quality Check**: We can inspect the quality of the images and identify any issues such as blurriness, incorrect colors, or artifacts.
* **Content Understanding**: It helps us understand the content of the images and ensures that they are relevant to the task at hand.

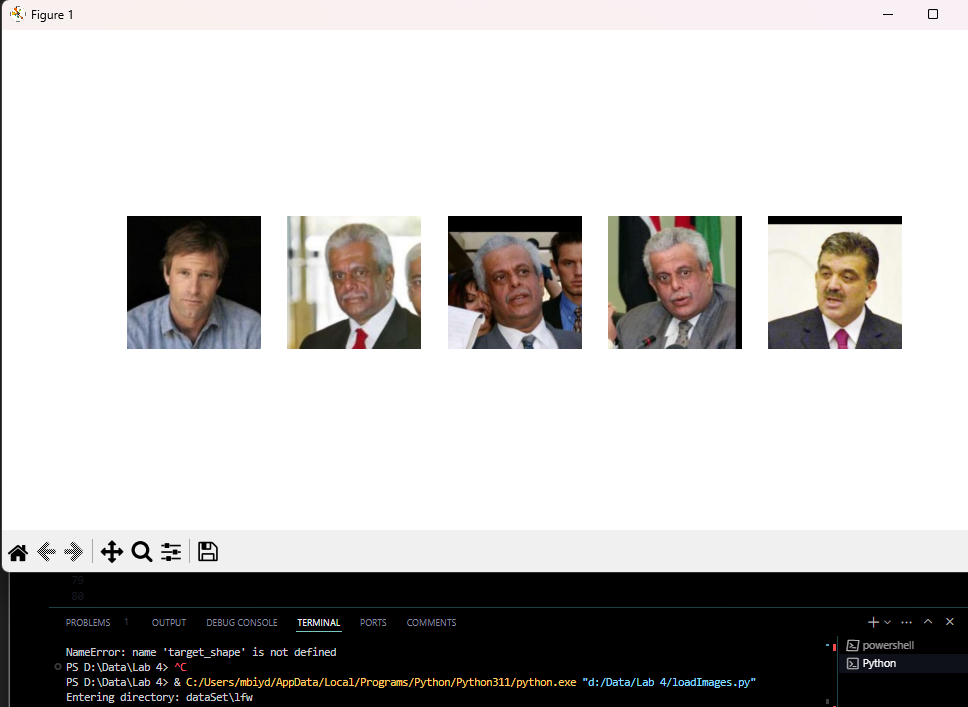
By visualizing the images, we can catch any issues early in the process and make necessary adjustments before proceeding to the next steps.

Figure 4 Results from visualizing 5 images

1. **Import Necessary Libraries**:
   * We start by importing the necessary libraries. These include os for file operations, [cv2](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) from OpenCV for image processing, matplotlib.pyplot for visualization, numpy for numerical operations, and various modules from sklearn for preprocessing and clustering.
2. **Define the Dataset Directory**:
   * data\_dir = 'dataSet': This line specifies the path to the directory containing the dataset of images. You can change this path to point to your specific dataset directory.
3. **Define the Target Shape**:
   * [target\_shape = (128, 128)](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line defines the target shape for resizing images. All images will be resized to this shape to ensure uniformity. You can adjust the dimensions as needed based on your dataset and requirements.
4. **Function to Load Images**:
   * We define a function load\_images\_from\_folder that takes two parameters: folder (the directory containing the images) and [target\_shape](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") (the desired shape for resizing images).
5. **Initialize an Empty List**:
   * images = []: We initialize an empty list to store the loaded and resized images.
6. **Check if Directory Exists**:
   * if not os.path.exists(folder): This line checks if the specified directory exists. If it does not, a message is printed, and the function returns the empty list.
7. **Iterate Over Files in Directory**:
   * for filename in os.listdir(folder): This loop iterates over all files and subdirectories in the specified directory.
8. **Handle Subdirectories**:
   * if os.path.isdir(img\_path): This line checks if the current item is a subdirectory. If it is, the function calls itself recursively to load images from the subdirectory. This allows the function to handle nested directories.
9. **Load and Resize Images**:
   * img = cv2.imread(img\_path): This line reads the image from the specified path using OpenCV's [imread](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") function.
   * if img is not None: This line checks if the image was loaded successfully. If it was, the image is resized to the target shape using [cv2.resize(img, target\_shape)](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
   * images.append(img\_resized): The resized image is appended to the images list.
10. **Handle Failed Image Loads**:
    * else: print(f"Failed to load image: {img\_path}"): If the image could not be loaded, a message is printed indicating the failure.
11. **Return the List of Images**:
    * return images: Finally, the function returns the list of loaded and resized images.
12. **Load Images from the Dataset**:
    * images = load\_images\_from\_folder(data\_dir, target\_shape): This line calls the load\_images\_from\_folder function with the specified dataset directory and target shape, and stores the returned list of images in the images variable.

**Importance of Loading and Resizing Images**

Loading and resizing images to a consistent shape is crucial for several reasons:

* **Uniform Processing**: Ensures that all images have the same dimensions, making it easier to process them uniformly in subsequent steps.
* **Memory Management**: Helps manage memory usage by resizing large images to a smaller, consistent size.
* **Algorithm Compatibility**: Many machine learning algorithms require input data to have a consistent shape. Resizing images ensures compatibility with these algorithms.

By loading and resizing the images correctly, we set a solid foundation for the rest of the image processing pipeline.

## Step 3: Preprocess Images

We flatten the images into a 1D array and standardize them to ensure uniform scaling.

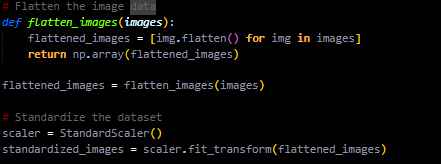


Figure 5 Script to flatten the images

Preprocessing images is a crucial step in preparing the data for clustering. This step involves flattening the images into a one-dimensional array and standardizing them to ensure uniform scaling. These preprocessing steps help in reducing the complexity of the data and making it suitable for clustering algorithms.

1. **Flatten the Image Data**:
   * Images are typically represented as multi-dimensional arrays (e.g., 2D for grayscale images or 3D for color images). To process these images using machine learning algorithms, we need to convert them into a one-dimensional array (flattening).
   * We define a function flatten\_images that takes a list of images and returns a list of flattened images.

*def* *flatten\_images*(***images***):

    flattened\_images = [img***.***flatten() **for** img **in** images]

**return** np***.***array(flattened\_images)

* **Function Definition**:
  + The function flatten\_images takes one parameter: images, which is a list of images.
* **Flatten Each Image**:
  + The function uses a list comprehension to iterate over each image in the images list and flattens it using the flatten() method. The flatten() method converts a multi-dimensional array into a one-dimensional array.
* **Return Flattened Images**:
  + The function returns a NumPy array of flattened images using np.array(flattened\_images). This ensures that the output is a consistent array that can be easily processed further.

1. **Standardize the Dataset**:
   * Standardization is a preprocessing step that involves scaling the data to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the clustering process and helps in improving the performance of the clustering algorithm.
   * We use the StandardScaler from the sklearn.preprocessing module to standardize the flattened images.

scaler = StandardScaler()

standardized\_images = scaler***.***fit\_transform(flattened\_images)

* **Initialize the Scaler**:
  + We create an instance of the StandardScaler class using [scaler = StandardScaler()](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
* **Fit and Transform the Data**:
  + We use the fit\_transform method of the scaler to fit the scaler to the flattened images and transform the data. The fit\_transform method computes the mean and standard deviation of the data and scales it accordingly.
  + The transformed data is stored in the standardized\_images variable.

**Visual Example**

Let's assume we have a dataset of images of different animals. After loading and resizing the images, we need to preprocess them before applying the clustering algorithm.

1. **Flattening the Images**:
   * Original Image Shape: (128, 128, 3) for a color image with 128x128 pixels and 3 color channels (RGB).
   * Flattened Image Shape: (49152,) for a one-dimensional array with 49152 elements (128 \* 128 \* 3).
2. **Standardizing the Images**:
   * The standardized images will have a mean of 0 and a standard deviation of 1. This ensures that all features (pixel values) contribute equally to the clustering process.

**Importance of Preprocessing**

Preprocessing the images is crucial for several reasons:

* **Uniform Data Representation**: Flattening the images ensures that all images are represented as one-dimensional arrays, making them suitable for machine learning algorithms.
* **Equal Feature Contribution**: Standardizing the images ensures that all features (pixel values) contribute equally to the clustering process, preventing any single feature from dominating the results.
* **Improved Algorithm Performance**: Preprocessing helps in improving the performance of the clustering algorithm by reducing the complexity of the data and ensuring uniform scaling.

By preprocessing the images correctly, we set a solid foundation for the clustering process, ensuring that the data is in a suitable format for the algorithm to work effectively.

## Step 4: Dimensionality Reduction

We use Principal Component Analysis (PCA) to reduce the number of features while retaining essential variance in the data.

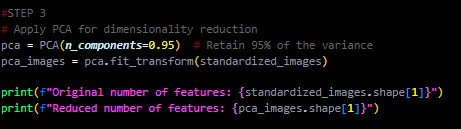


Figure 6 Script for dimensionality reduction



Figure 7 Results from dimensionality reduction

Dimensionality reduction is a crucial step in preparing high-dimensional data for clustering. It involves reducing the number of features (dimensions) while retaining the essential information. This step helps in simplifying the data, reducing computational complexity, and improving the performance of clustering algorithms. In this project, we use Principal Component Analysis (PCA) for dimensionality reduction.

**Detailed Explanation**

1. **Principal Component Analysis (PCA)**:
   * PCA is a statistical technique used to transform high-dimensional data into a lower-dimensional space. It does this by identifying the directions (principal components) along which the variance in the data is maximized.
   * By projecting the data onto these principal components, PCA reduces the number of dimensions while retaining most of the variability in the data.
2. **Initialize PCA**:
   * We create an instance of the PCA class from the sklearn.decomposition module.
   * pca = PCA(n\_components=0.95): This line initializes PCA with the parameter n\_components=0.95, which means we want to retain 95% of the variance in the data. This parameter can also be set to a specific number of components if desired.
3. **Fit and Transform the Data**:
   * [pca\_images = pca.fit\_transform(standardized\_images)](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line fits the PCA model to the standardized images and transforms the data into the lower-dimensional space.
   * The fit\_transform method computes the principal components and projects the data onto these components, resulting in a reduced-dimensionality representation of the data.
4. **Print the Number of Features**:
   * [print(f"Original number of features: {standardized\_images.shape[1]}")](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line prints the original number of features (dimensions) in the standardized images.
   * [print(f"Reduced number of features: {pca\_images.shape[1]}")](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line prints the reduced number of features after applying PCA.

**Visual Example**

Let's assume we have a dataset of images that have been flattened and standardized. Each image is represented as a one-dimensional array with 49152 features (128 \* 128 \* 3 for a color image).

1. **Original Data**:
   * Shape: (number of images, 49152)
   * Each image is represented as a one-dimensional array with 49152 features.
2. **After Applying PCA**:
   * Shape: (number of images, reduced number of features)
   * The number of features is reduced while retaining 95% of the variance in the data. For example, the reduced number of features might be around 100, depending on the dataset.

**Importance of Dimensionality Reduction**

Dimensionality reduction is important for several reasons:

* **Simplification**: Reduces the complexity of the data, making it easier to process and analyze.
* **Computational Efficiency**: Reduces the computational resources required for clustering by decreasing the number of dimensions.
* **Noise Reduction**: Helps in removing noise and redundant features, leading to better clustering performance.
* **Visualization**: Makes it possible to visualize high-dimensional data in 2D or 3D space, aiding in understanding the structure of the data.

By applying PCA, we ensure that the data is in a lower-dimensional space while retaining most of the essential information. This sets the stage for effective clustering in the subsequent steps

## Step 5: Apply HDBSCAN Clustering

We apply the HDBSCAN algorithm to the reduced data to group similar images into clusters.

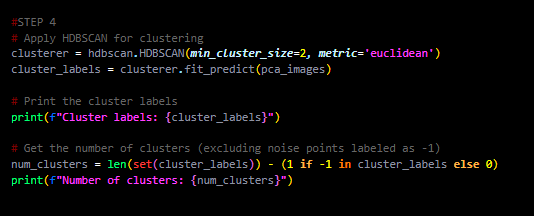


Figure 8 Script for HDBSCAN for clustering

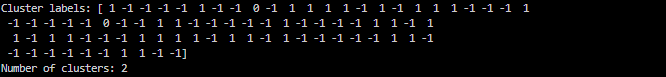


Figure 9 Results from HDBSCAN clustering

Clustering is the process of grouping similar data points together. In this step, we apply the HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) algorithm to the reduced-dimensionality data to group similar images into clusters. HDBSCAN is an advanced clustering algorithm that can find clusters of varying densities and is robust to noise.

1. **Initialize HDBSCAN**:
   * We create an instance of the HDBSCAN class from the hdbscan module.
   * clusterer = hdbscan.HDBSCAN(min\_cluster\_size=10, metric='euclidean'): This line initializes HDBSCAN with the following parameters:
     + min\_cluster\_size=10: The minimum size of clusters. Clusters with fewer than 10 points will be considered noise.
     + metric='euclidean': The distance metric to use. Euclidean distance is the default and most commonly used metric.
2. **Fit and Predict Clusters**:
   * [cluster\_labels = clusterer.fit\_predict(pca\_images)](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line fits the HDBSCAN model to the PCA-transformed images and predicts the cluster labels for each data point.
   * The fit\_predict method performs the clustering and returns an array of cluster labels. Each label corresponds to a cluster, and noise points are labeled as -1.
3. **Print the Cluster Labels**:
   * [print(f"Cluster labels: {cluster\_labels}")](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line prints the cluster labels assigned to each data point. The labels indicate which cluster each data point belongs to.
4. **Calculate the Number of Clusters**:
   * num\_clusters = len(set(cluster\_labels)) - (1 if -1 in cluster\_labels else 0): This line calculates the number of clusters by counting the unique labels, excluding the noise points labeled as -1.
   * [print(f"Number of clusters: {num\_clusters}")](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): This line prints the number of clusters found by the HDBSCAN algorithm.

**Visual Example**

Let's assume we have a dataset of images that have been preprocessed and reduced in dimensionality using PCA. Each image is now represented as a point in a lower-dimensional space.

1. **Original Data**:
   * Shape: (number of images, reduced number of features)
   * Each image is represented as a point in a lower-dimensional space.
2. **After Applying HDBSCAN**:
   * Cluster Labels: An array of cluster labels indicating which cluster each image belongs to.
   * Number of Clusters: The total number of clusters found by the algorithm, excluding noise points.

**Importance of Clustering**

Clustering is important for several reasons:

* **Grouping Similar Items**: Clustering helps in grouping similar images together, making it easier to analyze and understand the data.
* **Noise Handling**: HDBSCAN is robust to noise and can identify noise points that do not belong to any cluster.
* **Discovering Patterns**: Clustering can reveal hidden patterns and structures in the data, providing valuable insights.

By applying HDBSCAN, we can effectively group similar images into clusters, identify noise points, and gain insights into the structure of the data.

## Step 6: Visualize and Analyze Results

We create scatter plots to display the clustering of the reduced data and evaluate clustering quality using silhouette scores.

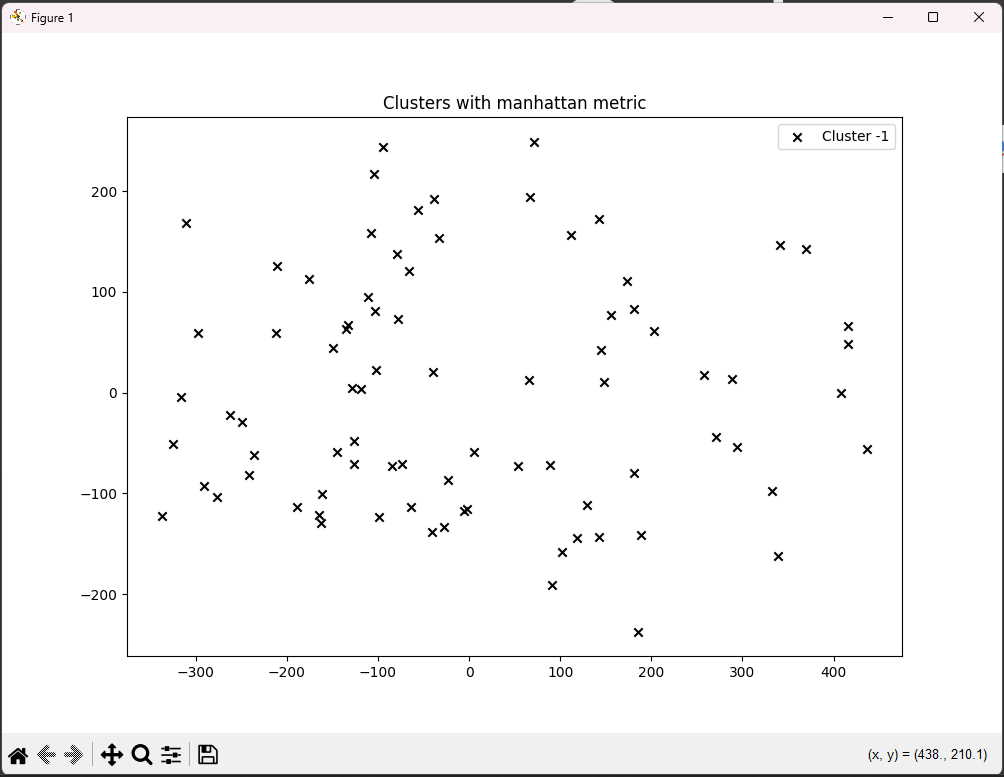
****

Figure 10 Clusters with manhattan matric

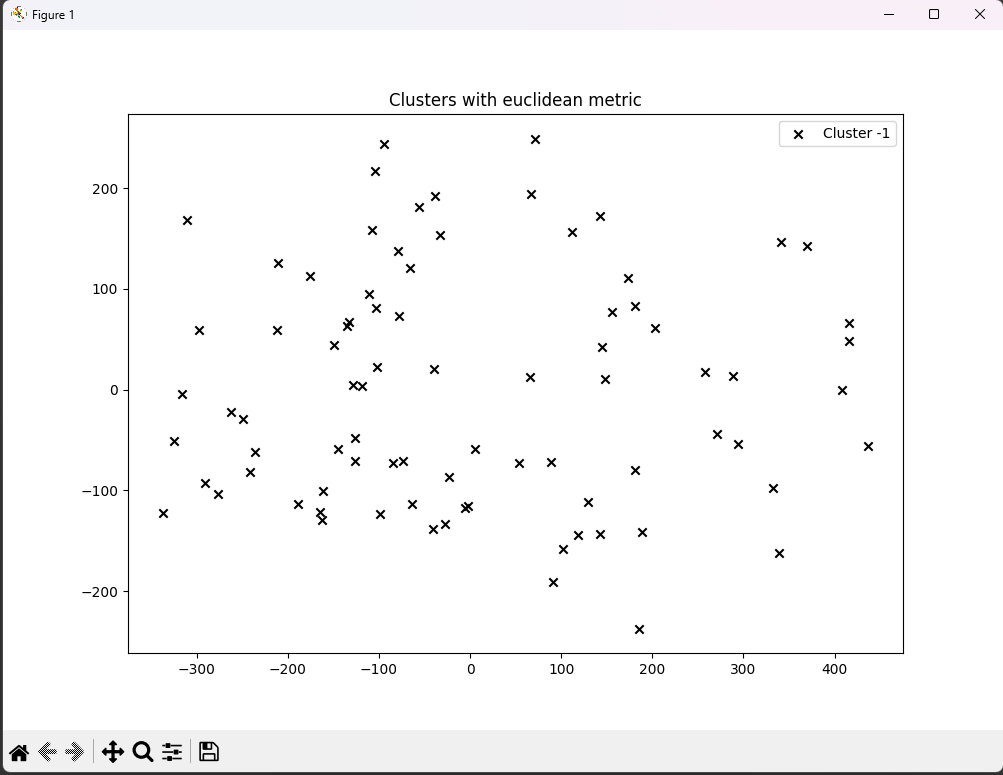


Figure 11 Clusters with Euclidean matrix

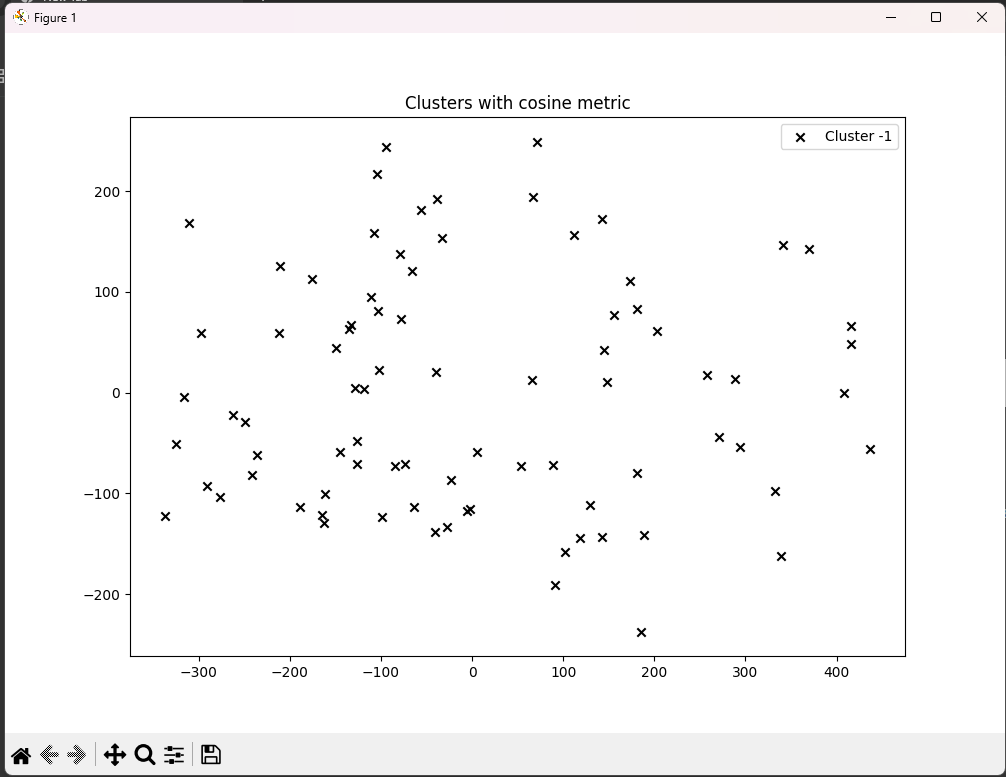
****

Figure 12 Clusters with Cosine metric

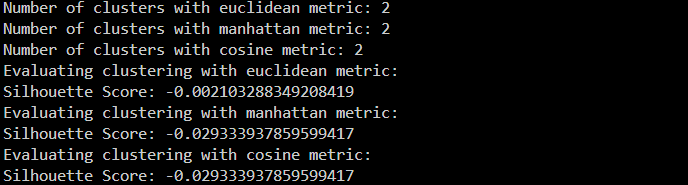
****

Figure 13 Results from euclidean, manhattan, cosine matrics

## Step 7: Test on New Data

We test the clustering algorithm on a new image by assigning it to one of the identified clusters.

# Test on new data

*def* *assign\_new\_image\_to\_cluster*(***new\_image,*** ***pca,*** ***scaler,*** ***clusterer,*** ***target\_shape***):

**if** new\_image **is** **None*:***

        print("Failed to load new image. Please check the file path.")

**return** **None**

    new\_image\_resized = cv2***.***resize(new\_image***,*** target\_shape)

    new\_image\_flattened = new\_image\_resized***.***flatten()***.***reshape(1***,*** -1)

    new\_image\_standardized = scaler***.***transform(new\_image\_flattened)

    new\_image\_pca = pca***.***transform(new\_image\_standardized)

    # Find the nearest cluster center

    cluster\_centers = clusterer***.***weighted\_cluster\_centers\_

    distances = np***.***linalg***.***norm(cluster\_centers - new\_image\_pca***,*** ***axis***=1)

    new\_cluster\_label = np***.***argmin(distances)

**return** new\_cluster\_label

# Load a new image (replace 'new\_image\_path' with the actual path to the new image)

new\_image\_path = 'j2.jpg'

new\_image = cv2***.***imread(new\_image\_path)

**if** new\_image **is** **not** **None*:***

    new\_cluster\_label = assign\_new\_image\_to\_cluster(new\_image***,*** pca***,*** scaler***,*** clusterer***,*** target\_shape)

**if** new\_cluster\_label **is** **not** **None*:***

        print(f'New image assigned to cluster: **{**new\_cluster\_label**}**')

**else*:***

    print("Failed to load new image. Please check the file path.")

**Step 8: Identify Representative Images**

We identify the most representative image in each cluster by finding the data point closest to the cluster center.

# Identify the most representative image in each cluster

*def* *find\_representative\_images*(***data,*** ***labels***):

    representative\_images = {}

**for** label **in** set(labels)***:***

**if** label == -1***:***

**continue**  # Skip noise points

        cluster\_data = data[labels == label]

        cluster\_center = cluster\_data***.***mean(***axis***=0)

        distances = np***.***linalg***.***norm(cluster\_data - cluster\_center***,*** ***axis***=1)

        representative\_image\_index = np***.***argmin(distances)

        representative\_images[label] = representative\_image\_index

**return** representative\_images

representative\_images = find\_representative\_images(pca\_images***,*** cluster\_labels)

print(f'Representative images for each cluster: **{**representative\_images**}**')

# 4. Conclusion

This project demonstrates how to cluster images using the HDBSCAN algorithm. We loaded and preprocessed images, reduced their dimensions using PCA, applied HDBSCAN for clustering, visualized and analyzed the results, tested the algorithm on new data, and identified representative images for each cluster. The project provides a comprehensive approach to image clustering, making it accessible even to non-developers.

# 5. References

* [HDBSCAN Documentation](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)
* [Scikit-learn Documentation](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)
* [OpenCV Documentation](vscode-file://vscode-app/c:/Users/mbiyd/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

# List of Figures

[Figure 1 Script to load the images from the data set and print them 3](#_Toc186273294)

[Figure 2 Results from step one ( 82 images loaded and 5 images visualized) 4](#_Toc186273295)

[Figure 3 Script to visualize the images 4](#_Toc186273296)

[Figure 4 Results from visualizing 5 images 4](#_Toc186273297)

[Figure 5 Script to flatten the images 5](#_Toc186273298)

[Figure 6 Script for dimensionality reduction 5](#_Toc186273299)

[Figure 7 Results from dimensionality reduction 5](#_Toc186273300)

[Figure 8 Script for HDBSCAN for clustering 6](#_Toc186273301)

[Figure 9 Results from HDBSCAN clustering 6](#_Toc186273302)

[Figure 10 Clusters with manhattan matric 7](#_Toc186273303)

[Figure 11 Clusters with euclidean matrix 7](#_Toc186273304)

[Figure 12 Clusters with Cosine metric 8](#_Toc186273305)

[Figure 13 Results from euclidean, manhattan, cosine matrics 8](#_Toc186273306)