CS 5660: Advanced Topics in Artificial Intelligence

Artwork Mapped using ML

Report

Submitted By—

Yamini Mandadi
Nikhil Dhiman
Akash Saraf

Index Page

Problem Definition

Informal description
To create an interactive 3D map of thousands of artworks that clusters them based on how similar they look to one another. This way, users can visually explore art without relying on tags or categories, just how the images actually appear.

Unsupervised Learning Problem

- **Formal Definition:**
  Unsupervised learning aims to identify patterns or structures in data without using labeled outputs. This includes tasks such as clustering, dimensionality reduction, or density estimation.

- **Input Data (X):**
  A large set of digital artwork images (120,000+ from the Wiki Art dataset). No metadata such as artist, title, or style is used — only the pixel-based image content is analyzed.

- **Objective**:
  The objective is to organize artwork into a spatial representation that reflects their visual similarity.
  - First, deep learning models (ResNet) are used to extract high-dimensional feature vectors from each image.
  - PCA and UMAP algorithm is applied to reduce these high-dimensional embeddings to a 3D space.
  - The resulting coordinates are used to build an interactive 3D landscape where the spatial proximity of images corresponds to their visual similarity.
  - The underlying goal is to group visually similar artworks closer together (maximizing similarity within local clusters) and separate dissimilar ones (maximizing inter-cluster distances), without relying on any predefined labels.

Motivation

Traditional methods of categorizing and exploring art, such as by period, artist, style, or genre, rely heavily on metadata that may not always capture the visual or emotional qualities of a piece. These systems are curated, subjective, and often rigid, making it difficult for users to discover artworks based on personal aesthetic preferences or to stumble upon similar works serendipitously. Our motivation stems from a desire to break this limitation and offer a new paradigm: organizing art by how it looks rather than how it's labeled. By analyzing the artworks themselves — independent of any historical, cultural, or curatorial context — we empower both casual viewers and researchers to explore art collections in a way that is inherently visual and intuitive. It's about turning art discovery into an experience of pure perception and curiosity.

Solution Benefits

Solving this problem brings several valuable benefits:

- Visual-First Exploration: Users can explore connections between artworks that are visually similar, regardless of metadata or categorization. This is particularly useful in uncovering hidden patterns across genres, time periods, and cultures.
- Unbiased Organization: Since no metadata is used, the system avoids bias introduced by human curators or cultural norms, instead relying solely on the raw visual features of the images.
- Scalability: This method can be scaled to extremely large datasets (e.g., tens of thousands of images), enabling museums, galleries, or educational platforms to present massive art collections in an organized, meaningful way.
- Interactive Discovery: The 3D interface makes the experience engaging and educational, helping users to understand aesthetic trends, styles, and clusters in an intuitive spatial manner.
- Cross-domain Applications: The same methodology can be adapted to other domains like fashion, design, architecture, or any other visual media, expanding the scope of use.

Solution Use

We envision the solution as an immersive application where users can freely navigate a 3D virtual landscape of artworks. Users would be able to zoom in/out, rotate the space, and click on individual artworks to view them in more detail. This interface could be integrated into digital museums, online art education platforms, and creative discovery tools.
In terms of lifetime and maintenance:

- The core embedding and dimensionality reduction need to be recalculated only when the dataset is updated (e.g., new artworks added).
- Visualization tools may evolve or be updated for performance improvements, but the main structure remains stable.
- The system is largely self-sustaining once deployed, requiring only minimal ongoing maintenance, primarily around adding new data and optimizing rendering performance.

Manual Solution Design

1. **The Data You Would Collect:**

To approach this problem manually, we would first collect many high-resolution artwork images from public online repositories like Wiki Art. In addition to the raw images, might also consider collecting any available visual metadata such as dominant colors, image dimensions, and basic texture indicators — but not any semantic metadata (like artist name or style) since the goal is to rely purely on visual similarity.

2. **Data Preparation Methods:**

Each image would need to be resized to a consistent dimension to standardize the comparison process, would convert them into a common color space (e.g., RGB or HSV) and normalize the brightness/contrast to reduce variability due to lighting or digitization differences. Basic feature extraction could include generating:

- Color histograms (to summarize color distribution)
- Edge detection maps (using filters like Sobel)
- Texture descriptors (e.g., using Gabor filters)

3. **Manual Algorithm or Decision Process:**
A handcrafted similarity function could be designed using weighted averages of the extracted features. For example:

- Compare color histograms using a distance metric like Chi-Square or Earth Mover's Distance.
- Use structural similarity index (SSIM) or pixel-wise correlation on edge maps.
- Aggregate texture statistics using basic Euclidean distance.

Once similarity scores are computed pairwise between all images, UMAP (Uniform Manifold Approximation and Projection) can be applied to reduce the high-dimensional similarity matrix to a 3D space, preserving the local and global structure of the data more effectively than linear methods. After dimensionality reduction, HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) can be used to identify clusters of visually similar images. The final output would be a static or semi-interactive 3D plot where images are placed based on computed visual proximity, with clusters optionally highlighted.

4. **Prototypes or Experiments Needed:**
Before committing to full-scale development, would build small-scale prototypes using a limited dataset (e.g., 100–200 artworks), manually evaluate whether the visual groupings produced by the similarity algorithm match intuitive human perception. A few users could be invited to explore the visual space and give feedback on whether related images appear spatially close.

Methods

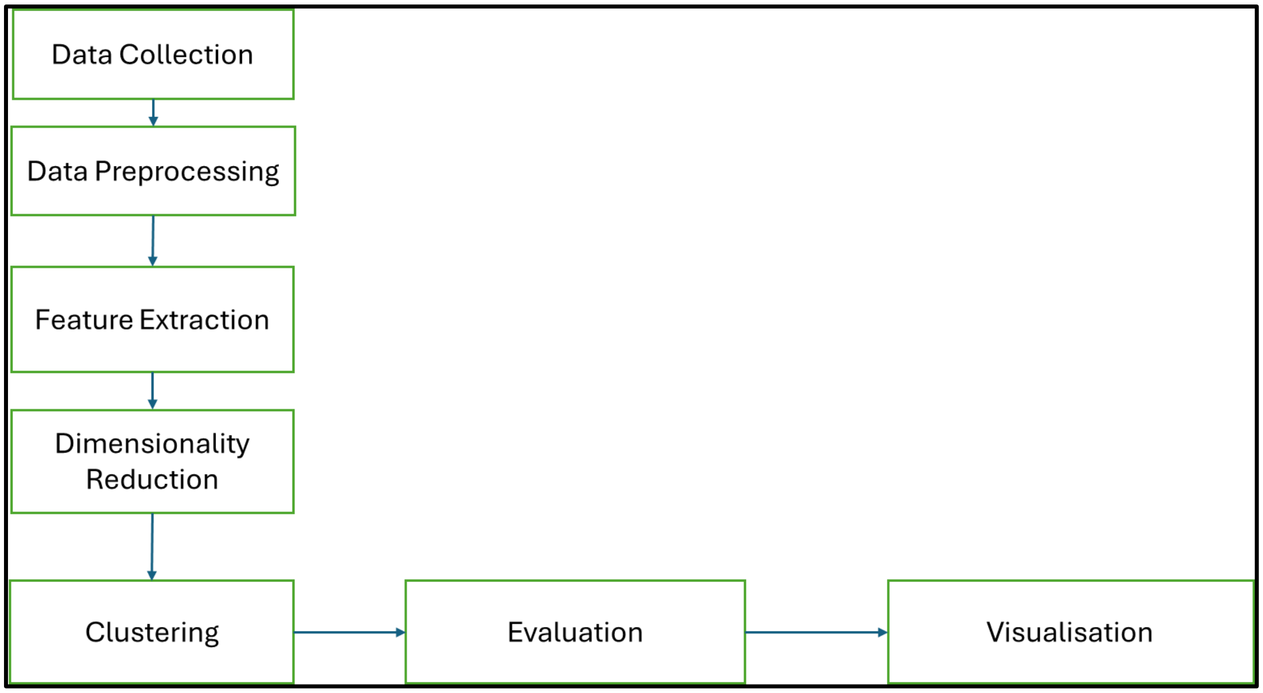
Literature review

[1] In the paper, "Considerably Improving Clustering Algorithms Using UMAP Dimensionality Reduction Technique: A Comparative Study" Dimensionality reduction simplifies high-

dimensional data by reducing the number of variables while retaining essential structure, improving efficiency, interpretability, and performance in tasks like clustering and visualization. Traditional methods like PCA capture variance through linear projections but struggle with nonlinear patterns. Nonlinear techniques such as t-SNE and Isomap better preserve data structure but may lack scalability. UMAP, a modern manifold learning method, effectively maintains both local and global relationships and scales well to large datasets. It significantly boosts clustering accuracy and reduces computation time when used as a preprocessing step. Hybrid methods combining UMAP, Canonical Correlation Analysis (CCA), and Variable Clustering enhance interpretability by generating meaningful synthetic features. Overall, dimensionality reduction is vital for efficient, accurate analysis of complex data.

[2] In the paper, "Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMap, and PaCMAP for Data Visualization" The paper provides a thorough empirical analysis of three popular dimension reduction algorithms—PCA, t-SNE, and UMAP—highlighting their respective strengths and limitations. PCA, a linear global method, efficiently captures global variance but fails to represent non-linear and local structures, making it unsuitable for visualizing complex manifolds. In contrast, t-SNE excels at preserving local structure and forming distinct clusters by modeling pairwise similarities with a heavy-tailed distribution; however, it often distorts global relationships and is sensitive to hyperparameters like perplexity. UMAP improves t-SNE in terms of speed and scalability, and though it also focuses on local neighborhood preservation, it shows modest improvements in global structure retention. The authors emphasize that all three methods face a trade-off between preserving local and global structures and cannot seamlessly balance both. Furthermore, the choice of loss function, graph components (which points are attracted or repelled), and initialization schemes play critical roles in the quality of the resulting embeddings. These insights help explain why these methods behave differently and guide the design of improved algorithms like PaCMAP, which aim to address these limitations.

Methodology/Design:



Data Collection:

The process begins with collecting image data, typically in the form of links or files from sources like Wiki Art.

Data Preprocessing:

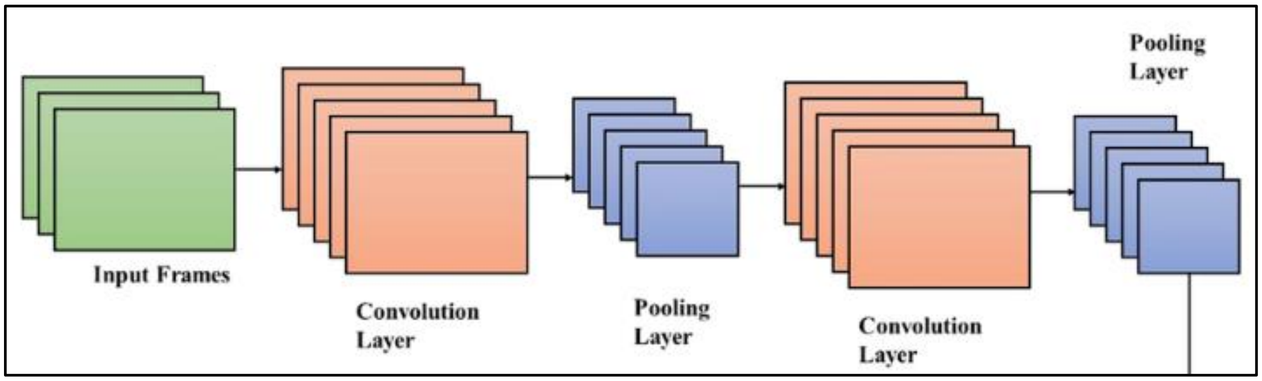
> **Data Cleaning:** It involves detecting and correcting errors or inconsistencies in the data, removing or imputing missing values, fixing corrupt entries or malformed formats and removing duplicates.
>
> **Data Transformation:** Converting data into a suitable structure or format, normalizing or scaling numerical values, converting images to tensors, resizing, and normalizing pixel values.
>
> **Data Reduction:** Reducing the volume or complexity of the data while preserving essential information. Techniques include feature selection, dimensionality reduction (e.g., PCA), sampling.
>
> **Data Storage:** Saving the processed data in appropriate formats (e.g., HDF5, Parquet, CSV), may also include logging skipped or failed entries.

Feature Extraction:

For image datasets, feature extraction involves identifying patterns, textures, shapes, or high-level concepts from the pixels. Rather than feeding raw pixel values into a model (which is inefficient and often ineffective), feature extraction compresses and summarizes the image into a compact, meaningful vector representation.

Common Methods-Traditional Feature Extraction and deep learning-based extraction (Modern Approach):

ResNet50 is a 50-layer convolutional neural network (CNN) widely used for image classification. A primary feature of ResNet50 is its use of residual blocks, which address the vanishing gradient problem in deep networks. The architecture includes an initial convolutional layer, followed by convolutional and pooling layers for initial feature extraction and down sampling. The core of the network is a series of bottleneck residual blocks, each containing three convolutional layers (1x1, 3x3, and 1x1) for dimensionality reduction, spatial feature extraction, and restoration, respectively. Skip connections in these blocks add the block's input to its output, facilitating the learning of residual mappings. The architecture concludes with average pooling to reduce spatial dimensions and a fully connected layer to generate the final output, such as class probabilities. This design, characterized by residual blocks and bottleneck architecture, enables ResNet50 to achieve computational efficiency and effectively learn complex features from datasets like ImageNet.



We chose to utilize ResNet50 as our feature extractor due to its proven ability to learn hierarchical and discriminative features from images. Its deep 50-layer architecture, coupled with the innovative residual connections, allows it to effectively capture both low-level details and high-level semantic information, overcoming the limitations of shallower networks. Furthermore, leveraging pre-trained weights on the extensive ImageNet dataset provides a strong initialization, enabling efficient transfer learning to our specific task and reducing the need for large-scale labeled data. The balance between its representational capacity and computational cost makes ResNet50 a practical and powerful choice for extracting meaningful features that can effectively drive the subsequent clustering process."

It reduces data dimensionality, enhances model performance, and captures essential patterns, making raw data interpretable and useful for machine learning tasks.

Dimensionality Reduction:

Dimensionality reduction is the process of transforming high-dimensional data into a lower-dimensional space while preserving its most important structures or patterns. It helps convert complex, high-dimensional datasets into more manageable forms without losing the essence of the data—making it faster, more interpretable, and more effective for downstream tasks like clustering, classification, or visualization.
PCA is a linear dimensionality reduction method that projects high-dimensional data onto directions of maximum variance, called principal components. It simplifies data by retaining the most informative features, making it ideal for preprocessing before clustering or visualization. PCA is fast, interpretable, and effective for linearly structured data, but it cannot capture complex nonlinear patterns.
UMAP is a nonlinear dimensionality reduction technique that preserves local and global structures by modeling data relationships as a graph and projecting them into a lower-dimensional space. It's commonly used for visualization and clustering of complex, high-dimensional data like image or text embeddings. UMAP excels at capturing nonlinear patterns but is slower than PCA and less interpretable.

Using PCA and UMAP after ResNet50 for feature extraction provides a way to effectively reduce the dimensionality of image data while preserving its essential information. PCA offers initial linear dimensionality reduction and noise reduction, while UMAP provides non-linear dimensionality reduction and excels at preserving local structure. This combination enables efficient processing, visualization, and analysis of complex image data.

The use of ResNet50, PCA, and UMAP as a combined pipeline for unsupervised clustering based on current literature.

ResNet50 serves as a reliable feature extractor, generating high-level semantic representations from image data. These features are typically high-dimensional, so PCA is applied to reduce dimensionality, remove noise, and improve computational efficiency. Although PCA is a linear method, it works well as a preprocessing step before more complex techniques.

UMAP, a nonlinear manifold learning method, is then used to preserve both local and global data structure in a lower-dimensional space. Studies show that UMAP improves clustering accuracy and runtime, making it superior to alternatives like t-SNE for large datasets.

Finally, clustering is performed on the UMAP-reduced data, where algorithms such as k-means or HDBSCAN benefit from clearer structure and faster computation. This pipeline is well-supported in literature and is effective for handling high-dimensional visual data in unsupervised learning tasks.

Clustering:

Clustering is an unsupervised learning step that groups similar data points based on feature similarity, revealing structure in unlabeled data. **HDBSCAN** is a density-based clustering algorithm that improves on DBSCAN by detecting clusters of varying shapes and densities without needing to predefine the number of clusters. It builds a hierarchy of density-based clusters and selects the most stable ones, automatically labeling low-density points as noise.

Evaluation metric:

In unsupervised learning, evaluation helps measure the quality of clustering results, even without ground-truth labels. One commonly used metric is the Silhouette Score, which assesses how well-separated and cohesive the clusters are.

The Silhouette Score measures how similar a data point is to its own cluster (cohesion) compared to other clusters (separation).

S = $(b - a)/\max(a, b)$

a: Mean distance to other points in the **same cluster**.

b: Mean distance to points in the **nearest different cluster**

- Score Range:

  +1 → Point is well-matched to its own cluster and far from others.

  0 → Point lies on the boundary between clusters.

  -1 → Point may be in the wrong cluster.

After clustering (e.g., with HDBSCAN), the silhouette score is computed on the clustered (non-noise) points to evaluate how well-separated the discovered groups are. Higher scores indicate more meaningful clustering.

Visualization:

Visualization is the final step in an unsupervised learning pipeline that helps interpret and explore the structure of high-dimensional data. After dimensionality reduction (e.g., via UMAP), data points are plotted in 2D, or 3D space to reveal patterns, clusters, or anomalies.

In this context, each point represents an image (or its feature embedding), and color is often used to show cluster assignments (e.g., from HDBSCAN), with noise labeled distinctly. Tools like Plotly allow for interactive 3D plots where users can rotate, zoom, and hover to inspect individual samples. This makes it easier to understand relationships between clusters, detect visual themes, and validate the effectiveness of the clustering process.

Code

1. **Data Collection:**

Traversing every image file, it can find with a ".jpg", ".jpeg", or ".png" extension and collects a valid unique image of 111668.

2. **Data Preprocessing:**

```
img = tf.io.read_file(path_decoded)

img = tf.image.decode_jpeg(img, channels=3)
```

While traversing the image, opens the image and converts the image byte string to RGB channel JPEG format with TensorFlow's decoder.

```
with open(SKIPPED_LOG_PATH, "a") as log_file:
    log_file.write(path_decoded + "\n")
return np.zeros((224, 224, 3), dtype=np.float32)
```

When failed to convert, it attempts to “repair” problematic image files by using PIL as a fallback when TensorFlow’s decoder can’t handle them. PIL opens the image, forces it into RGB mode (stripping out any odd channel formats), and then immediately re-saves it in standard JPEG format at quality level 95.If failed to convert to the image (for example, due to a corrupt header or unsupported format), skips or pad that image with zeroed (224,224,3) float32 array as a placeholder.

result = tf.py_function(_load, [path], tf.float32)
result.set_shape([224, 224, 3])

We resize the image to the target IMAGE_SIZE (224×224) and normalizes every image, and exposes a clean (224, 224, 3) float32 tensor that can form to batch, feed through your model, or save as features.

3. **Feature Extraction:**

model = ResNet50(weights='imagenet', include_top=False, pooling='avg')

For feature extraction, ResNet50 architecture, a deep convolutional neural network renowned for its performance in image recognition tasks. The use of a pretrained model leverages the rich feature hierarchy learned from a large and diverse dataset, potentially leading to improved performance and faster convergence compared to training a model from scratch.

A pretrained version of ResNet50, initialized with weights learned on the extensive ImageNet dataset. By excluding the top classification layers (include_top=False) and applying global average pooling (pooling='avg'), the model was configured to output a fixed-length feature vector for each input image. This vector encapsulates the high-level, learned representations of the image content, making it suitable for downstream tasks such as [mention your specific task, e.g., image classification, object detection, similarity search].

Large-scale image feature extraction using a pre-trained ResNet50 model and stores the resulting feature vectors in an HDF5 file. It begins by loading the ResNet50 model without its top classification layer and applies global average pooling to generate compact feature embeddings for each image. The images are processed in batches by pairing the original file paths with their corresponding preprocessed image tensors using TensorFlow datasets. Each batch is passed through the model, and the extracted features are saved sequentially in the HDF5 dataset. After all images are processed, it saves the final state and closes the file, providing a complete and efficient workflow for deep feature extraction.

4. **Dimensionality Reduction:**

Principal Component Analysis

Principal Component Analysis (PCA) is applied to reduce the dimensionality of a dataset while preserving at least 95% of the original variance. First, a PCA model is fitted to the data without limiting the number of components. This allows analysis of each component's explained variance, which is then used to calculate the cumulative explained variance.

```
pca_full = PCA()
pca_full.fit(features)
cumulative_variance = np.cumsum(pca_full.explained_variance_ratio_)
n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1
pca = PCA(n_components=n_components_95)
features_pca = pca.fit_transform(features)
```

A scree plot visualizes this cumulative variance, with a horizontal line indicating the 95% threshold for component selection. The code programmatically determines the minimum number of principal components required to meet this threshold. Finally, PCA is re-applied using this optimal number of components, transforming the original features into a lower-dimensional representation that retains the data's essential structure. This approach balances dimensionality reduction with information preservation, a technique commonly employed prior to clustering, visualization (e.g., UMAP), or use in other machine learning models.

Uniform Manifold Approximation and Projection (UMAP):

Unsupervised clustering using **UMAP** for dimensionality reduction and **HDBSCAN** for clustering, with optional **hyperparameter tuning via grid search**.

```
def run_umap_hdbscan(features, n_neighbors=15, min_dist=0.1, n_components=3,
    min_cluster_size=50, min_samples=10):


    import umap, hdbscan
    reducer = umap.UMAP(
        n_components=n_components,
        n_neighbors=n_neighbors,
        min_dist=min_dist,
        metric='cosine',
        random_state=42,
        low_memory=True,
        verbose=True
    )
    embedding = reducer.fit_transform(features)

    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=min_cluster_size,
        min_samples=min_samples,
        metric='euclidean'
    )
    labels = clusterer.fit_predict(embedding)

    return embedding, labels
```

The run_umap_hdbscan function streamlines the process of dimensionality reduction and clustering by integrating UMAP and HDBSCAN. First, UMAP reduces the dimensionality of the input features using specified parameters like n_neighbors, min_dist, and n_components, employing the 'cosine' metric and a fixed random state for reproducibility. The resulting lower-dimensional embedding is then fed into HDBSCAN, which identifies clusters based on density, guided by parameters such as min_cluster_size and min_samples, and using the 'euclidean' metric. The function returns both the UMAP-generated embedding and the cluster labels assigned by HDBSCAN.

To optimize the UMAP and HDBSCAN pipeline, a grid search was performed over a defined parameter space. The grid search function systematically evaluated combinations of the n_neighbors and min_dist parameters. Specifically, the search explored the following parameter grid: n_neighbors was set to [10] and min_dist was set to [0.05]. For each parameter combination, UMAP was applied to the PCA-reduced features (features_pca), followed by HDBSCAN clustering. The resulting embedding and cluster labels were then evaluated using a clustering performance metric. The parameter combination that yielded the highest performance score was selected as optimal. The function tracked and returned the best-performing parameter configuration, the corresponding score, the resulting embedding, and the associated cluster labels.

5. Evaluation:

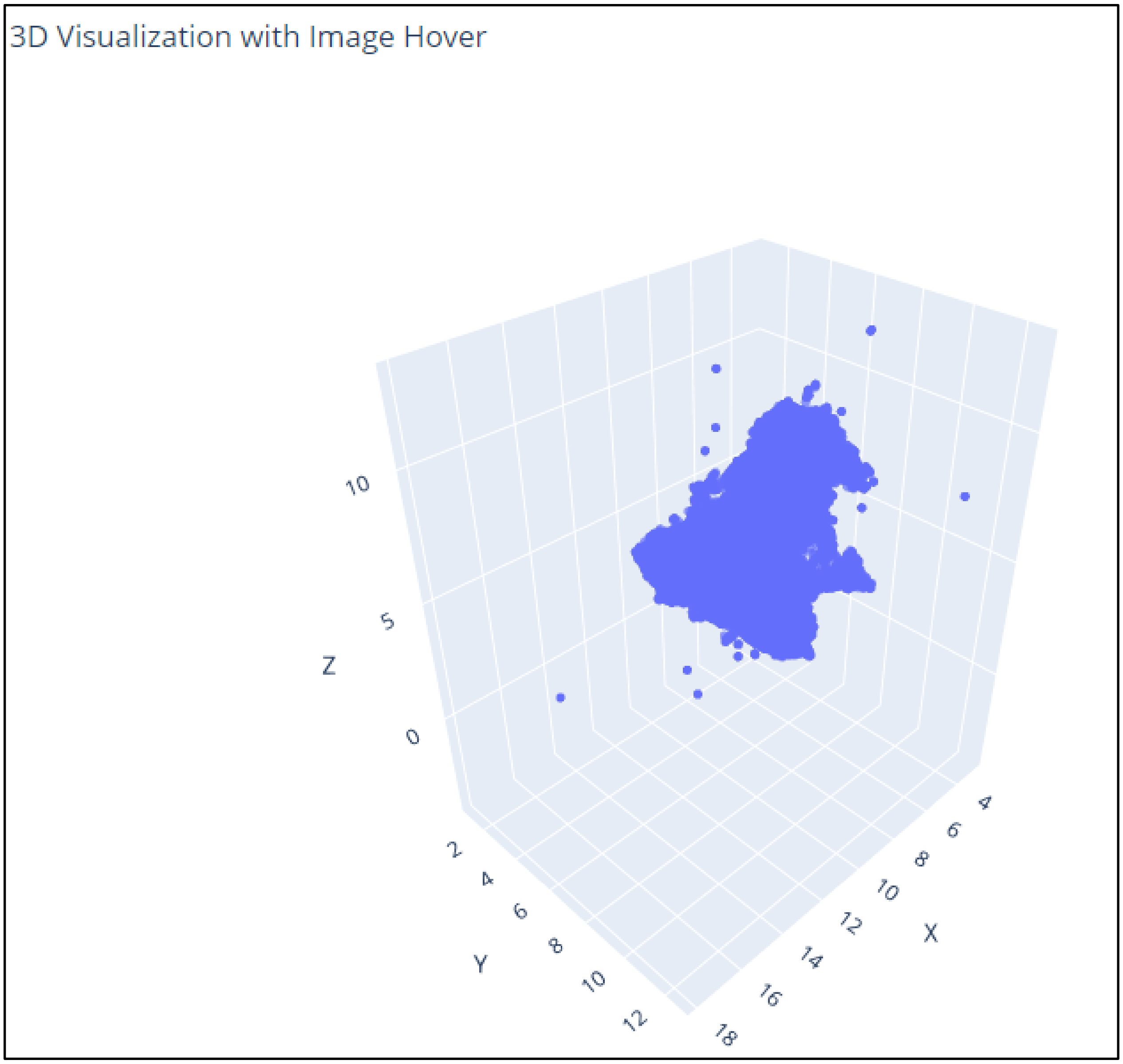
```
def evaluate_clustering(embedding, labels):
valid = labels != -1 # exclude noise
if valid.sum() < 2 or len(set(labels[valid])) < 2:
return -1 # invalid score
return silhouette_score(embedding[valid], labels[valid])
```

The clustering performance was evaluated using the silhouette score, calculated by the evaluate_clustering function. This function takes the embedding vectors and their corresponding cluster labels as input. Critically, the function excludes data points labeled as noise (-1) to ensure a focused evaluation on the quality of the identified clusters. If the valid data points are fewer than two, or if there are fewer than two distinct clusters among the valid points, the function returns a score of -1, indicating an invalid evaluation. This prevents errors or misinterpretations when the clustering result is trivial or degenerate. The silhouette score provides a measure of how similar each data point within a cluster is to other points in the same cluster, compared to points in other clusters, with higher scores indicating better-defined clusters.

Visualisation

```
def save_image_from_html(img_tag, index):

    try:

    match = re.search(r'base64,(.*)"', img_tag)

    base64_data = match.group(1)

    img_data = base64.b64decode(base64_data)

    img = Image.open(io.BytesIO(img_data)).convert("RGB")
    path = f"thumbnails/img_{index}.jpg"
    img.save(path)
    return path
```

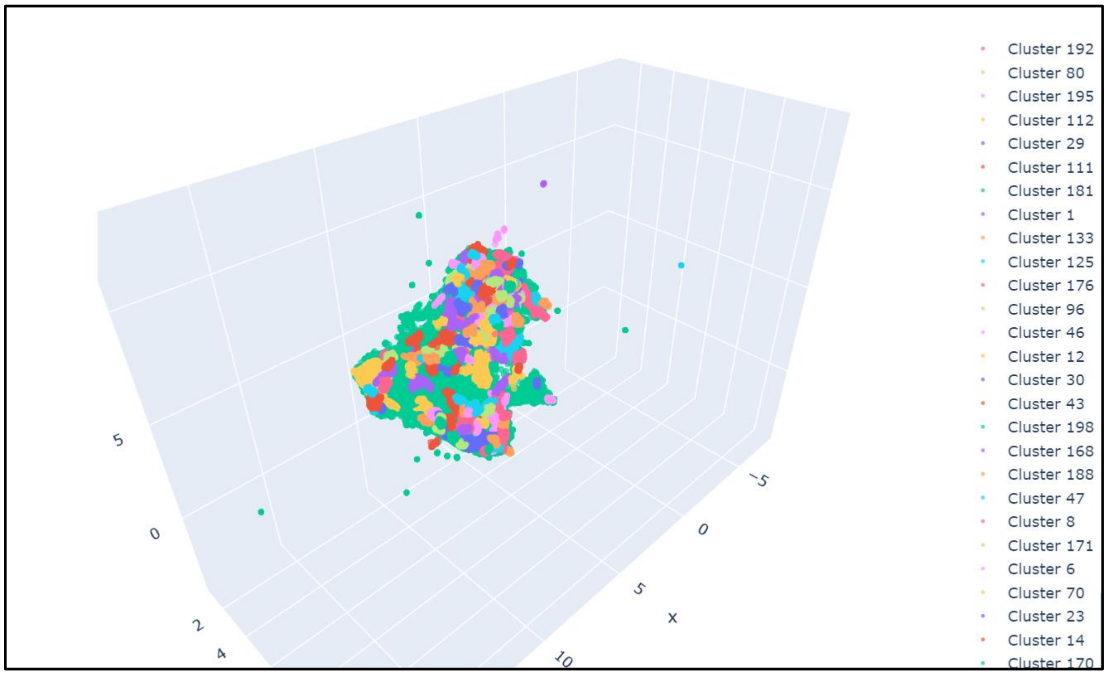
The save_image_from_html function is designed to extract base64 encoded images from HTML-like tags. It uses a regular expression to locate the base64 data, decodes it, and saves it as a JPEG file in the 'thumbnails' directory using a unique index-based filename. The function includes error handling to manage invalid image data, returning None in such cases and printing a notification. This process ensures that valid image data is extracted and saved for subsequent use, while gracefully handling any problematic entries.



Plotly Visualisation:

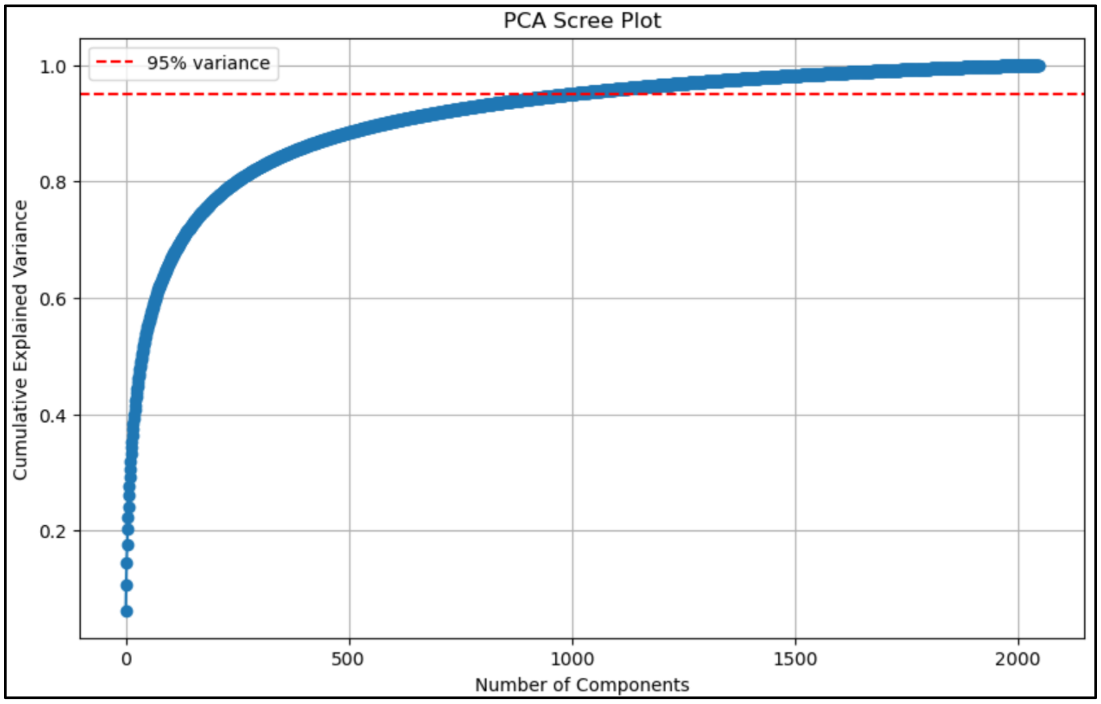
df= pd.DataFrame(best_emb, columns=["x", "y", "z"])
df["label"] = best_labels
df["label_str"] = df["label"].apply(lambda l: "Noise" if l == -1 else f" Cluster {l}")
fig = px.scatter_3d(df, x="x", y="y", z="z", color = "label_str", title="3D HDBSCAN Cluster Visualization", labels={"label_str": "Cluster"},opacity=0.7)

This code visualizes HDBSCAN clustering results in 3D using Plotly Express. It creates a DataFrame from embedding data and cluster labels, converting the numerical labels (including noise represented by -1) into descriptive strings in a new 'label_str' column. The px.scatter_3d function then generates an interactive 3D scatter plot, coloring each data point according to its cluster assignment (or as "Noise"). This visualization provides a clear and interactive way to examine the identified clusters in the three-dimensional embedding space.



Results/Analysis

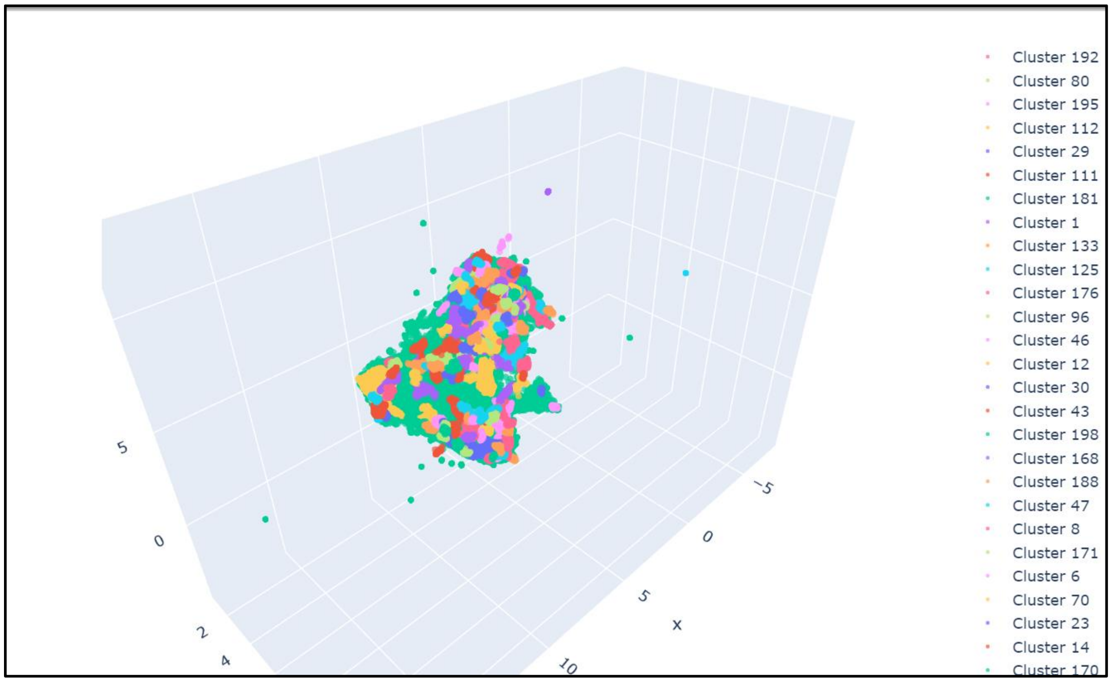
The scree plot below shows the cumulative explained variance as a function of the number of principal components (out of approximately 2,100 total). A red dashed line marks the 95% variance threshold.



The PCA scree plot reveals that the very first component alone captures roughly 5–7% of the total variance and the first ten components about 30%, highlighting substantial redundancy in the original feature set. A pronounced “elbow” appears around 200–300 components—by this point you already explain approximately 85–90% of the variance—after which the curve begins to flatten, and each additional axis adds progressively smaller gains. To reach a 95% explained-variance threshold, you would need to retain around 900–1,000 components, underscoring the trade-off between near-complete variance preservation and computational efficiency. Thus, reducing to 300 dimensions offers a strong balance of information retention and speed for downstream tasks, while expanding toward 900+ dimensions may be justified when maximal variance capture is critical.

For downstream analyses, striking the right balance in dimensionality reduction is key: if preserving nearly all variance (around 95%) is essential—such as before unsupervised clustering or anomaly detection—you may opt for roughly 900 principal components, whereas accepting a more aggressive reduction (retaining 85–90% variance) allows you to pare down to about 300 dimensions and thereby substantially speed up algorithms like UMAP or HDBSCAN with only minimal information loss. Furthermore, discarding the long tail of low-variance components acts as a form of noise filtering, helping to suppress spurious correlations and often leading to tighter, more stable clusters. A practical strategy is to start with 300 PCs and assess clustering performance (for example, via the silhouette score); if key structures appear degraded or performance suffers, you can gradually increase toward 900 components to reclaim additional variance and structure.

The 3D scatter plot displays the results of an image clustering analysis. Each dot represents an image, and the color of the dot indicates the cluster to which that image has been assigned.



Here's a breakdown of what we can observe:

Multiple Clusters: The presence of various colors clearly shows that the images have been grouped into numerous distinct clusters. The legend on the right provides a key, listing each cluster number and its corresponding color.

Cluster Density and Separation: We can visually assess how compact each cluster is and how well-separated the clusters are from each other in this 3D space.

- **Dense Clusters:** Some clusters appear to be tightly packed, suggesting that the images within these clusters are quite similar to each other based on the features used for clustering.
- **Sparse Clusters:** Other clusters might be more spread out, indicating greater variability among the images within them.
- **Well-Separated Clusters:** Ideally, we want to see distinct groups of colors with clear boundaries between them. This would imply that the clustering algorithm has effectively identified groups of dissimilar images.
- **Overlapping or Close Clusters:** If we observe regions where different colored dots are intermingled or very close, it might suggest that some images are difficult to assign definitively to a single cluster, or that the underlying image features have similarities across these clusters.

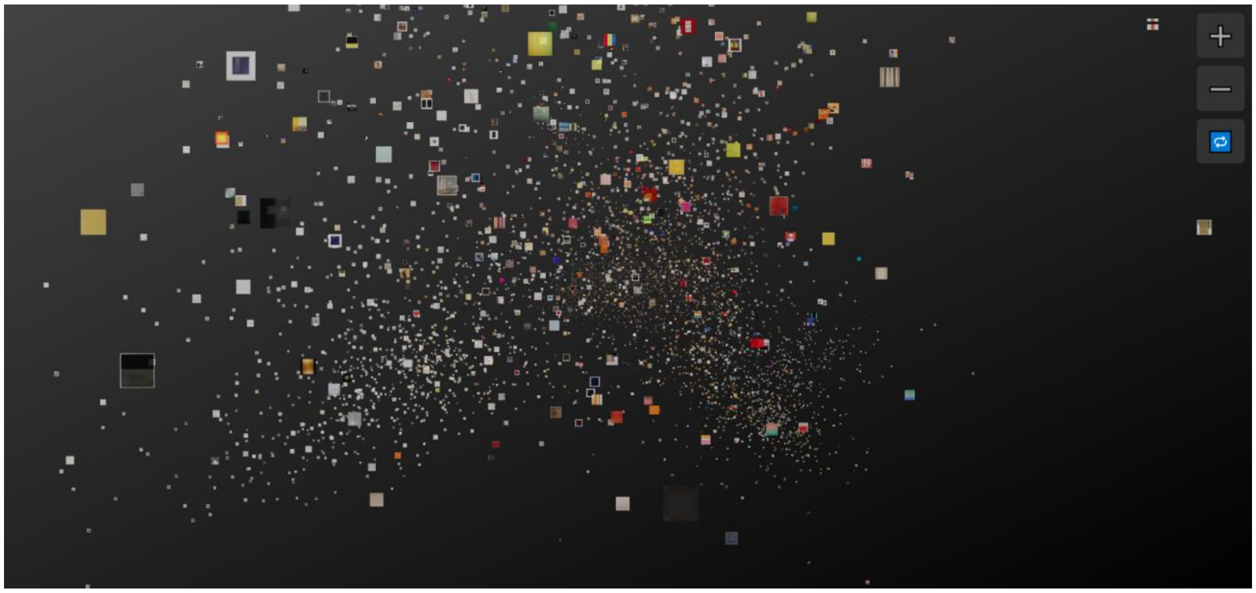
Dimensionality Reduction: Since we are visualizing the clusters in a 3D plot, it's highly likely that the original image features were of a much higher dimension. A dimensionality reduction technique (like PCA, UMAP) has probably been applied to project the images into this 3D space for visualization. It's important to remember that this 3D representation is a lower-dimensional approximation of the true relationships between the images.

Potential Insights: By examining this plot, we can gain a qualitative understanding of how the clustering algorithm has grouped the images. Further analysis would involve:

- Looking at representative images from each cluster to understand the semantic meaning of the groupings.
- Quantifying the quality of the clustering using metrics like silhouette score or Davies-Bouldin index.
- Investigating the features used for clustering to understand what aspects of the images are driving the formation of these groups.

In summary, this 3D visualization provides a valuable initial look at the image clustering results, allowing us to see the number of clusters, their relative density, and their separation in the reduced dimensional space. However, a deeper understanding requires further investigation of the cluster contents and quantitative evaluation.

Interactive 3D map of artwork



Challenges

1. Dataset Scale and Quality

Large Dataset (120,000 high-res images) and data quality challenges:: High storage and I/O overhead and slower processing for feature extraction & visualizationSkipped/corrupt image logs indicate missing or unreadable file inconsistent image formats or dimensions

2. Computational Demands

ResNet50 for Feature Extraction with high GPU/TPU memory requirements ,also require efficient batching to prevent memory overflow. Dimensionality reduction & clustering, UMAP/PCA are memory-intensive on large vectors, parameter tuning (e.g., UMAP, silhouette score) adds compute cost

3. System Integration

3D interactive map, performance bottlenecks in real-time rendering of large datasets and limited by browser and frontend capabilities.

4. User Experience

Exploration design map must reflect meaningful artistic/stylistic relationships, smooth, intuitive navigation across thousands of data points

5. Reproducibility & Scalability

Pipeline Complexity: Multi-stage pipeline: extraction → reduction → clustering → visualization. Ensuring reproducibility and adaptability to new datasets

References:

[1] "Considerably Improving Clustering Algorithms Using UMAP Dimensionality Reduction Technique: A Comparative Study" Springer Nature Switzerland AG 2020 A. El Moataz et al. (Eds.): ICISP 2020, LNCS 12119, pp. 317–325, 2020. https://doi.org/10.1007/978-3-030-51935-3_34

[2] Yingfan Wang, Haiyang Huang, Cynthia Rudin, Yaron Shaposhnik "Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMap, and PaCMAP for Data Visualization", 2021 Journal of Machine Learning Research 22(2021) 1-73

[3]https://www.kaggle.com/datasets/antoinegruson/-wikiart-all-images-120k-link

[4]https://www.researchgate.net/figure/Convolutional-neural-network-on-the-feature-extraction_fig2_367421134

Reflection:

This exercise revealed just how difficult it would be to implement a visual similarity-based artwork mapping system without machine learning. Manually designing features and similarity metrics is both subjective and error-prone, especially when dealing with the complexity and richness of fine art. It also doesn't scale well with large datasets, as pairwise comparison of high-dimensional handcrafted features becomes computationally expensive and often less meaningful.

Defining the problem for AI clarified the immense value of using deep learning: it enables automatic extraction of high-level visual features that are far more nuanced and abstract than anything we could handcraft. The challenge in articulating the problem was identifying how to move from raw image data to a meaningful spatial visualization — once that connection was made through embedding and dimensionality reduction, the AI-based approach became the obvious solution.