DS203 E7

Ayush singh
(22B2203)
Ashutosh Agarwal
(22B2187)

Preprocessing

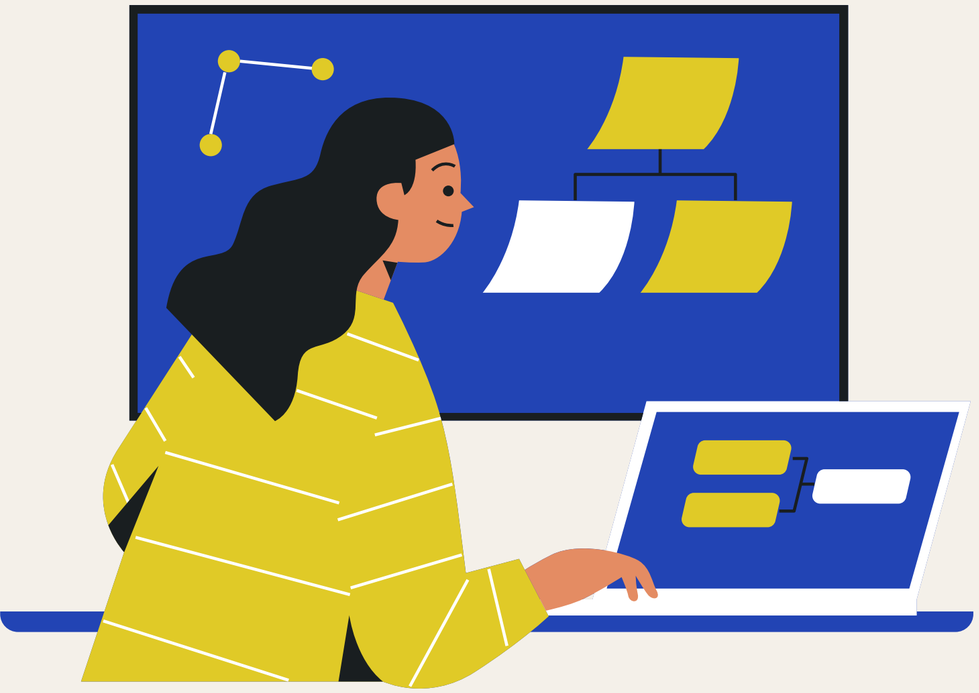


Image Preprocessing:

- Resized the images to have consistent dimensions for easier processing
- Converted the images to grayscale to simplify feature extraction and reduce computational complexity.
- Apply noise reduction techniques.
- Enhance contrast for better feature visibility.
- Normalize pixel values to a common range.

Data preprocessing simplifies the communication of analysis findings

Feature Extraction:

- Edge Detection: Used the techniques like Canny edge detection to extract edge features, which can capture layout boundaries and shapes effectively.
- Contour Detection: Identified the contours to capture shape information and extract features like area, perimeter, and number of vertices.
- Texture Analysis: Extracted the texture features using methods like GLCM (Gray-Level Co-occurrence Matrix) to capture layout details.
- Deep Learning Features: Utilized pre-trained deep learning models (e.g., VGG, ResNet) to extract high-level features from images.

```
from torchvision.models.detection import fasterrcnn_resnet50_fpn
```

✓ 0.0s

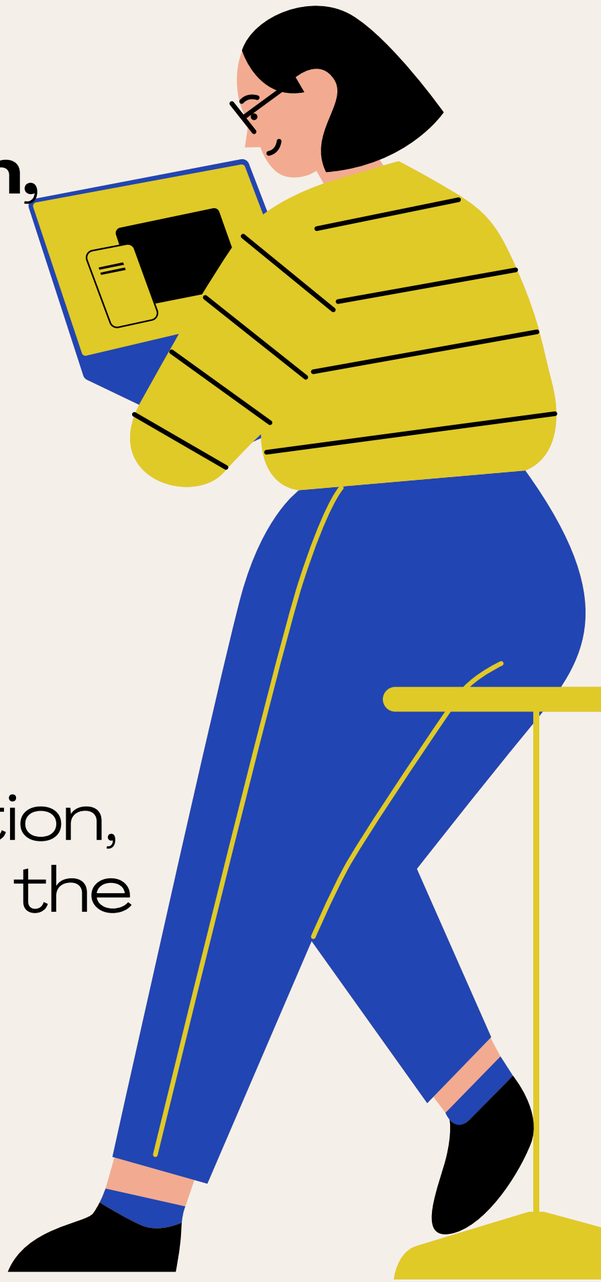
```
# Load pre-trained Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()
```

Tight-Fitting Box Formation

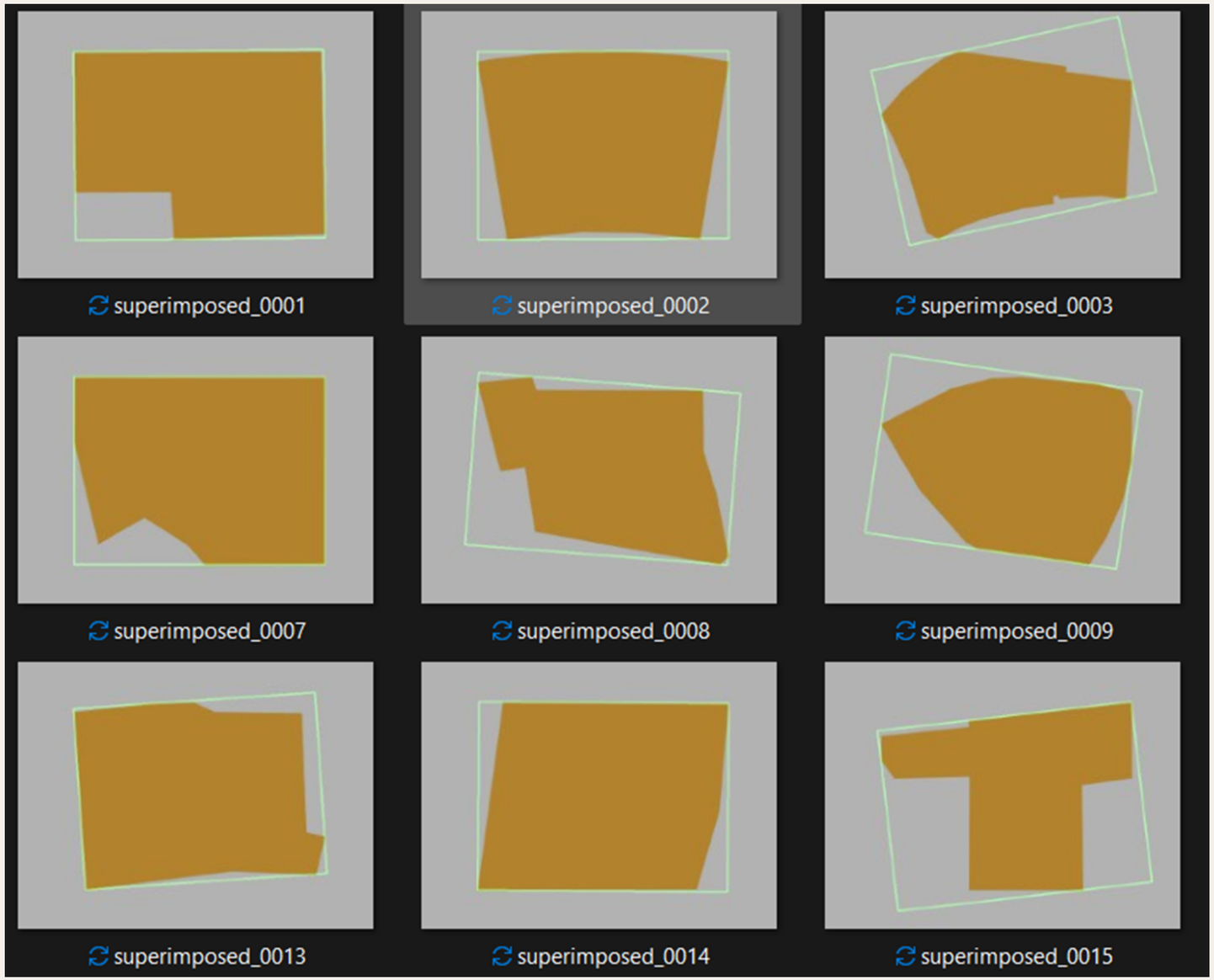
Our image processing pipeline performs **edge detection, contour localization, and minimum area bounding rectangle (MABR) visualization.**

Leveraging the Canny edge detection method, we accurately identify edges in grayscale images. Subsequently, contours are extracted using the cv2.findContours function with parameters for contour retrieval and approximation. The contours are then enclosed within MABRs using the cv2.minAreaRect function, which computes the minimum bounding box aligned with the object's orientation.

Data



The bounding box images as stored in the other directory can be viewed as :



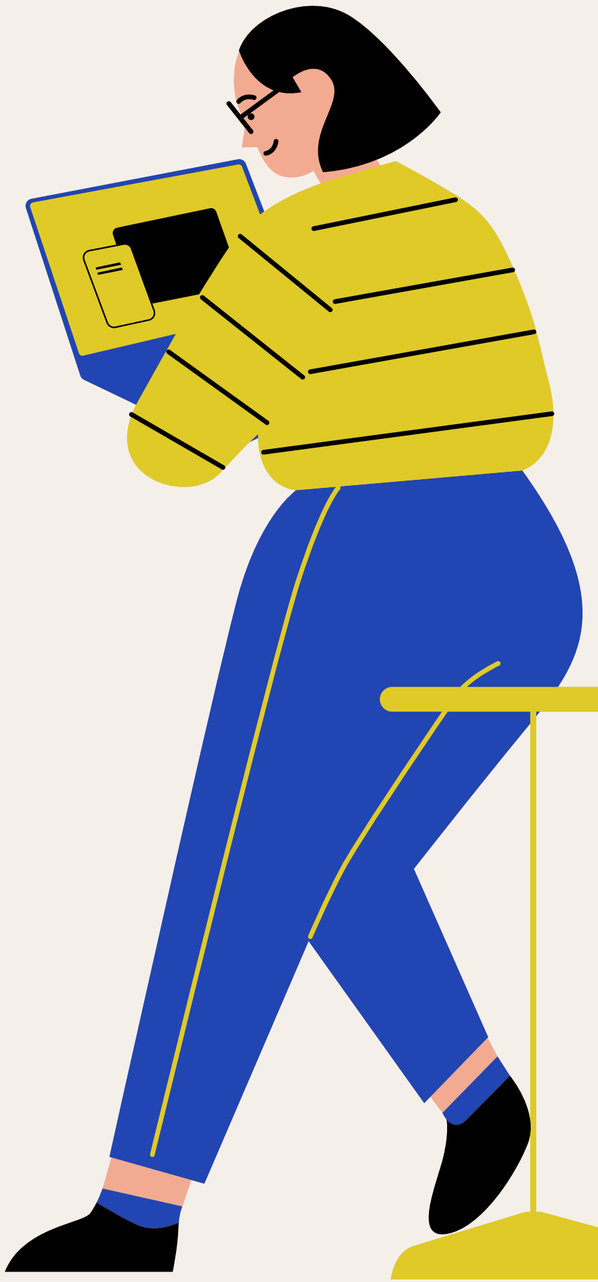
Visualization

Contour Plot Box Formation

Utilizing the **Canny edge detection** algorithm, we identify edges in the grayscale representation of the original images. Subsequently, contours are extracted using the cv2.findContours method with specified parameters for contour retrieval and approximation.

These contours, representing the boundaries of objects in the images, are then overlaid onto the original images as green lines using the cv2.drawContours function. This facilitates a visual representation of the detected contours

Data



The contour plot box images as stored in the other directory can be viewed as :

Visualization

Extracting Unique Images

The output features was of the shape (**1183,7**) i.e. the 7 features of all of the images. But there were many of them repeated ones.

So, we applied the unique property and stored the unique images in a different directory.

So, finally we were left with the only 173 unique images.

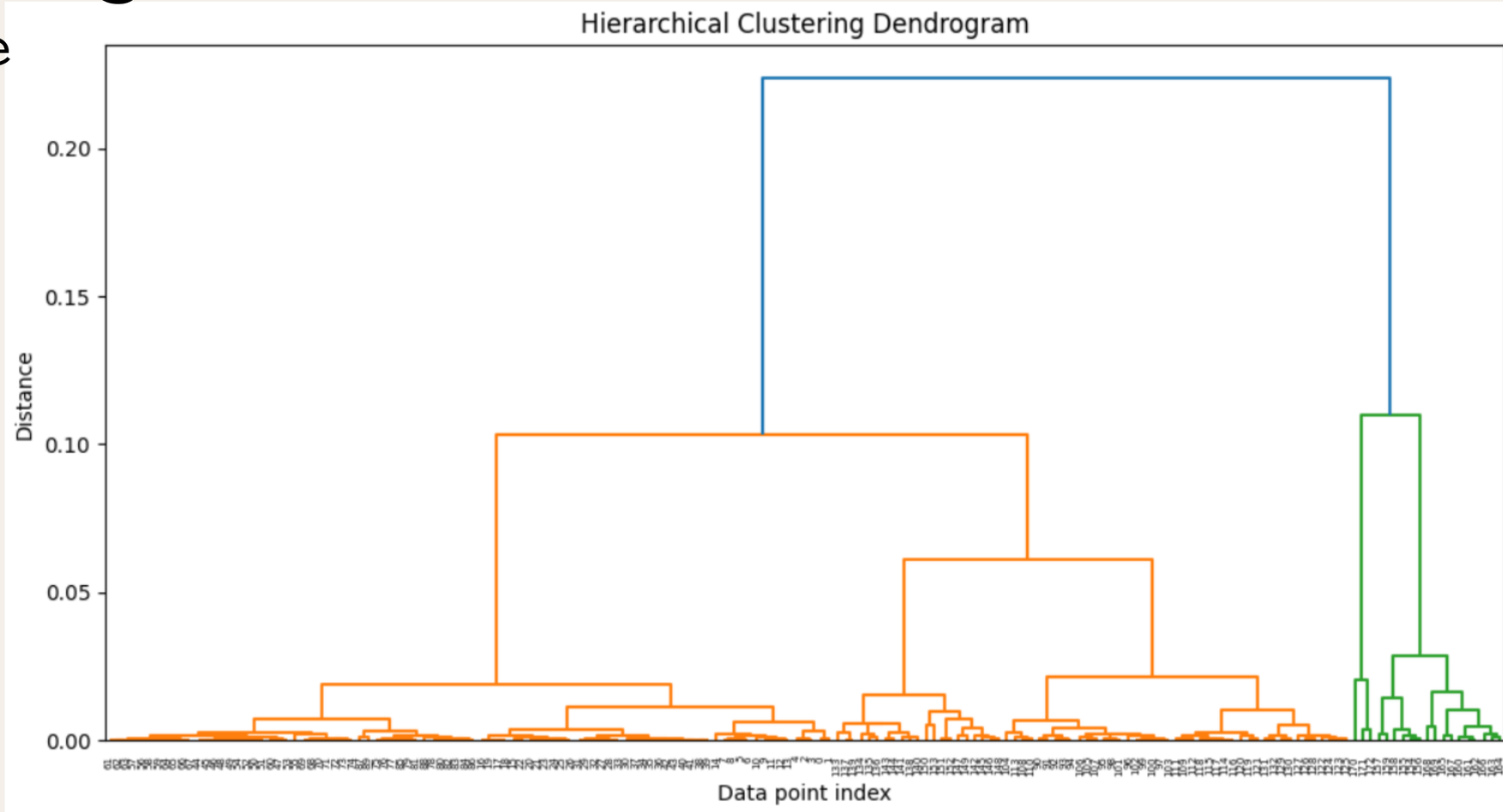
```
shape_features_orig.shape
```
✓ 0.0s
```
(1183, 7)
```
```
shape_features = np.unique(shape_features_orig, axis=0)
```
✓ 0.0s
```
shape_features.shape
```
✓ 0.0s
```
(173, 7)
```

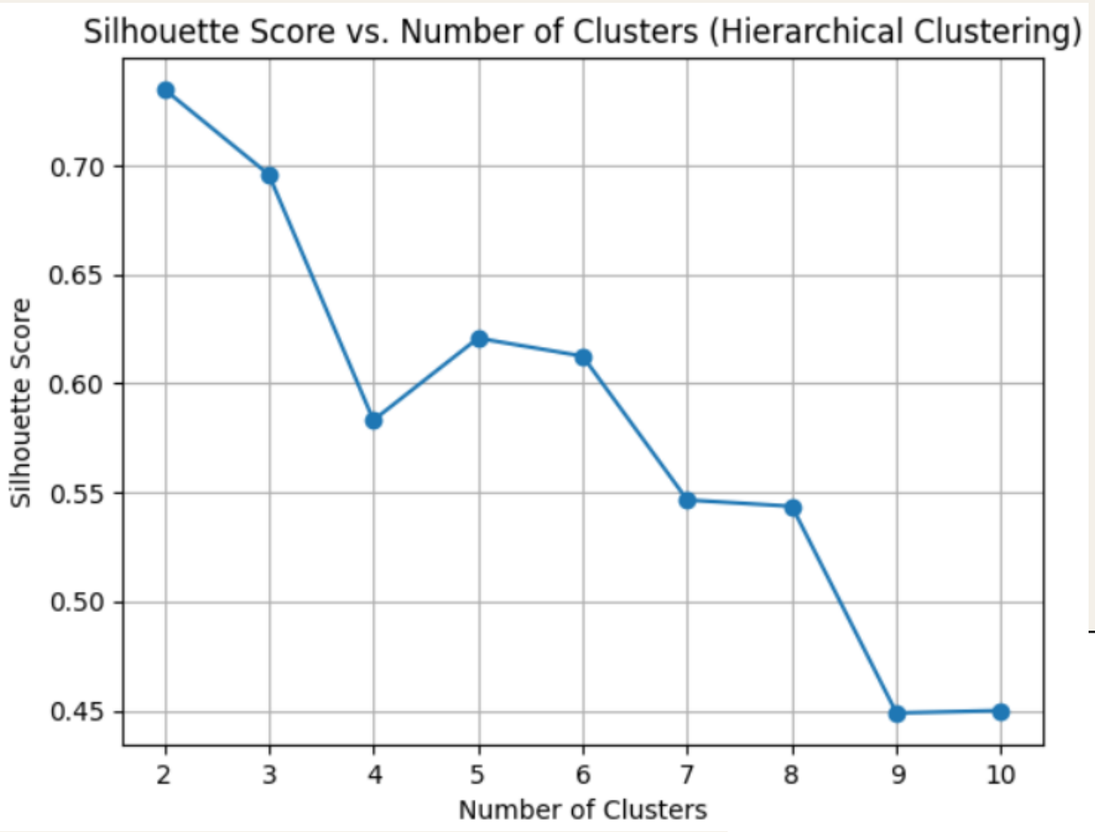
Clustering the Images

In order to classify the images in the clusters first of all we applied the **Hierarchical Clustering** method and plotted the dendrograms for it.

The error metric we used for this was the **Silhouette Score**.

From the dendrogram plot by visualization and optimum pruning, the no of cluster is 5





Silhouette Score vs no of Clusters for Hierarchical Method

Images belonging to the some of the groups

First 6 images belonging to cluster 8 (6 images in total)

First 10 images belonging to cluster 1 (46 images in total)

Spectral Clustering Method

Spectral clustering is based on the similarity of their features. It operates by constructing a similarity graph, where nodes represent data points and edges indicate pairwise similarities.

Spectral clustering leverages the eigenvalues and eigenvectors of the graph **Laplacian matrix** to embed the data points into a lower-dimensional space, where clustering is performed using traditional methods like k-means.

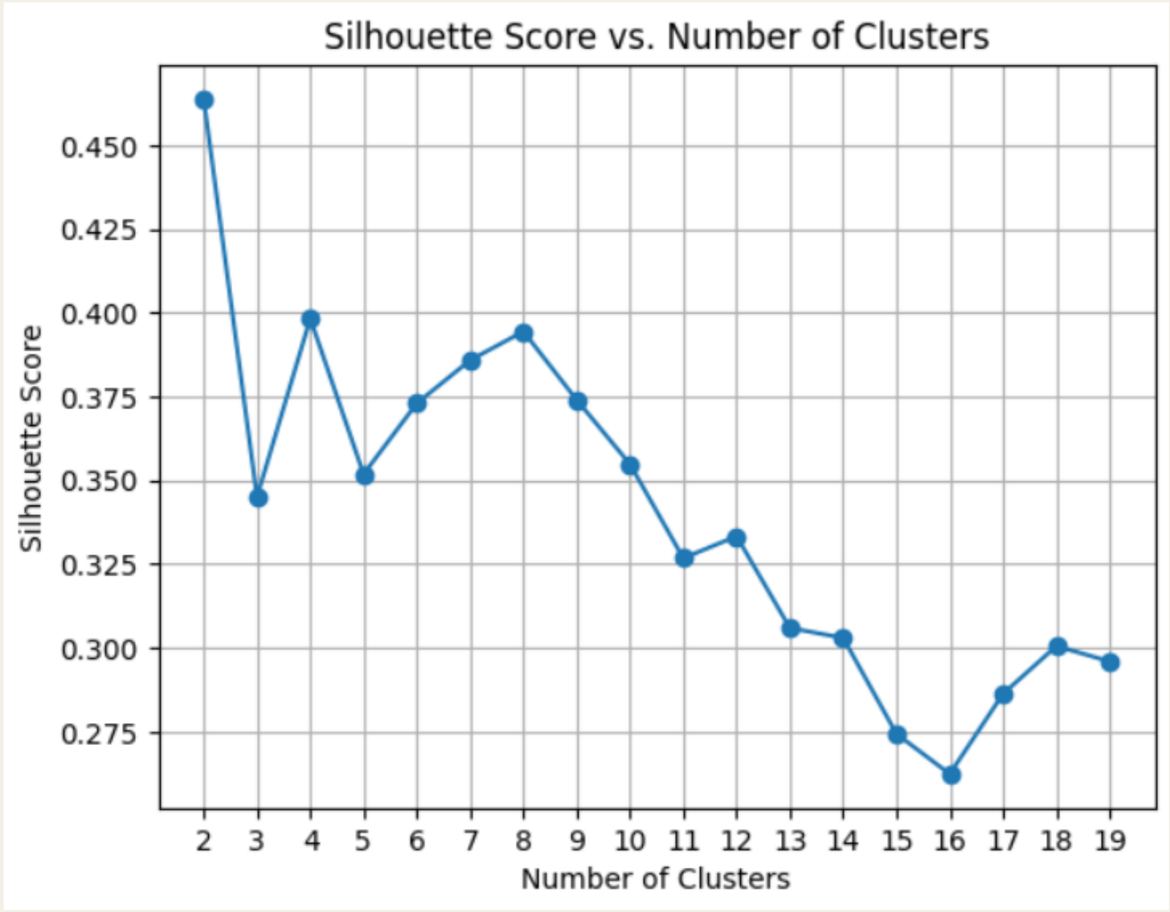
```
for n_clusters in cluster_range:
    # Create a Spectral Clustering model
    spectral_model = SpectralClustering(n_clusters=n_clusters, affinity='nearest_neighbors'

    # Fit the model to the data
    spectral_clusters = spectral_model.fit_predict(shape_features_array)

    # Compute the silhouette score
    silhouette_avg = silhouette_score(shape_features_array, spectral_clusters)

    # Append the silhouette score to the list
    silhouette_scores.append(silhouette_avg)

# Find the optimal number of clusters with the maximum silhouette score
optimal_num_clusters = cluster_range[np.argmax(silhouette_scores)]
max_silhouette_score = max(silhouette_scores)
```

From the graph of the Silhouette Score of the Spectral clustering method, we can see that after the dip of the graph from the n=2 it is max at the n=4.

So, considering the previous model too we arrive at the optimum no of clusters

as **4 and 5**

Silhouette Score vs no of Clusters for Spectral Clustering Method



Complexity ANALYSIS 01

Done using feature extraction from VGG19 dataset.

Below are the steps included for the same.

- Utilized VGG19 to extract features from building layout images.
- Features represent high-level characteristics of layouts.
- Applied Agglomerative Clustering to group layouts into complexity levels.
- Three complexity levels: Low, Medium, and High
- Present categorized layouts: Low Complexity, Medium Complexity, and High Complexity.
- Results aid in understanding layout complexity distributions.
- Visualize sample images from each complexity level.
- Illustrate the diversity of layouts within each complexity category.

Complexity ANALYSIS 01

RESULTS

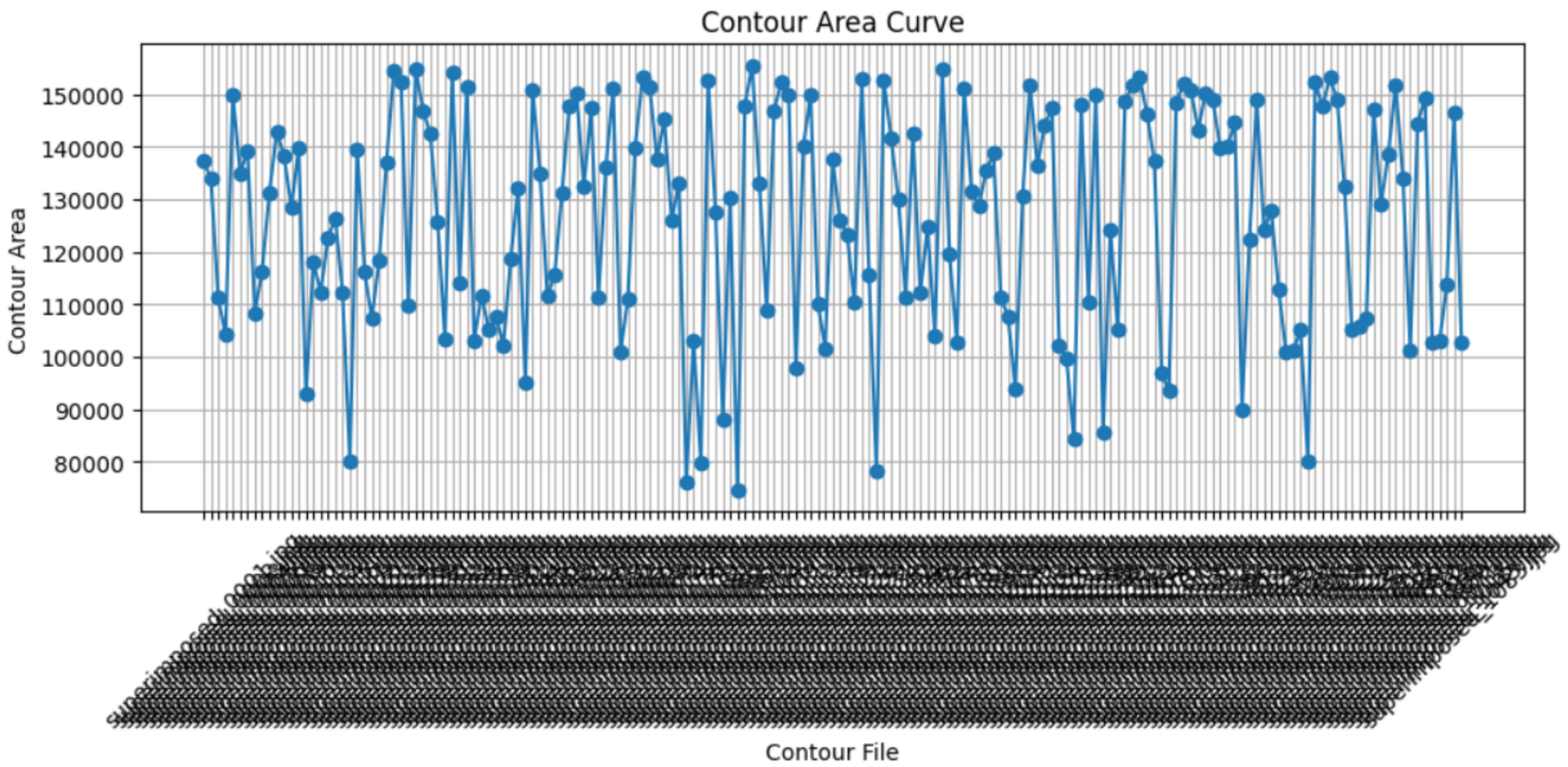
As quite visible from these diagrams , our model has worked quite well, with increasing complexity when edges/area ratio increases and hence our 2nd method in subsequent slides.

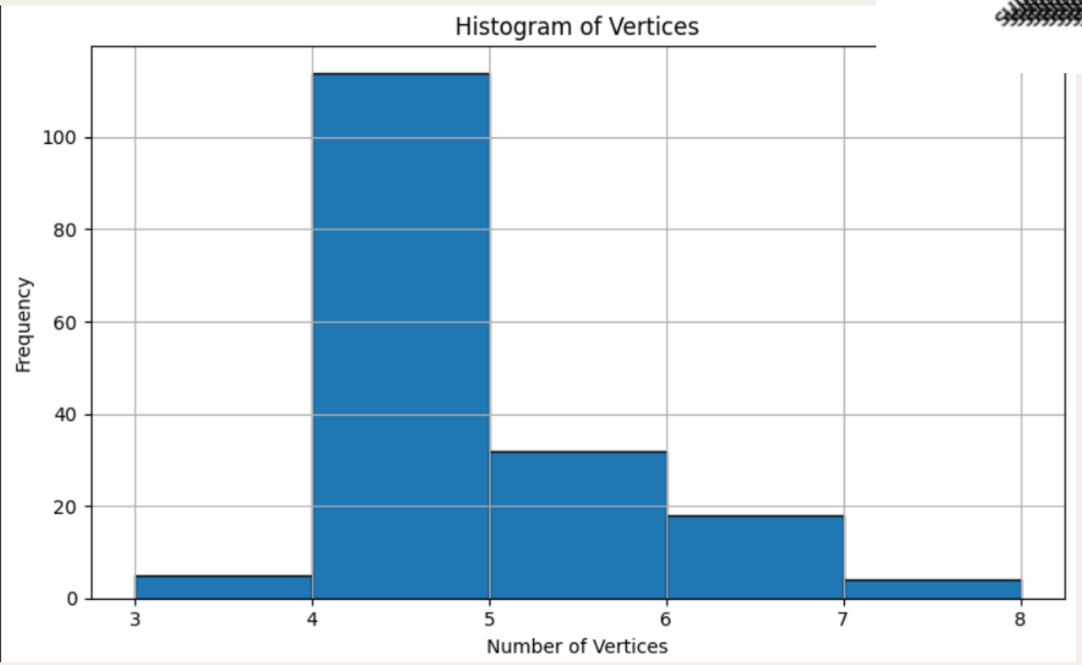
Complexity Analysis 02

For the contours we had generated we calculated the area of the each of the contour and the no of the vertices of the contours.

1. **calculate_contour_area(contour):** calculates the area enclosed by a contour. It uses the cv2.contourArea() function provided by OpenCV, which computes the area of the contour using the Green's theorem.
2. **detect_contour_edges(contour):** detects the edges of a contour. It first creates a blank canvas and draws the contour on it. Then, it applies the Canny edge detection algorithm to find the edges of the contour on the canvas. This process helps visualize the shape of the contour,

Plot of the area of the images.



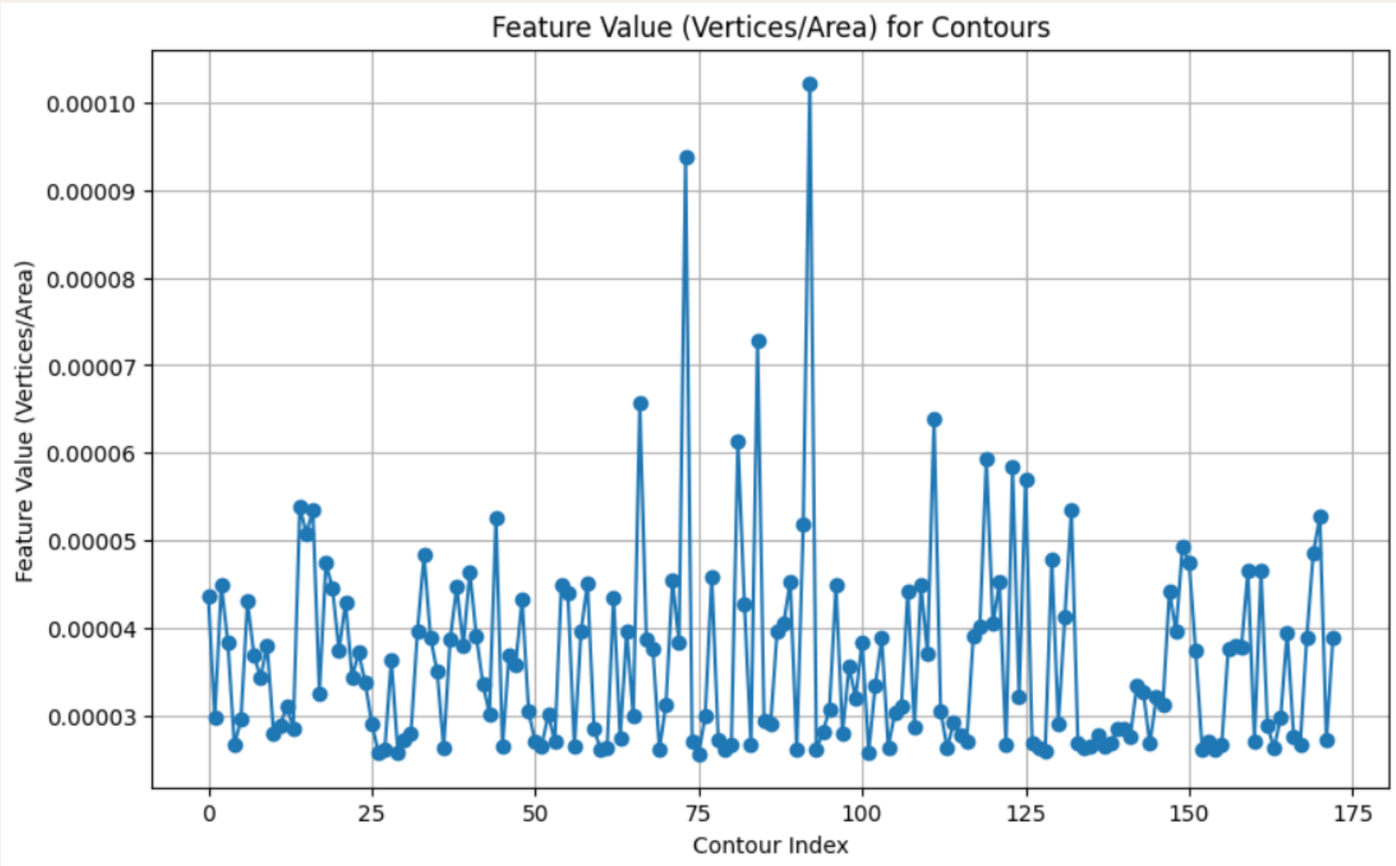


Histogram plot of the frequency of the no of vertices.

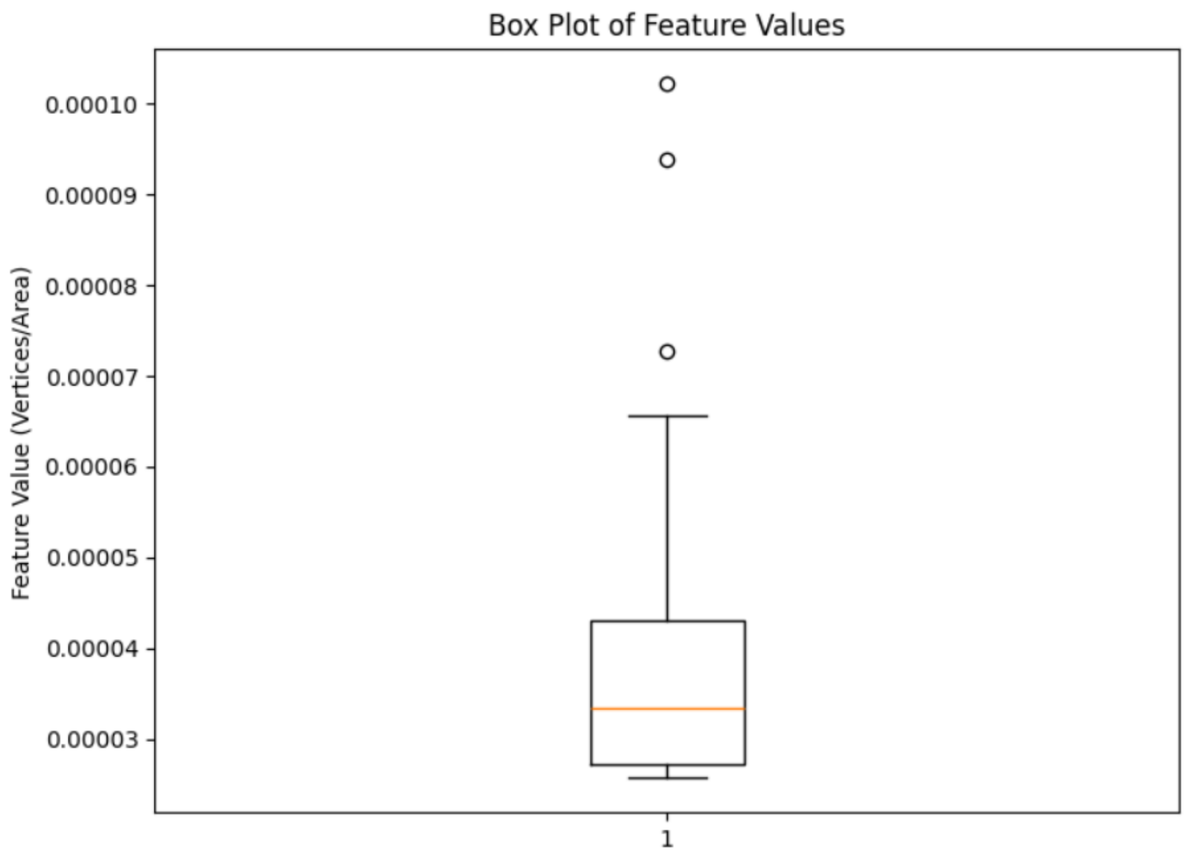
count_contour_vertices(contour): counts the number of vertices of a contour. It first approximates the contour using the Douglas-Peucker algorithm to reduce the number of points while preserving the shape. Then, it counts the number of vertices in the approximated contour, which gives an estimate of the complexity of the contour shape.

Then we created a feature value = **vertices/area.**

More the value of this feature more the complex the images will be as there in the smaller area there are large no of the vertices, making the image more complex.



Then after creating this feature we found the percentiles of images lying by using the box plots.

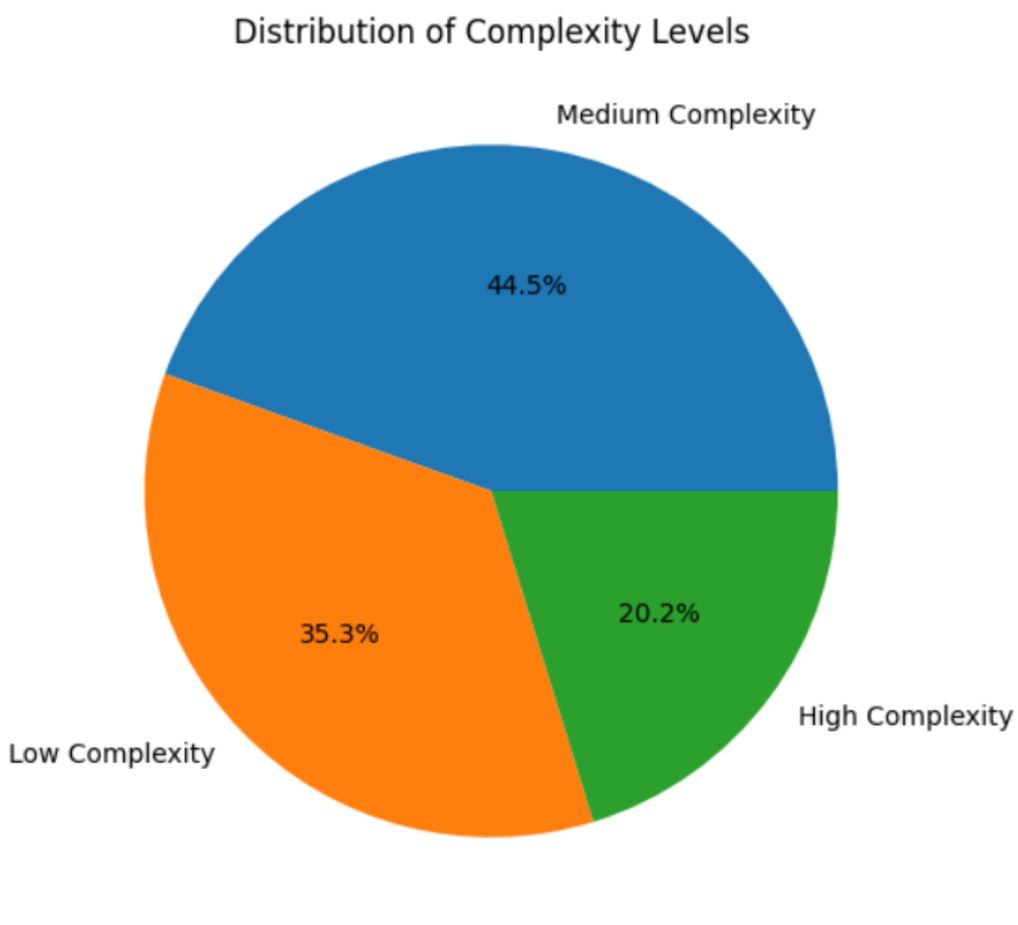



```
Number of Images within Each Percentile Region:
25th to 50th percentile: 43 images
50th to 75th percentile: 43 images
75th to 90th percentile: 26 images
90th to 95th percentile: 9 images
95th to 99th percentile: 7 images
```

Then manually we allotted the three regions for classes.

```
Number of Images in Each Complexity Level:
Complexity
Medium Complexity    77
Low Complexity       61
High Complexity      35
Name: count, dtype: int64
```

Pie chart showing the regions of the each of the complexities.



Generating outputs for the users as per the data entered by them i.e. the images which belong to same complexity and area near to them

1. Data Input:
    - Users provided input data, such as area and complexity level, through an interface or input form.
    - The input data included parameters such as the area of the layout and the desired complexity level (low, medium, or high).

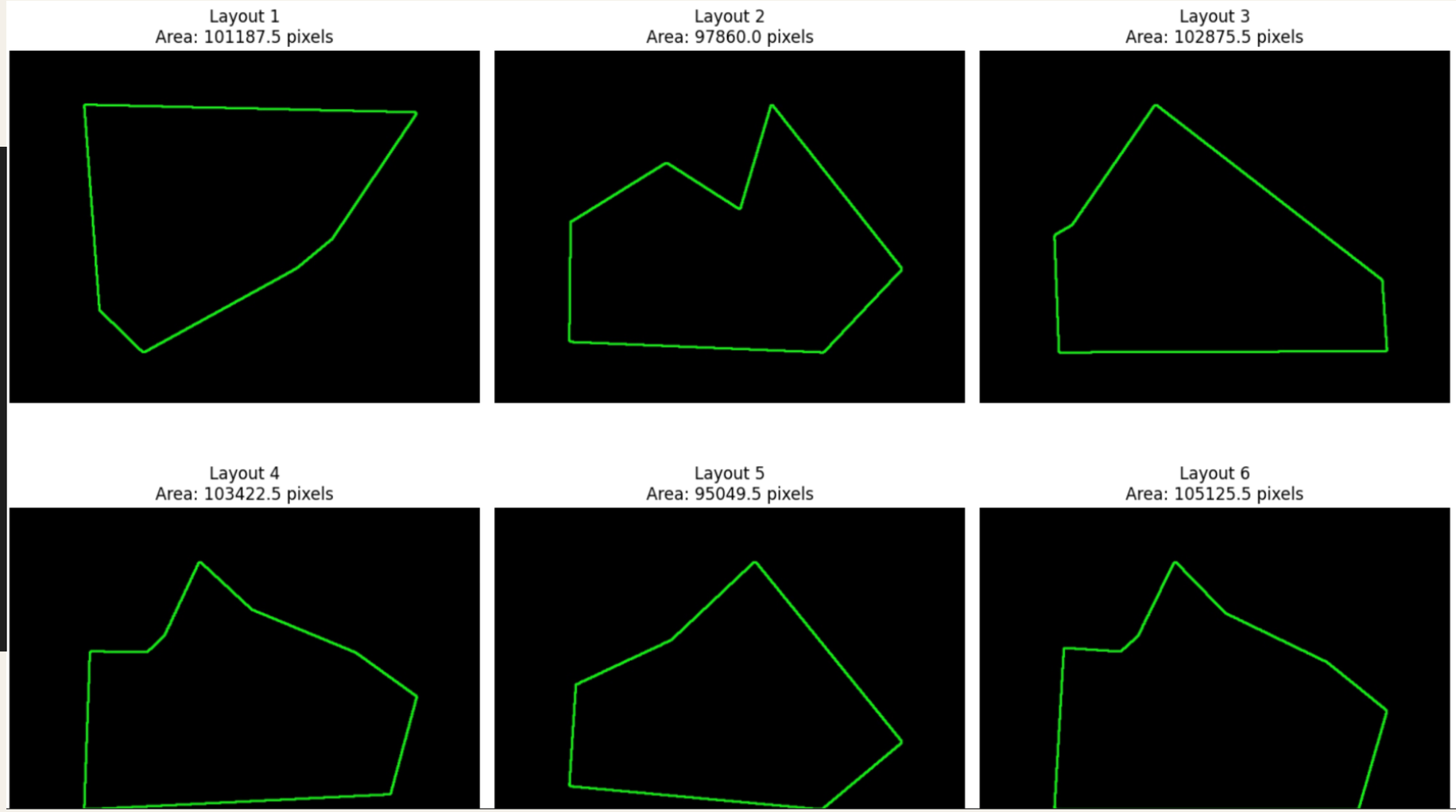
Data Processing: The input data was processed by the system to identify relevant images from a dataset. Based on the specified complexity level and area, the system filtered the dataset to select images that matched the user's criteria.

Image Generation: Once the relevant images were identified, the system retrieved them from the dataset. These images were then displayed or visualized to the user, providing a visual representation of layouts that met the specified criteria. Users could review the generated images to evaluate potential layout options based on their input parameters.

```
specified_area = float(input("Enter the layout area: "))
specified_complexity_index = int(input("Enter the complexity level (0 for Low, 1 for Medium, 2 for High): "))
```

For example : We entered the input and the corresponding 6 related images are on next slide:

```
Loaded image shape: (480, 640, 3)
Plotting image 1 in position 0, 0
Loaded image shape: (480, 640, 3)
Plotting image 2 in position 0, 1
Loaded image shape: (480, 640, 3)
Plotting image 3 in position 0, 2
Loaded image shape: (480, 640, 3)
Plotting image 4 in position 1, 0
Loaded image shape: (480, 640, 3)
Plotting image 5 in position 1, 1
Loaded image shape: (480, 640, 3)
Plotting image 6 in position 1, 2
```



Suggestions from the data mined

Generative Models:

Generative models, particularly **Generative Adversarial Networks** (**GANs**), can be trained on the existing layout data to generate entirely new building layouts. By learning the underlying patterns and structures present in the dataset, GANs can produce novel designs that adhere to the constraints and characteristics of the original layouts. Architects can use these generated designs as inspiration or starting points for their projects.

Recommendation System:

A recommendation system can be made using the data that can analyze the preferences and **past choices** of architects and recommend similar building layouts based on their design history. By leveraging machine learning algorithms, the system can identify patterns in the architect's preferences and suggest layouts that align with their **complexity** or functional preferences. This can streamline the design process by providing architects with relevant and personalized recommendations.

Suggestions from the data mined

Quality Assessment:

Quality assessment involves evaluating the quality of building layouts based on predefined criteria such as **complexity** and structural integrity. Machine learning models can be trained to assess these aspects of layouts by analyzing their features and characteristics. For example, a model could analyze the proportions of rooms, the flow of space, and the **structural stability** of a layout to determine its overall quality. This can help architects identify areas for improvement and ensure that their designs meet high standards.

Interactive Design Tools:

Interactive design tools empower architects to manipulate design parameters and visualize the impact on the layout in real-time. These tools can provide instant feedback on changes made to the layout, allowing architects to experiment with different configurations and make informed decisions. Machine learning can enhance these tools by predicting the consequences of design changes and suggesting alternatives based on the desired outcome. This can facilitate a more iterative and collaborative design process, leading to better-designed buildings.

CONCLUSIONS

The project successfully employed machine learning techniques to analyze building layout complexity.

Visual representation enhanced comprehension of complexity distributions.

LEARNINGS

Effective utilization of pre-trained models like VGG19 for feature extraction.
Understanding and implementation of clustering algorithms for grouping similar data points.
Importance of visual representation in conveying complex data insights.