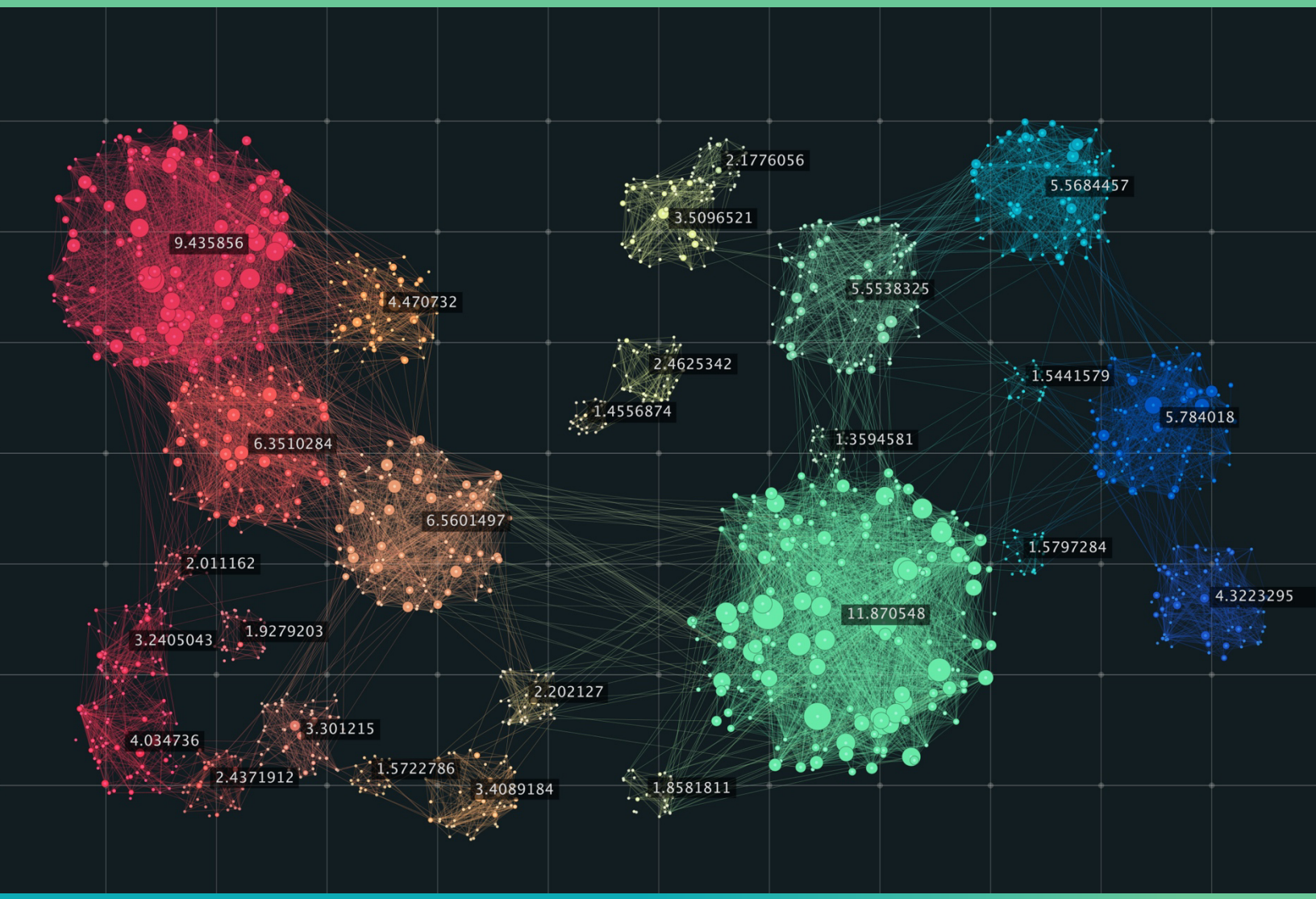
# CSE 480 MACHINE VISION




## *Image Cluster*


Morcous Osama Kamal Faik 19P6167

Amal Amr AbdelHamid Zewita 19P1504

Abdelhamid Salem Abdelhamid 15P8098

# Table of Contents

## List of Figures

# 1.0 PROMBLEM DEFINITION AND IMPORTANCE

Image clustering and classification are two fundamental problems in computer vision that involve analyzing and organizing images based on their visual content and characteristics.

## 1.1 Image Clustering

Image clustering is the process of assembling related pictures based on their visual characteristics. Without any prior knowledge of the labels or categories assigned to the photographs, the objective is to find patterns, similarities, and links among them. The goal of clustering algorithms is to group together pictures that have similar visual qualities, such as color, texture, form, or more complex semantic aspects.

In order to get high-level feature representations, picture clustering requires collecting pertinent features from images using methods like deep learning-based convolutional neural networks, CNNs. After that, related photos are grouped together into clusters using clustering algorithms like k-means, hierarchical clustering, or density-based algorithms, which employ these attributes as input. The clustering outcomes can be useful in a variety of applications, including unsupervised learning, picture organization, and image retrieval.

## 1.2 Image Classification

Picture classification entails categorizing or labelling images based on their visual content. The objective is to develop a model that can recognize and categories photos automatically into various groups or classes. In supervised image classification, each image is assigned a specific class label, and the model is trained on this labelled dataset. The model gains the ability to extract distinguishing characteristics from the photos and map them to the appropriate classes.

CNNs effectively classify images using hierarchical representations and optimization methods. They are used in face recognition, autonomous driving, object recognition, and medical imaging. Both clustering and classification are crucial in computer vision, handling specific aspects of image processing. Classification assigns unique titles to each image, while clustering groups related photos into sets. These skills are essential in various fields for effectively analyzing, organizing, and comprehending visual data.

# 2.0 METHODS AND ALGORITHMS

While image classification seeks to place photos into preset groups or classes, image clustering includes gathering together comparable images. It is crucial to comprehend these processes and algorithms if you want to analyze and arrange enormous picture sets efficiently. The essential ideas and applications of a variety of methodologies are examined in this research, including both conventional and deep learning-based approaches. There are various methods and algorithms, commonly employed approaches:

## 2.1 Image Clustering

1. K-means Clustering
2. Hierarchical Clustering
3. Density-based Spatial Clustering of Applications with Noise (DBSCAN)
4. Gaussian Mixture Models (GMM)
5. Spectral Clustering

### 2.1.1 K-means clustering

The K means clustering algorithm divides a set of n observations into k clusters, assigning similar data points to the specified number of groups. Clustering is a method of assigning data points to groups using data patterns, and K means clustering is one such algorithm. The process of assigning observations to the nearest center (mean) minimizes variances between data points and the cluster's centroid.

### 2.1.2 Hierarchical clustering

Hierarchical clustering is a data mining method that creates a hierarchical series of nested clusters. Iteratively combining closest clusters until a stopping criterion is reached, resulting in a dendrogram, a tree-like structure that illustrates the relationships among the clusters.

### 2.1.3 DBSCAN

DBSCAN is a popular clustering algorithm used for data analysis and pattern recognition. DBSCAN groups together images that are densely connected in the feature space, separating them from sparse regions. It groups data points based on their density, identifying clusters of high-density regions and classifying outliers as noise. DBSCAN is effective in discovering arbitrary-shaped clusters in data and is widely used in data mining, spatial data analysis, and machine learning applications.

### 2.1.4 GMM

GMM assumes that the data is generated from a mixture of Gaussian distributions, allowing images to be assigned to clusters based on the probability distribution. Gaussian mixture models can be used to cluster unlabeled data in much the same way as k-means. There are, however, a couple of advantages to using Gaussian mixture models over k-means.

### 2.1.5 Spectral cluster

Spectral Clustering is a variant of the clustering algorithm that uses the connectivity between the data points to form the clustering. It uses eigenvalues and eigenvectors of the data matrix to forecast the data into lower dimensions space to cluster the data points. It is based on the idea of a graph representation of data where the data point are represented as nodes and the similarity between the data points are represented by an edge.

## 2.2 Image Classification:

1. Convolutional Neural Networks (CNN)
2. Support Vector Machines (SVM)
3. Random Forests
4. Deep Neural Networks (DNN)
5. Transfer Learning

### 2.2.1 CNN

CNNs are highly effective in image classification tasks, learning hierarchical representations using convolutional layers, pooling layers, and fully connected layers. They are widely used in artificial intelligence modeling, particularly for creating image classifiers. CNNs are ubiquitous in the image data space and are widely used in object detection and recognition.

### 2.2.2 SVM

SVMs are supervised learning models that classify images by finding an optimal hyperplane to separate different classes. They can handle both linear and non-linear classification tasks. Extracting features like color values, edge detection, or textures is necessary for the SVM algorithm, which finds the hyperplane that maximizes the margin between support vectors.

### 2.2.3 Random forest

Random Forests are ensemble learning methods that combine multiple decision trees to make predictions. They can handle high-dimensional feature spaces and provide robust classification performance. The RF relies on many self-learning decision trees which in their sum make up a “Forest”. The idea behind using many decision trees (i.e. an ensemble) is that many base learners can come to one strong and robust decision compared to a single DT.

### 2.2.4 DNN

DNNs, also known as multilayer perceptron, consist of multiple hidden layers and are capable of learning complex representations. They have been successfully applied to image classification tasks.

### 2.2.5 Transfer learning

Transfer learning involves using pre-trained models that are trained on large-scale datasets, such as ImageNet. These models are fine-tuned on specific image classification tasks, leveraging the learned features from the pre-training. Transfer learning is a method of reusing the already acquired knowledge.

# 3.0 PROGRAM

## 3.1 GUI

Our GUI geometry is 300 × 200 It serves as an interface for performing image clustering. It starts by asking about the number of clusters for image clustering process and it allows the user to enter the number of clusters they want to create. Then there is a label that instructs the user to select the images they want to cluster.

A label asks the user to enter the number of images per cluster This label prompts the user to enter the desired number of images they want to display per cluster in the visualization. The "Set" button triggers the execution of a sequence of actions, including processing the entered values, clustering the images, and visualizing the results.

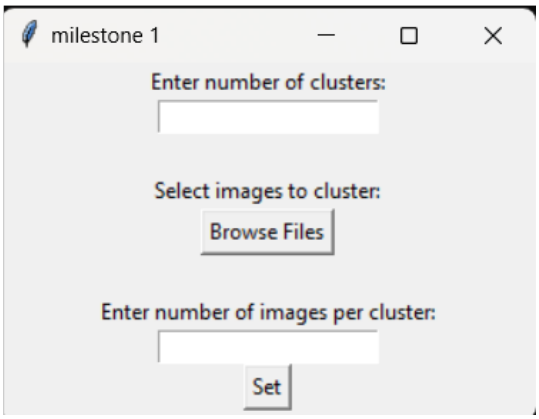



*Figure 1: GUI*

The GUI is very simple and easy to be used. The user chooses the number of clusters and the number of images per cluster that will be displayed I the output window.

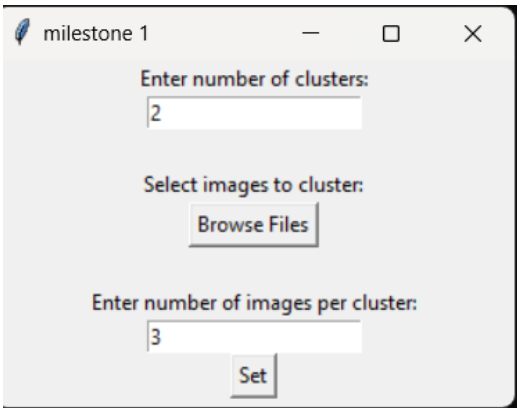



*Figure 3: Example for the GUI use*

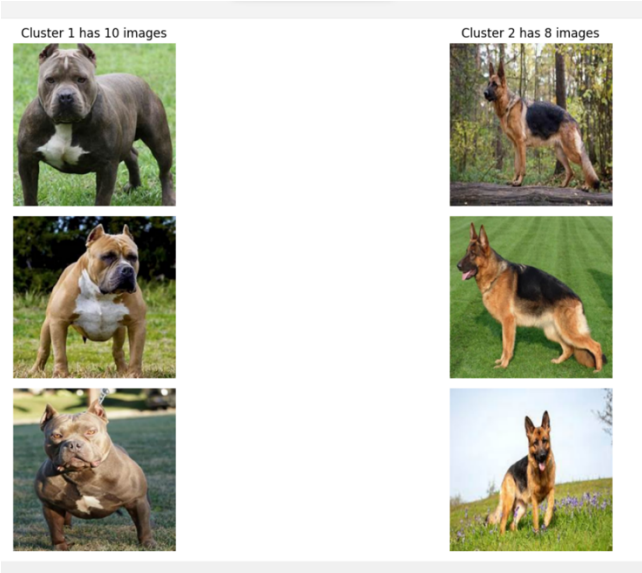



*Figure 2: Output Window*

## 3.2 Testing Section

We started testing from 20 images till 11000 images

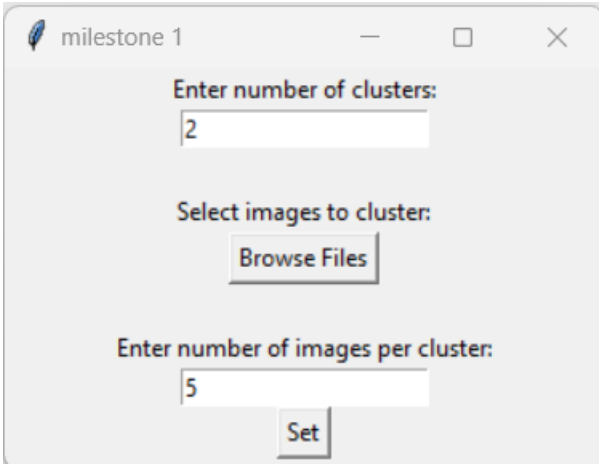



*Figure 5: Testing the 20 images*

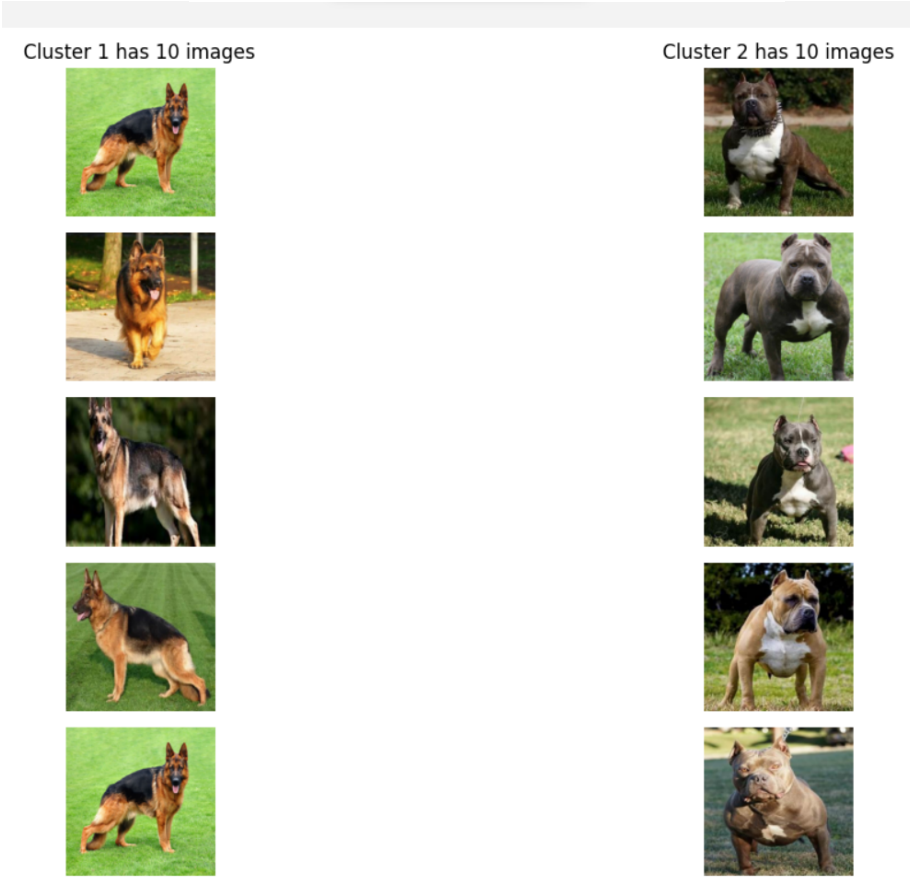



*Figure 4: Output of the 20 images*

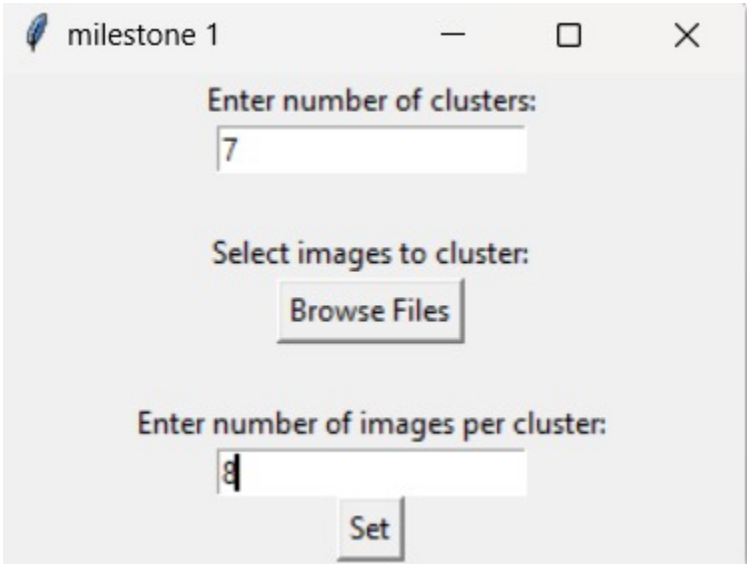



*Figure 6: Testing the 5,000 images*

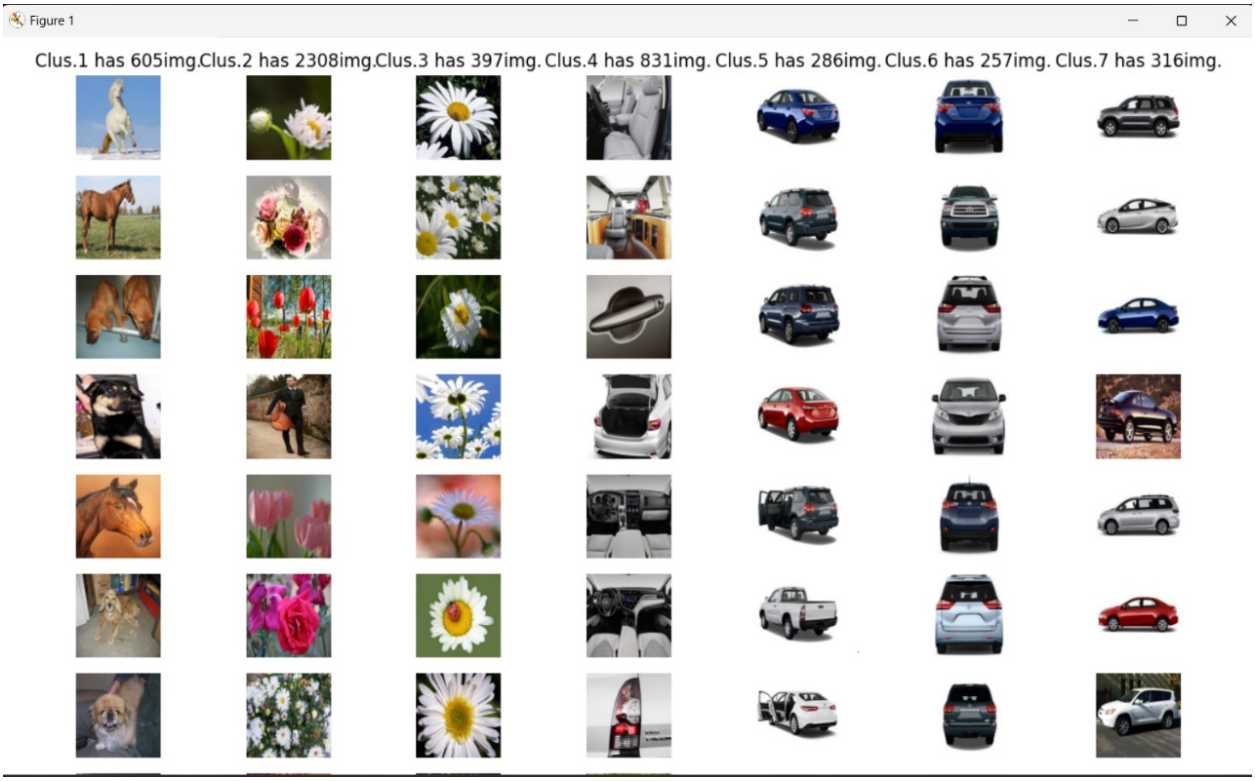



*Figure 7: Output of the 5,000 images*

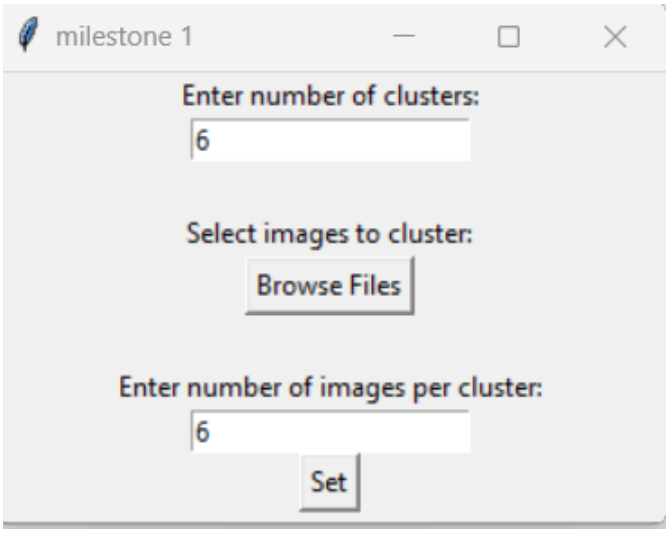



*Figure 9: Testing the 1,000 images*

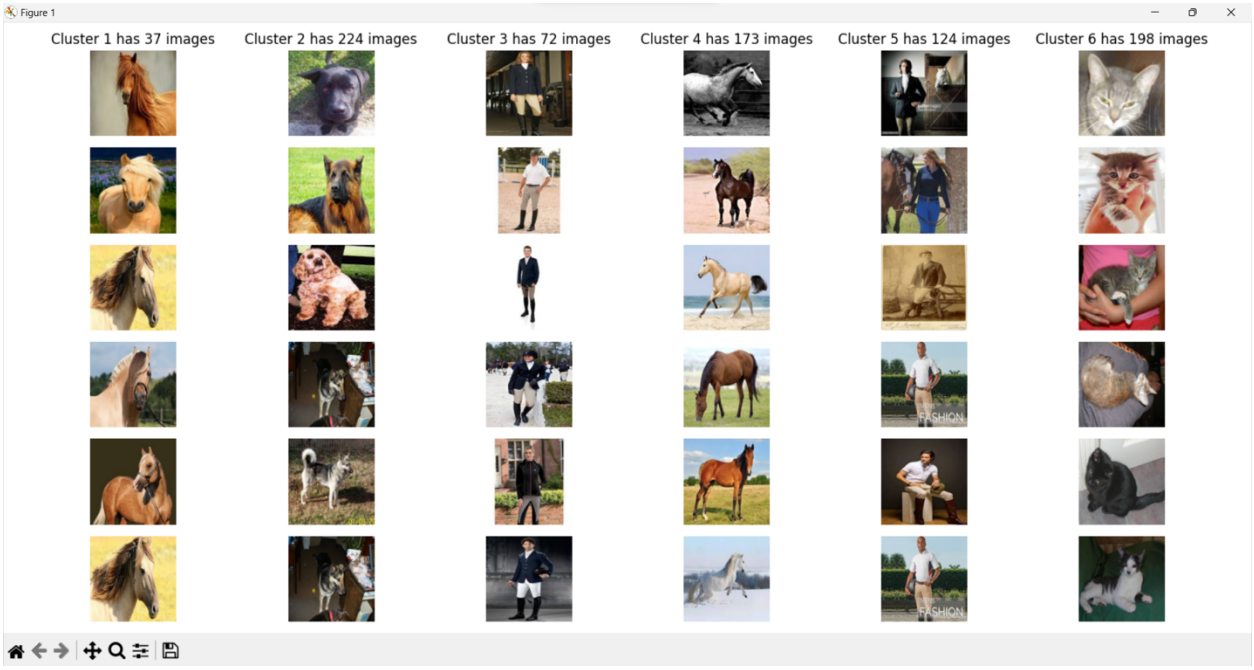



*Figure 8: Output of the 1,000 images*

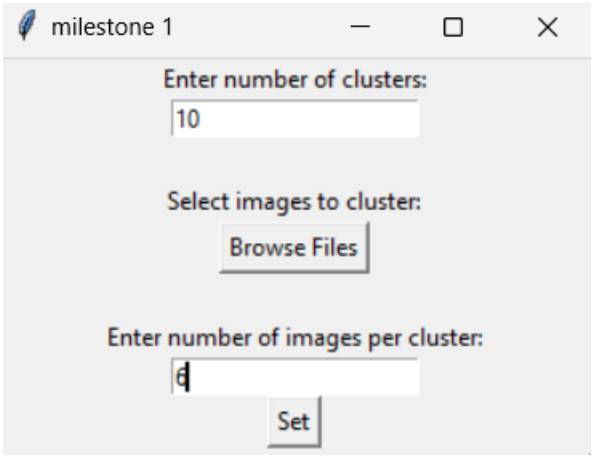



*Figure 10: Testing the 11,000 images*

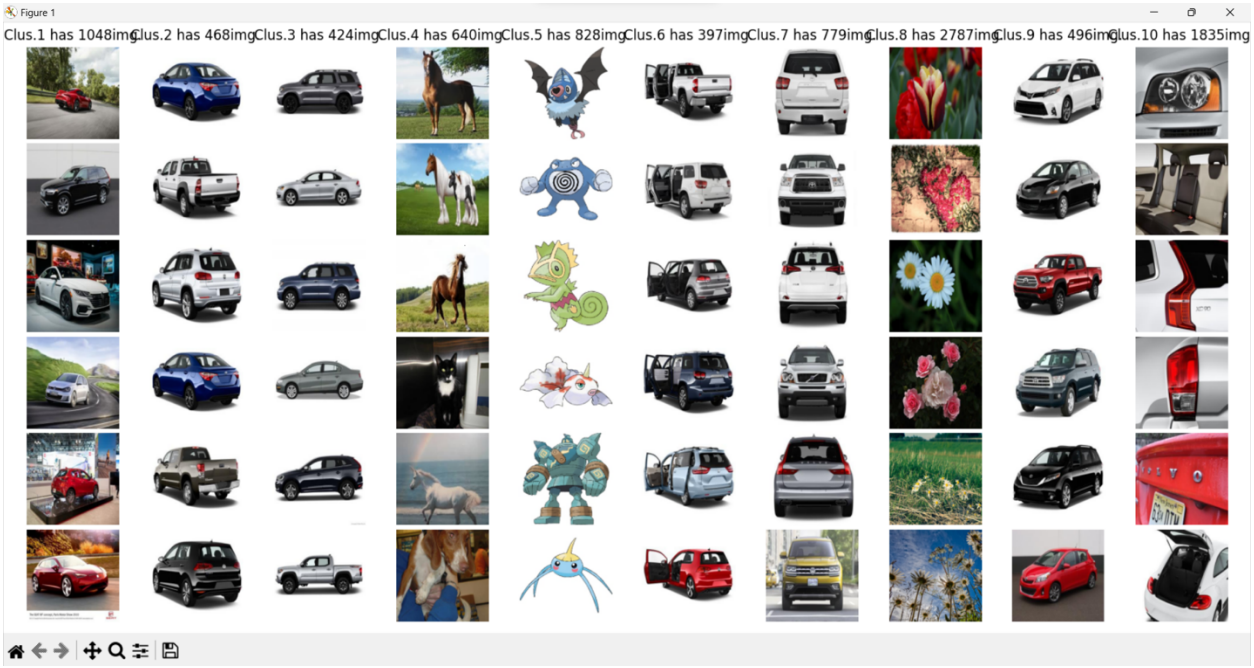



*Figure 11: Output of the 11,000 images*

## 4.0 CONCLUSION

Performing image clustering on a dataset of 1,200,000 pictures can be a resource-demanding computation for several reasons:

1. Memory Constraints: Image datasets are typically represented as matrices of pixel values, which can consume a significant amount of memory. With a large dataset like yours, the memory requirements can quickly exceed the available RAM on a typical laptop, leading to slow performance or crashes due to swapping data between RAM and disk.
2. Computational Power: Image clustering requires repetitive iterations, matrix operations, and distance calculations, which can strain CPUs or GPUs. Laptops might lack the processing power of dedicated workstations or cloud services, affecting performance.
3. Algorithm Complexity: Clustering algorithms like K-means, hierarchical clustering, or DBSCAN have time complexities that depend on the size of the dataset and the number of clusters. Larger datasets can lead to longer processing times, making it impractical to perform clustering on a laptop.
4. Storage and Disk I/O: Loading and saving large image datasets can strain the read/write speed of your laptop's storage device. Frequent disk I/O operations slow down the overall process.
5. Software Dependencies: Efficient image clustering relies on specialized libraries like scikit-learn, TensorFlow, or PyTorch for optimized computations. However, laptops might not fully utilize hardware acceleration when performing these tasks.

To address these limitations, Cloud Computing Cloud platforms like Amazon AWS, Google Cloud, or Microsoft Azure provide scalable computing resources for data-intensive tasks. You can rent powerful virtual machines with ample memory and processing power to perform clustering on larger datasets.

In summary, the limitations of our laptop for clustering a dataset of 1,200,000 pictures are primarily related to memory, computation, and processing power. Exploring alternative options like cloud computing or data preprocessing can help overcome these limitations and enable you to perform image clustering more efficiently.

## 5.0 Appendix

import the necessary libraries and modules

```
# import the necessary libraries and modules
import tkinter as tk                    #Libarary for Gui
from tkinter import filedialog
from PIL import ImageTk, Image          #pillow library for image manipulation
import numpy as np                      #library for operations
import matplotlib.pyplot as plt         #for plotting
from sklearn.cluster import KMeans
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input #models from
Keras for processing functions
from tensorflow.keras.applications.resnet50 import ResNet50
import random   # random module
import warnings #handle warnings
```

Introduce our global variables that store the paths of images, the clustered images, and the number of clusters and images. implement out functions

```
# Global variables
image_paths = []        #to put paths of images
clustered_images = []   #array to store in it the clustered images
num_clusters = 2        #to initialize number of cluster to avoid errors
num_images = 3          #to initialize number of images to show if it isn't chosen
```

1. Process function processes the value entered in entry1 and updates the number of clusters variable.
2. Change_number_of_images: This function processes the value entered in entry2 (a text input field) and updates the number of images variable.
3. Browse_files: This function opens a file dialog box using the file dialog module and allows the user to select multiple image files
4. Cluster image: this function performs the image clustering process.

```
def process_value():
    global num_clusters,entry1      #we put global variables
    num_clusters = int(entry1.get()) # to get the number inserted in entry 1


def change_number_of_images():
    global num_images,entry2
    num_images = int(entry2.get())

def browse_files():
    global image_paths
    image_paths = filedialog.askopenfilenames(filetypes=[("Image files", "*.jpg *.jpeg
*.png")])

def cluster_images():
    global clustered_images, num_clusters

    if len(image_paths) == 0:
        return
```

1. Load Model: It gets a pre-trained model (like VGG16 or ResNet50) that's really good at understanding images.
2. Prepare Images: It prepares the images for the model by resizing them to a standard size and adjusting their values.
3. Get Features: It uses the pre-trained model to understand the important features of each image.
4. Clustering: It uses a clustering method (KMeans) to group similar images together.
5. Organize Images: It sorts the images into different groups based on the clusters they belong to.
6. Visualize: It probably has a part that helps you see the grouped images, but that's not shown here.

```
# Load model for feature extraction (VGG16 or ResNet50)
    model = VGG16(weights='imagenet', include_top=False)
    #model = ResNet50(weights='imagenet', include_top=False)

    images = []
    #for (int i = 0 ; i < len(image_paths);i++)
    #   path = image_paths[i]
    for path in image_paths:
        img = Image.open(path)
        img = img.resize((224, 224))  # Resize image to fit model input size
        img_arr = np.array(img)
        img_arr = preprocess_input(img_arr)
        images.append(img_arr)

    images = np.array(images)
    features = model.predict(images)
    features = features.reshape(features.shape[0], -1)

    kmeans = KMeans(n_clusters=num_clusters, random_state=0)
    kmeans.fit(features)
    labels = kmeans.labels_

    clustered_images = [[] for _ in range(num_clusters)]
    for i, label in enumerate(labels):
        clustered_images[label].append(image_paths[i])

    visualize_clusters()
```

visualize_clusters :This function visualizes the clustered images. If no clustered images are available, the function returns early.

```
def visualize_clusters():
    global clustered_images

    if len(clustered_images) == 0:
        return

    plt.figure(figsize=(12, 8))
    for i, images in enumerate(clustered_images):
        for j in range(1,num_images+1):
            plt.subplot(num_images, len(clustered_images),
i+1+len(clustered_images)*(j-1))
            if j==1:
                plt.title(f"Clus.{i+1} has {len(clustered_images[i])}img.")
            plt.axis('off')
            if len(images) > 0:
                img = Image.open(images[random.randint(0,len(clustered_images[i])-1)])
                img = img.resize((200, 200))
                plt.imshow(img)

    plt.tight_layout()
    plt.show()

def sequence():
    process_value()
    change_number_of_images()
    cluster_images()
```

We created the main window using the tkinter library and we introduced various GUI elements, such as labels, text Entry, and buttons, and pack them into the main window.

```
# Create the main window
root = tk.Tk()
root.title("milestone 1")
root.geometry(f"{300}x{200}")


# Create GUI elements
label = tk.Label(root, text="Enter number of clusters:")
label.pack()

entry1 = tk.Entry(root)
entry1.pack()


label = tk.Label(root, text=" ")
label.pack()

label = tk.Label(root, text="Select images to cluster:")
label.pack()

browse_button = tk.Button(root, text="Browse Files", command=browse_files)
browse_button.pack()

label = tk.Label(root, text=" ")
label.pack()

label = tk.Label(root, text="Enter number of images per cluster:")
label.pack()

entry2 = tk.Entry(root)
entry2.pack()

button = tk.Button(root, text="Set", command=sequence)
button.pack()


# Start the main event loop
```