CSE 480 MACHINE VISION

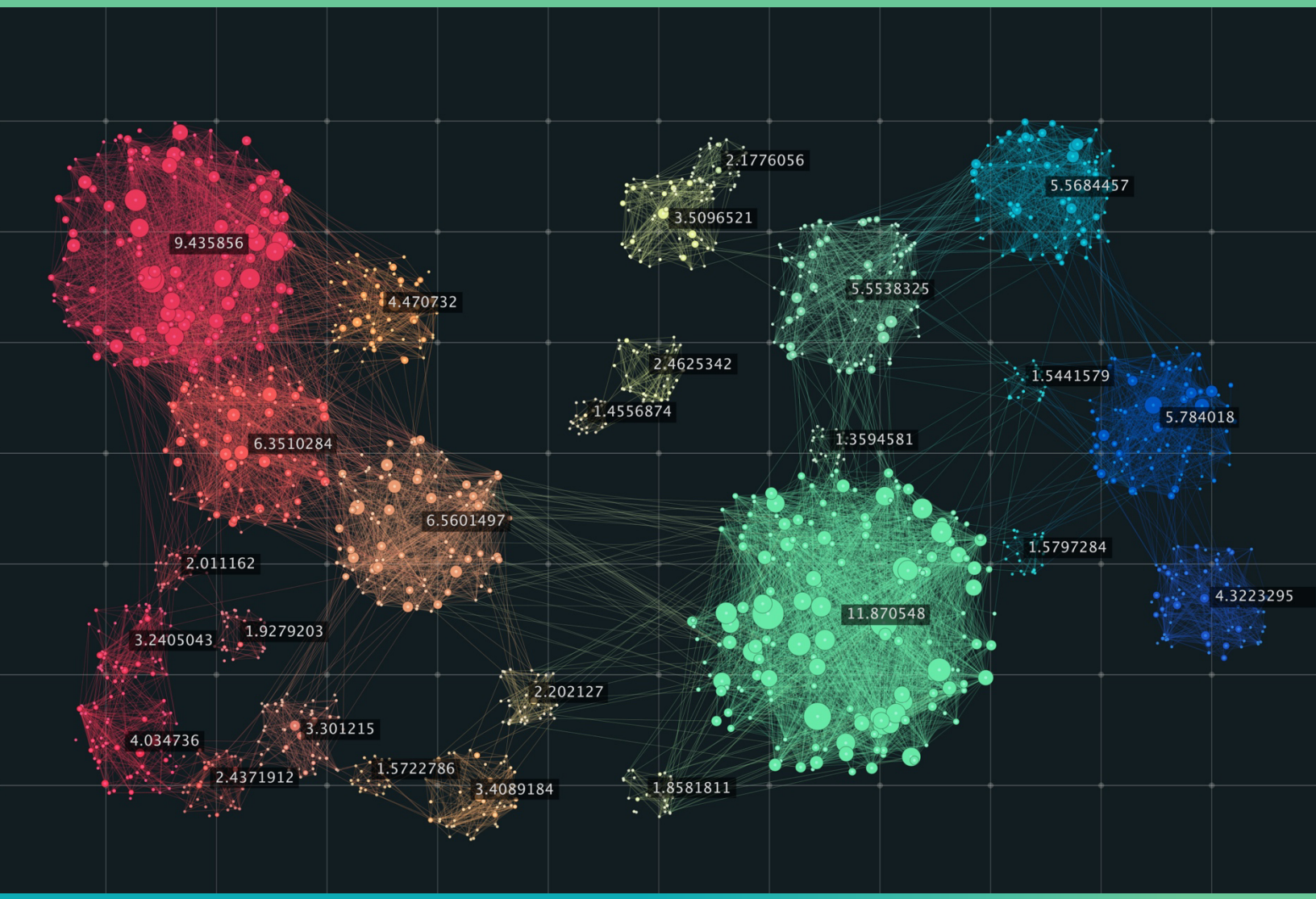

Image Search

Morcous Osama Kamal Faik 19P6167

Amal Amr AbdelHamid Zewita 19P1504

Abdelhamid Salem Abdelhamid 15P8098

Table of Contents

1.0 PROMBLEM DEFINITION AND IMPORTANCE *4*

1.1 Problem Definition 4

1.2 Importance 4

2.0 METHODS AND ALGORITHMS *5*

2.1 Image Index 5

2.2 Similarity Analysis and Retrieval 5

2.3 Query Processing and Ranking 6

2.4 Feature Extraction 7

2.5 Result Presentation 7

2.6 Scalability and Performance Optimization 7

3.0 PROGRAM *8*

3.1 GUI 8

3.2 Testing Section 8

4.0 CONCLUSION *11*

5.0 Appendix *12*

List of Figures

Figure 1: GUI Window 8
Figure 2: Searching for 3 Similar Images 8
Figure 3: Chosen Image 8
Figure 4: Output Window with the Searched Images from the database 9
Figure 5: Searching for 5 Similar Images 9
Figure 6: Chosen Image 9
Figure 7: Output Window with the Searched Images from the database 10
Figure 8: Chosen Images 10
Figure 9: Searching for 7 Similar Images 10
Figure 10: Output Window with the Searched Images from the database 11

1.0 PROMBLEM DEFINITION AND IMPORTANCE

In the digital age, effective techniques for searching and retrieving relevant pictures are needed due to the exponential expansion of visual material. Creating an image search engine addresses this issue.

1.1 Problem Definition

Developing image search engine for users to find photographs based on visual content has issues such as :

1. Image indexing improves search engine retrieval by tagging large photo libraries with metadata.
2. Search engine processes user queries, comparing them to indexed pictures.
3. Image similarity analysis uses algorithms to determine image similarities based on color, shape, texture, and other relevant elements.
4. Search engine's user-friendly results presentation allows browsing and examining photos.

1.2 Importance

The development of an image search engine holds significant importance for various reasons:

1. Efficient image retrieval using image search engines saves time and effort in locating digital images.
2. Image search engines organize and categorize visual content by indexing and assigning metadata, creating a structured database for easy discovery.
3. Image search engine improves user experience with intuitive interfaces and advanced search capabilities, accessible to users of all technical expertise.
4. Image search engines offer valuable inspiration for graphic design, content creation, academic research, and visual storytelling.
5. Image search engines enhance e-commerce by enabling users to search visually, provide personalized experiences, and facilitate product recommendations and cross-selling.
6. Image search engines aid in content filtering and copyright protection, identifying and flagging unauthorized use of copyrighted images.

2.0 METHODS AND ALGORITHMS

Image search engine development involves implementing methods and algorithms for efficient, accurate retrieval, affecting performance and significance.

2.1 Image Index

Image indexing is a fundamental step in building an image search engine. It involves assigning metadata and descriptors to each image in the database, enabling efficient retrieval based on user queries. The following methods and algorithms are commonly used for image indexing:

1. Feature Extraction: Images may be used to extract a variety of visual properties, including color, texture, form, and spatial arrangement. Color histograms, local binary patterns (LBPs), scale-invariant feature transform (SIFT), and convolutional neural networks (CNNs) are examples of feature extraction approaches. These attributes offer descriptions that accurately reflect the distinctive qualities of each image.
2. Vocabulary Construction: Visual words or descriptors are occasionally taken from photographs and utilized to build a visual language. A sample set of visual words is produced by grouping related visual descriptors using methods like k-means clustering or hierarchical clustering.
3. Inverted Indexing: The creation of an inverted index can improve search performance. It associates visual descriptors or visual phrases with the images that contain them, making it possible to retrieve information quickly during a search.

2.2 Similarity Analysis and Retrieval

The core functionality of an image search engine is to retrieve visually similar or related images based on user queries. Various methods and algorithms are employed for image similarity analysis and retrieval:

1. Content-Based Image Retrieval (CBIR): CBIR techniques compare the visual features of the query image with those of the indexed images. Similarity measures, such as Euclidean distance, cosine similarity, or chi-squared distance, are commonly used to calculate the

similarity between feature vectors. CBIR focuses on visual content rather than textual information associated with the images.

2. Deep Learning-Based Approaches: Deep learning models, particularly convolutional neural networks (CNNs), have shown remarkable performance in image analysis and retrieval tasks. Pre-trained CNN models, such as VGG16, ResNet, or InceptionV3, can extract high-level features from images, which can be used to measure image similarity. Techniques like transfer learning and fine-tuning allow leveraging pre-trained models for specific image search tasks.
3. Reverse Image Search: Reverse image search allows users to upload an image as a query and find visually similar images from a database. Hashing algorithms, such as perceptual hashing (e.g., pHash), generate compact representations of images. Similarity measures, such as hamming distance or cosine similarity, are then applied to compare the hash codes and retrieve visually similar images.

2.3 Query Processing and Ranking

Efficient query processing and result ranking are crucial for a smooth user experience. The following methods and algorithms are commonly used:

1. Query Expansion: Query expansion techniques enhance the search results by expanding the user's query with additional relevant terms or concepts. This can be achieved through techniques like synonym mapping, word embeddings, or leveraging external knowledge bases.
2. Relevance Feedback: Relevance feedback allows users to provide feedback on the retrieved images, indicating which images are relevant or irrelevant to their search. This feedback is then utilized to refine the search results and improve the ranking of subsequent queries.
3. Ranking Algorithms: Ranking algorithms determine the order in which the retrieved images are presented to the user. Various algorithms, such as TF-IDF, BM25, or learning-to-rank approaches, can be employed to assign relevance scores to images based on their similarity to the query and other factors.

2.4 Feature Extraction

1. Color Histograms: Histogram-based techniques capture the distribution of color information in images.
2. Local Binary Patterns (LBPs): LBPs encode local texture patterns in images.
3. Scale-Invariant Feature Transform (SIFT): SIFT extracts distinctive features invariant to scale, rotation, and affine transformations.
4. Convolutional Neural Networks (CNNs): Deep learning-based CNN models, such as VGG, ResNet, or Inception, extract high-level features from images.

2.5 Result Presentation

1. User Interface Design: Designing an intuitive and user-friendly interface to present search results, including images, metadata, and additional information.
2. Visualization Techniques: Displaying visually appealing representations of search results, such as grids, thumbnails, or interactive galleries.

2.6 Scalability and Performance Optimization

1. Indexing and Search Optimization: Implementing efficient data structures and algorithms to handle large image databases and perform real-time indexing and retrieval operations.
2. Distributed Computing: Utilizing distributed systems and parallel processing techniques to improve performance and scalability.

3.0 PROGRAM

3.1 GUI

Our GUI geometry is 400×400 It serves as an interface for performing image search engine. It starts by asking about the number of images you want in the output window. Then by clicking on the select button to choose image to search for similar images as your chosen image. By clicking on the browse button, you choose the testing collection of images. Finally you selected how many images will be displayed on the output window.

The GUI is very simple and easy to be used. The user chooses the images per search that will be displayed in the output window.

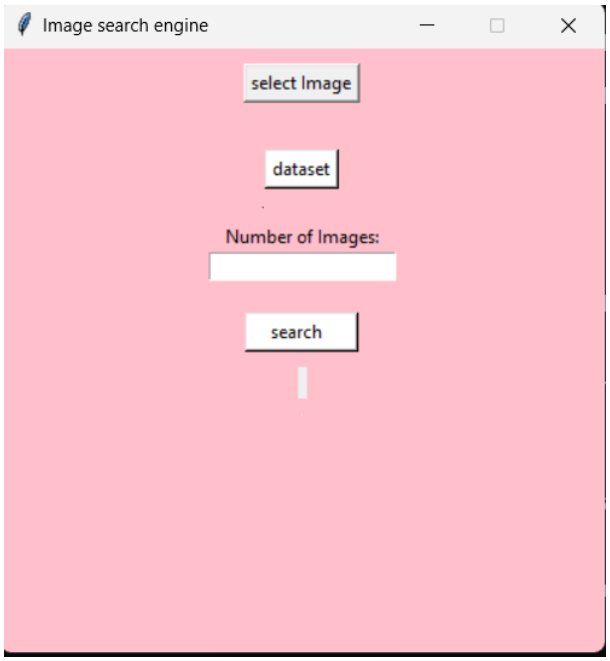

Figure 1: GUI Window

3.2 Testing Section

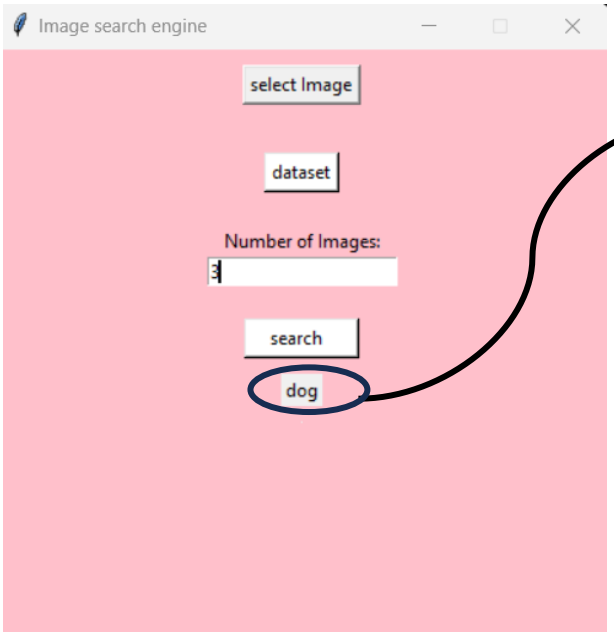

Figure 2: Searching for 3 Similar Images

This label that was used in the search

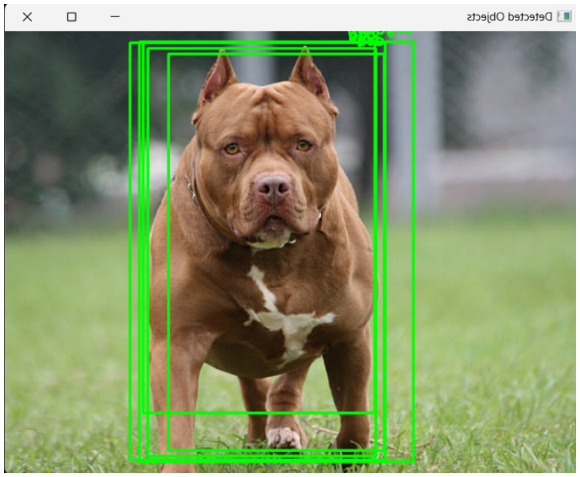

Figure 3: Chosen Image

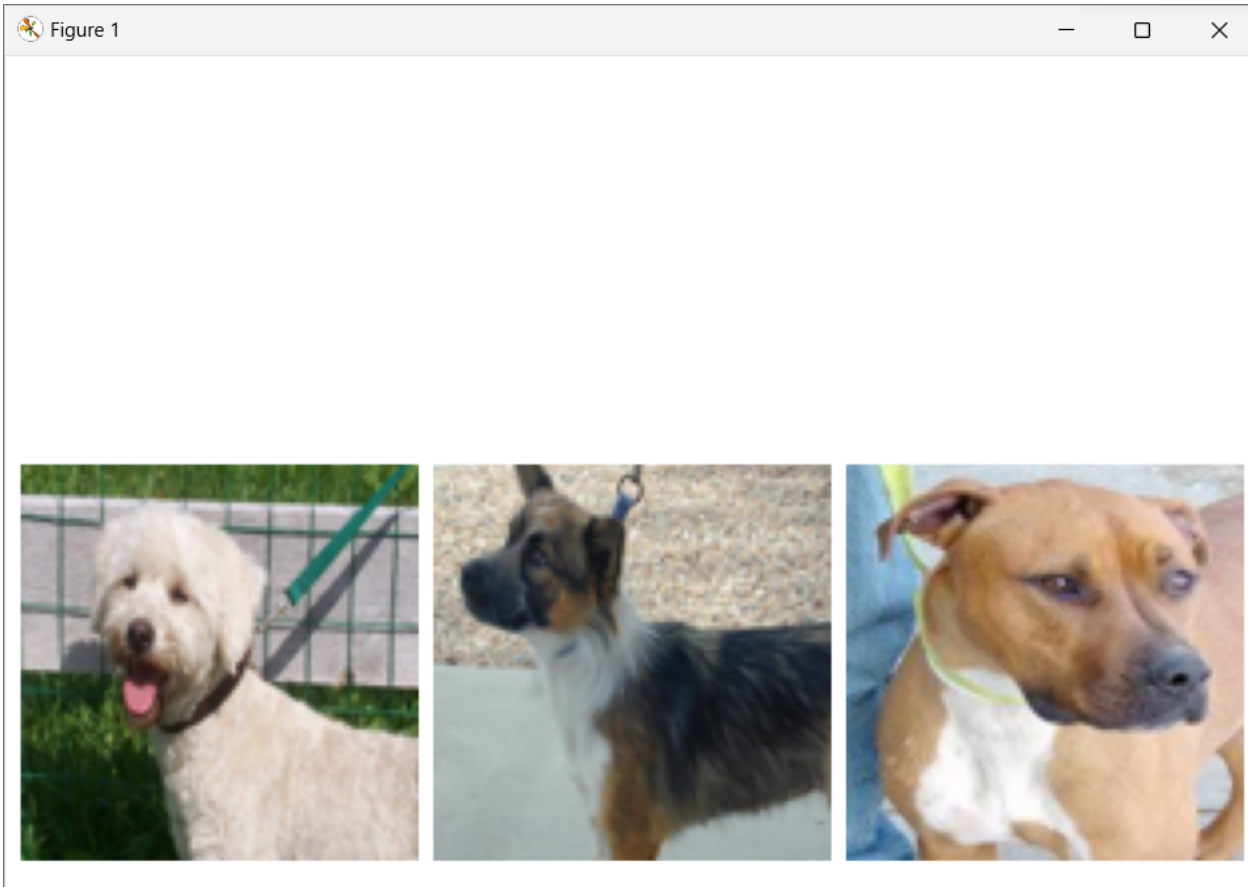

Figure 4: Output Window with the Searched Images from the database

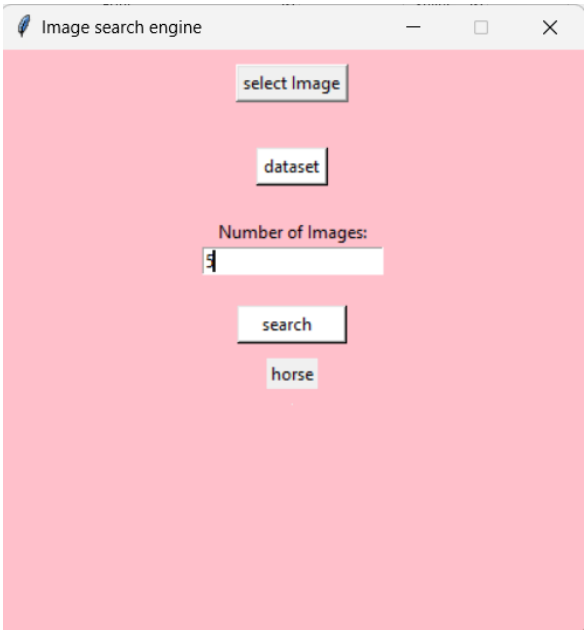

Figure 5: Searching for 5 Similar Images

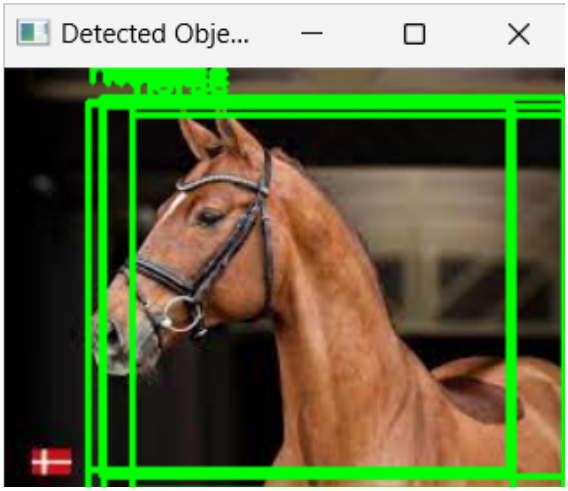

Figure 6: Chosen Image

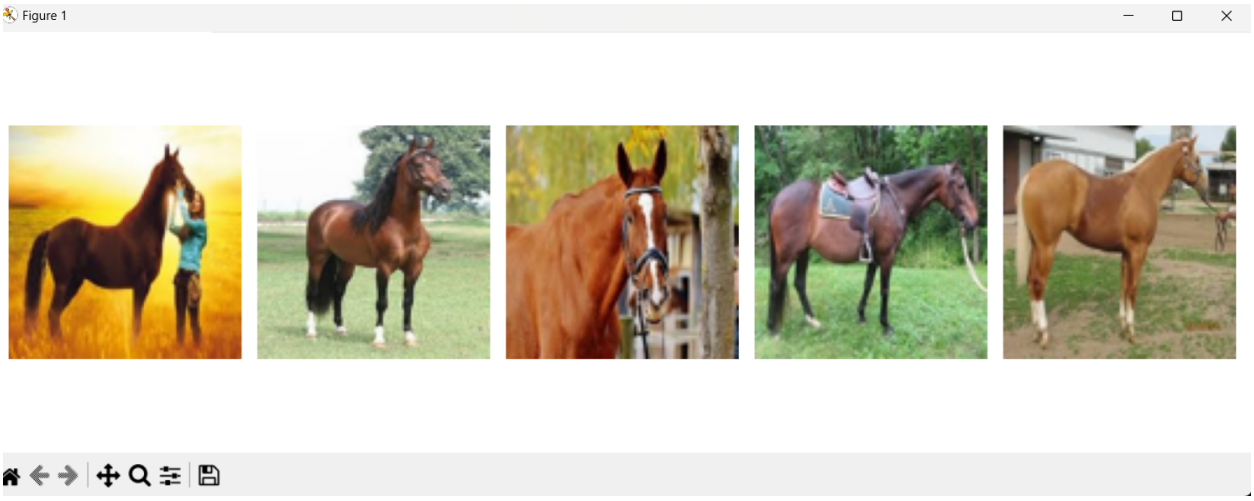

Figure 7: Output Window with the Searched Images from the database

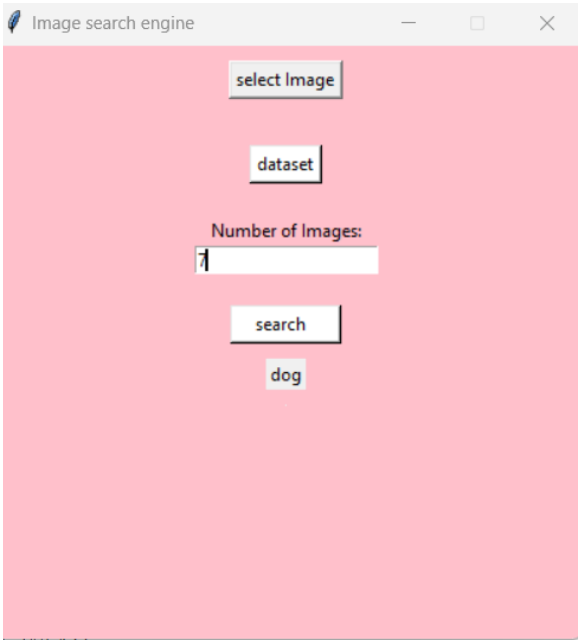

Figure 9: Searching for 7 Similar Images

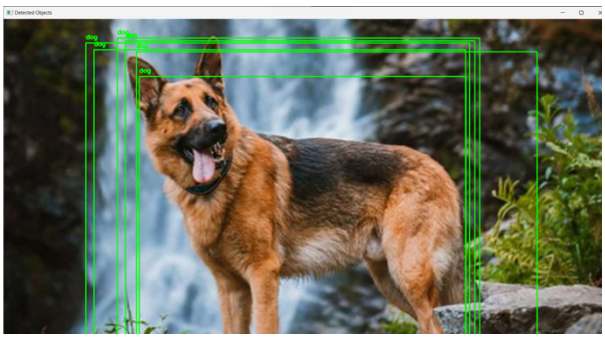

Figure 8: Chosen Images

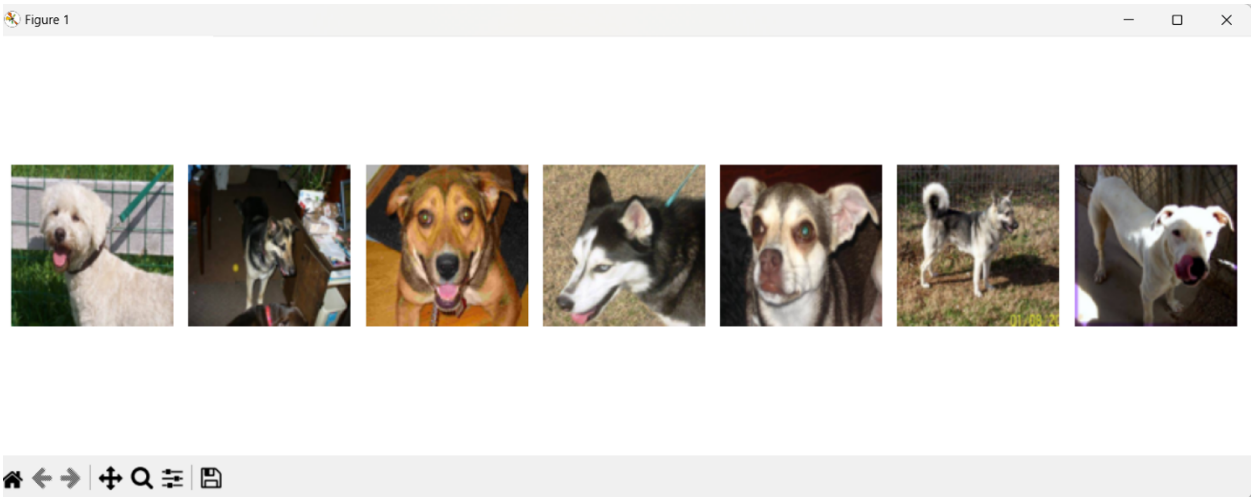

Figure 10: Output Window with the Searched Images from the database

4.0 CONCLUSION

In conclusion, the code presented provides a powerful tool for conducting image-based searches within a specified database. By leveraging a pre-trained deep learning model (InceptionV3) and a graphical user interface (GUI) built using tkinter, the code enables users to select an image and retrieve similar images from the database.

The code's functionality revolves around several key components. First, the GUI offers an intuitive interface for users to interact with the image search engine. It allows users to select a single image for analysis or browse and select multiple images as a query. The GUI provides visual feedback in the form of predicted labels for the selected image, aiding users in understanding the content of the image and refining their search.

Under the hood, the code employs the InceptionV3 model to analyze images. The selected image(s) are preprocessed, resized to match the model's input size, and passed through the model for predictions. The top predicted labels are extracted and displayed, allowing users to gain insights into the content of the images and refine their search parameters.

To facilitate the search process, the code stores the paths of images in the database that share the same predicted label as the initially selected image. This allows users to find visually similar images within the database. The number of images displayed can be controlled using a slider in the GUI, providing flexibility in the search results.

The code's integration of matplotlib for image visualization further enhances the user experience. It presents a subset of randomly chosen images from the database, allowing users to visually explore and assess the similarity of the retrieved images to the query image(s).

Overall, the code provides a comprehensive and user-friendly solution for conducting image-based searches within a specified database. By leveraging deep learning techniques, a well-designed GUI, and efficient image processing, it empowers users to search and retrieve visually similar images, opening up possibilities for various applications such as content-based image retrieval, image recommendation systems, and more.

5.0 Appendix

The provided code implements a GUI for an image search engine using the tkinter library. Let's go through the code and provide a detailed explanation:

1. Importing Libraries: The necessary libraries are imported, including tkinter for the GUI, filedialog for file selection, cv2 for image processing, numpy for numerical operations, os for file and directory operations, PIL for image manipulation, and matplotlib.pyplot for plotting images.
2. Loading YOLO Model: The YOLO object detection model is loaded using cv2.dnn.readNet() function. The weights and configuration files are provided as 'yolov3.weights' and 'yolov3.cfg', respectively. The class labels are read from 'coco.names' file.
3. GUI Functions:
    - take_image(): This function opens a file dialog for the user to select an image file and assigns the selected image path to the global variable image_path.

- take_dataset(): This function opens a file dialog for the user to select a folder containing a dataset of images and assigns the selected folder path to the global variable data_path.

4. Object Detection Functions:
    - detect_objects(image): This function takes an image as input, performs object detection using the YOLO model, and returns the bounding boxes and class IDs of the detected objects.
    - draw_boxes(image, boxes, class_ids): This function takes an image, bounding boxes, and class IDs as input, and draws rectangles and labels around the detected objects on the image.
5. Image Display Function:
    - show_images(unique_labels, dataset_folder, image_display_frame, n_lim): This function displays a subset of images from the dataset folder based on the unique labels of the detected objects. It uses matplotlib.pyplot to create subplots and show the images.
6. Search Function:
    - Search(): This function is triggered when the "search" button is clicked. It reads the image from the selected image path, performs object detection, draws bounding boxes on the image, and displays the image with boxes using cv2.imshow(). It then extracts unique class IDs and labels from the detected objects and updates the result label in the GUI. Finally, it calls the show_images() function to display a subset of images from the dataset based on the unique labels.
7. GUI Setup:
    - The main GUI window is created using tk.Tk().
    - GUI components such as buttons, labels, and an entry field are created and packed into the window using pack().
    - The Search() function is connected to the "search" button through the command parameter.
    - A frame is created to display the images.

8. GUI Event Loop:

- The GUI event loop is started using root.mainloop(), which handles user interactions and keeps the GUI window active.

Overall, the code provides a user-friendly interface for image selection, object detection, and visualization. Users can select an image, perform object detection, and view similar images from a dataset based on the detected objects.

```
import tkinter as tk  # GUI library
from tkinter import filedialog  # File dialog module
import cv2
import numpy as np
import os
from PIL import Image, ImageTk
import random
import matplotlib.pyplot as plt  # Plotting library

# Load the YOLO object detection model
net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')  # Load YOLO weights and
configuration
classes = open('coco.names').read().strip().split('\n')  # Read class names from file

def take_image():
    global image_path
    image_path = filedialog.askopenfilename()  # Open a file dialog to select an image

def take_dataset():
    global data_path
    data_path = filedialog.askdirectory()  # Open a file dialog to select a dataset
folder

def detect_objects(image):
    blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
# Preprocess image for YOLO
    net.setInput(blob)
    layer_names = net.getUnconnectedOutLayersNames()  # Get names of output layers
    outs = net.forward(layer_names)  # Perform forward pass through network

    boxes = []  # Bounding boxes of detected objects
    class_ids = []  # Class IDs of detected objects

    for out in outs:
        for detection in out:
            scores = detection[5:]  # Confidence scores for different classes
```

```
            class_id = np.argmax(scores)  # Get class ID with highest score
            confidence = scores[class_id]  # Confidence score for the detected class
            if confidence > 0.5:  # If confidence score is above threshold
                center_x = int(detection[0] * image.shape[1])  # Calculate center x-
coordinate of bounding box
                center_y = int(detection[1] * image.shape[0])  # Calculate center y-
coordinate of bounding box
                width = int(detection[2] * image.shape[1])  # Calculate width of
bounding box
                height = int(detection[3] * image.shape[0])  # Calculate height of
bounding box
                x = int(center_x - width / 2)  # Calculate top-left x-coordinate of
bounding box
                y = int(center_y - height / 2)  # Calculate top-left y-coordinate of
bounding box
                boxes.append([x, y, width, height])  # Add bounding box coordinates to
list
                class_ids.append(class_id)  # Add class ID to list
    return boxes, class_ids

def draw_boxes(image, boxes, class_ids):
    for i, box in enumerate(boxes):
        x, y, w, h = box
        class_id = class_ids[i]
        color = (0, 255, 0)  # Green color for the rectangle
        class_name = classes[class_id]  # Get class name from class ID

        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)  # Draw bounding box
rectangle on image
        cv2.putText(image, class_name, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)  # Write class name on image

    return image

def show_images(unique_labels, dataset_folder, image_display_frame, n_lim=1):
    for widget in image_display_frame.winfo_children():
        widget.destroy()  # Clear previously displayed images

    for label in unique_labels:
        label_folder = os.path.join(dataset_folder, label)  # Path to folder
containing images of current label
        image_files = [f for f in os.listdir(label_folder) if f.endswith(".jpg")]  #
List of image files in folder

        # Randomly select n_lim images from the list
```

```
        plt_arr = random.sample(image_files, n_lim)

        count = 0
        fig, axes = plt.subplots(1, n_lim, figsize=(12, 4))  # Create subplots for
image display
        for i in plt_arr:
            image_path = os.path.join(label_folder, i)
            img = Image.open(image_path)  # Open image from plt_arr
            img = img.resize((100, 100))  # Resize image for display
            axes[count].imshow(img)  # Display image on current subplot
            axes[count].axis('off')  # Turn off axis
            count = count + 1  # Increment count

        plt.tight_layout()  # Adjust layout
        plt.show()  # Show plot with images

def Search():
    global image_path
    image = cv2.imread(image_path)  # Read selected image

    # Perform object detection on the image
    boxes, class_ids = detect_objects(image)

    # Draw bounding boxes on the image
    image_with_boxes = draw_boxes(image, boxes, class_ids)

    # Display the image with bounding boxes
    cv2.imshow('Image with Boxes', image_with_boxes)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Extract unique class IDs and labels from the detected objects
    unique_class_ids = list(set(class_ids))
    unique_labels = [classes[class_id] for class_id in unique_class_ids]

    # Update the result label in the GUI
    result_label.config(text=", ".join(unique_labels))

    # Display a subset of images from the dataset based on the unique labels
    show_images(unique_labels, data_path, image_display_frame)


# Create the main GUI window
root = tk.Tk()
root.title("Image Search Engine")
```

```
# Create and pack the GUI components
image_button = tk.Button(root, text="Select Image", command=take_image)
image_button.pack()

dataset_button = tk.Button(root, text="Select Dataset", command=take_dataset)
dataset_button.pack()

search_button = tk.Button(root, text="Search", command=Search)
search_button.pack()

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

image_display_frame = tk.Frame(root)
image_display_frame.pack()

# Start the GUI event loop
root.mainloop()
```