Dog Recognition System in Python

The Dog shelter - Fifth leg (penktakoja.lt) is a project that hopes to spread warmth and companionship to homeless dogs and cats. On average, this dog shelter takes in about 800 dogs of various ages and breeds per year. About 600 hundred individuals are donated or adopted by dog lovers. The rest are under the care of shelter workers.

 by **Virgilijus Sakalauskas** and **Dalia Kriksciuniene**

Dog Recognition System: Purpose

The purpose of this project is to take care of homeless or lost dogs who may have been in the dog shelter "Pekta koja" and are photographed and registered there. The idea of the project is to try to identify a lost dog based on a database of dog photos in “Penkta koja” shelter.

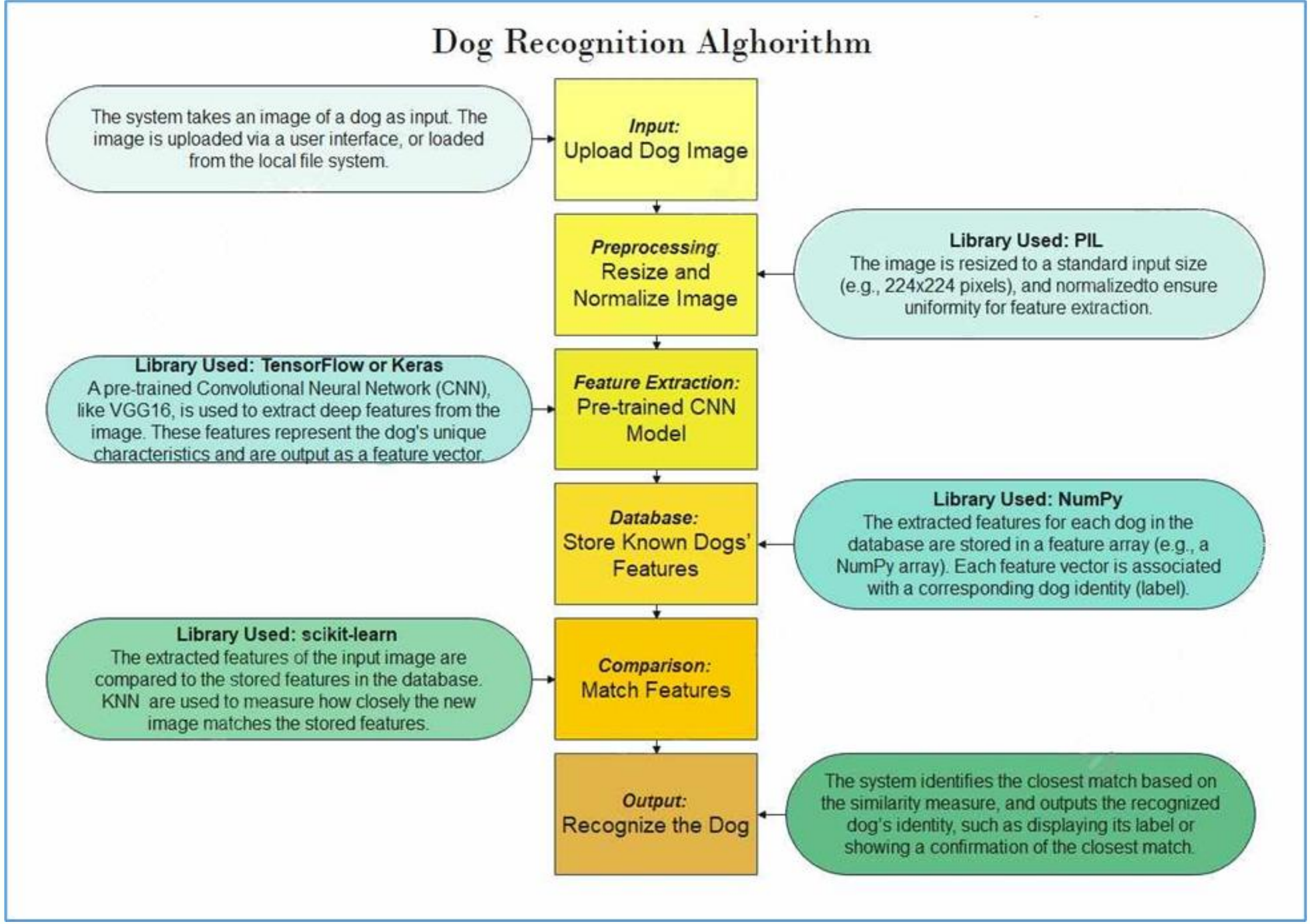
Goal

To identify a lost dog based on a database of dog photos.

Method

Leverage computer vision and machine learning to create a model that can compare a new dog's photo with the photos in the existing database and recognize the dog.

Implementation



Organizing Dog Photos

We propose to set up a folder structure where each dog has its own labeled folder - create a root folder, and inside it, create subfolders for each dog. Each subfolder should contain multiple images of that specific dog.

Root Folder

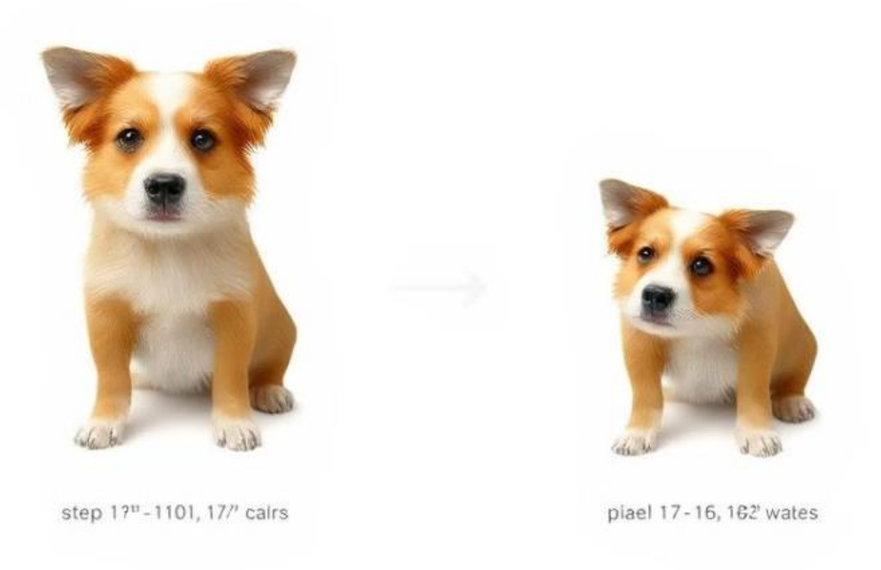
Contains subfolders for each dog.

Subfolders

Each subfolder represents a specific dog and contains multiple images of that dog.

Preprocessing

In the Preprocessing step, resizing and normalizing the image are crucial operations to prepare the image for input into a machine learning model (e.g., a Convolutional Neural Network like VGG16). These steps ensure that all images fed into the model have a consistent format and data range, which improves the model's performance and reliability. The resizing and normalizing the images we perform by Pillow (PIL) library.

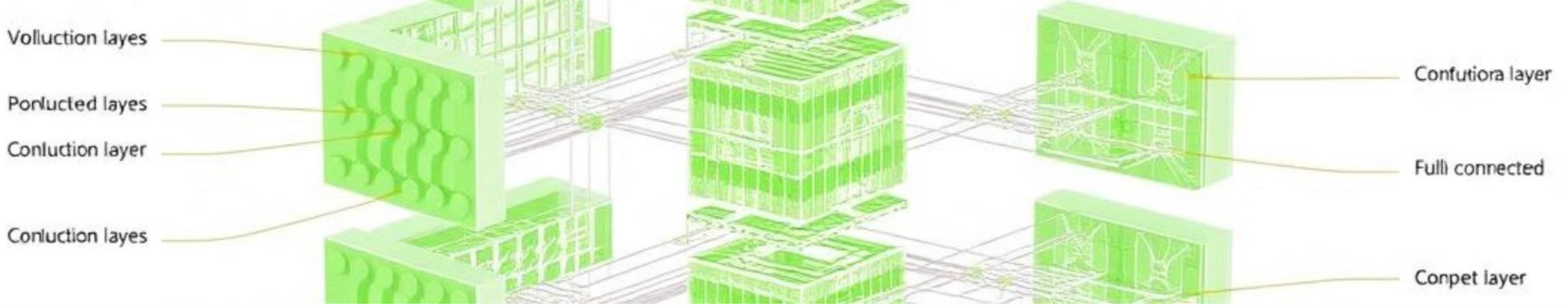


1 Resizing

Dog images usually vary in size. As the learning models like VGG16, ResNet, or MobileNet expect input images to be of a fixed size (e.g., 224x224 pixels for VGG16) we need to resize original images.

2 Normalizing

Pixel values in an image typically range from 0 to 255 (for 8-bit images). Different images can have widely different pixel intensity distributions. Normalization scales these values to a consistent range, often between 0 and 1 or around a zero-centered value, making it easier for the model to interpret the image data.



Feature Extraction

Feature extraction is the process of converting an image into a set of numerical values (a feature vector) that represent the most important information about the image. The neural network focuses on important patterns, like edges, textures, shapes and help identify key characteristics of the dog, such as the shape of its ears, face structure, fur patterns, and more.

We use a pre-trained CNN like VGG16 that has already been trained on a large dataset like ImageNet, which contains millions of images across thousands of categories. These pre-trained models have learned to extract general features from images that can be reused for specific tasks like dog recognition.

A pre-trained CNN consists of multiple layers, each responsible for extracting different levels of features:

1

Lower Layers

Detect simple patterns, like edges, corners, and textures.

2

Middle Layers

Detect more complex patterns, like shapes and parts of objects (e.g., dog ears, eyes).

3

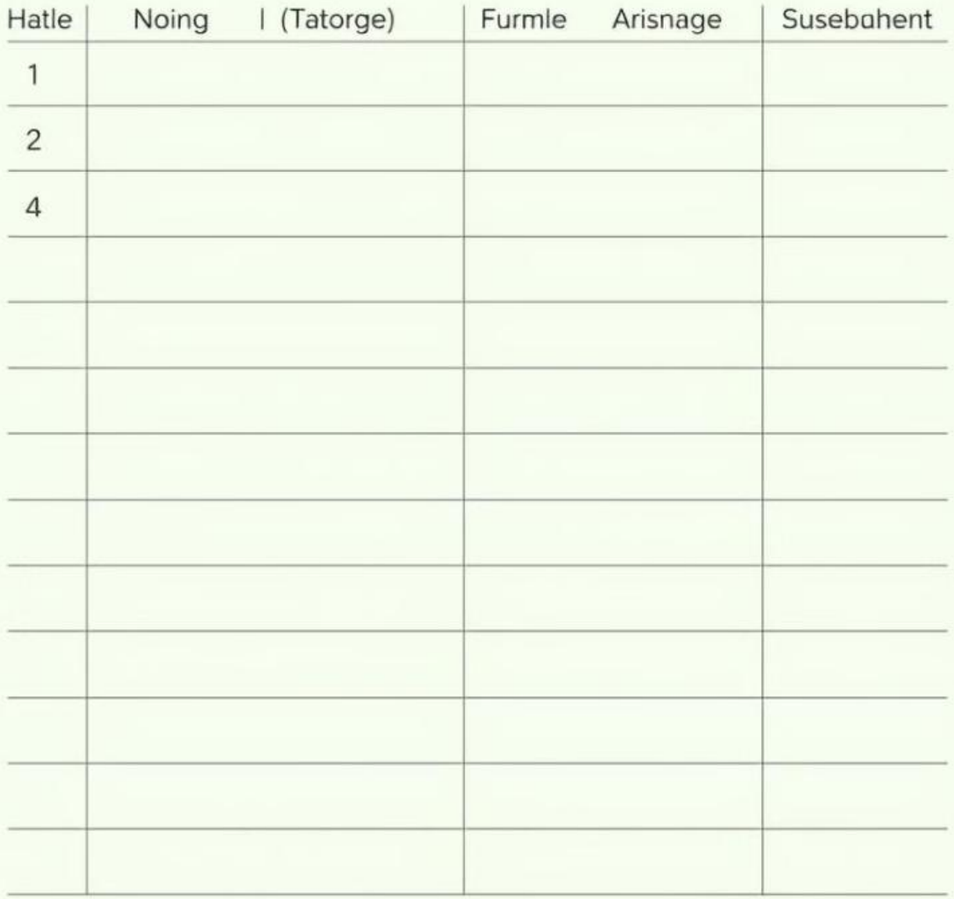
Higher Layers

Detect entire objects or very specific patterns, which are helpful for classification.

Database

A feature vector is a set of numerical values vector that captures the important traits of an image, such as patterns, textures, and shapes. These feature vectors are generated by the CNN model during the feature extraction step and serve as a compressed, abstract representation of the dog.
For example, after extracting features using VGG16, a feature vector for a dog image might look like this:

Dog ID	Feature Vector	Label
Dog1	[0.12, 0.34, 0.56, ..., 0.91]	Labrador Retriever
Dog2	[0.21, 0.45, 0.67, ..., 0.89]	Golden Retriever



We store these feature vectors and labels using NumPy library in a simple format like a NumPy array database where each row corresponds to a different dog. Alongside the features, you also keep a list of dog labels.

Comparison

Once the database is set up, we utilize scikit-learn library for algorithms like K-Nearest Neighbors (KNN) to compare the input dog's feature vector with all the stored feature vectors in the database. The algorithm calculates the similarity between the input vector and each stored vector, and the most similar one is identified as the matching dog.

In the K-Nearest Neighbors (KNN) algorithm, you don't rely on just one closest match. Instead, the algorithm considers the K closest matches to make a decision. Here are the most popular measures:

Euclidean Distance

The straight-line distance between two vectors in the feature space.

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

Manhattan Distance

The sum of the absolute differences between coordinates of the vectors.

$$\text{Manhattan Distance} = \sum_{i=1}^{n}|A_i - B_i|$$

Cosine Similarity

The cosine of the angle between two vectors, focusing on their orientation in the space.

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|}$$

Output

The Recognize the Dog step is the final part of the dog recognition system using the K-Nearest Neighbors (KNN) algorithm. In this step, the system determines which known dog is the best match for the new dog image and provides that as the output.

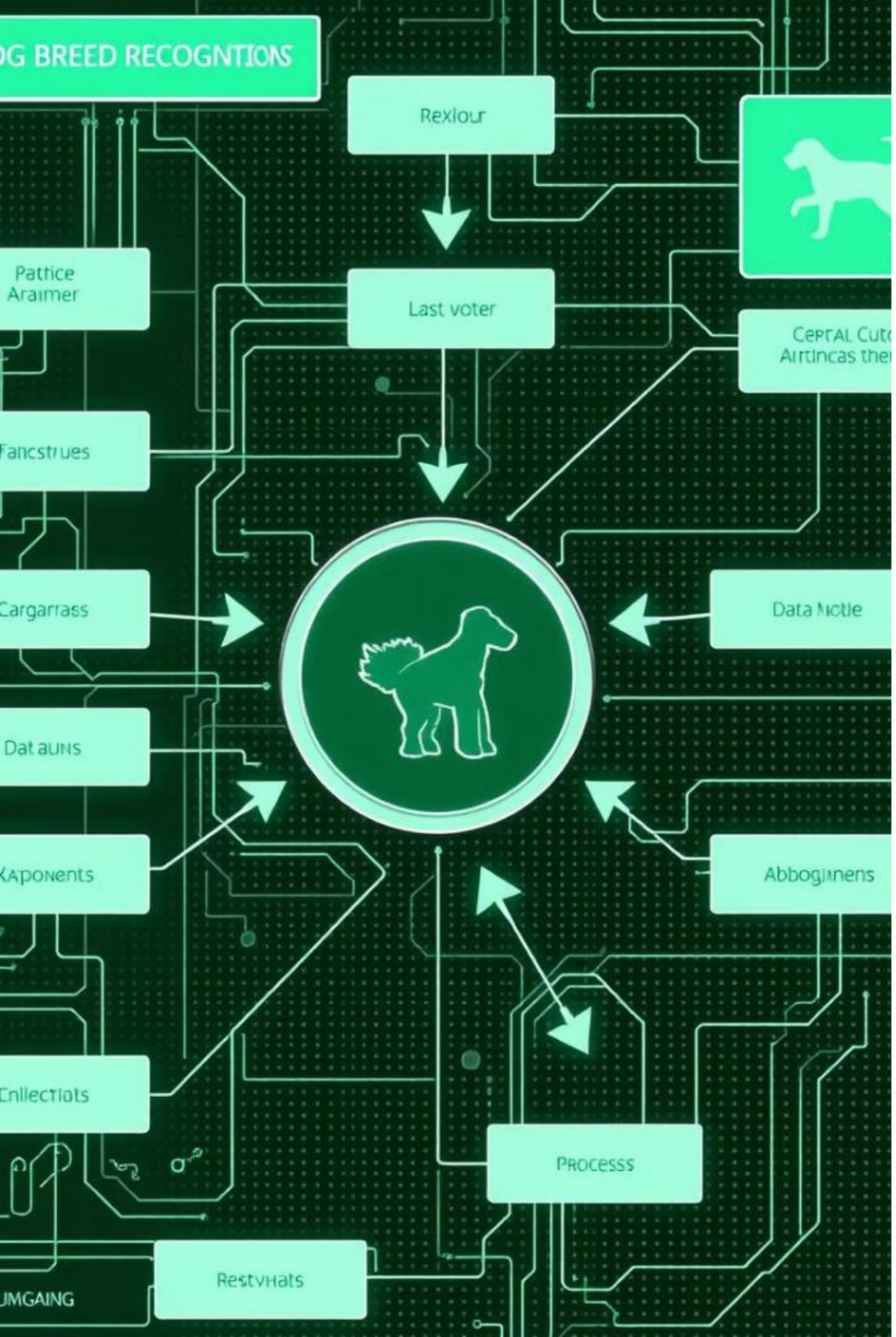
Happiness

1 Match

The system determines which known dog is the best match for the new dog image and provides that as the output.

2 Unknown Dog

If no dog’s distance is below the threshold (in our case 500), the system reports that the dog is unknown.



Strengths and Improvements

The implemented dog recognition system is a strong foundation for recognizing dogs based on image similarity. By combining deep feature extraction with KNN and an intuitive graphical interface, it provides a balance between accuracy and simplicity.

1 Strengths

Accuracy and Robustness, Simplicity with KNN, Extensibility, Unknown Dog Detection.

2 Improvements

Fine-tuning the Threshold, Data Augmentation, Handling Larger Databases, Improving Recognition with Fine-Tuning, Real-Time Recognition, GUI Enhancements.

Python program

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 11:53:02 2024

@author: Virgilijus
"""
import tkinter as tk
from tkinter import filedialog
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from sklearn.neighbors import KNeighborsClassifier
from PIL import Image
import numpy as np
import os
import matplotlib.pyplot as plt

# Load the pre-trained VGG16 model for feature extraction
model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Function to extract features from an image using PIL
def extract_features(image_path):
    img = Image.open(image_path)
    img = img.resize((224, 224))  # Resize image to 224x224 for VGG16
    img_array = np.array(img)  # Convert the image to NumPy array
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = tf.keras.applications.vgg16.preprocess_input(img_array)  # Preprocess for VGG16
    features = model.predict(img_array)
    return features.flatten()

# Load all dog images from the database
def load_database(database_path):
    dog_database = []
    labels = []
    image_paths = []

    for dog_folder in os.listdir(database_path):
        folder_path = os.path.join(database_path, dog_folder)
        if os.path.isdir(folder_path):
            for image_name in os.listdir(folder_path):
                image_path = os.path.join(folder_path, image_name)
                if image_name.endswith(('.jpg', '.jpeg', '.png', '.bmp', '.tiff', '.gif', 'JPG')):
                    features = extract_features(image_path)
                    dog_database.append(features)
                    labels.append(dog_folder)
                    image_paths.append(image_path)

    return dog_database, labels, image_paths

# Example database directory containing subfolders for each dog
#database_path = r"C:\OneDrive_VU\OneDrive - Vilnius University\Failai_baluteje\EPSILON_laboratori
root = tk.Tk()
root.withdraw()  # Hide the root window
print("Select a Dog Folder")
database_path= filedialog.askdirectory()
print("Select a Dog Photo")
file_selected = filedialog.askopenfilename()
```

```python
# Extract features from the database images and get the labels and image paths
database_features, labels, image_paths = load_database(database_path)

# Train a KNN classifier using the extracted features
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(database_features, labels)

distance_threshold = 500
# Function to recognize a new dog from an image
def recognize_dog(new_image_path):
    # Extract features from the new image
    new_features = extract_features(new_image_path)

    # Predict the closest dog from the database
    recognized_dog = knn.predict([new_features])

    # Find the corresponding image path for the recognized dog
  #  index = knn.kneighbors([new_features], n_neighbors=1, return_distance=False)[0][0]
    distances, indices = knn.kneighbors([new_features], n_neighbors=1, return_distance=True)
    nearest_distance = distances[0][0]

    if nearest_distance < distance_threshold:
        # If the nearest distance is below the threshold, recognize the dog
        recognized_dog = knn.predict([new_features])[0]
        recognized_dog_image_path = image_paths[indices[0][0]]
        return recognized_dog, recognized_dog_image_path, nearest_distance
    else:
        # If the nearest distance exceeds the threshold, return "Unknown Dog"
        return "Unknown Dog", None, nearest_distance

# Function to show the recognized dog's image
def show_image(image_path, title):
    if image_path:
        img = Image.open(image_path)
        plt.imshow(img)
        plt.title(title)
        plt.axis('off')  # Hide the axes
        plt.show()
    else:
        print(title)

# Test the recognition system with a new dog image
recognized_dog, recognized_dog_image_path, nearest_distance = recognize_dog(file_selected)

if recognized_dog == "Unknown Dog":
    print(f"Recognized Dog: {recognized_dog} (Distance: {nearest_distance:.4f})")
    show_image(None, "No matching dog found")
else:
    print(f"Recognized Dog: {recognized_dog} (Distance: {nearest_distance:.4f})")
    show_image(recognized_dog_image_path, f"Recognized Dog: {recognized_dog}")
```

Thank you for your attention!