Fuzzy Pattern Recognition System using Image Similarity

Overview

This project aims to estimate the similarity between two grayscale images using a fuzzy logic-based approach. Fuzzy logic allows for the handling of uncertainties and imprecise information, making it suitable for tasks like image comparison where exact matches may not be feasible.

Libraries Used

1. NumPy: For numerical computations and array manipulation.
2. scikit-fuzzy (skfuzzy): For implementing fuzzy logic systems.
3. Pillow (PIL): For image processing tasks such as loading images and applying filters.
4. Matplotlib: For displaying images and visualizing results.

Fuzzy System Design:

1. Inputs:
    a. img_diff: Represents the absolute difference in pixel intensity between the two images.
    b. edge_sim: Indicates the similarity between the edge features of the images.
2. Output:
    a. sim: Represents the overall similarity between the images.
3. Membership Functions:
    a. The input and output variables are fuzzified using three membership functions: low, medium, and high.
4. Rules:
    a. Three fuzzy rules are defined to map combinations of input variables to output similarity levels based on their linguistic labels.

Image Processing:

1. The provided images are loaded using Pillow and converted to grayscale.
2. The absolute pixel intensity difference between the images is computed to quantify their overall dissimilarity.
3. Edge detection using the FIND_EDGES filter is applied to both images to capture their structural features.
4. The similarity between the edge features is computed by comparing the pixel values of the edge-detected images.
5. A minimum threshold is enforced to prevent division by zero errors and ensure a meaningful similarity measure.

Fuzzy Inference:

1. The computed image difference and edge similarity values are fed into the fuzzy control system.
2. The control system applies the defined fuzzy rules to determine the overall similarity between the images.
3. The output similarity value is computed using the Mamdani inference method.

Results Visualization

1. The input images are displayed side by side using Matplotlib.
2. The computed similarity value is printed to the console, representing the degree of similarity between the images.

Python Implementation

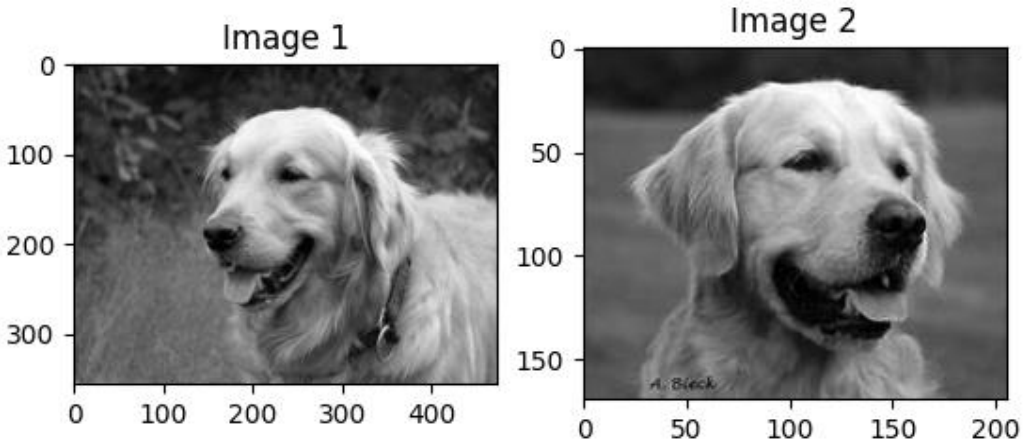
Source Code:

```
import numpy as np
from skfuzzy import control as ctrl
from PIL import Image, ImageFilter
import os
import matplotlib.pyplot as plt

# Create instance of the fuzzy system
img_diff = ctrl.Antecedent(np.arange(0, 256, 1), 'img_diff')
edge_sim = ctrl.Antecedent(np.arange(0, 101, 1), 'edge_sim')
sim = ctrl.Consequent(np.arange(0, 101, 1), 'sim')

# Setup variables
names = ['low', 'medium', 'high']
img_diff.automf(names=names)
edge_sim.automf(names=names)
sim.automf(names=names)

# Setup rules
rules = [
    ctrl.Rule(img_diff['low'] & edge_sim['low'], sim['high']),
    ctrl.Rule(img_diff['medium'] & edge_sim['medium'], sim['medium']),
    ctrl.Rule(img_diff['high'] & edge_sim['high'], sim['low'])
]

# Create control system
control_system = ctrl.ControlSystem(rules)
```

```
# Get input images
img1_path = "dog1.jpeg"
img2_path = "dog2.jpeg"

# Check if files exist
if not (os.path.isfile(img1_path) and os.path.isfile(img2_path)):
    print("One or both of the provided paths are invalid.")
else:
    # Load images
    try:
        img1 = Image.open(img1_path).convert("L")
        img2 = Image.open(img2_path).convert("L")
    except Exception as e:
        print(f"Error loading images: {e}")
    else:
        # Display both images
        plt.subplot(1, 2, 1)
        plt.imshow(img1, cmap='gray')
        plt.title("Image 1")
        plt.subplot(1, 2, 2)
        plt.imshow(img2, cmap='gray')
        plt.title("Image 2")
        plt.show()

        # Check if images are identical
        if img1 == img2:
            print("The provided images are identical.")
        else:
            # Compute features for comparison
            img_diff_input = np.abs(np.mean(img1) - np.mean(img2))
```

```
# Compute edge similarity
edge_img1 = img1.filter(ImageFilter.FIND_EDGES)
edge_img2 = img2.filter(ImageFilter.FIND_EDGES)

# Resize images to have the same dimensions
min_width = min(img1.width, img2.width)
min_height = min(img1.height, img2.height)
edge_img1 = edge_img1.resize((min_width, min_height))
edge_img2 = edge_img2.resize((min_width, min_height))

# Convert images to numpy arrays
edge_array1 = np.array(edge_img1)
edge_array2 = np.array(edge_img2)

# Compute edge similarity
similarity = np.sum(edge_array1 == edge_array2) / (min_width * min_height) * 100
# Ensure a minimum threshold for similarity to avoid total area zero error
min_similarity_threshold = 1.0          # adjustable
edge_sim_input = max (similarity, min_similarity_threshold)

# Compute similarity value
sim_ctrl = control_system
sim_estimator = ctrl.ControlSystemSimulation(sim_ctrl)
sim_estimator.input['img_diff'] = img_diff_input
sim_estimator.input['edge_sim'] = edge_sim_input
sim_estimator.compute()
sim_value = sim_estimator.output['sim']
print("Similarity value:", round(sim_value,2))
```

Output:



Similarity value: 81.14