Enhancing Car Image Comparison with Fuzzy Logic

Name: Raman

Reg. No: 23MDT1048

1. *Application Context:*
    - In the context of comparing car images, the goal might be to assess the similarity between two images of cars. This could be useful in various scenarios such as:
        - Identifying similar models in a car database.
        - Detecting changes or modifications between two versions of the same car model.
        - Verifying the authenticity of car images in online marketplaces or classified ads.
2. *Features of Interest:*
    - When comparing car images, specific features might be of interest, such as:
        - Color: Comparing the color distribution or dominant colors of the cars.
        - Shape: Assessing the overall shape and silhouette of the cars.
        - Details: Analyzing specific details like headlights, grille, wheels, etc.
        - Logo: Identifying and comparing logos or emblems on the cars.
3. *Fuzzy System Design:*
    - The fuzzy system would be designed to take into account these specific features relevant to car image comparison.
    - Input variables could include measures of color similarity, shape similarity, detail similarity, and logo similarity.
    - Membership functions for these input variables would be tailored to capture the fuzzy nature of the similarity between car images.
    - Rules would be defined to determine the overall similarity between car images based on the combination of these features.
4. *Edge Detection vs. Feature Extraction:*
    - While edge similarity might be useful for general image comparison tasks, in the context of car images, other feature extraction methods might be more relevant.

- Instead of edge detection, techniques like color histograms, shape descriptors (e.g., contours), and logo detection algorithms might be employed to extract meaningful features for comparison.
- The choice of feature extraction methods depends on the specific characteristics of the images and the requirements of the comparison task.

5. *Usage Scenarios:*

- The fuzzy logic-based approach to comparing car images could be integrated into various applications:
    - Automotive marketplaces: Providing users with more accurate search results by considering not just textual descriptions but also visual similarities between car images.
    - Car insurance: Automating the process of assessing damages by comparing before-and-after images of vehicles involved in accidents.
    - Car design and manufacturing: Analyzing design changes between different iterations of car models for quality control or design optimization purposes.

CODE:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from PIL import Image, ImageFilter
import os

def compute_intensity_difference(image1, image2):
    # Convert images to numpy arrays
    image1_array = np.array(image1)
    image2_array = np.array(image2)

    # Compute mean intensity values
    mean_intensity1 = np.mean(image1_array)
    mean_intensity2 = np.mean(image2_array)

    # Compute intensity difference
    intensity_diff = np.abs(mean_intensity1 - mean_intensity2)
```

```
    return intensity_diff

class ImageSimilarityFuzzySystem:
    def __init__(self):
        self.intensity_diff = ctrl.Antecedent(np.arange(0, 256, 1), 'Intensity Difference')
        self.edge_similarity = ctrl.Antecedent(np.arange(0, 101, 1), 'Edge Similarity')
        self.similarity = ctrl.Consequent(np.arange(0, 101, 1), 'Similarity')
        self._setup_variables()
        self._setup_rules()

    def _setup_variables(self):
        names = ['low', 'medium', 'high']
        self.intensity_diff.automf(names=names)
        self.edge_similarity.automf(names=names)
        self.similarity.automf(names=names)

    def _setup_rules(self):
        self.rules = [
            ctrl.Rule(self.intensity_diff['low'] & self.edge_similarity['low'], self.similarity['high']),
            ctrl.Rule(self.intensity_diff['medium'] & self.edge_similarity['medium'], self.similarity['medium']),
            ctrl.Rule(self.intensity_diff['high'] & self.edge_similarity['high'], self.similarity['low'])
        ]

    def create_system(self):
        return ctrl.ControlSystem(self.rules)

    def compute_similarity(self, intensity_diff_input, edge_similarity_input):
        similarity_ctrl = self.create_system()
        similarity_estimator = ctrl.ControlSystemSimulation(similarity_ctrl)
        similarity_estimator.input['Intensity Difference'] = intensity_diff_input
        similarity_estimator.input['Edge Similarity'] = edge_similarity_input
        similarity_estimator.compute()
        similarity_value = similarity_estimator.output['Similarity']
        return  similarity_value

def compute_features(image1, image2):
```

```
    # Compute features for comparison (e.g., intensity difference, edge similarity)
    intensity_diff = compute_intensity_difference(image1, image2)

    # Compute edge similarity
    edge_similarity = compute_edge_similarity(image1, image2)

    return intensity_diff, edge_similarity

def compute_edge_similarity(image1, image2):
    if image1 == image2:
        # If both images are identical, return a default similarity value
        return 100.0

    # Apply edge detection filters
    edge_image1 = image1.filter(ImageFilter.FIND_EDGES)
    edge_image2 = image2.filter(ImageFilter.FIND_EDGES)

    # Resize images to have the same dimensions
    min_width = min(image1.width, image2.width)
    min_height = min(image1.height, image2.height)
    edge_image1 = edge_image1.resize((min_width, min_height))
    edge_image2 = edge_image2.resize((min_width, min_height))

    # Convert images to numpy arrays
    edge_array1 = np.array(edge_image1)
    edge_array2= np.array(edge_image2)

    # 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  # You can adjust this threshold as needed
    edge_similarity = max(similarity, min_similarity_threshold)
    return edge_similarity
def main():
    # Create instance of the fuzzy system
    fuzzy_system = ImageSimilarityFuzzySystem()
```

```
    # Get input paths from the user
    image1_path = input("Enter path to first image: ").strip()
    image2_path = input("Enter path to second image: ").strip()

    # Check if files exist
    if not (os.path.isfile(image1_path) and os.path.isfile(image2_path)):
        print("One or both of the provided paths are invalid.")
        return

    # Load images
    try:
        image1 = Image.open(image1_path).convert("L")
        image2 = Image.open(image2_path).convert("L")
    except Exception as e:
        print(f"Error loading images: {e}")
        return

    # Check if images are identical
    if image1 == image2:
        print("The provided images are identical.")
        return

    # Compute features for comparison
    intensity_diff, edge_similarity = compute_features(image1, image2)
    similarity_value = fuzzy_system.compute_similarity(intensity_diff, edge_similarity)
    print("Similarity value:", similarity_value)


if __name__ == "__main__":
    main()
```

Output:

"C:\Users\Satoshi\OneDrive\Desktop\Data\PERSONAL_GROWTH\mini-projects\Images\Leaf Disease Using Fuzzy System\Car_imge_similar.py"

Enter path to first image: C:\Users\Satoshi\Downloads\IMG_20201228_115441.jpg

Enter path to second image: C:\Users\Satoshi\Downloads\IMG_20201205_083344.jpg

Similarity value: 68.61611665792122