***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*