HW5: Image Processing (EE604)

Chirag Garg (210288)

October 15, 2024

Code

```
import numpy as np
from skimage import io, segmentation, color, exposure
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt

def compute_saliency_map(image_path, n_segments):
    # Load the image and segment it into superpixels using SLIC
    image = io.imread(image_path)
    segments = segmentation.slic(image, n_segments=n_segments, compactness=10)

    # Convert the image to LAB color space for better color differentiation
    lab_image = color.rgb2lab(image)

    # Compute the mean color and spatial location for each superpixel
    superpixel_features = []
    for segment_id in np.unique(segments):
        mask = segments == segment_id
        color_mean = np.mean(lab_image[mask], axis=0)  # Average color in LAB space
        spatial_mean = np.mean(np.argwhere(mask), axis=0)  # Center coordinates of the
        ↪  superpixel
        superpixel_features.append(np.concatenate([color_mean, spatial_mean]))

    superpixel_features = np.array(superpixel_features)

    # Calculate distances between superpixels in terms of color and spatial proximity
    color_distances = cdist(superpixel_features[:, :3], superpixel_features[:, :3],
    ↪  metric='euclidean')
    spatial_distances = cdist(superpixel_features[:, 3:], superpixel_features[:, 3:],
    ↪  metric='euclidean')

    # Compute the effective distance and aggregate saliency values for each superpixel
    effective_distances = color_distances * np.exp(-spatial_distances /
    ↪  np.max(spatial_distances))
    saliency_values = np.sum(effective_distances, axis=1)

    # Assign saliency values back to the pixels in the original segments
    saliency_map = np.zeros_like(segments, dtype=float)
    for segment_id, saliency in enumerate(saliency_values):
        saliency_map[segments == segment_id] = saliency

    # Normalize the saliency map to a [0, 1] range for display
    saliency_map = (saliency_map - np.min(saliency_map)) / (np.max(saliency_map) -
    ↪  np.min(saliency_map))

    return saliency_map
```

```
def enhance_saliency_map(saliency_map):
    # Apply contrast stretching based on the 40th and 60th percentile values
    p1, p2 = np.percentile(saliency_map, (40, 60))
    enhanced_map = exposure.rescale_intensity(saliency_map, in_range=(p1, p2))

    # Apply adaptive histogram equalization to enhance local contrast
    enhanced_map = exposure.equalize_adapthist(enhanced_map, clip_limit=0.03)

    # Set values below a certain threshold to maximum brightness (1.0) for enhancement
    threshold = 0.05  # Adjustable threshold value
    enhanced_map[enhanced_map < threshold] = 1.0

    return enhanced_map


image_path = "bird.jpg"  # image path
saliency_map = compute_saliency_map(image_path,10)
enhanced_map = enhance_saliency_map(saliency_map)

# Display the original image, saliency map, and enhanced saliency map
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(io.imread(image_path))
ax1.set_title("Original Image")
ax2.imshow(saliency_map, cmap='gray')
ax2.set_title("Saliency Map")
ax3.imshow(enhanced_map, cmap='gray')
ax3.set_title("Enhanced Saliency Map")
plt.tight_layout()
plt.show()
```

Output

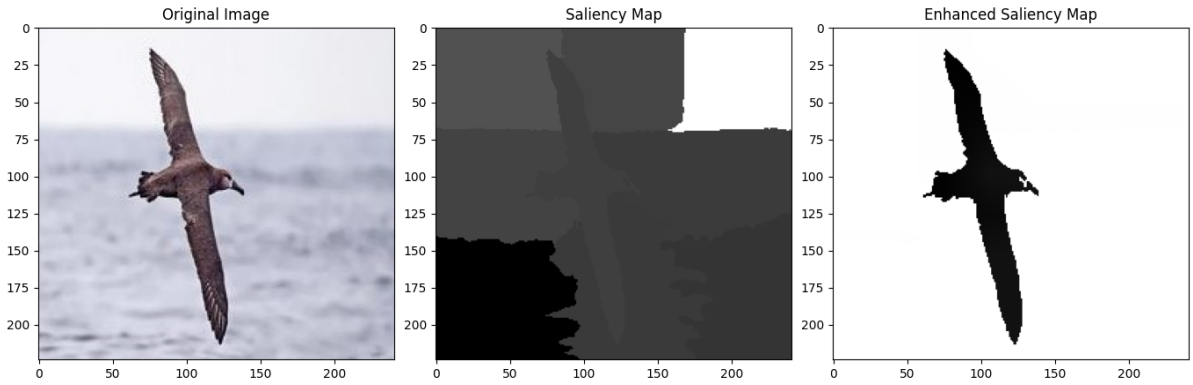

Figure 1: RESULT