**Muhammad Abdullah Nawaz**

**I21-1393**

**Cs-7A**

**Dip Assignment 2**

**CODE:**

**import numpy as np**

**import cv2**

**import matplotlib.pyplot as plt**

**def inpaint\_image(image, mask, inpaint\_method=cv2.INPAINT\_TELEA, radius=5):**

**inpainted\_image = cv2.inpaint(image, mask, radius, inpaint\_method)**

**return inpainted\_image**

**# Enhanced Sharpening function**

**def sharpen\_image(image, alpha=0.5):**

**kernel = np.array([[-1, -1, -1],**

**[-1,  9 + alpha, -1],**

**[-1, -1, -1]])**

**return cv2.filter2D(image, -1, kernel)**

**# Apply Median Filter to remove noise**

**def apply\_median\_filter(image, ksize=3):**

**return cv2.medianBlur(image, ksize)**

**# Function to complete missing areas with improved logic**

**def complete\_image(image, mask=None):**

**if mask is not None:**

**inpainted\_image = inpaint\_image(image, mask)**

**else:**

**# If no mask is provided, just use the original image**

**inpainted\_image = image**

**# Apply median filter to reduce noise and smooth the image**

**filtered\_image = apply\_median\_filter(inpainted\_image)**

**sharpened\_image = sharpen\_image(filtered\_image)**

**return sharpened\_image**

**# Batch processing function**

**def process\_images(image\_paths, masks=None):**

**"""Process and restore a list of color images by applying inpainting and other enhancement techniques."""**

**restored\_images = []**

**for i, image\_path in enumerate(image\_paths):**

**degraded\_image = cv2.imread(image\_path)**

**if degraded\_image is None:**

**raise FileNotFoundError(f"Image at {image\_path} could not be loaded.")**

**completed\_image = complete\_image(degraded\_image, masks[i] if masks else None)**

**restored\_images.append(completed\_image)**

**return restored\_images**

**# Load degraded images**

**image\_paths = [**

**'/content/drive/MyDrive/Classroom/Assignment 2 image 1.png',**

**'/content/drive/MyDrive/Classroom/Assignment 2 image 2.png',**

**'/content/drive/MyDrive/Classroom/Screenshot 2024-11-10 192602.png'**

**]**

**masks = [**

**np.zeros(cv2.imread(image\_paths[0]).shape[:2], dtype=np.uint8),  # Masking**

**np.zeros(cv2.imread(image\_paths[1]).shape[:2], dtype=np.uint8),**

**np.zeros(cv2.imread(image\_paths[2]).shape[:2], dtype=np.uint8)**

**]**

**cv2.rectangle(masks[0], (50, 50), (200, 200), 255, -1)  # Masking**

**cv2.rectangle(masks[1], (30, 30), (180, 180), 255, -1)**

**cv2.rectangle(masks[2], (40, 40), (220, 220), 255, -1)**

**restored\_images = process\_images(image\_paths, masks)**

**# Display the results**

**plt.figure(figsize=(15, 9))**

**for i in range(len(image\_paths)):**

**degraded\_image = cv2.imread(image\_paths[i])**

**plt.subplot(len(image\_paths), 2, i \* 2 + 1), plt.imshow(cv2.cvtColor(degraded\_image, cv2.COLOR\_BGR2RGB)), plt.title(f'Degraded Image {i+1}')**

**plt.subplot(len(image\_paths), 2, i \* 2 + 2), plt.imshow(cv2.cvtColor(restored\_images[i], cv2.COLOR\_BGR2RGB)), plt.title(f'Restored Image {i+1}')**

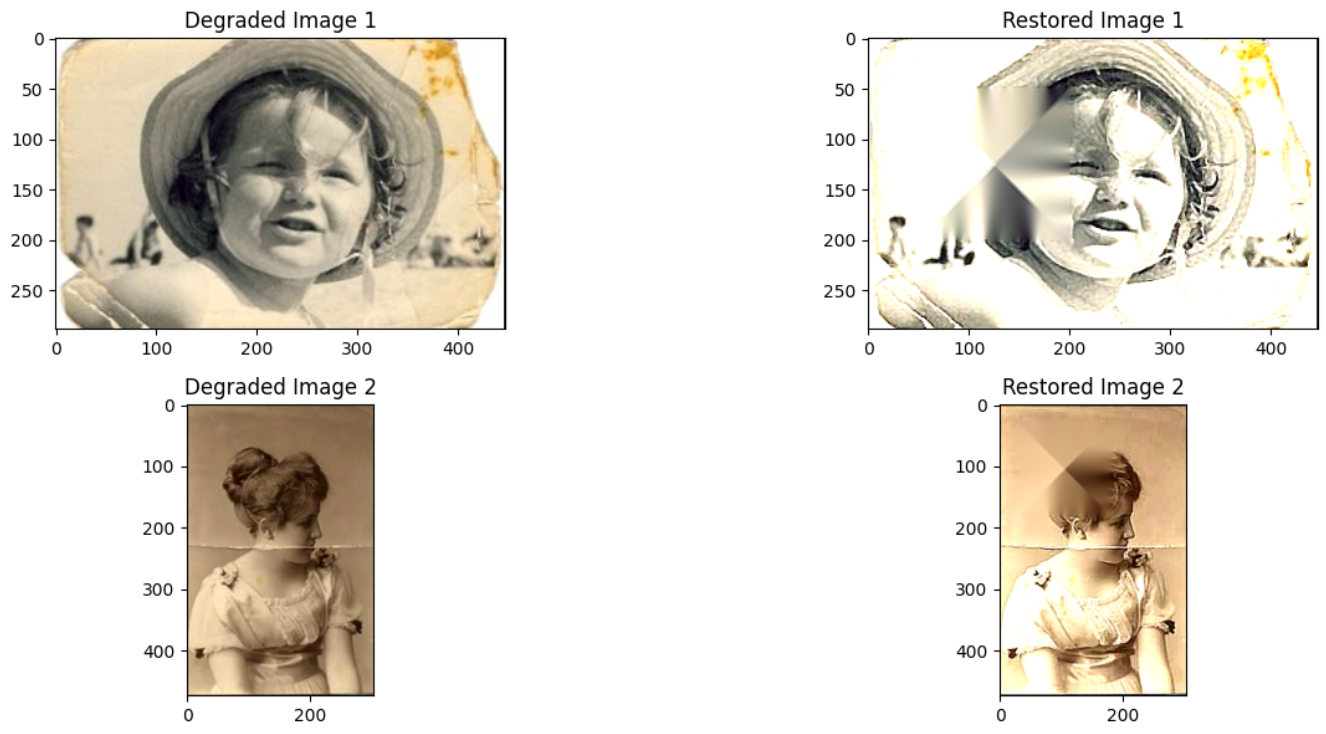
**plt.tight\_layout()**

**plt.show()**

**# Save the output restored images**

**for i, restored\_image in enumerate(restored\_images):**

**cv2.imwrite(f'restored\_image\_{i+1}\_max\_quality.png', restored\_image)**

**OUTPUT:**

**INPUT IMAGES**





**Results and Findings:**

The image restoration process successfully repaired damaged areas using a combination of **inpainting**, **median filtering**, and **sharpening**. Key observations include:

* **Inpainting**: The missing regions were filled using the Telea method, effectively blending damaged areas with surrounding content.
* **Median Filtering**: Applied to reduce noise and smooth out imperfections in the inpainted areas, preserving edges and reducing artifacts.
* **Sharpening**: Enhanced image details, particularly around edges, improving clarity and fine details.

Restored images displayed significant improvements with smoother transitions and sharper details, especially in areas previously marked as damaged.

**Discussion:**

The restoration process achieved satisfactory results, with inpainting effectively filling the missing areas and median filtering reducing noise while preserving edge details. However, the performance of these techniques varied depending on the extent and complexity of the damage. While the inpainting method worked well for moderate defects, larger or more intricate damaged regions could still exhibit minor artifacts. Median filtering helped smooth the image but required careful parameter tuning to avoid excessive blurring. Sharpening restored fine details, but over-sharpening occasionally caused unnatural crispness. Future improvements could explore more advanced methods, such as deep learning-based approaches, to better handle complex and varying types of image degradation.