Lab Report 4 : Image Filtering (Mean Filter, Median Filter and Gaussian Filter), Image Resizing (Replication Method, Linear Interpolation Method and Pixel Mapping Method)

Course Title: Digital Image Processing Lab

Course Code: CSE-406

Date of Performance: September 15, 2024

Date of Submission: September 22, 2024

Submitted to:

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University

Submitted by:

Class Roll	Exam Roll	Name
370	202182	Rubayed All Islam

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Experiment Name

a) Illustration of image filtering using Mean Filter method, Median Filter method, and Gaussian Filter method.

b) Image Resizing using Replication method, Linear Interpolation method, and Pixel Mapping method.

Objectives

a) To apply and compare different image filtering methods (Mean Filter, Median Filter, Gaussian Filter) on a given image and analyze their effects.

b) To explore and implement various image resizing methods (Replication, Linear Interpolation, Pixel Mapping) and evaluate their performance on image scaling.

Experiment 1: Image filtering using Mean Filter, Median Filter, and Gaussian Filter method

MATLAB Code:

Listing 1: Image_Filtering.m

```
% Load the image
image = imread('image.jpg');
image_gray = rgb2gray(image);  % Convert to grayscale if needed

% Define the mean filter (5x5 kernel)
mean_filter = fspecial('average', [5 5]);

% Apply the mean filter
mean_filtered = imfilter(image_gray, mean_filter);

% Apply the median filter (5x5 kernel)
median_filtered = medfilt2(image_gray, [5 5]);

% Define the Gaussian filter (5x5 kernel, sigma=1)
gaussian_filter = fspecial('gaussian', [5 5], 1);

% Apply the Gaussian filter
gaussian_filtered = imfilter(image_gray, gaussian_filter);

% Plot all the images in one frame
figure;

% Original image
subplot(2, 2, 1);
imshow(image_gray);
title('Original Image');
```

```
27
28 % Mean filtered image
29 subplot(2, 2, 2);
30 imshow(mean_filtered);
31 title('Mean Filtered Image');
32
33 % Median filtered image
34 subplot(2, 2, 3);
35 imshow(median_filtered);
36 title('Median Filtered Image');
37
38 % Gaussian filtered image
39 subplot(2, 2, 4);
40 imshow(gaussian_filtered);
41 title('Gaussian Filtered Image');
42
43 % Save the composite figure
44 saveas(gcf, 'filtered_images_composite.jpg');
```

Python Code:

Listing 2: Image_Filtering.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from PIL import Image
4  from scipy.ndimage import convolve, median_filter,
       gaussian_filter
5  import os
6
7  def mean_filter(image_array, kernel_size):
8      if kernel_size % 2 == 0:
9          raise ValueError("Kernel size must be odd.")
10
11     kernel = np.ones((kernel_size, kernel_size),
           dtype=np.float32) / (kernel_size * kernel_size)
12     filtered_image_array = np.zeros_like(image_array)
13
14     for i in range(3):  # Assuming image_array has 3 channels
           (RGB)
15         filtered_image_array[:, :, i] = convolve(image_array[:,
               :, i], kernel, mode='reflect')
16
17     return filtered_image_array
18
19 def apply_median_filter(image_array, kernel_size):
20     if kernel_size % 2 == 0:
21         raise ValueError("Kernel size must be odd.")
22
23     filtered_image_array = np.zeros_like(image_array)
24
```

```
25      for i in range(3):  # Assuming image_array has 3 channels (RGB)
26          filtered_image_array[:, :, i] = median_filter(image_array[:, :, i], size=kernel_size)
27
28      return filtered_image_array
29
30  def apply_gaussian_filter(image_array, sigma):
31      filtered_image_array = np.zeros_like(image_array)
32
33      for i in range(3):  # Assuming image_array has 3 channels (RGB)
34          filtered_image_array[:, :, i] = gaussian_filter(image_array[:, :, i], sigma=sigma)
35
36      return filtered_image_array
37
38  # Load an image using PIL
39  image_path = 'image.jpg'  # Replace with your image path
40  image = Image.open(image_path)
41
42  # Convert the image to a NumPy array
43  image_array = np.array(image)
44
45  # Ensure the image is RGB
46  if image_array.ndim == 3 and image_array.shape[2] == 3:
47      # Apply filters
48      mean_filtered_array = mean_filter(image_array, kernel_size=5)
49      median_filtered_array = apply_median_filter(image_array, kernel_size=5)
50      gaussian_filtered_array = apply_gaussian_filter(image_array, sigma=1.0)
51  else:
52      raise ValueError("The input image is not an RGB image.")
53
54  # Create outputs directory in the same directory as the script
55  output_dir = os.path.join(os.getcwd(), 'outputs')
56  if not os.path.exists(output_dir):
57      os.makedirs(output_dir)
58
59  # Convert results back to images
60  mean_filtered_image = Image.fromarray(np.uint8(mean_filtered_array))
61  median_filtered_image = Image.fromarray(np.uint8(median_filtered_array))
62  gaussian_filtered_image = Image.fromarray(np.uint8(gaussian_filtered_array))
63
64  # Save the results
65  mean_filtered_image.save(os.path.join(output_dir, 'mean_filtered_image.jpg'))
```

```
median_filtered_image.save(os.path.join(output_dir,
    'median_filtered_image.jpg'))
gaussian_filtered_image.save(os.path.join(output_dir,
    'gaussian_filtered_image.jpg'))

# Display the images
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(image_array)
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title('Mean Filtered Image')
plt.imshow(mean_filtered_array)
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title('Median Filtered Image')
plt.imshow(median_filtered_array)
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title('Gaussian Filtered Image')
plt.imshow(gaussian_filtered_array)
plt.axis('off')

plt.show()
```

Output:

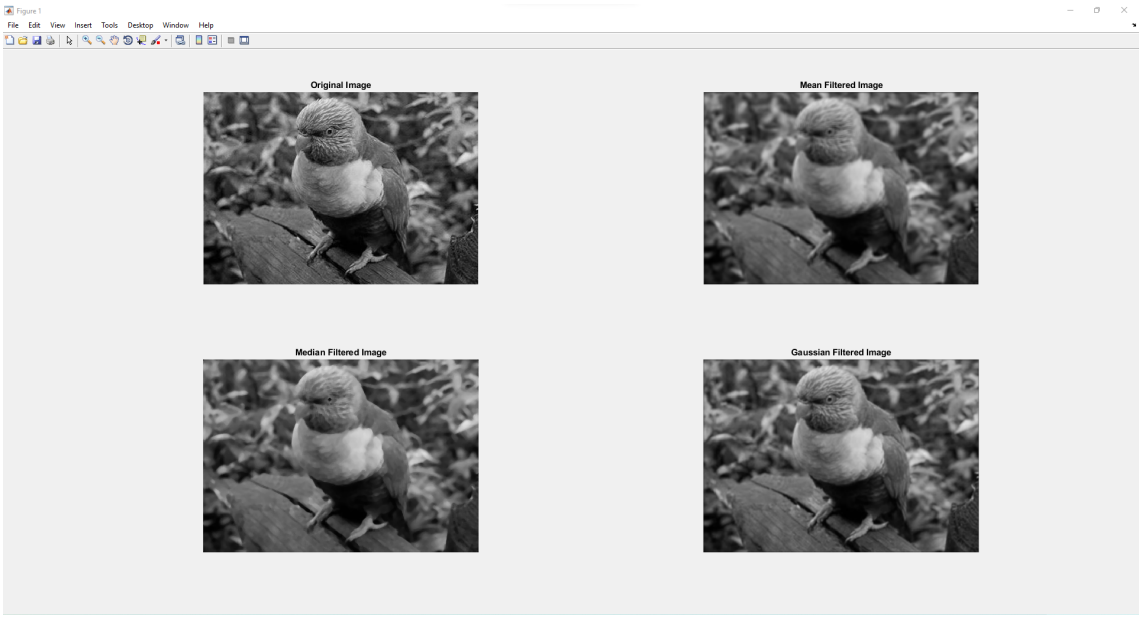

Figure 1: Image Filtering of Gray Scale Image

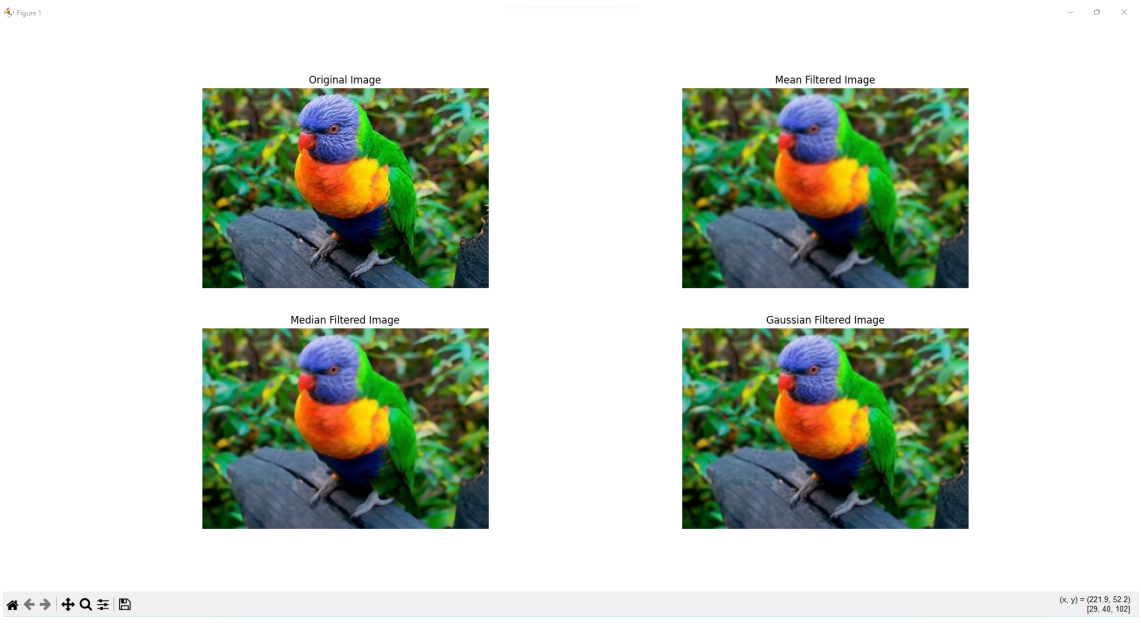

Figure 2: Image Filtering of RGB Image

Experiment 2: Image Resizing using Replication, Linear Interpolation and Pixel Mapping method

MATLAB Code:

Listing 3: Image_Resizing.m

```
% Step 1: Get User Input for Image File and New Size

% Prompt user for the image file name
imageFileName = input('Enter the image file name (e.g., ''peppers.png''): ', 's');

% Read the image
originalImage = imread(imageFileName);

% Prompt user for the new size
newHeight = input('Enter the new height for resizing: ');
newWidth = input('Enter the new width for resizing: ');
newSize = [newHeight, newWidth]; % New size [height, width]

% Convert to double if needed for better interpolation results
if ~isa(originalImage, 'double')
    originalImage = im2double(originalImage);
end

% Step 2: Resize using Replication Method (Nearest-Neighbor Interpolation)
resizedImageReplication = imresize(originalImage, newSize, 'nearest');

% Step 3: Resize using Linear Interpolation Method (Bilinear Interpolation)
resizedImageLinear = imresize(originalImage, newSize, 'bilinear');

% Step 4: Resize using Pixel Skipping Method
% Pixel skipping involves subsampling

% Calculate the skipping ratio
originalSize = size(originalImage);
rowRatio = originalSize(1) / newSize(1);
colRatio = originalSize(2) / newSize(2);

% Use floor to ensure indices are within bounds
rowSkip = floor(rowRatio);
colSkip = floor(colRatio);

% Generate the resized image by skipping pixels
resizedImagePixelSkipping = originalImage(1:rowSkip:end, 1:colSkip:end, :);
```

```
39
40 % Adjust size to match exactly if necessary
41 % Resize pixel skipping result to newSize using imresize if
      necessary
42 resizedImagePixelSkipping = imresize(resizedImagePixelSkipping,
      newSize, 'nearest');
43
44 % Step 5: Display the Results
45 figure;
46
47 % Display original image
48 subplot(2, 2, 1);
49 imshow(originalImage);
50 title('Original Image');
51
52 % Display resized image using replication (nearest neighbor)
53 subplot(2, 2, 2);
54 imshow(resizedImageReplication);
55 title('Resized Image (Replication)');
56
57 % Display resized image using linear interpolation
58 subplot(2, 2, 3);
59 imshow(resizedImageLinear);
60 title('Resized Image (Linear Interpolation)');
61
62 % Display resized image using pixel skipping method
63 subplot(2, 2, 4);
64 imshow(resizedImagePixelSkipping);
65 title('Resized Image (Pixel Skipping)');
```

Python Code:

Listing 4: Image_Resizing.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from PIL import Image
4
5  def linear_interpolation_resize(image_array, new_width,
       new_height):
6      """
7      Resize an image using linear interpolation method.
8      """
9      old_height, old_width, channels = image_array.shape
10     resized_image_array = np.zeros((new_height, new_width,
           channels), dtype=image_array.dtype)
11     row_scale = old_height / new_height
12     col_scale = old_width / new_width
13
14     for y in range(new_height):
15         for x in range(new_width):
```

```
            orig_y = y * row_scale
            orig_x = x * col_scale
            y0 = int(orig_y)
            x0 = int(orig_x)
            y1 = min(y0 + 1, old_height - 1)
            x1 = min(x0 + 1, old_width - 1)
            dy = orig_y - y0
            dx = orig_x - x0
            w00 = (1 - dx) * (1 - dy)
            w01 = dx * (1 - dy)
            w10 = (1 - dx) * dy
            w11 = dx * dy

            for c in range(channels):
                pixel_value = (w00 * image_array[y0, x0, c] +
                               w01 * image_array[y0, x1, c] +
                               w10 * image_array[y1, x0, c] +
                               w11 * image_array[y1, x1, c])
                resized_image_array[y, x, c] = pixel_value
    return resized_image_array

def pixel_skipping_resize(image_array, new_width, new_height):
    """
    Resize an image using the pixel skipping method.
    """
    old_height, old_width, channels = image_array.shape
    resized_image_array = np.zeros((new_height, new_width,
       channels), dtype=image_array.dtype)
    row_scale = old_height / new_height
    col_scale = old_width / new_width

    for y in range(new_height):
        for x in range(new_width):
            src_y = int(y * row_scale)
            src_x = int(x * col_scale)
            resized_image_array[y, x] = image_array[src_y, src_x]
    return resized_image_array

def replicate_resize(image_array, new_width, new_height):
    """
    Resize an image using the replication method.
    """
    old_height, old_width, channels = image_array.shape
    resized_image_array = np.zeros((new_height, new_width,
       channels), dtype=image_array.dtype)
    row_scale = old_height / new_height
    col_scale = old_width / new_width

    for y in range(new_height):
        for x in range(new_width):
            src_y = int(y * row_scale)
```

```
65              src_x = int(x * col_scale)
66              # Use nearest neighbor replication
67              resized_image_array[y, x] = image_array[src_y, src_x]
68      return resized_image_array
69
70  # Load an image using PIL
71  image_path = 'rgb-image.jpg'  # Replace with your image path
72  try:
73      image = Image.open(image_path)
74  except FileNotFoundError:
75      print(f"Error: The file {image_path} was not found.")
76      exit()
77
78  image_array = np.array(image)
79
80  # Get and print the original image dimensions
81  original_height, original_width, _ = image_array.shape
82  print(f"Original image dimensions:
        {original_width}x{original_height}")
83
84  # Input the new dimensions for resizing
85  try:
86      new_width = int(input("Enter the new width: "))
87      new_height = int(input("Enter the new height: "))
88  except ValueError:
89      print("Error: Please enter valid integer values for width
          and height.")
90      exit()
91
92  # Perform the resizing using different methods
93  resized_image_linear = linear_interpolation_resize(image_array,
        new_width, new_height)
94  resized_image_pixel_skip = pixel_skipping_resize(image_array,
        new_width, new_height)
95  resized_image_replicate = replicate_resize(image_array,
        new_width, new_height)
96
97  # Convert back to image for saving purposes
98  Image.fromarray(np.uint8(resized_image_linear)).
        save('outputs/linear_interpolation_resized.jpg')
99  Image.fromarray(np.uint8(resized_image_pixel_skip)).
        save('outputs/pixel_skipping_resized.jpg')
100 Image.fromarray(np.uint8(resized_image_replicate)).
        save('outputs/replication_resized.jpg')
101
102 # Plot and compare results
103 plt.figure(figsize=(15, 10))
104
105 # Original Image
106 plt.subplot(2, 2, 1)
107 plt.title('Original Image')
```

```
plt.imshow(image_array)
plt.axis('off')
plt.text(0.5, -0.1, f"Dimensions:
   {original_width}x{original_height}", ha='center',
   va='center', fontsize=12, transform=plt.gca().transAxes)

# Resized using Linear Interpolation
plt.subplot(2, 2, 2)
plt.title('Resized Image using Linear Interpolation')
plt.imshow(resized_image_linear.astype(np.uint8))
plt.axis('off')
plt.text(0.5, -0.1, f"Dimensions: {new_width}x{new_height}",
   ha='center', va='center', fontsize=12,
   transform=plt.gca().transAxes)

# Resized using Pixel Skipping
plt.subplot(2, 2, 3)
plt.title('Resized Image using Pixel Skipping')
plt.imshow(resized_image_pixel_skip.astype(np.uint8))
plt.axis('off')
plt.text(0.5, -0.1, f"Dimensions: {new_width}x{new_height}",
   ha='center', va='center', fontsize=12,
   transform=plt.gca().transAxes)

# Resized using Replication
plt.subplot(2, 2, 4)
plt.title('Resized Image using Replication')
plt.imshow(resized_image_replicate.astype(np.uint8))
plt.axis('off')
plt.text(0.5, -0.1, f"Dimensions: {new_width}x{new_height}",
   ha='center', va='center', fontsize=12,
   transform=plt.gca().transAxes)

plt.tight_layout()
plt.show()
```

Output:

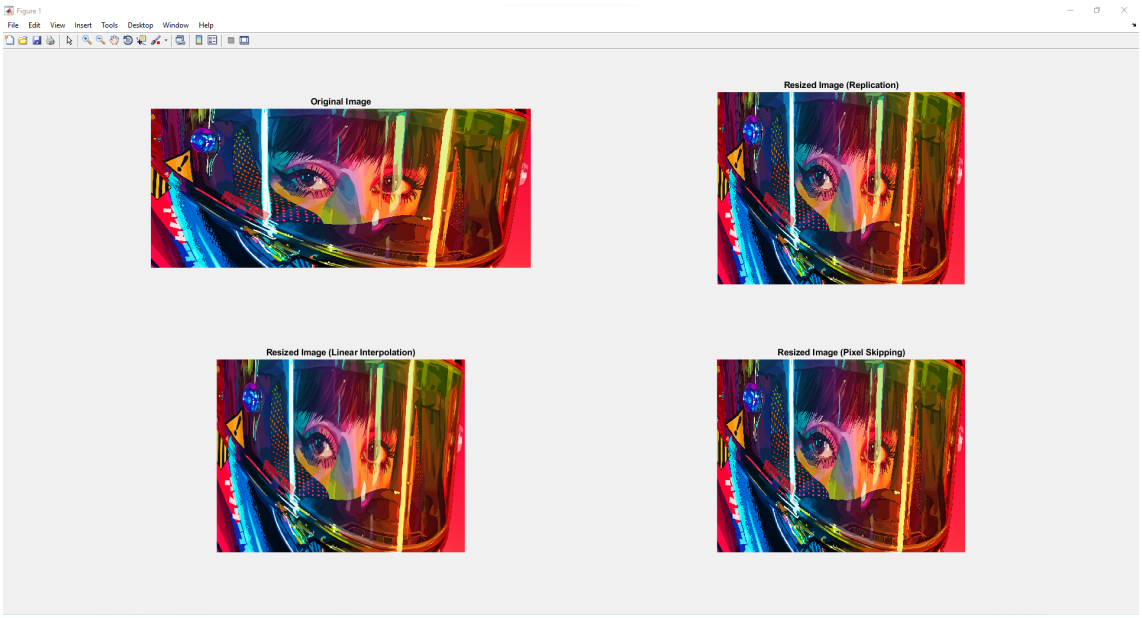

Figure 3: Image Resizing Output for Matlab Code

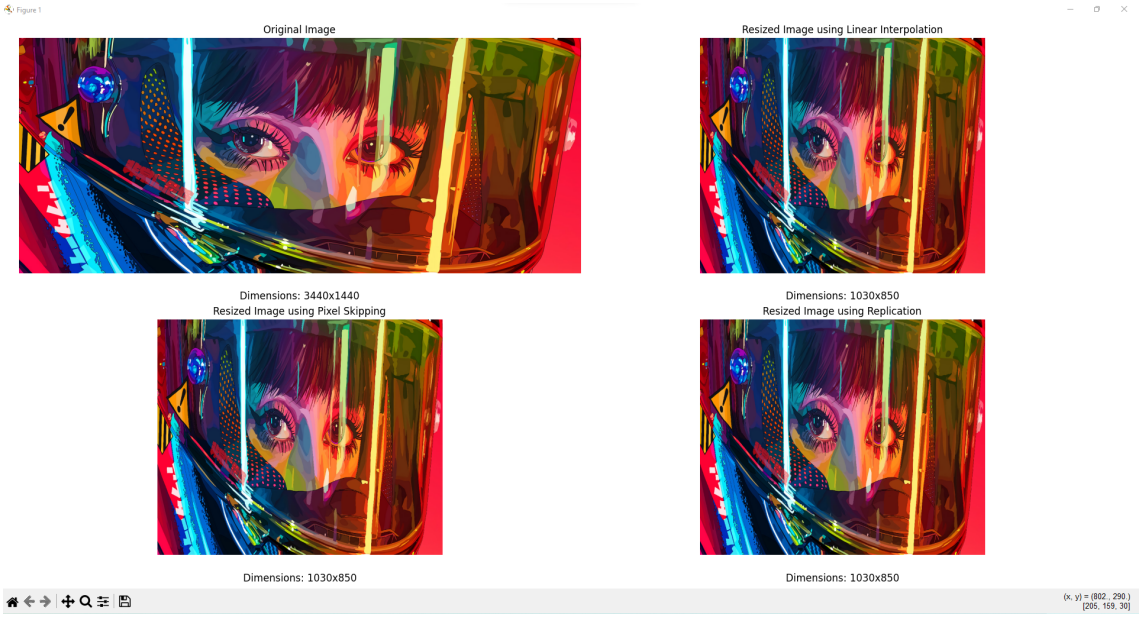

Figure 4: Image Resizing Output for Python Code

Results

A) Image Filtering Results

- **Original Image** – The original grayscale and RGB both images are used as the base for filtering.
- **Mean Filtered Image** – After applying the Mean Filter, we observe that noise in the image is smoothed, but fine details may also be slightly blurred.
- **Median Filtered Image** – The Median Filter is effective in removing salt-and-pepper noise while preserving the edges of the image.
- **Gaussian Filtered Image** – Gaussian Filter smooths the image by reducing high-frequency components while introducing a Gaussian blur effect, depending on the sigma value used.

Comparison: Each filter shows different levels of smoothing and noise reduction, with the Median Filter being most effective for removing noise without sacrificing details, and Gaussian Filter introducing smoothness while preserving more of the original image's structure compared to the Mean Filter.

B) Image Resizing Results

- **Replication Method** – This method results in a pixelated or blocky appearance, as it simply replicates the nearest pixel value for scaling.
- **Linear Interpolation Method** – Produces a smoother resized image by calculating weighted averages of neighboring pixel values, reducing the pixelation effect.
- **Pixel Mapping Method** – Involves subsampling pixels, which can lead to loss of detail but ensures fast and efficient resizing for certain applications.

Comparison: Linear interpolation provides the best quality among the methods, preserving more detail and producing smooth transitions in pixel values. Replication can be fast but results in blockiness, while pixel mapping is simple but may sacrifice accuracy.