EEE391 - MATLAB Assignment 2 - Görkem Kadir Solun 22003214

Question 1

Part 1

In a convolution operation, each pixel in the image is replaced by a weighted sum of its neighboring pixels, determined by a filter or kernel. The filter size defines how many neighboring pixels influence the new pixel value.

When the filter size N is slight, such as $N = 3$ (a $3x3$ filter), only a limited neighborhood contributes to the calculation. This results in mild smoothing while preserving structures and details, keeping the image sharp and clear.

With a larger filter size, such as $N = 10$, the convolution considers a broader neighborhood of pixels. This leads to more pronounced smoothing, causing the image to appear somewhat blurry, though still recognizable. Fine aspects and noise are substantially reduced.

When $N = 50$, the filter spans a vast area, producing heavy smoothing and noticeable blurring. Only typical forms and crude structures remain visible at this scale, while finer details are lost.

In conclusion, because the filter averages pixel values over a greater and greater region, the image gets increasingly blurrier as N grows.

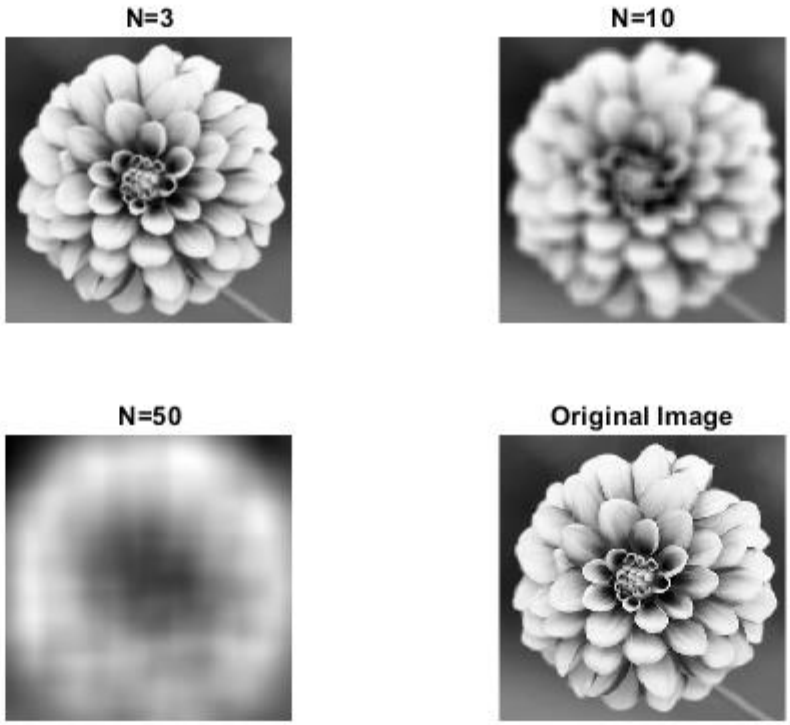

Figure 1

The original $conv2$ function accounts for only minor relative changes introduced by image padding. These changes arise because padding slightly alters the dimensions of the input image, which can affect the output size and the boundary conditions of the convolution operation. Despite this, the two versions of the function (with and without padding adjustments) remain almost identical in logic and execution, as expected. This similarity ensures that the core convolution process remains consistent, regardless of the subtle differences introduced by padding.

Part 2

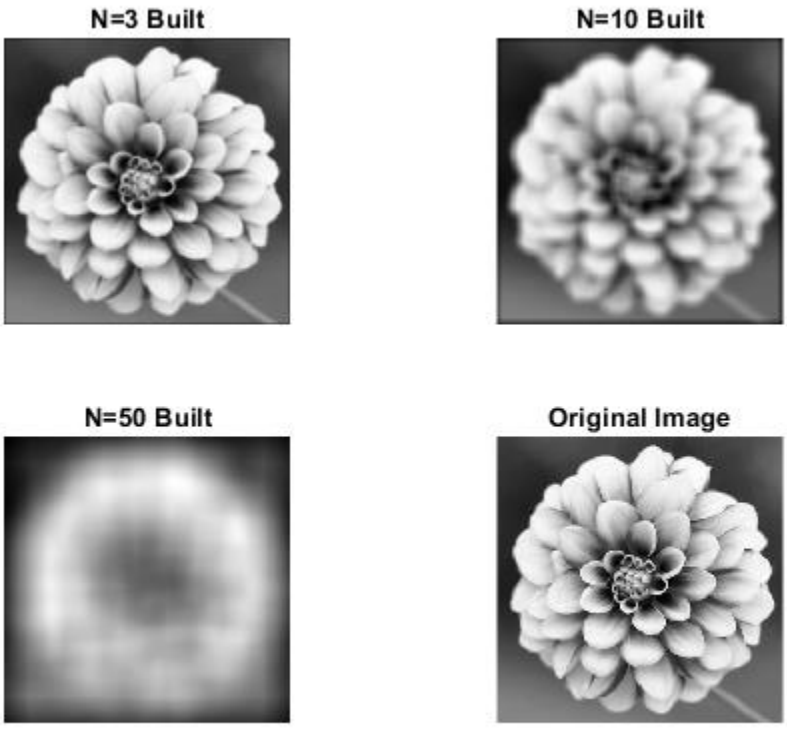

Figure 2

When using a filter with a kernel size of $N = 3$, the filter performs mild smoothing, effectively reducing subtle noise while maintaining most of the image's fine details and edges. This is particularly beneficial in applications where preserving texture and detail is essential, such as medical imaging or photography requiring clarity and precision.

As the kernel size increases to $N = 10$, the filter becomes more aggressive in reducing noise. While more significant noise artifacts and visual inconsistencies are removed, some finer details and subtle textures blur. This level of smoothing may be appropriate for images where a cleaner, more uniform appearance is required, but it might compromise the sharpness of intricate patterns or edges.

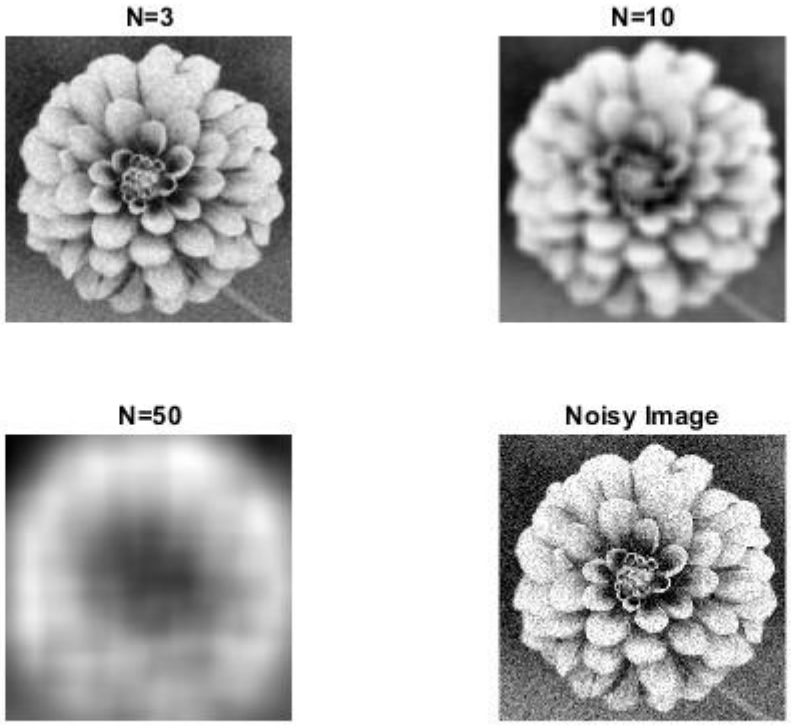

Figure 3

When the kernel size is further increased to $N = 50$, the filter reduces noise significantly, eliminating almost all visible noise. However, this comes at the cost of substantially blurring edges and finer details, making the image appear overly smooth or softened. This level of filtering is suitable for scenarios where noise suppression is prioritized over detail preservation, such as preparing images for compression or creating artistic effects.

Increasing the kernel size (N) generally enhances the smoothing effect: larger filters are more effective at suppressing noise, but they also lead to a more significant loss of fine details, edges, and textural information. Therefore, selecting the optimal kernel size involves a trade-off between noise reduction and detail preservation.

For most applications that require balancing noise reduction with maintaining image clarity, $N = 3$ is the best choice. This setting strikes a balance by effectively reducing noise while keeping the essential details and edges of the image intact, ensuring the image remains sharp and visually accurate.

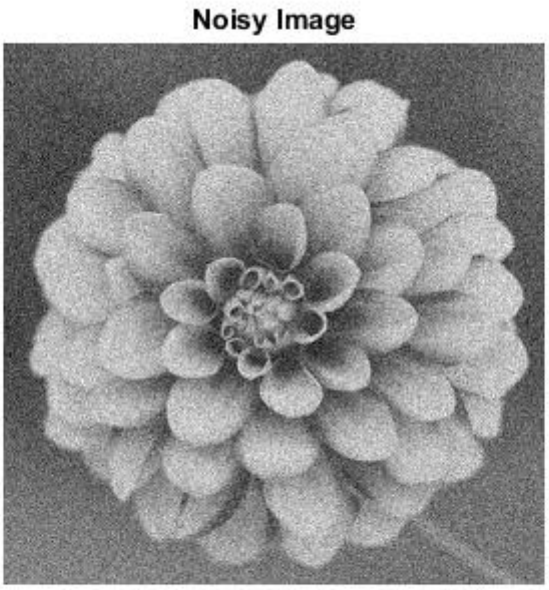

Figure 4

Question 2

The two displayed images, smoothed combined and summed filtered, appear identical because of the linear property of convolution in mathematical operations. Convolution is a linear process, meaning it adheres to the principle of superposition, which states that the result of applying a convolution filter to the sum of two inputs is the same as summing the results of applying the filter to each input separately.

In this case, if we add two images and then apply a convolutional filter producing smoothed combined, the result is mathematically equivalent to convolving each image individually with the same filter and summing the outputs, producing summed filtered. This is due to the commutative and distributive properties of linear operations.

This behavior underscores the consistency and predictability of linear systems, as the operation's outcome does not depend on the order in which addition and convolution are applied. Equivalence is especially useful in image processing, where it simplifies computations and ensures reproducibility of filtered results.

Quaternion convolution, Linearity:

$$(a\,f\,+\,b\,g)\,\star\,h\,=\,a\,(f\,\star\,h)\,+\,b\,(g\,\star\,h), with\,a,b\,\in\,\mathbb{H}$$

$$h\,\star\,(a\,f\,+\,b\,g)\,=\,a\,(h\,\star\,f)\,+\,b\,(h\,\star\,g), with\,a,b\,\in\,\mathbb{R}$$

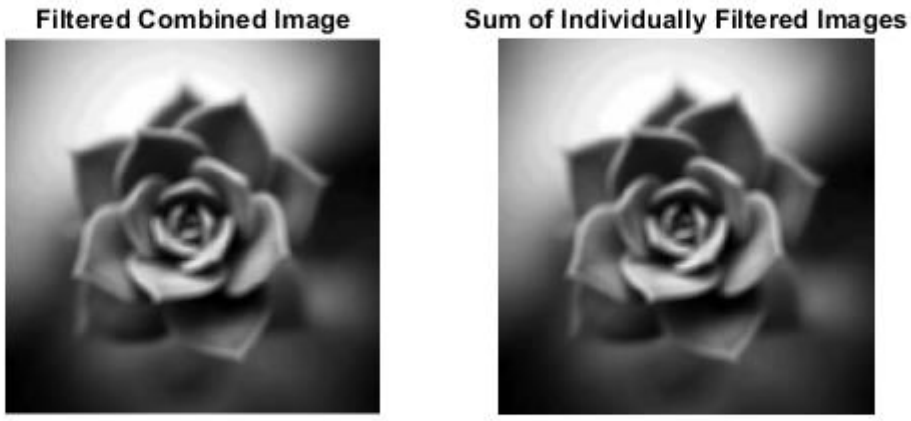

Figure 5

In the first approach, when generating the output referred to as smoothed combined, the images are combined before applying the smoothing filter. Specifically, two images, image1, and image2, are weighted and combined into a single matrix by performing the operation:

$$combined\ image\ =\ 10\ times\ image1\ +\ 5\ times\ image2$$

Once this combined image has been formed, a smoothing filter, such as a Gaussian or averaging filter, is applied to the entire combined image. This filter smooths the result by averaging the pixel values within a localized neighborhood, reducing noise, and smoothing out sharp transitions.

In the second approach, the order of operations is reversed when generating the output referred to as summed filtered. Here, the smoothing filter is applied individually to each weighted image first. Specifically, the filter is applied separately to:

- 10 times image1, resulting in a smoothed version of the weighted image1.
- 5 times image2, resulting in a smoothed version of the weighted image2.

The two filtered images are then summed to produce the final output:

$$Summed\ filtered\ =\ filter\ applied\ to\ (10\ times\ image1)\ +\ filter\ applied\ to\ (5\ times\ image2)$$

Both methods yield the same result due to the mathematical property of convolution linearity. Convolution is a linear operation, which means the order of applying a linear filter (such as the

smoothing filter) and summing weighted images does not affect the outcome. This property can be expressed as:

$$filter\ applied\ to\ (a\ times\ f\ +\ b\ times\ g) = a\ times\ (filter\ applied\ to\ f)\ +\ b\ times\ (filter\ applied\ to\ g),$$

$$where\ a\ and\ b\ are\ constants, and\ f\ and\ g\ represent\ functions\ or\ images.$$

This explains why the two images produced by the smoothed combined and summed filtered approaches are identical. The identical results confirm that, in this specific scenario, combining the images and applying the filter does not impact the outcome.

Question 3

Figure 6

Preprocessing

The preprocessing step involved preparing the input text image to enhance the accuracy of letter detection. Specifically:

- Cropping: The original text image was cropped to isolate the horizontal portion of the text. This was done to eliminate any potential false detections of vertically aligned t letters. The cropping operation extracted the image data within rows 1 to 55 and columns 1 to 200, focusing only on the relevant section.
- Rescaling: After cropping, the text and reference letter t images were rescaled. The pixel intensity values were normalized to fall from 0 to 1. This normalization ensured compatibility with convolution operations, making the matching process more accurate and consistent across varying image intensities.

Procedure

The following steps outline the process for detecting occurrences of the letter t in the cropped text image using template matching with convolution:

1. Focus on Horizontal Text: The cropped image was designed to isolate the horizontal text, minimizing the likelihood of detecting false positives from vertically aligned letters.
2. Convolution with Template: The horizontal portion of the text image was convolved with the reference image of the letter t. This convolution operation identifies regions in the text where the template (the letter t) closely matches the image content.
3. Normalization of Convolution Result: Using the rescale function, the convolution output was normalized to a [0, 1] scale. This step ensures that the convolution values are standardized, facilitating accurate thresholding for peak detection.
4. Thresholding to Identify Peaks: A threshold of 95% (0.95) was applied to the normalized convolution result to identify significant peaks. Peaks above this threshold correspond to regions where the letter t was detected with high confidence.
5. Counting Peaks: The number of peaks above the threshold was counted to determine how many times the letter t appeared in the horizontal text section.

Application Name

The described process is part of a Template Matching Application. This application utilizes convolution-based methods to detect specific patterns within a larger image, in this case, the letter t. Template matching is a powerful technique commonly used in image processing to locate objects or patterns by comparing a reference template to regions within the target image.

Code

```
% Görkem Kadir Solun 22003214
```

```
clear;
clc;
close all;

% Read and prepare the original image
I = imread('flower.jpg');
A = mat2gray(rgb2gray(I));

% Define filter sizes
filter_sizes = [3, 10, 50];

% ------------------------------------------------------------
% QUESTION 1
% PART 1: Custom convolution on original image
% ------------------------------------------------------------

perform_custom_convolution(A, filter_sizes, 'Original Image');

% Compare with built conv2 results
figure;

for i = 1:length(filter_sizes)
   f = ones(filter_sizes(i)) / (filter_sizes(i) ^ 2);
   output_image_built = conv2(A, f, 'same');
   subplot(2, 2, i);
   imshow(output_image_built, []);
   title(['N=', num2str(filter_sizes(i)), ' Built']);
end

subplot(2, 2, 4), imshow(A), title('Original Image');

% ------------------------------------------------------------
% QUESTION 1
% PART 2: Add noise and repeat
% ------------------------------------------------------------

noisy_image = A + 0.1 * randn(size(A));
figure;
imshow(noisy_image, []), title('Noisy Image');
perform_custom_convolution(noisy_image, filter_sizes, 'Noisy Image');

% Part 2 (continued): Linearity test with combined images
image1 = im2double(imread('image1.png'));
image2 = im2double(imread('image2.png'));
filter_10 = ones(10, 10) / 100;

smoothed_combined = custom_convolution(10 * image1 + 5 * image2, filter_10);
smoothed_image1 = custom_convolution(10 * image1, filter_10);
smoothed_image2 = custom_convolution(5 * image2, filter_10);
```

```
figure;
subplot(1, 2, 1), imshow(smoothed_combined, []), title('Filtered Combined Image');
subplot(1, 2, 2), imshow(smoothed_image1 + smoothed_image2, []), title('Sum of Individually Filtered Images');

% Part 3: Detect occurrences of 't' in a text image
text_image = im2double(imread('Text.png'));
letter_t = im2double(imread('lettert.png'));
horizontal_text = text_image(1:55, 1:200);

conv_result = custom_convolution(horizontal_text, letter_t);
normalized_result = rescale(conv_result);
detected_peaks = normalized_result > 0.95;
num_t_horizontal = sum(detected_peaks(:));

figure;
subplot(1, 3, 1), imshow(text_image), title('Original Text Image');
subplot(1, 3, 2), imshow(horizontal_text), title('Horizontal Text');
subplot(1, 3, 3), imshow(detected_peaks), title(['Detected "t": ', num2str(num_t_horizontal)]);

%% Local Functions

function output_image = custom_convolution(image, filter)
  [filter_rows, filter_columns] = size(filter);
  [image_rows, image_columns] = size(image);
  output_image = zeros(image_rows - filter_rows + 1, image_columns - filter_columns + 1);

  for i = 1:size(output_image, 1)

    for j = 1:size(output_image, 2)
      local_region = image(i:i + filter_rows - 1, j:j + filter_columns - 1);
      output_image(i, j) = sum(sum(local_region .* filter));
    end

  end

end

function perform_custom_convolution(image, filter_sizes, name)
  figure;

  for i = 1:length(filter_sizes)
    f = ones(filter_sizes(i)) / (filter_sizes(i) ^ 2);
    output_image = custom_convolution(image, f);
    subplot(2, 2, i);
    imshow(output_image, []);
    title(['N=', num2str(filter_sizes(i))]);
  end

  subplot(2, 2, length(filter_sizes) + 1);
```

```
   imshow(image);
   title(name);
end
```