Bilateral Filter for Grayscale images

A REIMPLEMENTATION AND ANALYSIS

December 7, 2022

Part I. a Bilateral Filter Review

Traditional low-pass filters were unable to produce optimal results at edges, often blurring them in an undesirable manner. Existing solutions such as anisotropic diffusion were mostly iterative, which introduced instability and were inherently inefficient. [1] Therefore, there was a strong motivation to develop a technique that could smooth images without losing edge details in a simple and non-iterative manner. To address this issue, C. Tomasi and R. Manduchi [2] proposed the bilateral filter, a type of non-linear smoothing filter that was designed to reduce noise in images while preserving edges. The filter worked by weighting the contribution of each pixel in the input image to the corresponding pixel in the output image, using two Gaussian functions: one to weigh the spatial distance between pixels, and the other to weigh the intensity difference between pixels. The resulting multiplied weighted average was used to compute the output pixel value, which resulted in a smoothed image that still retained fine details and sharp edges. The main contribution of the paper was the mathematical definition of this non-iterative, local, and simple method of edge-preserving smoothing. Another contribution was the extension of the method to color images to reduce phantom colors. Since its development, bilateral filters have been widely used in denoising, texture synthesis, and tone mapping. [3] The authors presented formulas and supported their theories with convincing image outputs using different parameters. However, the method was non-linear, which made it difficult to evaluate mathematically and to fully understand its effects on an image. Additionally, the authors claimed that the computational cost could be significantly decreased, which turned out to be an overly optimistic evaluation. The naive implementation of the bilateral filter was impractical for processing large images.

Part I. b Fast Approximation of Bilateral Filter Review

Researchers from MIT proposed a new version of the bilateral filter to improve its speed. [4] It consisted of two main steps: a separable linear convolution using a 3D Gaussian kernel, and a non-linear normalization using division and slicing. The division was the process of normalization and slicing was the reduction from 3D back to 2D space. The novel technique used a *3D bilateral grid* in the convolution process. This grid was created by introducing a range dimension that represents the grayscale of each pixel in the image. This allowed the bilateral filter to be performed in a higher dimension, making it separable. In addition, the image was downsampled using a box filter before convolution. After the convolution, the data was upsampled linearly. These operations were done in constant time, while convolution was performed at an increased computational rate. Although this approach sacrificed some accuracy due to the lower resolution, the researchers claimed that the changes were not significant and the approximation significantly improved memory and time consumption. The recent paper leads to the following contributions: Conceptually, it defined bilateral filter as a convolution in a higher-dimensional space. Practically, the enhanced algorithm produced results in seconds while maintaining high precision, making it an overall better computation. The researchers also performed extensive testing of the proposed method using various types of images. They found that it performed well in terms of both numerical accuracy and running time complexity. The validation process was enhanced by its superior performance over other existing approaches. However, one limitation of the novel approach was that it might take longer to process RGB images with very small kernels. [5]
The flowcharts for the methods in the two papers can be found in the appendix.

Part II. Re-implementation and Validation

1. Code Implementation in python with OpenCV

```
import sys
import numpy as np
import matplotlib.pyplot as plt
import cv2

def bilateral_filter(image, kernel_size, sigma_intensity, sigma_range):
    half_size = kernel_size // 2
    RGB = 1
    dim = len(image.shape)
    if dim != 2:
        RGB = image.shape[2]
    height = image.shape[0]
    width = image.shape[1]
    image = image.reshape(height, width, RGB)
    output_image = np.zeros(image.shape)

    for i in range(half_size, height - half_size):
        for j in range(half_size, width - half_size):
            for k in range(RGB):
                total_weight = 0.0
                total_feature = 0.0
                for x in range(-half_size, half_size+1):
                    for y in range(-half_size, half_size+1):
                        #Range weight (Gaussian)
                        weight_range = -(x ** 2 + y ** 2) / (2 * (sigma_range ** 2))
                        #Intensity difference (Also Gaussian)
                        intensity_feature = -(int(image[i][j][k]) - int(image[i + x][j + y][k])) ** 2 / (2 *
(sigma_intensity ** 2))
                        weight = np.exp(weight_range + intensity_feature)
                        total_weight += weight
                        total_feature += (weight * image[i + x][j + y][k])
                        #Normalization
                curr_scale = total_feature / total_weight
                output_image[i][j][k] = curr_scale
    return output_image.astype(np.uint8)

def arguments(args: list) -> tuple:
            #Simply for testing from command line
    filename = args[1] if args[1:] else "random.png"
    sigma_intensity = float(args[2]) if args[2:] else 75.0
    sigma_range = float(args[3]) if args[3:] else 75.0
    if args[4:]:
        k_s = int(args[4])
```

```
        #Kernel size must be an odd number
        if k_s % 2 == 0:
            k_s += 1
    else:
        k_s = 9
    return filename, sigma_intensity, sigma_range, k_s

if __name__ == '__main__':
        file_path, sigma_intensity, sigma_range, kernel_size = arguments(sys.argv)
        image = cv2.imread(file_path, 0)
        #bilateral = cv2.bilateralFilter(image, kernel_size, sigma_intensity, sigma_range)
        my_bilateral = bilateral_filter(image, kernel_size, sigma_intensity, sigma_range)
        figure = plt.figure(figsize=(10, 10))
        plt.subplot(1, 2, 1), plt.title('Original image')
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.axis('off')
        plt.subplot(1, 2, 2), plt.title('My bilateral filter')
        plt.imshow(cv2.cvtColor(my_bilateral, cv2.COLOR_BGR2RGB)), plt.axis('off')
        #Just some code to save the output
        plt.gca().set_axis_off()
        plt.subplots_adjust(top = 1, bottom = 0, right = 1, left = 0,
                hspace = 0, wspace = 0)
        plt.margins(0,0)
        plt.gca().xaxis.set_major_locator(plt.NullLocator())
        plt.gca().yaxis.set_major_locator(plt.NullLocator())
        plt.show()
        figure.savefig('result.png', bbox_inches = 'tight',
        pad_inches = 0)
```

2. Representation of result

Geometric spread = σ_d, *photometric spread* = σ_r.

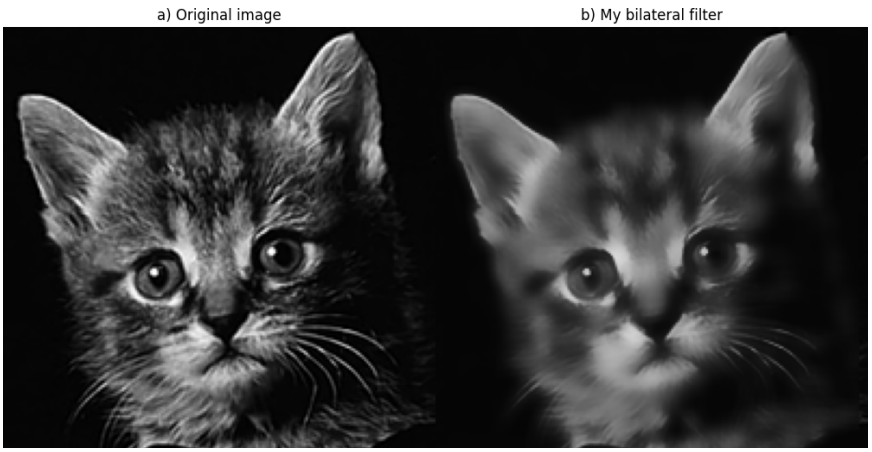

Figure 1: A picture of cat before (a) and after the re-implementation (b), with Kernel size = 9, geometric spread = photometric spread = 75

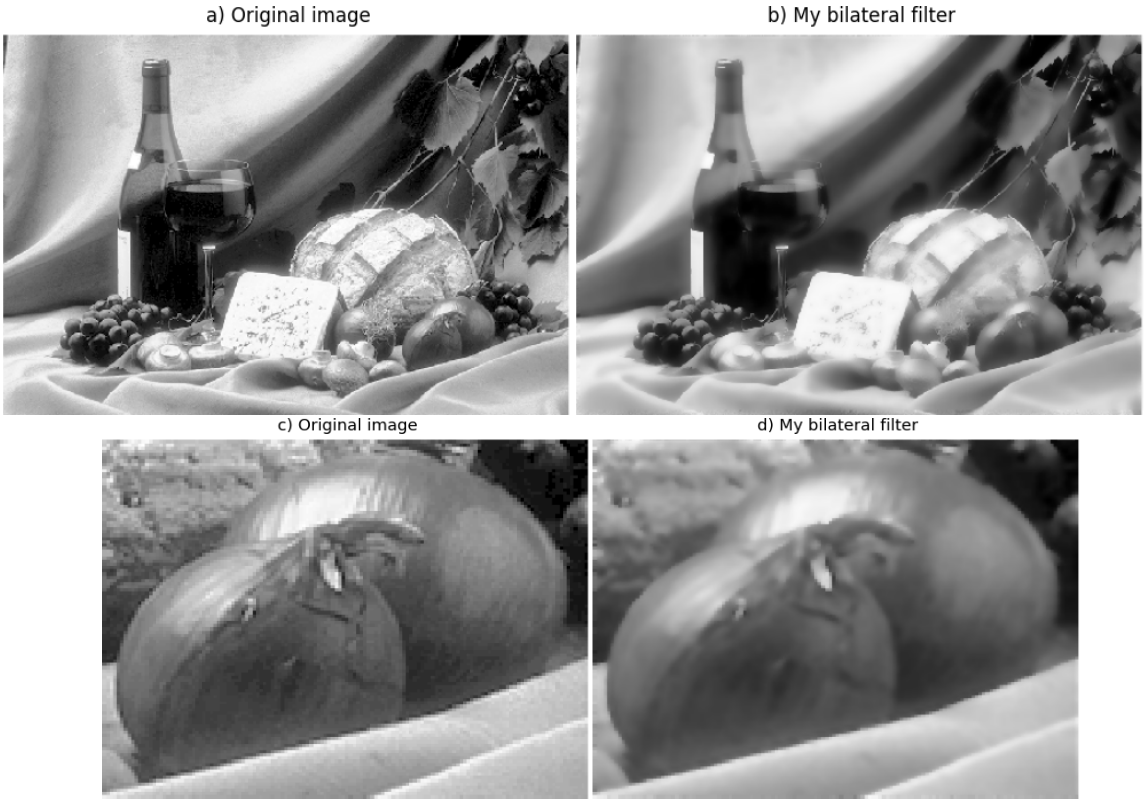

Figure 2: A picture of food before (a) and after the re-implementation (b), with geometric spread = 3 and photometric spread = 50. (c) and (d) shows the details of (a) and (b).

Figure 1 shows the effect of our re-implemented bilateral filter on a cat image. This image was chosen because it was used in the original paper. Upon examination, we can see that most textures and details on the fur are lost, but significant edges around the ears, eyes, nose, and whiskers are preserved. This demonstrates that the re-implementation successfully achieves the goal of denoising the image while preventing edges from being heavily affected by blurring.

Figure 2 shows the effect of the filter on a variety of objects. The scattered features on the bread and the veins on the leaves disappear, but the contours of the objects remain unaffected. A closer look at the leaves reveals that the shades are preserved due to similarities in neighbor intensities, also shown by the left edges on the left onion in (d).

Overall, the results suggest that our re-implemented bilateral filter is effective at smoothing the image and reducing noise while preserving edges. This is in line with the goals of the bilateral filter, indicating the success of our re-implementation.

3. Validation of the algorithm

For validation, we first compare our implementation with existing baseline method in OpenCV, cv2.bilateralFilter().

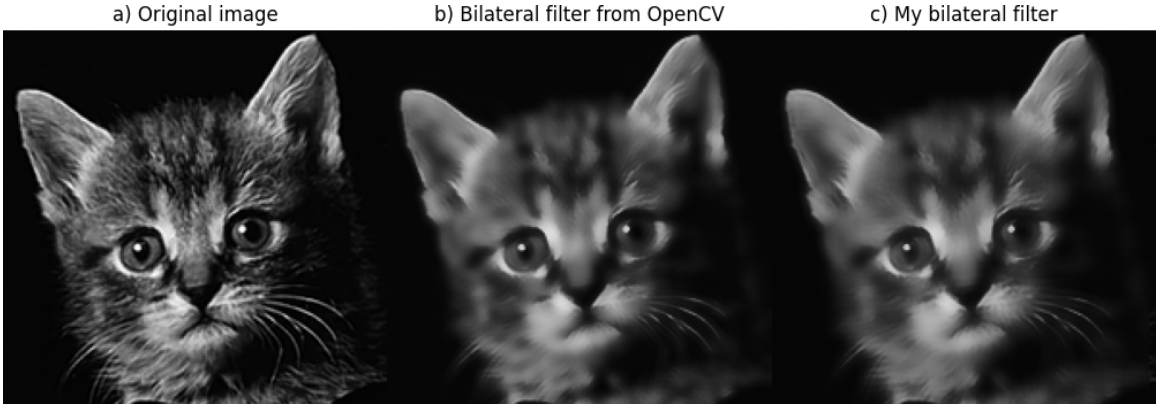

Figure 3: A picture of cat before (a) and after the re-implementation (c), compared to baseline function in OpenCV (b), with same parameters in Figure 1.

Figure 4: A picture of camera man before (a) and after the re-implementation (c), compared to baseline function in OpenCV (b), geometric spread = 3 and photometric spread = 100.

Our implementation of the bilateral filter produces results that are visually identical to the baseline method, as shown in Figure 3. The baseline method preserves edges while smoothing textures, which is matched by our implementation. To further demonstrate this, we tested our implementation using another image with different parameters in Figure 4. The result is still visually similar to the baseline method. This is due to crisp edges around the camera, the smoothing of the background buildings, and the presence of triangular edges from one building in the background. Therefore, we can conclude that our implementation is accurate and produces results in a similar fashion to the baseline method.

To further validate our implementation, we tested the effects of different parameter values on the output, as shown in Figure 5. We chose parameter values like those used in the original paper.

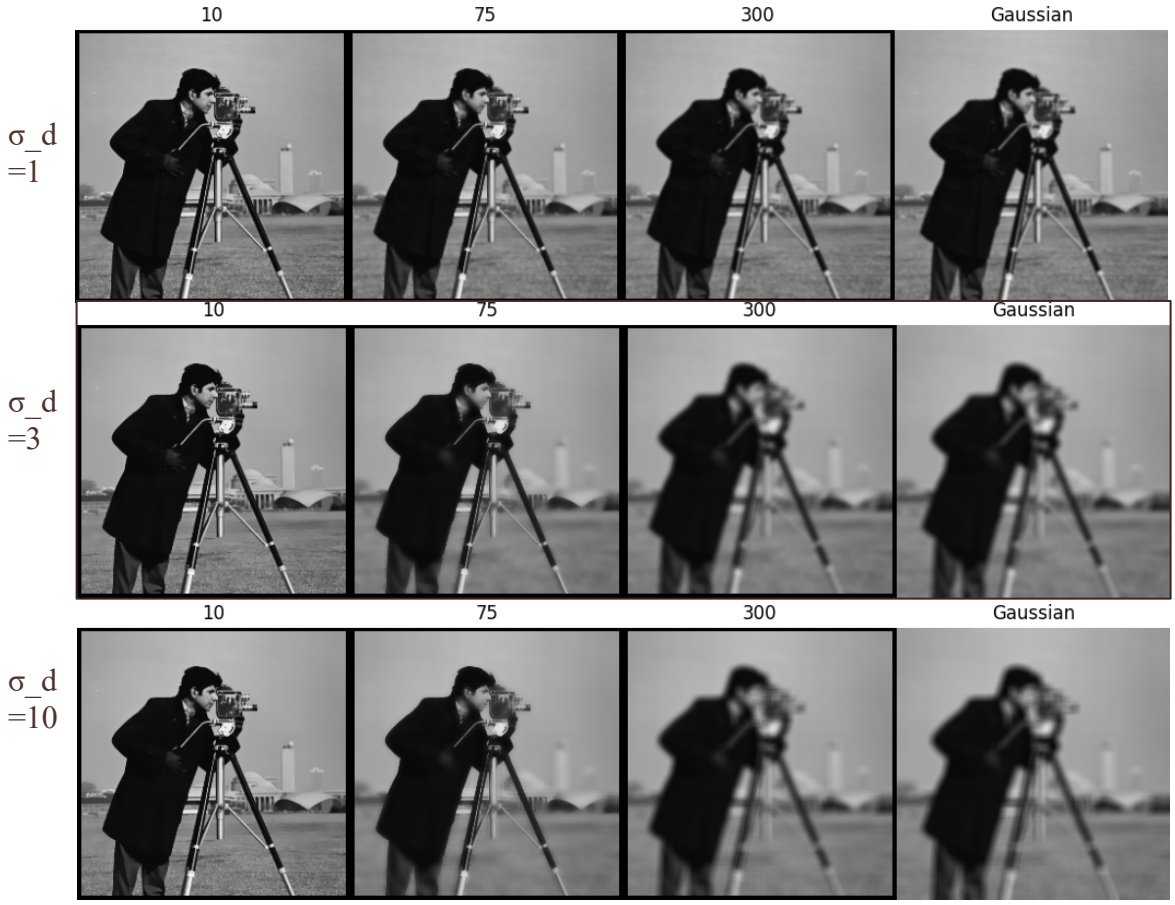

Figure 5: Bilateral filtering of the camera man under different parameters.
Vertically σ_d = 1,3,10, horizontally σ_r = 10, 75, 300. Gaussian filter for control.

We try to investigate the behavior of our implementation in relation to its parameters, specifically σ_r and σ_d, to see if they align with the mathematical properties of a bilateral filter. Our findings show that when σ_r is high, the bilateral filter behaves like a Gaussian filter. This is because a large σ_r makes the Gaussian close to constant across the intensity interval, as indicated by the formula of the bilateral filter. In our experiments, we found that all σ_r=300 examples were visually identical to the corresponding Gaussian filter.

Additionally, we found that a larger σ_d results in larger features being smoothed. This can be observed in the second column of Figure 5, where the background buildings are blurred with increasing σ_d.

One key aspect of the bilateral filter is that the weights from geometric and photometric pixels are multiplied together. This means that if the weight is close to 0, smoothing will not be applied. Therefore, when a large σ_d is combined with a small σ_r, the smoothing will be highly restricted even with a large σ_d. This can be seen in the first column of Figure 5, where the smoothing is extremely limited across samples with increasing σ_d.

Through our experiments with various parameters, we have confirmed that our implementation satisfies the essential characteristics of the bilateral filter. This validation supports the effectiveness of our algorithm.

4. Reflection on re-implementation

The bilateral filter is a widely used tool in image processing and computer vision, with applications in denoising and edge preservation. In this work, we present a re-implementation of the bilateral filter and validate its effectiveness through theoretical analysis and experimental results.

Our implementation is based on the original paper describing the bilateral filter, and we were able to reproduce similar results using the same test images. We also compare our results to those of a baseline method and find that they are visually identical. One advantage of our implementation is that it is independent of external libraries, using only NumPy arrays for the data structure. Additionally, our code is simple and easy to understand, with the core function only consisting of around 20 lines.

However, a significant limitation of our implementation is that it is computationally slow. Naive implementations of the bilateral filter are known to be impractical due to their high computational cost, and our implementation already takes a few minutes to produce results for low-resolution images. We did not use any high-resolution images in our test due to this, although high-resolution images would produce more distinguishable results in general. We have also chosen to use the default padding when using the filter, which can lead to the loss of some information at the borders of the image. It is also worth considering that the bilateral filter can be extended to work with color images. We have provided a third RGB dimension in our code but have not yet focused on testing with color images.

Despite these limitations, the implementation itself is a successful representation of a bilateral filter that can reduce noise and preserve edges. This is beneficial because we learned in class that in several areas such as medical image processing, edges are key features that should be kept undistorted. From the lecture, we learned about the Gaussian filter for denoising. The bilateral filter shares some similarities with the Gaussian filter in the relationship between weight and spatial distance from the center pixel. In addition, pixels in neighborhoods will have a larger impact. However, the presence of intensity weight in the bilateral filter allows pixels that have similar grayscale to influence the current pixel more. Thus, we can remove noise from the image while still maintaining its key features. This can improve the quality of the image and make it easier to analyze or process.

Since the code itself is short and straightforward, there weren’t major difficulties in re-implementing the algorithm in the paper. In the process, there was a problem with applying the Gaussian function to both spatial and intensity weights because it was defined separately so it wasn't working on both. After switching to the direct calculation of Gaussian values within the

loop, the issue is solved. Another obstacle is determining the range of i,j,x, and y in the loop, which creates some confusion. However, we eventually overcome it.
In conclusion, we have validated our implementation both theoretically and practically, and have demonstrated its effectiveness in reducing noise while preserving edges. Although the naive implementation is slow, there are existing rapid approximation methods that can improve the computational efficiency of the bilateral filter. Additionally, the bilateral filter can be extended to work with color images or combined with other filters to achieve better results. This makes the bilateral filter a versatile and useful tool in a wide range of applications.

Bibliography

[1] R. T. Chin and C. L. Yeh. Quantitative evaluation of some edge preserving noise-smoothing techniques. CVGIP, 23:67–91, 1983.

[2] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Sixth international conference on computer vision (IEEE Cat. No. 98CH36271), 1998: IEEE, pp. 839-846.

[3] M. Zhang and B. K. Gunturk, "Multiresolution bilateral filtering for image denoising," (in eng), IEEE Trans Image Process, vol. 17, no. 12, pp. 2324-33, Dec 2008, doi: 10.1109/tip.2008.2006658.

[4] S. Paris and F. Durand, A fast approximation of the bilateral filter using a signal processing approach, European Conference on Computer Vision, (2006), pp. 568–580

[5] S. Paris, A gentle introduction to bilateral filtering and its applications, in ACM SIGGRAPH 2007 courses, 2007, pp. 3-es.

Appendix

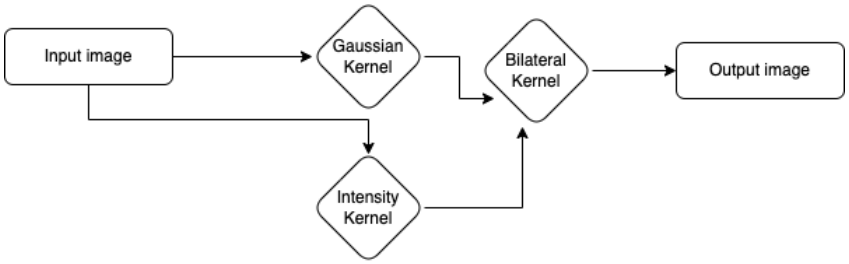

Figure 1a: Flowchart of original bilateral filter

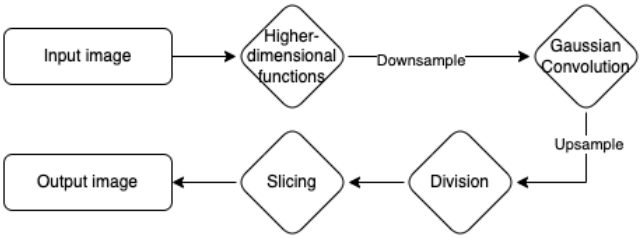

Figure 1b: Flowchart of fast approximation of bilateral filter