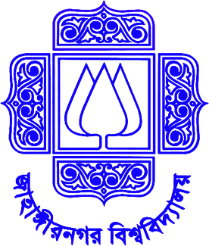
Institute of Information Technology

Jahangirnagar University

Savar, Dhaka-1342



Lab Manual

Course Code: ICT-4202

Course Title: Digital Image Processing Lab

**Lab No.: 6**

**Lab Title: Image Denoising Using Median, Bilateral, NLM and Total Variance.**

Prepared by

**Mehrin Anannya**

Assistant Professor

Institute of Information Technology

Jahangirnagar University

**Lab Title: Image Denoising Using Median, Bilateral, NLM, Total Variance and BM3D**: To introduce students with Image processing filters named Median, Bilateral, NLM and Total Variance.

**Lab Contents:**

* Denoising using median filter
* Denoising using Bilateral filter
* Denoising using NLM filter
* Denoising using total variance filter
* Denoising using bm3d filter

**Theory with Hands on Practice:**

**Introduction to Median Filtering:**

Denoising with median filtering is a common technique used in image processing to reduce the impact of impulse noise, such as salt-and-pepper noise. Impulse noise typically manifests as randomly occurring white and black pixels in an image, and median filtering is particularly effective in mitigating this type of noise while preserving image details.

The key idea behind median filtering is to replace each pixel value in an image with the median value of its neighborhood. This process is typically performed using a sliding window, often referred to as a kernel or a structuring element. For each pixel, the values within the neighborhood are sorted, and the median value is assigned to the pixel being processed.

**Denoising using median Filtering:**

See how median is much better at cleaning salt and pepper noise compared to Gaussian.

import cv2

from skimage.filters import median

#Needs 8 bit, not float.

img\_gaussian\_noise = cv2.imread('images/Osteosarcoma\_01\_25Sigma\_noise.tif', 0)

img\_salt\_pepper\_noise = cv2.imread('images/Osteosarcoma\_01\_8bit\_salt\_pepper.tif', 0)

img = img\_salt\_pepper\_noise

#img =img\_gaussian\_noise

#3 means square kernel size.g. 3X3

median\_using\_cv2 = cv2.medianBlur(img, 3)

from skimage.morphology import disk

#Disk creates a circular structuring element, similar to a mask with specific radius

median\_using\_skimage = median(img, disk(3), mode='constant', cval=0.0)

cv2.imshow("Original", img)

cv2.imshow("cv2 median", median\_using\_cv2)

cv2.imshow("Using skimage median", median\_using\_skimage)

cv2.waitKey(0)

cv2.destroyAllWindows()

This Python code is using the OpenCV (cv2) and scikit-image libraries to demonstrate median filtering on an input image. Median filtering is a non-linear filtering technique commonly used for image processing tasks like noise reduction.

**Here's a breakdown of the code:**

Here's an explanation of each module and function used in the code:

1. cv2: This module is from the OpenCV library and provides various computer vision functionalities.

2. skimage.filters.median: This function is from the scikit-image library and performs median filtering.

3. cv2.imread: Function to read an image file. It reads an image from the specified file and returns it.

4. 'images/Osteosarcoma\_01\_25Sigma\_noise.tif': The path to the image file containing Gaussian noise.

5. 'images/Osteosarcoma\_01\_8bit\_salt\_pepper.tif': The path to the image file containing salt and pepper noise.

6. img\_salt\_pepper\_noise: Variable storing the grayscale image with salt and pepper noise.

7. img\_gaussian\_noise: Variable storing the grayscale image with Gaussian noise.

8. img: Variable used to select the image for further processing (either salt and pepper noise or Gaussian noise).

9. cv2.medianBlur: Function from OpenCV to perform median filtering on an image. It replaces each pixel's value with the median value under the kernel area.

10. skimage.morphology.disk: Function to create a circular structuring element (a disk) with a specified radius.

11. median: Function from scikit-image to perform median filtering on an image. It replaces each pixel's value with the median value under the disk-shaped neighborhood.

This code helps visualize the effects of median filtering on images with salt-and-pepper noise and Gaussian noise. The `disk` structuring element in scikit-image creates a circular region used for median filtering, which is applied to reduce noise in the images.

**Introduction to Bilateral Filtering:**

The bilateral filter is a non-linear edge-preserving smoothing filter commonly used in image processing and computer vision applications. It was introduced as a means to reduce noise and blur while preserving important edges and details in an image. The bilateral filter achieves this by considering both the intensity differences and spatial distances between pixels.

**Denoising using Bilateral Filtering:**

Bilateral is slow and not very efficient at salt and pepper.

The provided code example demonstrates the use of bilateral filtering in both OpenCV (cv2.bilateralFilter) and scikit-image (skimage.restoration.denoise\_bilateral). Users can adjust the parameters of the bilateral filter based on the specific characteristics of the image and the desired trade-off between smoothing and edge preservation.

import cv2

img\_gaussian\_noise = cv2.imread('images/Osteosarcoma\_01\_25Sigma\_noise.tif', 0)

img\_salt\_pepper\_noise = cv2.imread('images/Osteosarcoma\_01\_8bit\_salt\_pepper.tif', 0)

img = img\_salt\_pepper\_noise

bilateral\_using\_cv2 = cv2.bilateralFilter(img, 5, 20, 100, borderType=cv2.BORDER\_CONSTANT)

#d - diameter of each pixel neighborhood used during filtering

# sigmaCOlor - Sigma of grey/color space.

#sigmaSpace - Large value means farther pixels influence each other (as long as the colors are close enough)

from skimage.restoration import denoise\_bilateral

bilateral\_using\_skimage = denoise\_bilateral(img, sigma\_color=0.05, sigma\_spatial=15, channel\_axis=None)

#sigma\_color = float - Sigma for grey or color value.

#For large sigma\_color values the filter becomes closer to gaussian blur.

#sigma\_spatial: float. Standard ev. for range distance. Increasing this smooths larger features.

cv2.imshow("Original", img)

cv2.imshow("cv2 bilateral", bilateral\_using\_cv2)

cv2.imshow("Using skimage bilateral", bilateral\_using\_skimage)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Here's a breakdown of the code:**

1. Import the OpenCV library (`cv2`) for computer vision tasks.
2. Read two grayscale images (`img\_gaussian\_noise` and `img\_salt\_pepper\_noise`) using `cv2.imread`.
3. Select the image with Gaussian noise (`img = img\_gaussian\_noise`).
4. Apply bilateral filtering using OpenCV (`cv2.bilateralFilter`) to reduce noise:

- `img`: Input image.

- `d`: Diameter of each pixel neighborhood (5).

- `sigmaColor`: Sigma of color space (20).

- `sigmaSpace`: Sigma of spatial space (100).

- `borderType`: Border handling type(`cv2.BORDER\_CONSTANT`).

1. Import the `denoise\_bilateral` function from `skimage.restoration`.
2. Apply bilateral filtering using scikit-image (`denoise\_bilateral`):

- `img`: Input image.

- `sigma\_color`: Sigma for grey or color value (0.05).

- `sigma\_spatial`: Standard deviation for range distance (15).

- `multichannel`: Indicates whether the image is multi-channel (`False` for grayscale).

7. Display the original image and the two filtered images using `cv2.imshow`.

8. Wait for a key event (`cv2.waitKey(0)`) and close all OpenCV windows (`cv2.destroyAllWindows()`) when a key is pressed.

In this code, bilateral filtering is applied to the input image using both OpenCV and scikit-image, showcasing how this technique can be utilized for denoising. The parameters may need to be adjusted based on the characteristics of the noise and the desired level of smoothing while preserving edges.

**Introduction to NLM Filtering:**

Works well for random gaussian noise but not as good for salt and pepper

https://www.iro.umontreal.ca/~mignotte/IFT6150/Articles/Buades-NonLocal.pdf

The non-local means algorithm replaces the value of a pixel by an average of a selection of other pixels values: small patches centered on the other pixels are compared to the patch centered on the pixel of interest, and the average is performed only for pixels that have patches close to the current patch.

The Non-Local Means (NLM) filter is a popular image denoising technique that aims to reduce noise while preserving edges and fine details. It is particularly effective in scenarios where traditional linear filters might fail to maintain image features. The key idea behind the NLM filter is to exploit redundancies in the image by comparing and averaging similar patches, rather than relying solely on local pixel neighborhoods.

**Denoising using NLM filtering:**

import cv2

import numpy as np

from skimage import io, img\_as\_float

from skimage.restoration import denoise\_nl\_means, estimate\_sigma

img\_gaussian\_noise = img\_as\_float(io.imread('images/Osteosarcoma\_01\_25Sigma\_noise.tif', as\_gray=True))

img\_salt\_pepper\_noise = img\_as\_float(io.imread('images/Osteosarcoma\_01\_8bit\_salt\_pepper.tif', as\_gray=True))

img = img\_gaussian\_noise

sigma\_est = np.mean(estimate\_sigma(img, channel\_axis=True))

#sigma\_est = 0.1

denoise\_img = denoise\_nl\_means(img, h=1.15 \* sigma\_est, fast\_mode=True,

patch\_size=5, patch\_distance=3, channel\_axis=None)

"""

When the fast\_mode argument is False, a spatial Gaussian weighting is applied

to the patches when computing patch distances. When fast\_mode is True a

faster algorithm employing uniform spatial weighting on the patches is applied.

Larger h allows more smoothing between disimilar patches.

"""

#denoise\_img\_as\_8byte = img\_as\_ubyte(denoise\_img)

cv2.imshow("Original", img)

cv2.imshow("NLM Filtered", denoise\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

This Python code uses the Non-Local Means (NLM) filter from scikit-image to denoise an image affected by Gaussian noise. The NLM filter is particularly effective in preserving edges and details while reducing noise.

**Here's an explanation of the code:**

1. img\_as\_float: A function from scikit-image to convert an image to a floating-point representation.

2. denoise\_nl\_means: A function from scikit-image for non-local means denoising. It reduces noise in the image based on similar patches.

3. estimate\_sigma: A function from scikit-image to estimate the standard deviation of Gaussian noise in an image.

4. img: Variable used to select the image with Gaussian noise for further processing.

5. \sigma\_est: Variable storing the estimated standard deviation of Gaussian noise in the selected image.

6. denoise\_nl\_means: Function to apply non-local means denoising to the image. Parameters include:

- `img`: Input image.

- `h`: Filtering strength (1.15 times the estimated standard deviation).

- `fast\_mode`: If True, a faster algorithm is applied with uniform spatial weighting.

- `patch\_size`: Size of patches used for denoising (5x5).

- `patch\_distance`: Maximum distance in pixels where to search for similar patches (3).

In this code, the Non-Local Means filter is applied to the image with Gaussian noise, resulting in denoised output. The `h` parameter controls the strength of the filter, and the other parameters affect the size and distance of the patches considered during the denoising process. The `fast\_mode` parameter toggles between faster computation with uniform spatial weighting and slower computation with spatial Gaussian weighting.

**Introduction to Total Varience Filter:**

Total Variation (TV) denoising is a popular non-linear image denoising technique that aims to preserve edges and fine details while effectively reducing noise. It's particularly useful for denoising images corrupted by random noise or other types of imperfections.

Works well for random gaussian noise but not as good for salt and pepper.

https://hal.archives-ouvertes.fr/hal-00437581/document

**Denoising Works using Total Variance filtering**

import cv2

from skimage import io, img\_as\_float

from skimage.restoration import denoise\_tv\_chambolle

from matplotlib import pyplot as plt

img = img\_as\_float(io.imread('images/Osteosarcoma\_01\_25Sigma\_noise.tif', as\_gray=True))

plt.hist(img.flat, bins=100, range=(0,1)) #.flat returns the flattened numpy array (1D)

denoise\_img = denoise\_tv\_chambolle(img, weight=0.1, eps=0.0002, n\_iter\_max=200, channel\_axis=False)

"""

denoise\_tv\_chambolle(image, weight=0.1, eps=0.0002, n\_iter\_max=200, multichannel=False)

weight: The greater weight, the more denoising (at the expense of fidelity to input).

eps: Relative difference of the value of the cost function that determines the stop criterion.

n\_iter\_max: Max number of iterations used for optimization

"""

plt.hist(denoise\_img.flat, bins=100, range=(0,1)) #.flat returns the flattened numpy array (1D)

cv2.imshow("Original", img)

cv2.imshow("TV Filtered", denoise\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

Let's break down the provided code step by step:

1. Import necessary libraries:

import cv2

from skimage import io, img\_as\_float

from skimage.restoration import denoise\_tv\_chambolle

from matplotlib import pyplot as plt

2. Read the input image:

- `io.imread`: This function from the `skimage` library is used to read the image file.

- `img\_as\_float`: This function converts the image to a floating-point representation, which is often required for numerical calculations in image processing.

3. Display histogram of original image:

- `plt.hist`: This function from the `matplotlib` library is used to plot a histogram of pixel intensities in the image.

- `img.flat`: This flattens the image array to a 1D array, suitable for histogram computation.

- `bins=100, range=(0,1)`: These parameters specify the number of bins and the range of pixel intensities for the histogram.

4. Apply Total Variation (TV) denoising using Chambolle's method:

- `denoise\_tv\_chambolle`: This function from the `skimage.restoration` module applies Total Variation denoising using Chambolle's method.

- `weight`: This parameter controls the amount of denoising. Higher values result in more aggressive denoising.

- `eps`: This parameter sets the relative difference of the cost function, which determines the stopping criterion for the optimization process.

- `n\_iter\_max`: This parameter specifies the maximum number of iterations used for optimization.

- `multichannel=False`: This parameter indicates that the input image is not a multichannel (color) image.

5. Display histogram of denoised image:

- Similar to the histogram of the original image, this line plots a histogram of pixel intensities in the denoised image.

6. Display original and denoised images using OpenCV:

- `cv2.imshow`: This function from the `cv2` (OpenCV) library is used to display images in separate windows.

- `cv2.waitKey(0)`: This function waits for a key press indefinitely.

- `cv2.destroyAllWindows()`: This function closes all OpenCV windows.

Overall, this code demonstrates how to read an image, apply Total Variation denoising using Chambolle's method, and visualize the original and denoised images along with their histograms. It uses libraries like `skimage`, `matplotlib`, and `cv2` for image processing and visualization.

**Introduction to Block matching and 3D filtering (BM3D) Filter:**

BM3D (Block-Matching and 3D Filtering) is an advanced denoising algorithm widely used for image restoration tasks. It operates on the principle of collaborative filtering, exploiting similarities between blocks of pixels in 3D groups. The algorithm begins by grouping similar blocks in the noisy image using a block-matching technique. For each group, a 3D transform is applied, and collaborative filtering is performed to reduce noise and enhance the image. BM3D is particularly effective in scenarios where images are corrupted by various types of noise, and it offers a robust solution for denoising while preserving important image details. The `bm3d` library in Python provides an implementation of this powerful algorithm, allowing users to easily integrate it into their image processing workflows.

**Denoising using Block matching and 3D filtering (BM3D):**

**Modules:**

* **pip install bm3d**

from skimage import io, img\_as\_float

import bm3d

import cv2

noisy\_img = img\_as\_float(io.imread("images/Osteosarcoma\_01\_25Sigma\_noise.tif", as\_gray=True))

BM3D\_denoised\_image = bm3d.bm3d(noisy\_img, sigma\_psd=0.2, stage\_arg=bm3d.BM3DStages.HARD\_THRESHOLDING)

"""

bm3d library is not well documented yet, but looking into source code....

sigma\_psd - noise standard deviation

stage\_arg: Determines whether to perform hard-thresholding or Wiener filtering.

stage\_arg = BM3DStages.HARD\_THRESHOLDING or BM3DStages.ALL\_STAGES (slow but powerful)

All stages performs both hard thresholding and Wiener filtering.

"""

cv2.imshow("Original", noisy\_img)

cv2.imshow("Denoised", BM3D\_denoised\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

The provided code utilizes the `bm3d` library to perform denoising on an image. Here's an explanation of the code:

1. skimage: The scikit-image library, used for image processing tasks.

2. io: A submodule of scikit-image for input and output operations on images.

3. img\_as\_float: A function from scikit-image to convert an image to a floating-point representation.

4. bm3d\*: A third-party library for image denoising using the Block-Matching and 3D Filtering algorithm.

5. cv2: The OpenCV library, providing computer vision functionalities.

6. noisy\_img: Variable storing the input image with Gaussian noise in floating-point format.

7. bm3d.bm3d: The main function from the `bm3d` library for denoising. Parameters include:

- `noisy\_img`: The input image with noise.

- `sigma\_psd`: The noise standard deviation used for denoising (0.2 in this case).

- `stage\_arg`: Determines whether to perform hard-thresholding or Wiener filtering. It can take values `bm3d.BM3DStages.HARD\_THRESHOLDING` or `bm3d.BM3DStages.ALL\_STAGES`.

The `bm3d` library implements a sophisticated denoising algorithm that involves block matching and 3D collaborative filtering. The `sigma\_psd` parameter is essential for specifying the standard deviation of the noise, and `stage\_arg` allows you to choose between hard-thresholding and Wiener filtering. This library is particularly effective for denoising images corrupted by various types of noise.

Tasks: 1. Take two images and compare between any two filtering techniques and write necessary comments on it mentioning which filtering techniques works well on the images.