DIGITAL IMAGE PROCESSING PROJECT

Prepared By:

1. NIGATU AGIZE	**SGS/0238/2014A**
2. TAMIRESILASSIE TILAHUN	**SGS/0241/2014A**
3. MICHAEL SIYOUM	**SGS/0235/2014A**

Submitted to: Million Meshesha (PHD)

ABSTRACT

In this paper, we tried to discuss Gaussian and Poisson noises that present in the image. Also, we discuss different types of filters which are used to remove these noises from original images. The removal of noise from images is a major task in the field of image processing because it affects the quality of the image and leads to the loss of some of its important information through the impact of noise on it. to remove the noise in the images, different image filtering techniques are used. In this paper, the main challenge was to conduct analytical study on the work of filters mean, medium, bilateral and Gaussian filter used to remove the types of noise such as Poisson noise (shot noise) and Gaussian noise. We will try to explain the causes of these types of noise and their impact on the images. In addition to how to remove them from the images will also be a comparison between the types of filters and their ability to remove noise according to specific criteria using python.

Table of Contents

Introduction

Digital Image Processing involves the reconstruction of digital information for enhancing the quality of the image with the help of a computer. The processing helps in boosting the clarity, sharpness of image, and draw out the details of interest for feature extraction and further study, the resultant is a new digital image that maybe displayed or saved in a format specified for images or might also be further modified using additional techniques. To improve certain elements and to eliminate noise, the digital images are put through different image processing operations.

Some sources of noise in digital images are:

1. Noise arises during the time of image acquisition or transmission.
2. At the time of image acquisition, the light level and sensor temperature are the major factors affecting the amount of noise in the resulting image.
3. The imaging senor may be affected by environmental conditions during image acquisition.
4. If the image is acquired directly in a digital format, the mechanism for gathering the data can introduce noise.
5. Electronic transmission of image data can introduce noise.
6. Interference in the transmission channel may also corrupt the image.

Statement of the problem

Spatial noise is described by the statistical behavior of the gray level values in the noise component of the degraded image. Noise can be modeled as a random variable with a specific probability distribution function. There is different type of spatial noise, for this assignment we are using Gaussian noise and Poisson noise.

Gaussian noise in digital image arise during image acquisition e.g. sensor noise caused by poor illumination or high temperature and electronic circuit noise and Poisson noises sometimes called shot noise arise from uncertain measurement of light having random fluctuation of photons.

Noise is introduced in the image at the time of image acquisition or transmission. Different factors may be responsible for introduction of noise in the image. The number of pixels corrupted in the image will decide the quantification of the noise. The principal sources of noise in the digital image are:

a) The imaging sensor may be affected by environmental conditions during image acquisition.
b) Insufficient Light levels and sensor temperature may introduce the noise in the image.
c) Interference in the transmission channel may also corrupt the image.

d) If dust particles are present on the scanner screen, they can also introduce noise in the image.

Noise Types

1. Salt and pepper noise

Salt and pepper noise is sometimes called impulse noise or spike noise or random noise or independent noise. In salt and pepper noise (sparse light and dark disturbances), pixels in the image are very different in color or intensity unlike their surrounding pixels. Salt and pepper degradation can be caused by sharp and sudden disturbance in the image signal. Generally this type of noise will only affect a small number of image pixels. When viewed, the image contains dark and white dots, hence the term salt and pepper noise. Typical sources include flecks of dust inside the camera and overheated or faulty (Charge-coupled device) CCD elements. An image containing salt-and-pepper noise will have dark pixels in bright regions and vice versa. This type of noise can be caused by dead pixels, it known as impulsive noise. It appearances is randomly scattered white or black pixel over the image. It sometimes happens for memory cell failure, for synchronization errors in image digitizing or transmission. This type of noise can be caused by analog to digital converter errors, bit error in transmission. Python implementation and result of adding Salt and pepper noise on an image:

In [49]:
```python
import numpy as np
import random
import cv2
from skimage.util import random_noise
from prettytable import PrettyTable
def noisy(noise_typ,path):
    if noise_typ == "s&p":
        temp = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        image = resize_img(temp, 450, 450)
        # Getting the dimensions of the image
        row, col = image.shape
        # Randomly pick some pixels in the image for coloring them white
        # Pick a random number between 500 and 10000
        number_of_pixels = random.randint(500, 10000)
        for i in range(number_of_pixels):
            # Pick a random x and y coordinate
            y_coord = random.randint(0, row - 1)
            x_coord = random.randint(0, col - 1)
            # Color that pixel to white
            image[y_coord][x_coord] = 255
        # Randomly pick some pixels in the image for coloring them black
        # Pick a random number between 500 and 10000
        number_of_pixels = random.randint(500, 10000)
        for i in range(number_of_pixels):

            # Pick a random y coordinate
            y_coord = random.randint(0, row - 1)

            # Pick a random x coordinate.
            x_coord = random.randint(0, col - 1)

            # Color that pixel to black
            image[y_coord][x_coord] = 0
        return image
def resize_img(image, width, height):
    resized_image = cv2.resize(image,(width,height))
    return resized_image
```

```
In [33]: ##### adding salt and pepper noise #####
original_img = cv2.imread("F:\\Icons\\pictures\\HDWALL\\second_test.jpg", cv2.IMREAD_GRAYSCALE)
original_img = resize_img(original_img, 450, 450)
image_with_noise = noisy('s&p',"F:\\Icons\\pictures\\HDWALL\\second_test.jpg")
noiseCompare = np.concatenate((original_img,image_with_noise),axis = 1)
cv2.imshow("Original image vs salt and pepper noise added image",noiseCompare)
cv2.waitKey(0)
cv2.destroyAllWindows()
########
```

2. Poisson Noise

The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted number of photons per unit time. These rays are injected in patient's body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuation of photons. Result gathered image has spatial and temporal randomness. This noise is also called as quantum (photon) noise or shot noise. This noise obeys the Poisson distribution and is given as:

$$P(f_{(pi)}) = k) = \frac{\lambda^k{}_i e^{-\lambda}}{k!}$$

Python implementation and result of adding Salt and pepper noise on an image:

In [49]:
```
import numpy as np
import random
import cv2
from skimage.util import random_noise
from prettytable import PrettyTable
def noisy(noise_typ,path):
    if noise_typ == "poisson":
        temp = cv2.imread(path)
        image = resize_img(temp,450,450)
        noise_mask = np.random.poisson(image).astype(np.uint8)
        noise_img = image + noise_mask
        return noise_img
def resize_img(image, width, height):
    resized_image = cv2.resize(image,(width,height))
    return resized_image
```

In [37]:
```
##### adding poisson noise #######
original_img2 = cv2.imread("F:\\Icons\\pictures\\HDWALL\\second_test.jpg")
original_img2 = resize_img(original_img2, 450, 450)
image_with_noise2 = noisy('poisson',"F:\\Icons\\pictures\\HDWALL\\second_test.jpg")
noiseCompare2 = np.concatenate((original_img2,image_with_noise2),axis = 1)
cv2.imshow("Original image vs poisson noise added image",noiseCompare2)
cv2.waitKey(0)
cv2.destroyAllWindows()
##########
```

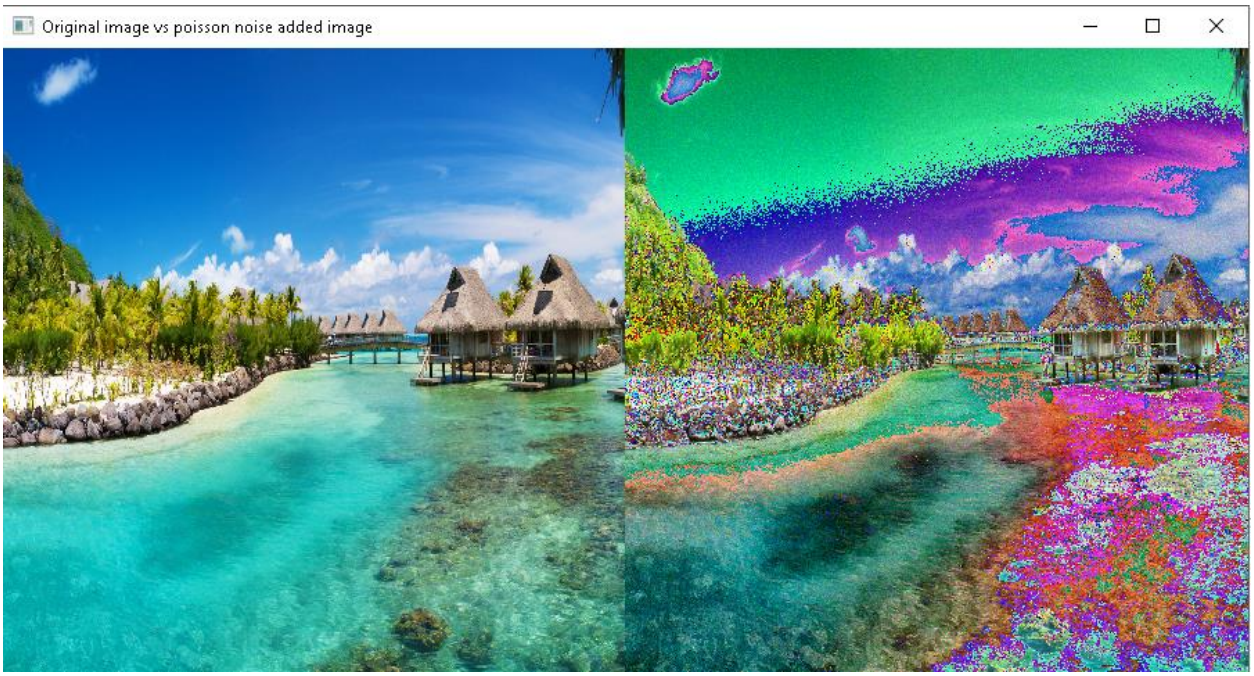


Image Filtering Algorithms

1. Mean filtering

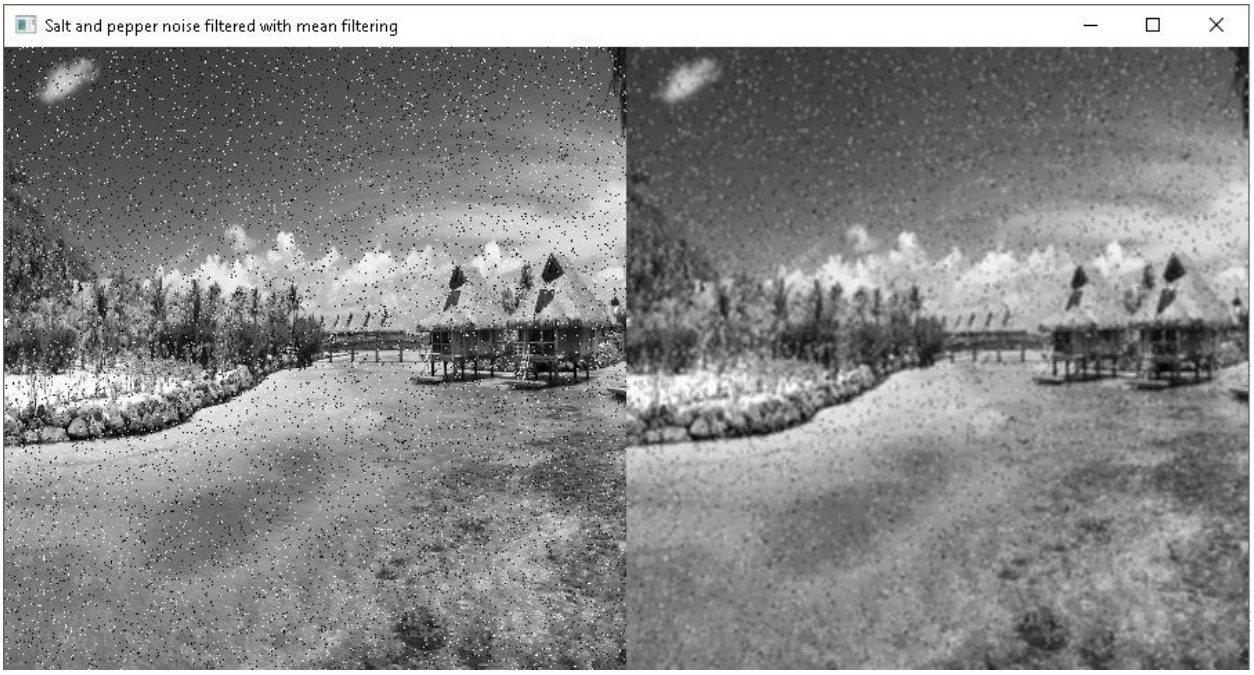
Average (or mean) filtering is a method of 'smoothing ‘images by reducing the amount of intensity variation between neighboring pixels.

This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element Mean filter is a linear filter algorithm. That means it increases a template on the target pixel, and then uses all the average of pixel value instead of the original pixel value. The template size and shape can often be based on the characteristics of images to be processed to determine size.

$$f(x, y) = \frac{1}{m\,n} \sum_{(s,t)\in S_{xy}} g(s, t)$$

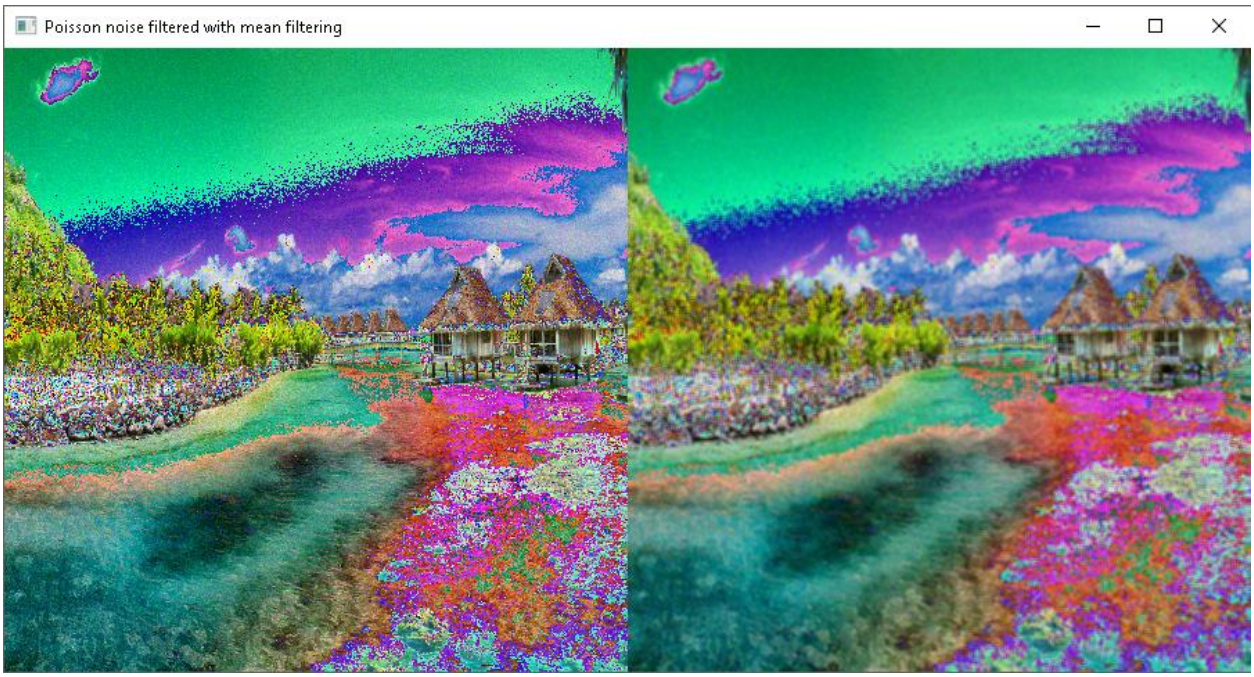
Python implementation Filtering Salt and pepper noise using mean filtering algorithm and result:

```
In [47]: #### mean filtering ###
         imgMeanBlur = cv2.blur(image_with_noise,(3,3))
         meanCompare = np.concatenate((image_with_noise, imgMeanBlur), axis=1)
         cv2.imshow("Salt and pepper noise filtered with mean filtering", meanCompare)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
         ##########
```



Python implementation Filtering Poisson noise using mean filtering algorithm and result:

```
In [54]: #### mean filtering ###
         imgMeanBlur2 = cv2.blur(image_with_noise2,(3,3))
         meanCompare2 = np.concatenate((image_with_noise2, imgMeanBlur2), axis=1)
         cv2.imshow("Poisson noise filtered with mean filtering", meanCompare2)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
         ##########
```

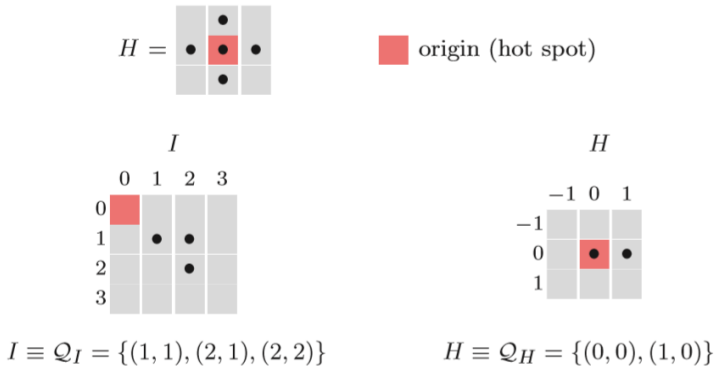


2. **Morphological Filter**

The idea of the morphological filter is shrinking and let grow process. The word “shrink” means using median filter to round off the large structures and to remove the small structures and in grow process, remaining structures are grow back by the same amount.

- ♣ Structuring element

In morphological filter, each element in the matrix is called “structuring element” instead of coefficient matrix in the linear filter. The structuring elements contain only value 0 and 1. And the hot spot of the filter is the dark shade element.



The binary image is described as sets of two-dimensional coordinate point. This is called "Point Set" Q and point set consist of the coordinate pair p = (u,v) of all foreground pixels. Some operations of point set are similar to the operation in others image. For inverting binary image is complement operation and combining two binary image use union operator. Shifting binary image I by some coordinate vector d by adding vector d to point p or reflection of binary image I by multiply -1 to point p.

- ♣ Dilation and Erosion:

Dilation: is a morphological operator which works for the grow process as we mentioned before. The equation of this operator is defined as

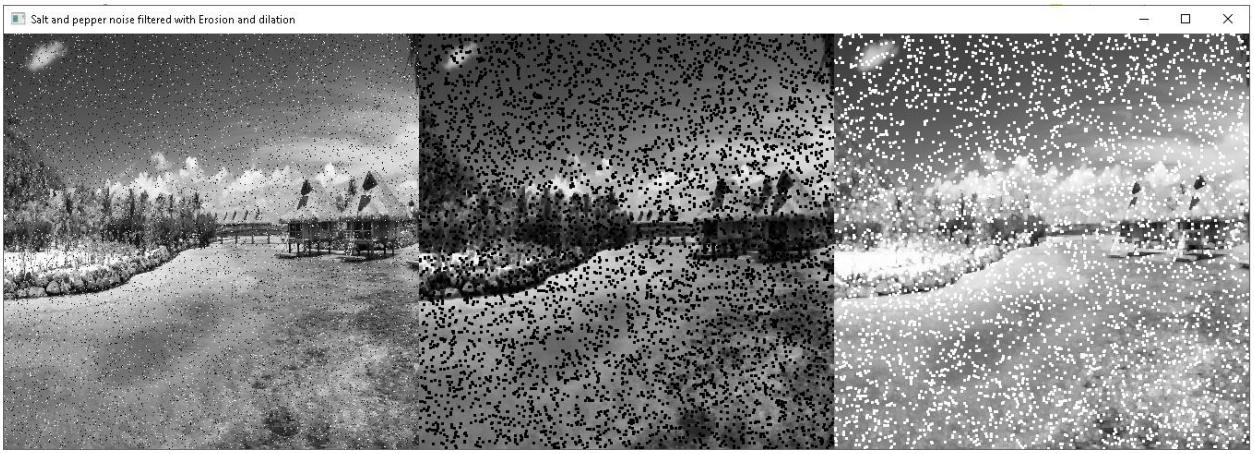
$$I \oplus H \equiv \{(\boldsymbol{p}+\boldsymbol{q}) \mid \text{for every } \boldsymbol{p} \in I, \boldsymbol{q} \in H\}.$$

Erosion: is a morphological operator which works for the shrink process as we mentioned before as well and the equation is defined as

$$I \ominus H \equiv \{\boldsymbol{p} \in \mathbb{Z}^2 \mid (\boldsymbol{p}+\boldsymbol{q}) \in I, \text{ for every } \boldsymbol{q} \in H\}.$$

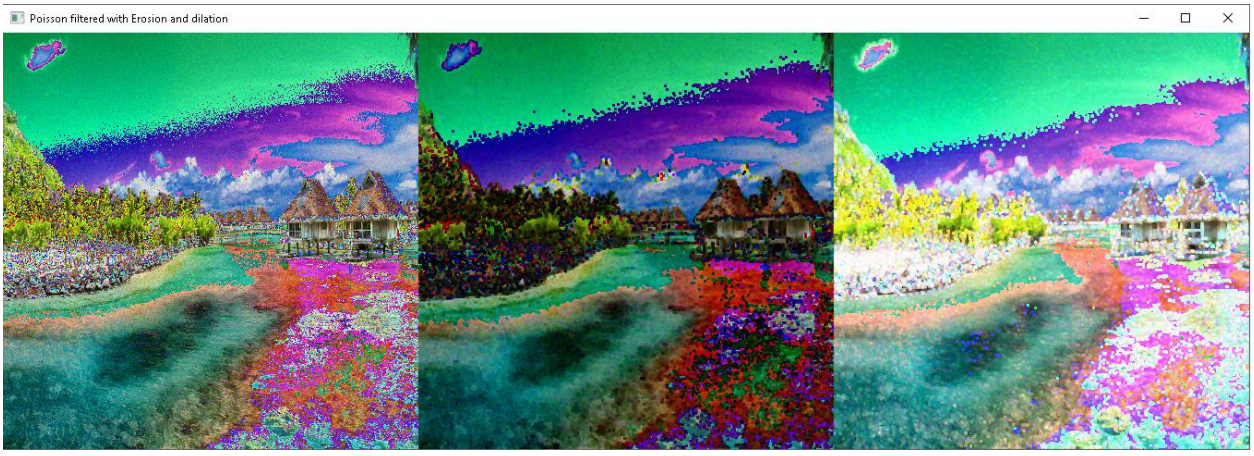
Python implementation Filtering Salt and pepper noise using Morphological analysis (Erosion and Dilation) filtering algorithm and result:

In [35]:
```
#### Morphological analysis (erosion and dilation) ###
kernel = np.ones((3,3),np.uint8)
imgDilate = cv2.dilate(image_with_noise,kernel,iterations=1)
imgErode = cv2.erode(image_with_noise,kernel,iterations=1)
morphCompare = np.concatenate(([image_with_noise, imgErode, imgDilate]), axis=1)
cv2.imshow("Salt and pepper noise filtered with Erosion and dilation", morphCompare)
cv2.waitKey(0)
cv2.destroyAllWindows()
####################################
```



Python implementation Filtering Poisson noise using Morphological analysis (Erosion and Dilation) and result:

```
In [40]: #### Morphological analysis (erosion and dilation) ###
         kernel2 = np.ones((3,3),np.uint8)
         imgDilate2 = cv2.dilate(image_with_noise2,kernel2,iterations=1)
         imgErode2 = cv2.erode(image_with_noise2,kernel2,iterations=1)
         morphCompare2 = np.concatenate((image_with_noise2, imgErode2, imgDilate2), axis=1)
         cv2.imshow("Poisson filtered with Erosion and dilation", morphCompare2)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
         ####################################
```

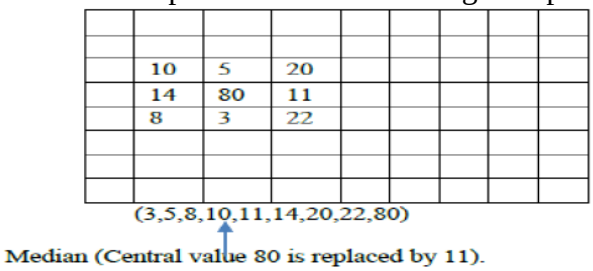


3. Median filter

Median filter is a filter which changes the pixel value with the median of neighborhood pixels. The major drawback of median filter is that the all pixels are substituted by the median of the window even if the pixel under concern is uncorrupted. This will worsen the overall visual quality of the image. In addition, the simple median filter fails to preserve the edges.

The algorithm for the median filter is as follows:

1. Select a two-dimensional window W of size 3*3. Assume that the pixel being processed is Cx,y.
2. Compute Wmed the median of the pixel values in window W.
3. Replace Cx,y by Wmed.
4. Repeat steps 1 to 3 until all the pixels in the entire image are processed.

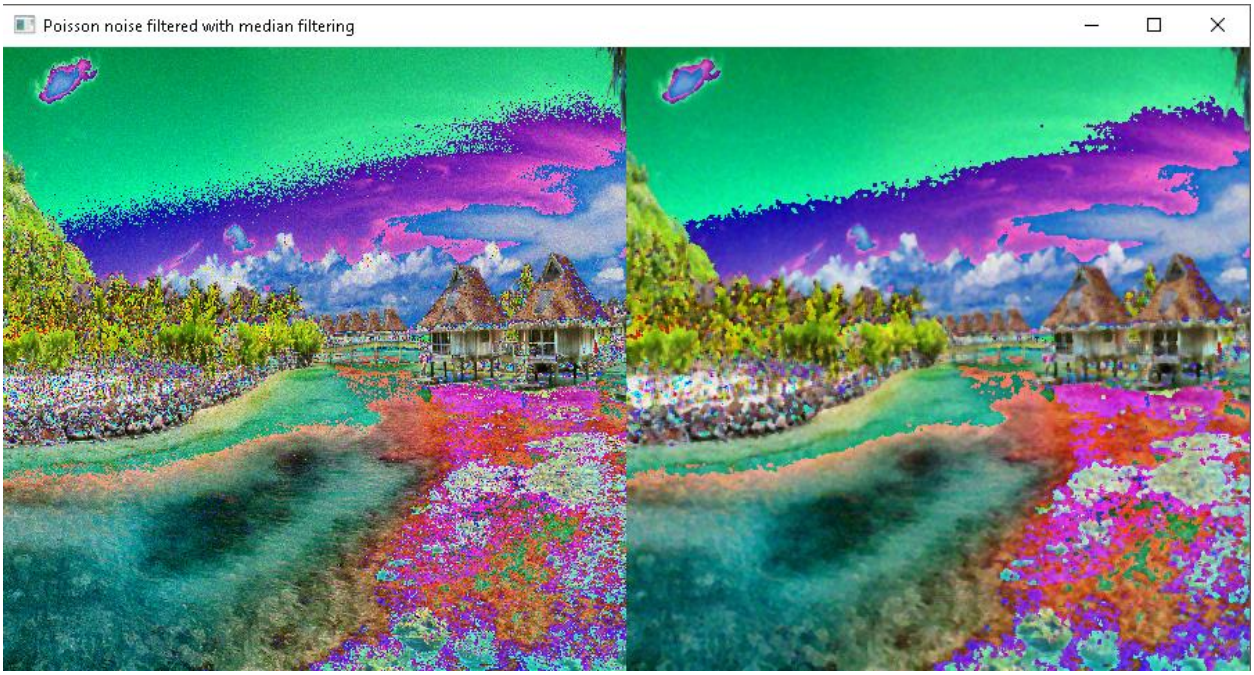


Python implementation Filtering Salt and pepper noise using median filtering algorithm and result:

```
In [56]: #### median filtering ###
         imgMedBlur= cv2.medianBlur(image_with_noise, 3)
         medianCompare = np.concatenate((image_with_noise, imgMedBlur), axis=1)
         cv2.imshow("Salt and pepper noise filtered with median filtering", medianCompare)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
         ####################################
```

Python implementation Filtering Poisson noise using median filtering algorithm and result:

```
In [39]: #### median filtering ###
         imgMedBlur2 = cv2.medianBlur(image_with_noise2, 3)
         medianCompare2 = np.concatenate((image_with_noise2, imgMedBlur2), axis=1)
         cv2.imshow("Poisson noise filtered with median filtering", medianCompare2)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
         ###################################
```



Experimental Results

We used two objective measurement tools MSE (Mean Square Error) and PSNR (Peak signal-to-noise ratio) to propose the best one from the above filtering we have been used so far for each type of noises. The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error.

- ♣ MSE and PSNR for Salt and pepper noise:

In [42]:
```
##### MSE and PSNR for salt and pepper noise ##########
from skimage.metrics import mean_squared_error

sandpTable = PrettyTable(["S&P noise", "MSE", "PSNR"])
sandpTable.add_row([
    "mean filter",
    mean_squared_error(original_img,imgMeanBlur),
    peak_signal_noise_ratio(original_img, imgMeanBlur)
])
sandpTable.add_row([
    "morphological filter (erode)",
    mean_squared_error(original_img, imgErode),
    peak_signal_noise_ratio(original_img, imgErode)
])
sandpTable.add_row([
    "morphological filter (dilate)",
    mean_squared_error(original_img, imgDilate),
    peak_signal_noise_ratio(original_img, imgDilate)
])
sandpTable.add_row([
    "median filter",
    mean_squared_error(original_img, imgMedBlur),
    peak_signal_noise_ratio(original_img, imgMedBlur)
])
print(sandpTable)
```

```
+------------------------------+--------------------+--------------------+
|           S&P noise          |        MSE         |        PSNR        |
+------------------------------+--------------------+--------------------+
|          mean filter         | 306.2088049382716  | 23.270626863243283 |
|  morphological filter (erode) | 6216.698335802469  | 10.195205670521062 |
| morphological filter (dilate) | 2171.913037037037  | 14.76237928667048  |
|         median filter        | 184.90006419753087 |  25.4614329892113  |
+------------------------------+--------------------+--------------------+
```

- ♣ MSE and PSNR for Poisson noise :

In [43]:
```
##### MSE and PSNR for poisson noise ##########
from skimage.metrics import peak_signal_noise_ratio
poissonTable = PrettyTable(["Poisson noise", "MSE", "PSNR"])
poissonTable.add_row([
    "mean filter",
    mean_squared_error(original_img2,imgMeanBlur2),
    peak_signal_noise_ratio(original_img2, imgMeanBlur2)
])
poissonTable.add_row([
    "morphological filter (erode)",
    mean_squared_error(original_img2, imgErode2),
    peak_signal_noise_ratio(original_img2, imgErode2)
])
poissonTable.add_row([
    "morphological filter (dilate)",
    mean_squared_error(original_img2, imgDilate2),
    peak_signal_noise_ratio(original_img2, imgDilate2)
])
poissonTable.add_row([
    "median filter",
    mean_squared_error(original_img2, imgMedBlur2),
    peak_signal_noise_ratio(original_img2, imgMedBlur2)
])
print(poissonTable)
```

Poisson noise	MSE	PSNR
mean filter	4463.386228806585	11.6341589173254
morphological filter (erode)	8240.163586831275	8.971445273075755
morphological filter (dilate)	6960.046610699588	9.704682128231749
median filter	5105.623275720164	11.050315941348593

Conclusion

We have introduced three filtering algorithm such as median filtering, Mean filtering and Morphological filtering for removing Poisson and salt and pepper noise from images. Depending on the MSE and PSNR values we can propose the best filtering algorithm.

In the case of MSE, as the value of MSE decreases, the lower the error between filtered and original image. In the case of PSNR, The higher the PSNR value, the better the quality of filtered, or reconstructed image.

Therefore, depending on the results we can conclude that **Median filtering** is the best filtering algorithm among others for removing salt and pepper noise since, it has lowest MSE and highest PSNR value and **Mean filtering** is better for removing Poisson noise meanwhile, it has lowest MSE and highest PSNR value

Reference

[1]. Luisier, F., Blu, T. and Unser, M. (2011), Image denoising in mixed Poisson-Gaussian noise, IEEE Trans. Image Process., Vol. 20, No. 3, pp. 696–708

[2]. Kamboj P. & Rani V., (2013) A Brief study of various noise models and filtering techniques, Journal of Global Research in Computer Science, Vol. 4, No.

[3]. Ajay, K, and Brijendra. (2015), Noise Models in Digital Image Processing.

[4]. Dominic, A etl., (2018), Measuring the Performance of Image Contrast Enhancement Technique, Vol. 118.

[5]. Kalpana, C & Mrs. Nidhi, S, (2015), Performance Evaluation and Comparison of Different Noise, apply on PNG Image Format used in Deconvolution Wiener filter (FFT) Algorithm, Vol.4

[6] J. W. Tukey, 'Nonlinear (Nonsuperposable)', Methods for Smoothing Data. Conference Record EASCON, pp. 673-685, 1974

Name	Group Member Contribution
Tamiresilassie Tilahun	Document preparation ❖ Median Filtering Experiment of algorithms ❖ Python experiment on the above filtering algorithms Conclusion
Nigatu Agize	Document preparation ❖ Mean Filtering Experiment of algorithms ❖ Python experiment on the above filtering algorithms Conclusion
Michael Siyoum	Document preparation ❖ Morphological Filtering Experiment of algorithms ❖ Python experiment on the above filtering algorithms Conclusion