Marco Galindo ID: 310540013
2021, November 2nd Image Processing

Programming Assignment #1

It is common knowledge that digital images are currently a vital part of most people's lives, they capture moments in a way that was not possible not so long ago. However, sometimes these images get taken under less-than-ideal conditions, and bad lighting, digital noise and bad cameras can often lead to images with alterations in their intended form. The purpose of this assignment is to explore and implement digital enhancement and restoration techniques that can improve an image that is noisy or has lightning problems to obtain a cleaner and better image.

The algorithms used in this assignment use a range of techniques, some use one-to-one-pixel functions and some other use convolutional operations that modify a pixel based on their surrounding neighbors.

Objectives

The main objectives of this assignments are the following:

- Implement digital image improvement and enhancement algorithms discussed during class.
- Implement improvements to these methods to obtain even better results than their corresponding original algorithm.
- Learn the difference of implementing methods for different images represented in different color models.

Review of the methods used

- Histogram equalization

In order to improve contrast on images, one popular method is histogram equalization. The method represents the intensity distribution of an image through a histogram, and then it equalizes these values to be uniformly distributed. An image that is too light, too dark, has low-contrast or has high-contrast can be easily identified by looking at its histogram, as shown in Figure 1. The aim of the method is to ensure a uniform cumulative sum of the intensity histogram, so that the resulting image has a high-dynamic range and can correctly show really dark and really bright spots.

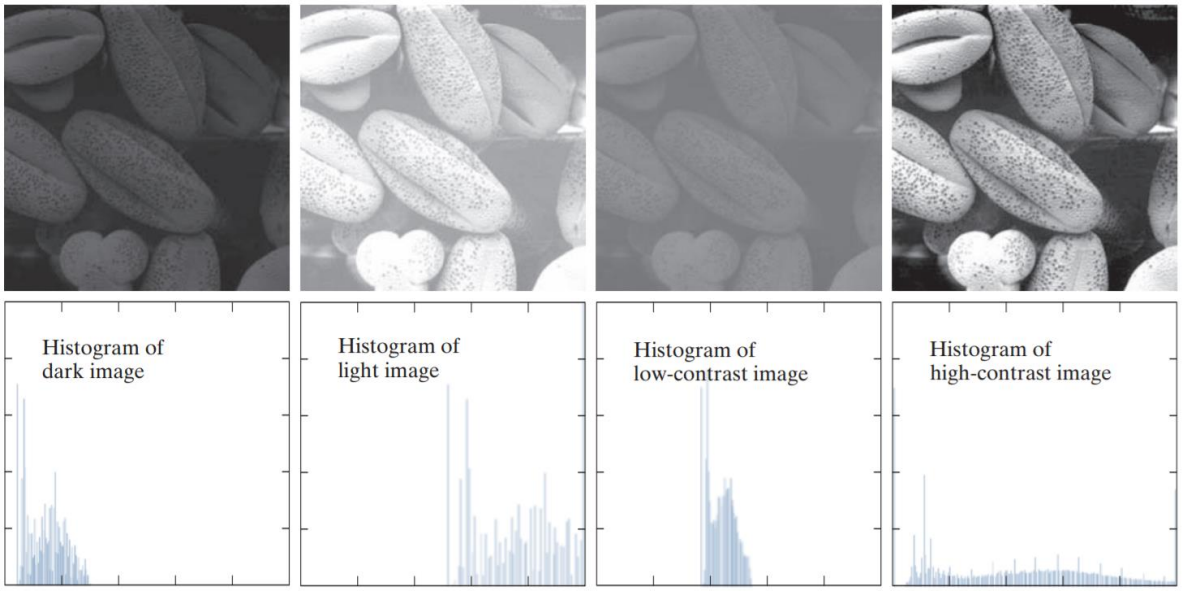

Figure 1 From left to right: Histogram corresponding to a (1) Dark image (2) Light image (3) Low-contrast image (4) High-contrast image

- Order-statistics filters for impulse noise - Median filters

Order-statistics filters define the new value of a pixel based on not just the value of the pixel itself but also considering the values of neighbor pixels. The most used order-statistics filter is the median filter, that defines an MxM filter where N is usually an odd number, where the new value of the pixel is replaced by the median value of the intensity values in its neighborhood. These filters are particularly effective against salt-and-pepper noise where white and black dots appear on an image.

An improvement to this algorithm is the implementation of adaptive median filters, where the median value is first diagnosed to check whether it is a product of noise itself or a non-noisy pixel that is fit to replace other pixels in the kernel of the convolution operation.

- Sharpening spatial filters - Laplacian filter

The intuition behind sharpening spatial filters is to highlight transitions in intensity values. This approach consists of first defining a kernel that can be applied using convolution with the original image to result in a sharpened output. The Laplacian has been shown to be the best operator for this method, being defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Using the definition of a second derivative:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

It can be proven that a convolution kernel in the form of Figure 2(a) can be used to represent the formula. However, it can also include diagonal values into consideration as shown in Figure 2(b), adjusting the center value to be double by taking into consideration the extra pair of $-2f(x)$ each diagonal derivative would add to the kernel.

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1

Figure 2 Laplacian kernels. (a) 4-neighbor kernel (b) 8-neighbor kernel

The result of the convolution operation would a representation of the edges that need to be intensified in the original image, thus, to obtain a result, we need to do the following operation:

$$g(x,y) = f(x,y) - \nabla^2 f$$

Where $g(x,y)$ and $f(x,y)$ are the input and sharpened images, respectively.

- Power-law transformations

Power-law transformations, also called Gamma corrections, have the form:

$$s = cr^{\gamma}$$

Usually, it is assumed that $c = 1$ and r and s are rescaled to be within $[0,1]$. The result of such transformations is shown in Figure 3 for different values of γ. Essentially, the output image for values of $\gamma < 1$ has its darker pixels mapped to brighter values while leaving already bright pixels like their original values. For values of $\gamma > 1$ the effect is the opposite, where bright pixels are mapped to darker values. For the case of $\gamma = 1$, the equation reduces to an identity equation, leaving the image untouched.

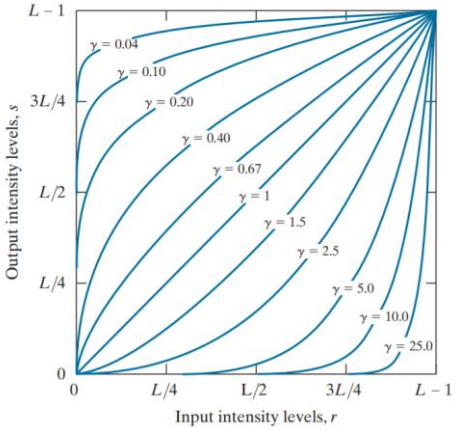

Figure 3 Gamma transformations for different γ values

- Gaussian smoothing filter

The Gaussian smoothing filter is a 2-D convolutional filter which purpose is to blur the original image to remove details and noise. The resulting value of each pixel will be defined by a weighted sum of pixels in its neighborhood. The resulting formula can be defined as:

$$g(x,y) = \sum_i \sum_j \frac{1}{2\pi\sigma^2} e^{-\frac{x_i^2 + y_j^2}{2\sigma^2}}$$

In theory, the Gaussian function is non-zero for every pixel, but in practice, a kernel three times the size of σ is sufficient, due to the small value of the function. The effects of the Gaussian smoothing filter are defined by the standard deviation of the function.

Explanation of the experiments done

- Histogram equalization

This method was applied to two different images with contrast problems, the first one is a color image and the second one is a black and white image. The color image is enhanced by modifying the values in its intensity channel, while pixels values are directly modified in the black and white image. The original and equalized histograms for the first image can be observed in the Figure 4. A comparation between the original and improved image can be seen in the Figure 5.

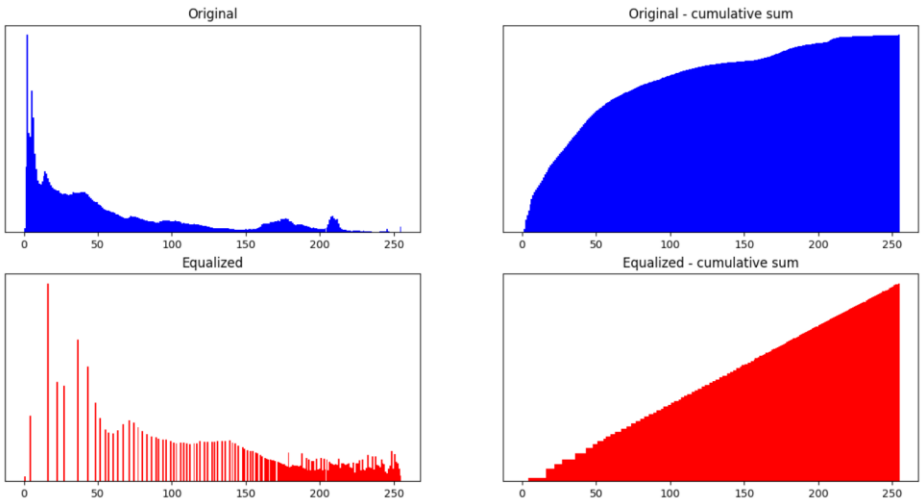

Figure 4 Top row: original image's intensity histogram and cumulative sum. Bottom row: Enhanced intensity histogram and cumulative sum.

Figure 5 Intensity histogram equalization for a color image

The second image enhancement can be seen in the Figure 6.

Figure 6 Intensity histogram equalization for a black and white image

- Median filters

To evaluate the effectiveness of the median filters an image that had impulse noise was selected. The median filters show excellent results in deleting impulse noise, as seen in Figure 7. In Figure 7(b) we can see the result of applying a 3x3 median filter on the image, the result was improved in Figure 7(c), where an adaptive median 3x3 filter was applied, helping the image to retain more detail where noise was lower. As a result of the median filter, there is a loss of detail around the edges of the objects, to counter this loss, a Laplace sharpening filter was applied to the image to obtain Figure 7(d).

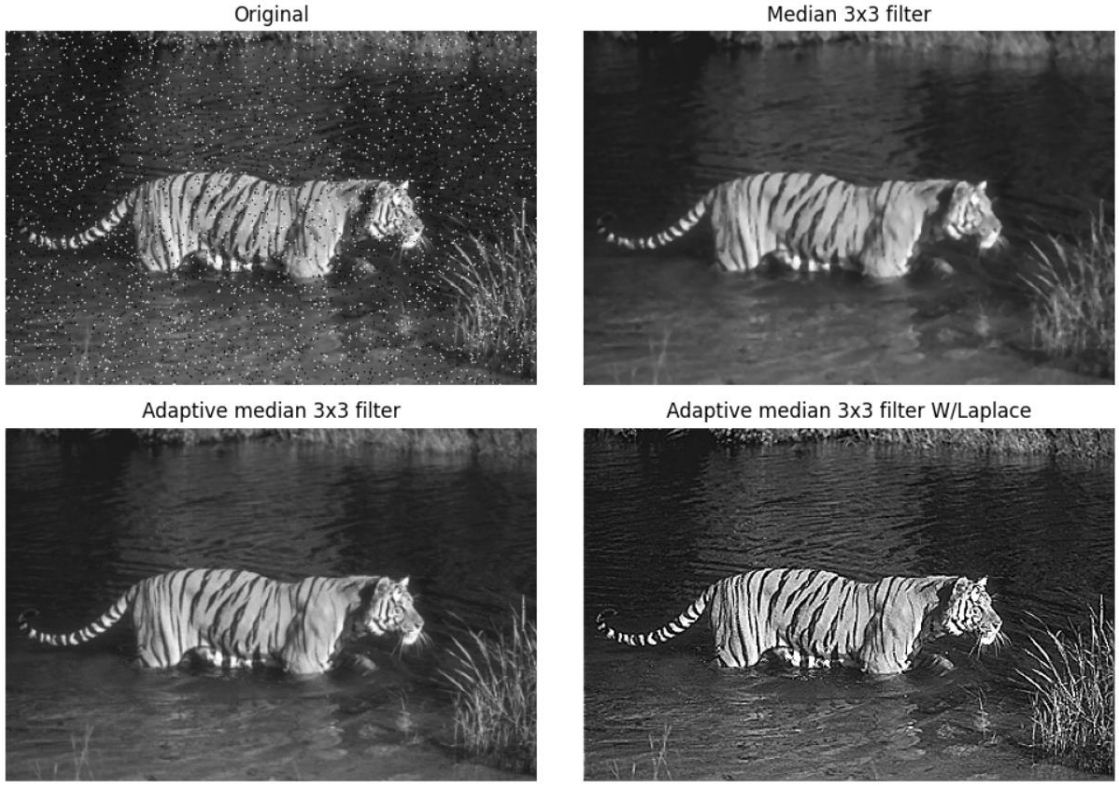

Figure 7 Median filter results on a noisy image. (a) Original image (b) Result of a 3x3 median filter (c) Result of a 3x3 filter with an adaptive median method (d) Result of applying sharpening to an adaptive 3x3 median filter

Following an algorithm proposed by Arumugam Rajamani et. Al, it was possible to improve an image containing impulse noise on its three-color channels with little distortion. This method also uses a median filter to eliminate noise and treats each channel independently. The original and denoised image can be seen in Figure 8. For these experiments, a lot of images were enhanced, but some presented too much noise to be effectively enhanced, the presented result is the best application of the algorithm.

Figure 8 Color image with salt and pepper noise

- Laplacian filter

A sharpening Laplacian filter was applied on a to enhance a coins' image as seen on Figure 9. Two different configurations of the Laplacian filter were used, the first had the form of Figure 2(a) and the second one was configured using the Figure 2 (b) model. In this case, because the photo was very blurry, the result of including diagonal information in the filter yielded a better result, as seen on the Figure 9(d), with a more defined mask in Figure 9(e).

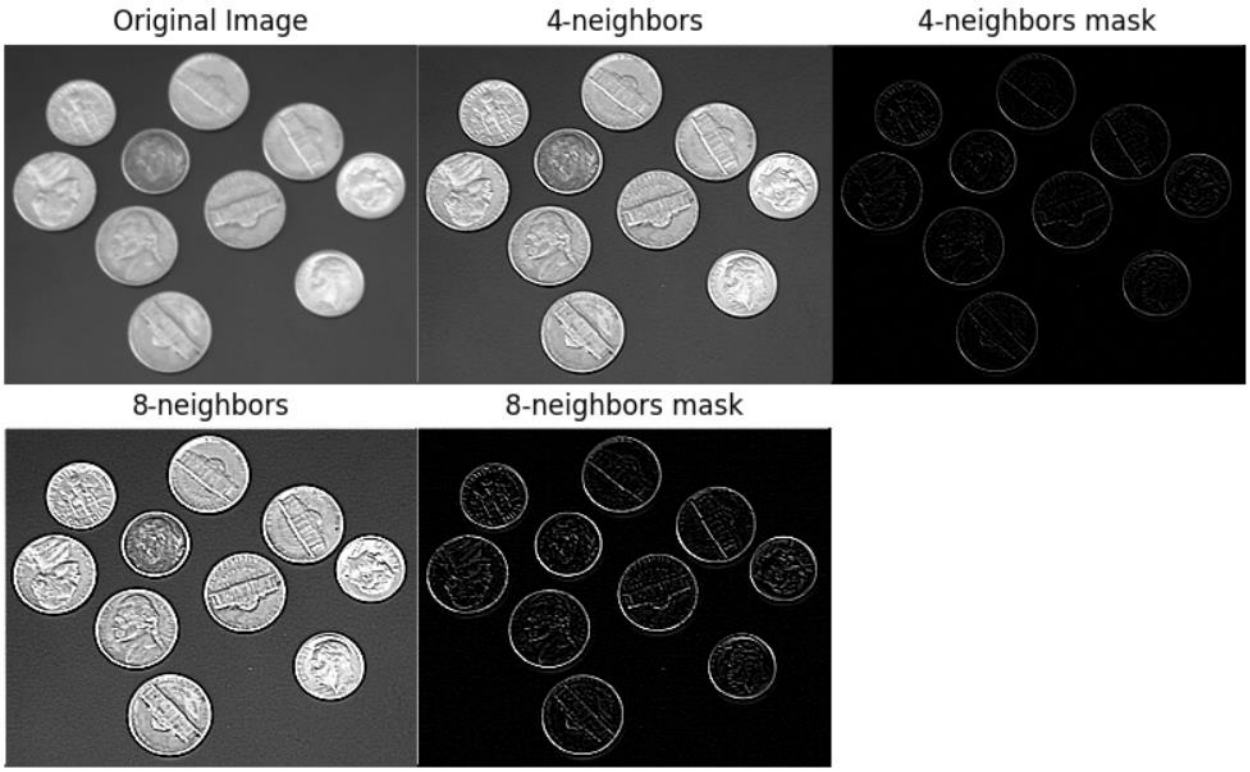

Figure 9 Top row from left to right: (a) Original image. (b) Sharpened output using a 4-neighbors Laplacian filter. (c) Mask from the 4-neighbors filter. Bottom row: (d) Sharpened image using an 8-neighbors Laplacian filter. (e) Mask from the 8-neighbors filter.

- Gamma corrections

The Gamma transformation method was implemented to handle color images in the HSV or RGB color model. For HSV images, the transformation is applied to the intensity channel, whereas for RGB images, the transformation is applied to only a single-color channel. In the example shown in Figure 10(b), gamma correction with $\gamma = 0.7$ was applied on the red channel to balance the intensity of every channel. In Figure 10(c), a histogram equalization was applied on the previous result to further enhance the image.

Figure 10 From left to right. (a) Original image (b) Gamma correction on red channel. (c) Gamma correction on red channel and histogram equalization

In Figure 11 we can see the effects of applying gamma correction with $\gamma = 0.5$ on the intensity channel of an HSV-represented image. It can be noted that already well-lit parts, such as the forehead, are not over-exposed, while unseen details of the cheek are highlighted in the enhanced image. For this image different values of γ were used.

Figure 11 Gamma correction on the intensity channel of an HSV image

- Gaussian smoothing filter

The Gaussian smoothing filter helps remove noise from an image by blurring it, an image with heavy noise was tested for this method. The enhanced image can be seen in Figure 12, where a 7x7 Gaussian smoothing filter was used. The result can be seen as less noisy, but the cost is the sharpness of the details. For this experiment, different filter sizes and intensities were applied.

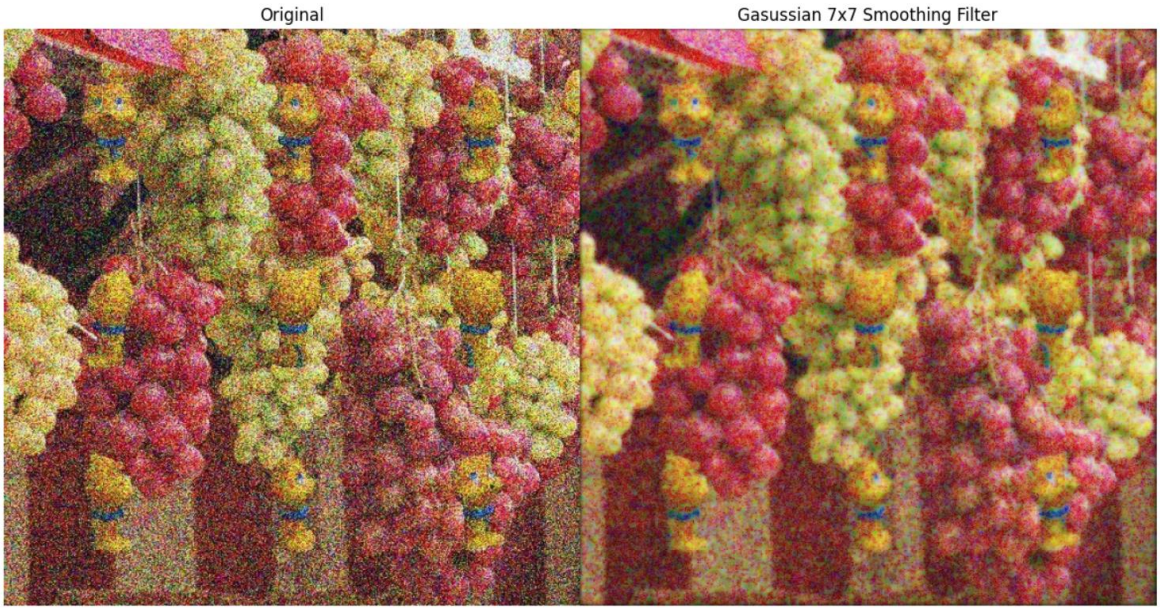

Figure 12 Gaussian smoothing filter

Discussions

The results obtained for all the experiments satisfactorily enhance digital images, each method in a different way. However, it is important to be aware of the limitations of each method, as it is usually a good idea to combine different techniques on one single image to obtain a good resulting output. Despite one algorithm not being enough to enhance an image to its fullest, it is good to know that these algorithms keep improving over time, just like every technique out there, to further enhance images; for example, the very popular histogram equalization method now has more complex implementations that deliver better results, namely the AHE and CLAHE.

One disadvantage of most of the methods implemented in this work is that they have parameters that need to be adjusted and change depending on the initial image, so one set of parameters will not work for every image even if the underlying algorithm is the same. Moreover, there are some images that cannot be recovered due to the severe noise or degradation they present.

It should be noted that for these experiments, there were a lot of color model conversions, modifying an image's representation from RGB to HSV color models and the other way around. These conversions make some enhancements possible and simpler, where modifying an image's intensity channel is more convenient than individually modifying each color channel independently.

In summary, all the techniques implemented become part of a toolbox, where they need can be used to enhance an image, by combining and tuning them to fit the specific task at hand.

References

- Sujaya Kumar Sathua, Arabinda Dash, & Aishwaryarani Behera. (2017). Removal of Salt and Pepper noise from Gray-Scale and Color Images: An Adaptive Approach.
- Arumugham, R., Vellingiri, K., Habeebrakuman, W., & Mohan, K. (2012). A new denoising approach for the removal of impulse noise from color images and video sequences. Image Analysis & Stereology, 31, 185-191.
- Abhisek Jana, Applying Gaussian Smoothing to an Image using Python from scratch, 2019. http://www.adeveloperdiary.com/data-science/computer-vision/applying-gaussian-smoothing-to-an-image-using-python-from-scratch/ (Accessed October 2021).
- Maria Sagrario Millan Garcia-Verela, & Edison Valencia (2004). Laplacian filter based on color difference for image enhancement. In *5th Iberoamerican Meeting on Optics and 8th Latin American Meeting on Optics, Lasers, and Their Applications* (pp. 1259 – 1264). SPIE.

Code

```
import matplotlib.pyplot as plt
import numpy as np
import math
import cv2
import os

############ Methods for image enhancement ############
def calculate_histogram(array, number_of_bins):
    """Takes an array and a number of bins and returns the Probability Density Function of the histogram"""
    min_val = np.amin(array)
    max_val = np.amax(array)
    interval = (max_val - min_val) / number_of_bins
    bins = np.linspace(min_val, max_val, number_of_bins + 1)
    frequencies = np.zeros(number_of_bins)
    correspoding_bins = np.digitize(array, bins)

    # Increments the corresponding bin's value by one
    for resulting_bin in correspoding_bins:
    if resulting_bin == number_of_bins + 1:
    frequencies[-1] += 1
    else:
    frequencies[resulting_bin - 1] += 1

    pdf = frequencies/interval/frequencies.sum() # Probability Density Function

    return pdf, bins


def cumulative_sum(array):
    """Calculates the cumulative sum of the array"""
    cumsum = np.zeros(array.shape)
    cumsum[0] = array[0]
    for i in range(1, array.shape[0]):
    cumsum[i] = array[i] + cumsum[i - 1]
```

```
    return cumsum


def histogram_eq(image):
    """ Take an Numpy HSV image array and performs histogram equalization.
    The image is converted to HSV to change the Value channel but already returned as RGB for easy display
    """
    image_array = np.copy(image)
    total_width, total_height, _ = image_array.shape
    X_STEP = 10
    Y_STEP = 10

    intensity_channel = image_array[:,:,2]
    histogram, bins = calculate_histogram(intensity_channel.flatten(), 256)

    original_hist = intensity_channel.flatten()

    new_values = cumulative_sum(histogram) # cumulative distribution function
    new_values = new_values / new_values[-1] # normalize all new values to be from [0,1]

    # convert old values into the new equalized values
    intensity_equalized = np.interp(intensity_channel.flatten(), bins[:-1], 255 * new_values)

    # Update the image's value channel with the equalized one
    image_array[:,:,2] = intensity_equalized.reshape(intensity_channel.shape)


    equalized_hist = intensity_channel.flatten()
    return cv2.cvtColor(image_array, cv2.COLOR_HSV2RGB), original_hist, equalized_hist

def is_pixel_corrupted(pixel):
    """Function that determines if a pixel is corrupted or not"""
    return pixel == 0 or pixel == 255

def salt_pepper_denoise(image, color = False, adaptive = False, filter_size=3):
    """Remove salt & pepper noise by using a median filter."""
```

```
image_array = np.array(image)
if color:
# Implementation algorithm by Arumugam Rajamani et. Al
image_array = np.pad(image_array, pad_width=((1,1),(1,1),(0,0)), mode='constant')
X_STEP = filter_size
Y_STEP = filter_size
for channel in range(3): # For all three colors
for Xo in range(image_array.shape[0] - 2):
for Yo in range(image_array.shape[1] - 2):
Xf = Xo + X_STEP
Yf = Yo + Y_STEP
region = image_array[Xo:Xf,Yo:Yf,channel] # Window region
sorted_diag = np.sort(np.array([region[0,0],region[1,1],region[2,2]])) # Sorted diagonal values
clear_pixels = np.array([(sorted_diag>0) & (sorted_diag<255)]).reshape(3) # Values that don't have noise

# If none of the pixels have noise, there's nothing to do, continue with the next window
if np.all(clear_pixels):
continue

# Choose a healthy pixel from the sorted pixels, starting from the median
replacement = None
if clear_pixels[1]:
replacement = sorted_diag[1]
elif clear_pixels[0]:
replacement = sorted_diag[0]
elif clear_pixels[2]:
replacement = sorted_diag[2]

# If all three pixels from the diagonal are corrupted, just replace the the upper-left pixel
# from the window with a non-noisy 4-neighborhood pixel.
if replacement is None and is_pixel_corrupted(image_array[Xo,Yo,channel]):
if (not is_pixel_corrupted(image_array[Xo+1,Yo, channel])):
image_array[Xo,Yo,channel] = image_array[Xo+1, Yo, channel]
elif (Xo > 0 and not is_pixel_corrupted(image_array[Xo-1,Yo,channel])):
image_array[Xo,Yo,channel] = image_array[Xo-1, Yo, channel]
elif (Yo > 0 and not is_pixel_corrupted(image_array[Xo,Yo-1,channel])):
```

```
    image_array[Xo,Yo,channel] = image_array[Xo, Yo-1, channel]
    else:
    image_array[Xo,Yo,channel] = image_array[Xo, Yo+1, channel]
    elif replacement is not None:
    # If there is a non-noisy pixel in the diagonal, replace other noisy values with this one
    for i in range(3):
    if is_pixel_corrupted(region[i,i]):
    image_array[Xo+i,Yo+i,channel] = replacement

    return image_array
    else:
    # For B&W images
    image_array = cv2.cvtColor(image_array, cv2.COLOR_BGR2GRAY)
    resulting_array = np.copy(image_array)
    if adaptive:
    # Use an Adaptive Median Filter
    image_array = np.pad(image_array, 1, mode='constant')
    height, width = resulting_array.shape

    for Xo in range(height):
    for Yo in range(width):
    filter_size = 3
    while True:
    X_STEP = filter_size
    Y_STEP = filter_size
    Xf = Xo + X_STEP
    Yf = Yo + Y_STEP
    region = image_array[Xo:Xf,Yo:Yf] # Window size
    Zxy = resulting_array[Xo, Yo]
    Zmed = np.median(region)
    if Zmed == 0 or Zmed == 255:
    # If the Zmed is an extreme value, increase window and continue the cycle or leave the original Zxy
    if Xo-1 > 0 and Yo-1 > 0 and Xf+1 < resulting_array.shape[0] and Yf+1 < resulting_array.shape[1]:
    filter_size += 2
    else:
    # Leave the original value
```

```
    break
    else:
    if Zxy == 0 or Zxy == 255:
    # If the Zmed is not corrupted and Zxy is corrupted replace the value
    resulting_array[Xo, Yo] = Zmed
    break
    else:
    # Is Zxy is not corrupted leave it
    break
    else:
    image_array = np.pad(image_array, 1, mode='constant')
    X_STEP = filter_size
    Y_STEP = filter_size
    height, width = resulting_array.shape
    for Xo in range(height):
    for Yo in range(width):
    Xf = Xo + X_STEP
    Yf = Yo + Y_STEP
    region = image_array[Xo:Xf,Yo:Yf]

    # Replace the value with the median of the filter
    resulting_array[Xo, Yo] = np.median(region)

    return resulting_array

def laplacian(image, filter_config, filter_size = 3, color = False):
    if color:
    # Process BGR image
    image_array = np.array(image)
    sharpened_image = np.copy(image_array)
    image_array = np.pad(image_array, pad_width=((1,1),(1,1),(0,0)), mode='constant')
    resulting_mask = np.zeros(sharpened_image.shape)
    for channel in range(2): # For all three color channels
    resulting_channel = np.zeros((image_array.shape[0]-2,image_array.shape[1]-2))
    X_STEP = filter_size
    Y_STEP = filter_size
```

```
    height, width = resulting_channel.shape

    # The filter config contains the matrix definition for the filter
    laplacian_filter = -1 * np.array(filter_config).reshape((filter_size,filter_size))
    for Xo in range(height):
    for Yo in range(width):
    Xf = Xo + X_STEP
    Yf = Yo + Y_STEP
    if Yf > width:
    continue
    if Xf > height:
    continue
    region = image_array[Xo:Xf,Yo:Yf,2]

    # Apply the filter
    resulting_channel[Xo,Yo] = np.sum(np.multiply(region, laplacian_filter))

    sharpened_image[:,:,channel] = image_array[1:-1,1:-1,channel] + resulting_channel
    sharpened_image[:,:,channel] = sharpened_image[:,:,channel] / np.max(sharpened_image[:,:,channel]) * 255
    resulting_mask[:,:,channel] = resulting_channel
    # Return the sharpenend image and the laplacian mask used
    return cv2.cvtColor(sharpened_image.astype('uint8'), cv2.COLOR_BGR2RGB), cv2.cvtColor(resulting_mask.astype('uint8'),
cv2.COLOR_BGR2RGB)
    else:
    image_array = np.array(image)
    image_array = cv2.cvtColor(image_array, cv2.COLOR_BGR2GRAY)
    resulting_array = np.zeros(image_array.shape)
    image_array = np.pad(image_array, 1, mode='constant')
    X_STEP = filter_size
    Y_STEP = filter_size
    height, width = resulting_array.shape

    # The filter config contains the matrix definition for the filter
    laplacian_filter = -1 * np.array(filter_config).reshape((filter_size,filter_size))
    for Xo in range(height):
    for Yo in range(width):
```

```
    Xf = Xo + X_STEP
    Yf = Yo + Y_STEP
    if Yf > width:
    continue
    if Xf > height:
    continue
    region = image_array[Xo:Xf,Yo:Yf]

    # Apply the filter
    resulting_array[Xo,Yo] = np.sum(np.multiply(region, laplacian_filter))

    sharpened_image = (image_array[1:-1,1:-1] + resulting_array)

    # Return the sharpenend image and the laplacian mask used
    return sharpened_image, resulting_array

def check_gamma_values(image):
    """ Shows an histogram of the three color channels of an image,
    to help decide if gamma correction would work or not.
    """
    plt.style.use('seaborn-deep')
    plt.hist([image[:,:,0].flatten(),image[:,:,1].flatten(),image[:,:,2].flatten()],
    bins=256, range=(0, 255), alpha=0.5, label=['red','green','blue'], color=['red','green','blue'])
    plt.legend(loc='upper right')
    plt.show()

def gamma_correction(image, gamma=1, c=1, channel=2):
    """ Takes an image and applies gamma correction to it. Can be applied to the value channel on
    an HSV image or on a single color channel on an RGB image.
    """
    image_array = np.array(image)
    channel_to_modify = image_array[:,:,channel] / 255

    # do gamma correction on value channel
    val_gamma = c * channel_to_modify ** gamma
    image_array[:,:,channel] = val_gamma * 255
```

```
    return image_array


def get_kernel(size, sigma):
    """ Get the filter with the Gaussian formula applied to its original values. """

    # Creating a vector of the desired size and evenly spaced
    kernel = np.linspace(-(size // 2), size // 2, size)

    # Calculate the gaussian for each vector element
    for i in range(size):
    kernel[i] = 1 / (np.sqrt(2 * np.pi) * sigma) * np.e ** (-np.power((kernel[i]) / sigma, 2) / 2)

    # Transform the vector into a matrix, to use in in the convolution process
    kernel = np.outer(kernel.T, kernel.T)

    # Normalizing the kernel
    kernel *= 1.0 / kernel.max()
    return kernel


def gaussian_blur(image, filter_size, color):
    """ Perform Gaussian Blur on an image. """
    kernel = get_kernel(filter_size, math.sqrt(filter_size))
    image_array = np.array(cv2.cvtColor(image, cv2.COLOR_BGR2HSV))
    if color:
    # For color images, perform the process on the value channel of an HSV image
    height, width, _ = image_array.shape
    X_STEP, Y_STEP = kernel.shape

    resulting_array = np.zeros(image_array.shape)
    resulting_array[:,:,0] = image_array[:,:,0]
    resulting_array[:,:,1] = image_array[:,:,1]
    pad_height = int((X_STEP - 1) / 2)
    pad_width = int((Y_STEP - 1) / 2)
```

```
    padded_image = np.zeros((height + (2 * pad_height), width + (2 * pad_width)))
    padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padded_image.shape[1] - pad_width] =
image_array[:,:,2]

    # Perfom the convolutions
    for Xo in range(height):
    for Yo in range(width):
    Xf = Xo + X_STEP
    Yf = Yo + Y_STEP
    resulting_array[Xo, Yo, 2] = np.sum(kernel * padded_image[Xo:Xf, Yo:Yf])
    resulting_array[:,:,2] = resulting_array[:,:,2] * 255 / np.max(resulting_array[:,:,2])

    return resulting_array
    else:
    # For B&W images
    if len(image_array.shape) == 3:
    image_array = cv2.cvtColor(image_array, cv2.COLOR_BGR2GRAY)

    height, width = image_array.shape
    X_STEP, Y_STEP = kernel.shape

    resulting_array = np.zeros(image_array.shape)

    pad_height = int((X_STEP - 1) / 2)
    pad_width = int((Y_STEP - 1) / 2)

    padded_image = np.zeros((height + (2 * pad_height), width + (2 * pad_width)))
    padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padded_image.shape[1] - pad_width] = image_array

    # Perfom the convolutions
    for Xo in range(height):
    for Yo in range(width):
    Xf = Xo + X_STEP
    Yf = Yo + Y_STEP
    resulting_array[Xo, Yo] = np.sum(kernel * padded_image[Xo:Xf, Yo:Yf])
```

```
    return resulting_array



########### Image Enhancement Results ###########
gaussian_blur_color = False
if gaussian_blur_color:

    image_original = cv2.imread(os.path.join(os.path.dirname(__file__), "fruits.png")).astype('uint8')
    # image_gaussian = gaussian_blur(image_original, 5, True)
    image_gaussian2 = gaussian_blur(image_original, 7, True)

    fig = plt.figure(figsize=(15,15))
    rows = 1
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(cv2.cvtColor(image_original, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Original")

    # fig.add_subplot(rows, columns, 2)
    # plt.imshow(cv2.cvtColor(image_gaussian.astype('uint8'), cv2.COLOR_HSV2RGB), vmin=0, vmax=255)
    # plt.axis('off')
    # plt.title("Gaussian 5x5")

    fig.add_subplot(rows, columns, 2)
    plt.imshow(cv2.cvtColor(image_gaussian2.astype('uint8'), cv2.COLOR_HSV2RGB), vmin=0, vmax=255)
    plt.axis('off')
    plt.title("Gasussian 7x7 Smoothing Filter")

    plt.subplots_adjust(wspace=0, hspace=0)
    plt.show()


laplacian_tests = False
```

```
if laplacian_tests:
    image_parrots = cv2.imread(os.path.join(os.path.dirname(__file__), "coins.png"))
    filter4_4neighbors, filter4_4neighbors_mask = laplacian(image_parrots,[0,1,0,1,-4,1,0,1,0], color = False)
    filter8_all, filter8_all_mask = laplacian(image_parrots,[1,1,1,1,-8,1,1,1,1], color = False)
    fig = plt.figure(figsize=(10, 7))
    # setting values to rows and column variables
    rows = 2
    columns = 3

    fig.add_subplot(rows, columns, 1)
    plt.imshow(cv2.cvtColor(image_parrots, cv2.COLOR_BGR2GRAY), cmap='gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("Original Image")

    fig.add_subplot(rows, columns, 2)
    plt.imshow(filter4_4neighbors, cmap='gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("4-neighbors")


    fig.add_subplot(rows, columns, 3)
    plt.imshow(filter4_4neighbors_mask, cmap='gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("4-neighbors mask")

    fig.add_subplot(rows, columns, 4)
    plt.imshow(filter8_all, cmap='gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("8-neighbors")


    fig.add_subplot(rows, columns, 5)
    plt.imshow(filter8_all_mask, cmap='gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("8-neighbors mask")
```

```
    plt.subplots_adjust(wspace=0, hspace=-0.2)
    plt.show()


doSaltPepperColor = False
if doSaltPepperColor:
    image_parrots = cv2.imread(os.path.join(os.path.dirname(__file__), "balloons_noisy.png"))
    image_parrots = cv2.cvtColor(image_parrots, cv2.COLOR_BGR2RGB)
    image_parrots_denoise_color = salt_pepper_denoise(image_parrots, color=True, adaptive=False)

    fig = plt.figure()
    rows = 1
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(image_parrots)
    plt.axis('off')
    plt.title("Original")

    fig.add_subplot(rows, columns, 2)
    plt.imshow(image_parrots_denoise_color)
    plt.axis('off')
    plt.title("Denoised image")

    plt.show()


doSaltPepper = False
if doSaltPepper:

    image_1 = cv2.imread(os.path.join(os.path.dirname(__file__), "tigernoise.png"))
    image_1_denoise = salt_pepper_denoise(image_1, color=False, adaptive=False)
    image_1_denoise_adapt = salt_pepper_denoise(image_1, color=False, adaptive=True)
    image_1_denoise_adapt_laplace, mask = laplacian(cv2.cvtColor(image_1_denoise_adapt, cv2.COLOR_GRAY2BGR),[0,1,0,1,-
4,1,0,1,0], color = False)
```

```
    fig = plt.figure()
    rows = 2
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(image_1, cmap = 'gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("Original")

    fig.add_subplot(rows, columns, 2)
    plt.imshow(image_1_denoise, cmap = 'gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("Median 3x3 filter")

    fig.add_subplot(rows, columns, 3)
    plt.imshow(image_1_denoise_adapt, cmap = 'gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("Adaptive median 3x3 filter")

    fig.add_subplot(rows, columns, 4)
    plt.imshow(image_1_denoise_adapt_laplace, cmap = 'gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.title("Adaptive median 3x3 filter W/Laplace")

    plt.show()

gamma_correction_tests = False
if gamma_correction_tests:
    image = cv2.imread(os.path.join(os.path.dirname(__file__), "cara.png"))
    image_gamma_corrected = gamma_correction(cv2.cvtColor(image, cv2.COLOR_BGR2HSV), gamma=0.5, channel=2)
    # image_gamma_corrected_eq = histogram_eq(cv2.cvtColor(image_gamma_corrected.astype('uint8'), cv2.COLOR_RGB2HSV))[0]


    fig = plt.figure()
    rows = 1
    columns = 2
```

```
    fig.add_subplot(rows, columns, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Original image")

    fig.add_subplot(rows, columns, 2)
    plt.imshow(cv2.cvtColor(image_gamma_corrected, cv2.COLOR_HSV2RGB))
    plt.axis('off')
    plt.title("γ=0.5 on the intensity channel")

    plt.show()


histogram_eq_tests = False
if histogram_eq_tests:
    image_1 = cv2.imread(os.path.join(os.path.dirname(__file__), "mujer2.jpg"))
    image_1_eq, img_1_hist, img_1_normalized = histogram_eq(cv2.cvtColor(image_1.astype('uint8'), cv2.COLOR_BGR2HSV))
    image_2 = cv2.imread(os.path.join(os.path.dirname(__file__), "carro.png"))
    image_2_eq, img_2_hist, img_2_normalized = histogram_eq(cv2.cvtColor(image_2.astype('uint8'), cv2.COLOR_BGR2HSV))

    fig = plt.figure()
    rows = 2
    columns = 2

    fig.add_subplot(rows, columns, 1).axes.get_yaxis().set_visible(False)
    plt.hist(img_1_hist, bins=256, range=(0, 255), color='blue')
    plt.title("Original")

    fig.add_subplot(rows, columns, 2).axes.get_yaxis().set_visible(False)
    plt.hist(img_1_hist, bins=256, range=(0, 255), color='blue', cumulative=True)
    plt.title("Original - cumulative sum")


    fig.add_subplot(rows, columns, 3).axes.get_yaxis().set_visible(False)
    plt.hist(img_1_normalized, bins=256, range=(0, 255), color='red')
```

```
plt.title("Equalized")

fig.add_subplot(rows, columns, 4).axes.get_yaxis().set_visible(False)
plt.hist(img_1_normalized, bins=256, range=(0, 255), color='red', cumulative=True)
plt.title("Equalized - cumulative sum")

plt.show()

fig = plt.figure()
rows = 1
columns = 2

fig.add_subplot(rows, columns, 1)
plt.imshow(cv2.cvtColor(image_1.astype('uint8'), cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title("Original Image 1")

fig.add_subplot(rows, columns, 2)
plt.imshow(image_1_eq)
plt.axis('off')
plt.title("Equalized Image 1")

plt.show()

fig = plt.figure()
rows = 1
columns = 2

fig.add_subplot(rows, columns, 1)
plt.imshow(image_2, cmap = 'gray', vmin=0, vmax=255)
plt.axis('off')
plt.title("Original Image 2")

fig.add_subplot(rows, columns, 2)
plt.imshow(image_2_eq, cmap = 'gray', vmin=0, vmax=255)
plt.axis('off')
```

```
    plt.title("Equalized Image 2")

    plt.show()
```