ECE 438 - Laboratory 10
Image Processing (Week 1)

Last Updated on April 12, 2022

In [ ]:
```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt


# refer to https://matplotlib.org/3.1.0/gallery/mplot3d/surface3d.html
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

In [ ]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2
```

1. Introduction

This is the first part of a two week experiment in image processing. During this week, we will cover the fundamentals of digital monochrome images, intensity histograms, pointwise transformations, gamma correction, and image enhancement based on filtering.

In the second week , we will cover some fundamental concepts of color images. This will include a brief description on how humans perceive color, followed by descriptions of two standard color spaces. The second week will also discuss an application known as image halftoning.

2. Introduction to Monochrome Images

An image is the optical representation of objects illuminated by a light source. Since we want to process images using a computer, we represent them as functions of discrete spatial variables. For monochrome (black-and-white) images, a scalar function $f[i, j]$ can be used to represent the light intensity at each spatial coordinate (i, j). Figure 1 illustrates the convention we will use for spatial coordinates to represent images.

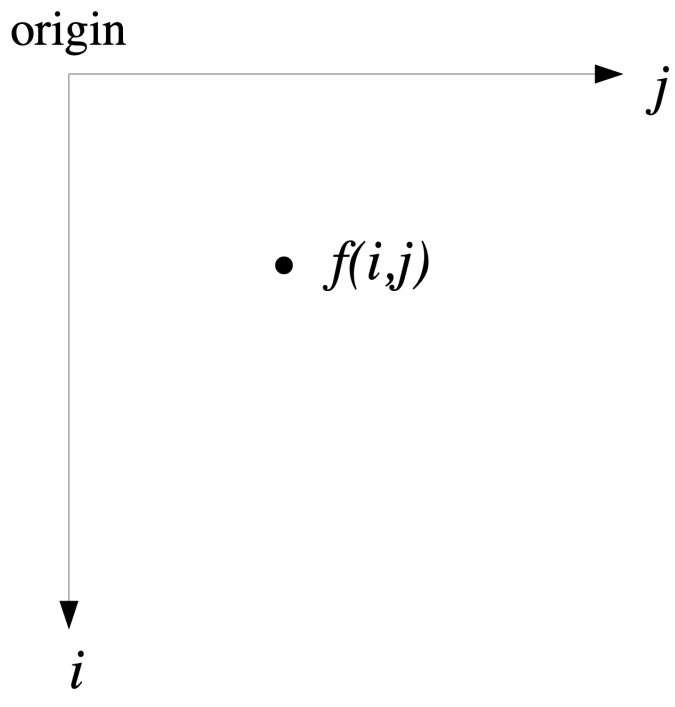


Figure 1: Spatial coordinates used in digital image representation

If we assume the coordinates to be a set of positive integers, for example $i = 0, \ldots, M-1$ and $j = 0, \ldots, N-1$, then an image can be conveniently represented by a matrix.

$$f[i, j] = \begin{bmatrix} f[0,0] & f[0,1] & \cdots & f[0,N-1] \\ f[1,0] & f[1,1] & \cdots & f[1,N-1] \\ \vdots & \vdots & \ddots & \vdots \\ f[M-1,0] & f[M-1,1] & \cdots & f[M-1,N-1] \end{bmatrix} \tag{1}$$

We call this an $M \times N$ image, and the elements of the matrix are known as *pixels*.

The pixels in digital images usually take on integer values in the finite range,

$$0 \leq f[i, j] \leq L_{\max} \tag{2}$$

where 0 represents the minimum intensity level (black), and $L_{\max}$ is the maximum intensity level (white) that the digital image can take on. The interval $[0, L_{\max}]$ is known as a gray scale.

In this lab, we will concentrate on 8-bit images, meaning that each pixel is represented by a single byte. Since a byte can take on 256 distinct values, $L_{\max}$ is 255 for an 8-bit image.

Exercise 2.1

In order to process images within Python, we need to first understand their numerical representation.

1. Load the image file `yacht.tif`, which contains an 8-bit monochrome image, using `plt.imread()` (https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.imread.html).

In [ ]:
```
# insert your code here
```

2. Display the type of this variable by printing its attribute `dtype` (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.dtype.html).

In [ ]:
```
# insert your code here
```

Notice that the `A` matrix elements are of type `uint8` (unsigned integer, 8 bits). This means that Python is using a single byte to represent each pixel. Be careful that we should not perform numerical computation on numbers of type `uint8`, so we usually need to convert the matrix to a floating point representation, using the method `astype` (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.astype.html).

3. Create a double precision representation of the image. Display the type of the matrix.

In [ ]:
```
# insert your code here
```

In future sections, we will be performing computations on our images, so we need to remember to convert them to type double before processing them.

4. Display `yacht.tif` using the following commands:

```
plt.imshow(B, cmap='gray', vmin=0, vmax=255)
plt.show()
```

In [ ]:
```
# insert your code here
```

The `plt.imshow()` command works for both type `uint8` and `double` images. The `cmap` argument specifies how the image is displayed. It is important to note that if any pixel values are outside the range `vmin` to `vmax` (after processing), they will be clipped to `vmin` or `vmax` respectively in the displayed image. `vmin` and `vmax` are two arguments of the function `plt.imshow()`. **It is necessary to set `vmin=0` and `vmax=255` to display a grayscale image. Without setting these two arguments, the `imshow` function tends to normalize the input matrix first, thus outputing a wrong grayscale image.**

It is also important to note that a floating point pixel value will be rounded down ("floored") to an integer before it is displayed. Therefore the maximum number of gray levels that will be displayed on the monitor is 255, even if the image values take on a continuous range.

Now we will practice some simple operations on the `yacht.tif` image.

5. Print the value of $f[35, 79]$ for this `yacht.tif` image.

In [ ]:
```
# insert your code here
```

6. Downsample this image by selecting every other row and column. Then, display it.

In [ ]:
```
# insert your code here
```

7. Make a horizontally flipped version of the image by reversing the order of each column. Then, display it.

In [ ]:
```
# insert your code here
```

8. Similarly, create a vertically flipped image. Display it.

In [ ]:
```
# insert your code here
```

9. Create a "negative" of the image by subtracting each pixel from 255. Then, display it.

In [ ]:
```
# insert your code here
```

10. Multiply each pixel of the original image by 1.5. Display it.

In [ ]:
```
# insert your code here
```

11. What effect did multiplying each pixel of the original image by 1.5 have?

insert your answer here

3. Pixel Distributions

3.1. Histogram of an Image

The histogram of a digital image shows how its pixel intensities are distributed. The pixel intensities vary along the horizontal axis, and the number of pixels at each intensity is plotted vertically, usually as a bar graph. A typical histogram of an 8-bit image is shown in Figure 2.

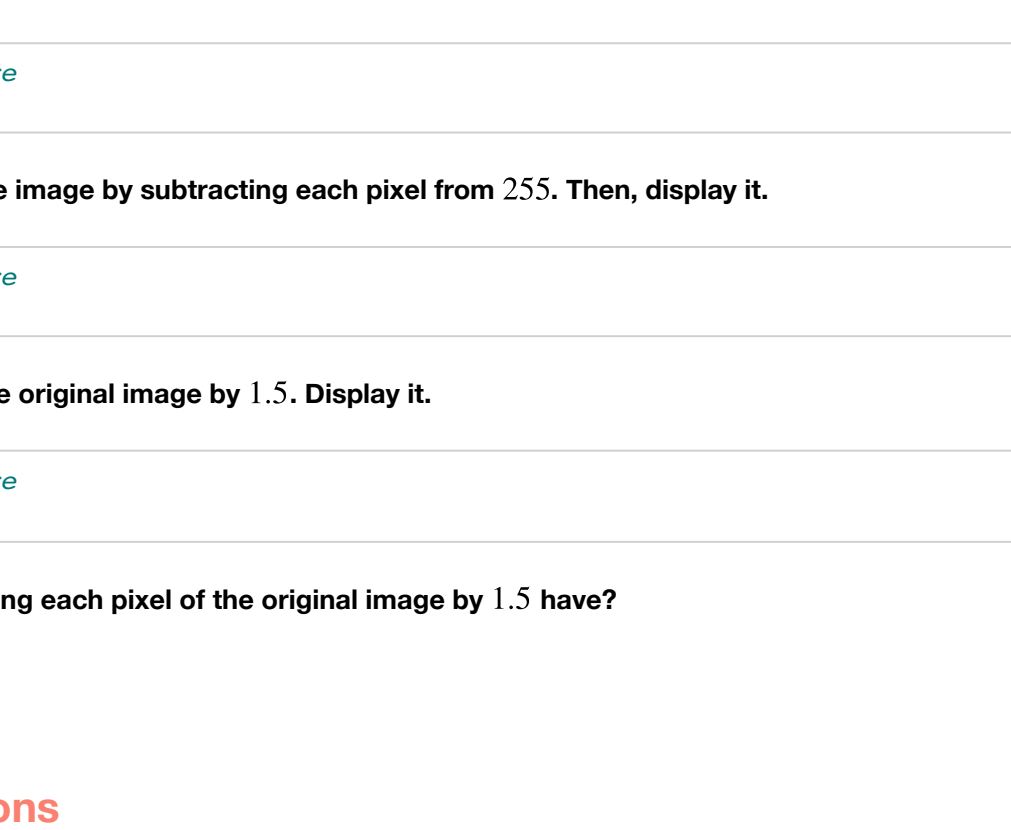


Figure 2: Histogram of an 8-bit image

Exercise 3.2: Histogram of an Image

1. Load the grayscale image `house.tif` and display it.

In [ ]:
```
# insert your code here
```

2. Plot the histogram of the image. Lable the axes of the histogram and give it a title.

Note: You may use `plt.hist()` (https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.hist.html) function. However, this function requires a vector as input. An example of using `plt.hist()` to plot a histogram of a matrix would be

```
plt.hist(x.reshape(-1), bins=np.arange(256))  # reshape(-1) reshapes the original multi-dimension a
rray to 1D
plt.title("title")
plt.xlabel("xlabel")
plt.ylabel("ylabel")
plt.show()
```

In [ ]:
```
# insert your code here
```

3.3. Pointwise Transformations

A pointwise transformation is a function that maps pixels from one intensity to another. An example is shown in Figure 3. The horizontal axis shows all possible intensities of the original image, and the vertical axis shows the intensities of the transformed image. This particular transformation maps the "darker" pixels in the range $[0, T_1]$ to a level of zero (black), and similarly maps the "lighter" pixels in $[T_2, 255]$ to

white. Then the pixels in the range $[T_1, T_2]$ are "stretched out" to use the full scale of $[0, 255]$. This can have the effect of increasing the contrast in an image.

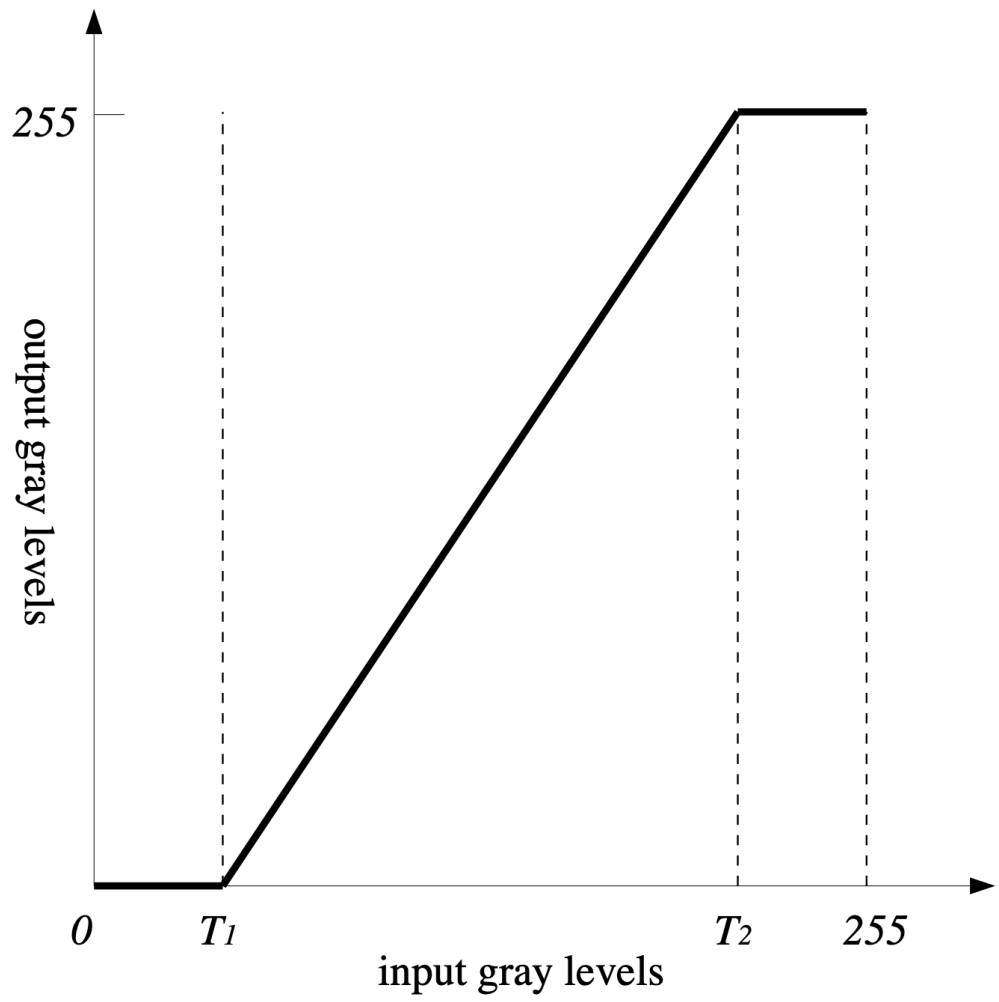


Figure 3: Pointwise transformation of image

Pointwise transformations will obviously affect the pixel distribution, hence they will change the shape of the histogram. If a pixel transformation can be described by a oneto-one function, $y = f(x)$, then it can be shown that the input and output histograms are approximately related by the following:

$$H_{\text{out}}(y) \approx \left. \frac{H_{\text{in}}(x)}{|f'(x)|} \right|_{x=f^{-1}(y)} \tag{3}$$

Since x and y need to be integers in (3), the evaluation of $x = f^{-1}(y)$ needs to be rounded to the nearest integer.

The pixel transformation shown in Figure 3 is not a one-to-one function. However, equation (3) still may be used to give insight into the effect of the transformation. Since the regions $[0, T_1]$ and $[T_2, 255]$ map to the single points 0 and 255, we might expect "spikes" at the points 0 and 255 in the output histogram. The region $[1, 254]$ of the output histogram will be directly related to the input histogram through equation (3).

First, notice from $x = f^{-1}(y)$ that the region $[1, 254]$ of the output is being mapped from the region $[T_1, T_2]$ of the input. Then notice that $f'(x)$ will be a constant scaling factor throughout the entire region of interest. Therefore, the output histogram should approximately be a stretched and rescaled version of the input histogram, with possible spikes at the endpoints.

Exercise 3.4: Pointwise Transformations

1. Complete the function below that will perform the pixel transformation shown in Figure 3.

Hints:

- Determine an equation for the graph in Fig. 3, and use this in your function. Notice you have three input regions to consider. You may want to create a separate function to apply this equation.
- If your function performs the transformation one pixel at a time, be sure to allocate the space for the output image at the beginning to speed things up.

In [ ]:
```
def pointTrans(x, T1, T2):
    """
    Parameters
    ---
    x: the input
    T1: the lower threshold
    T2: the upper threshold

    Returns
    ---
    y: the output
    """

    y = None
    return y
```

2. Load the image file `narrow.tif`. Display the image and its histogram.

In [ ]:
```
# insert your code here
```

3. Use your `pointTrans()` function to spread out the histogram using `T1 = 70` and `T2 = 180`. Display the new image and its

histogram.

```
In [ ]: # insert your code here
```

4. What qualitative effect did the transformation have on the original image? Do you observe any negative effects of the transformation?

insert your answer here

5. Compare the histograms of the original and transformed images. Why are there zeros in the output histogram?

insert your answer here

4. Gamma(γ) Correction

The light intensity generated by a physical device is usually a nonlinear function of the original signal. For example, a pixel that has a gray level of 200 will not be twice as bright as a pixel with a level of 100. Almost all computer monitors have a power law response to their applied voltage. For a typical cathode ray tube (CRT), the brightness of the illuminated phosphors is approximately equal to the applied voltage raised to a power of 2.5. The numerical value of this exponent is known as the gamma (γ) of the CRT. Therefore the power law is expressed as

$$I = V^{\gamma} \tag{4}$$

where I is the pixel intensity and V is the voltage applied to the device.

If we relate equation (4) to the pixel values for an 8-bit image, we get the following relationship,

$$y = 255\left(\frac{x}{255}\right)^{\gamma} \tag{5}$$

where x is the original pixel value, and y is the pixel intensity as it appears on the display. This relationship is illustrated in Figure 4.

In order to achieve the correct reproduction of intensity, this nonlinearity must be compensated by a process known as γ correction. Images that are not properly corrected usually appear too light or too dark. If the value of γ is available, then the correction process consists of applying the inverse of equation (5). This is a straightforward pixel transformation, as we discussed in Section 3.2.

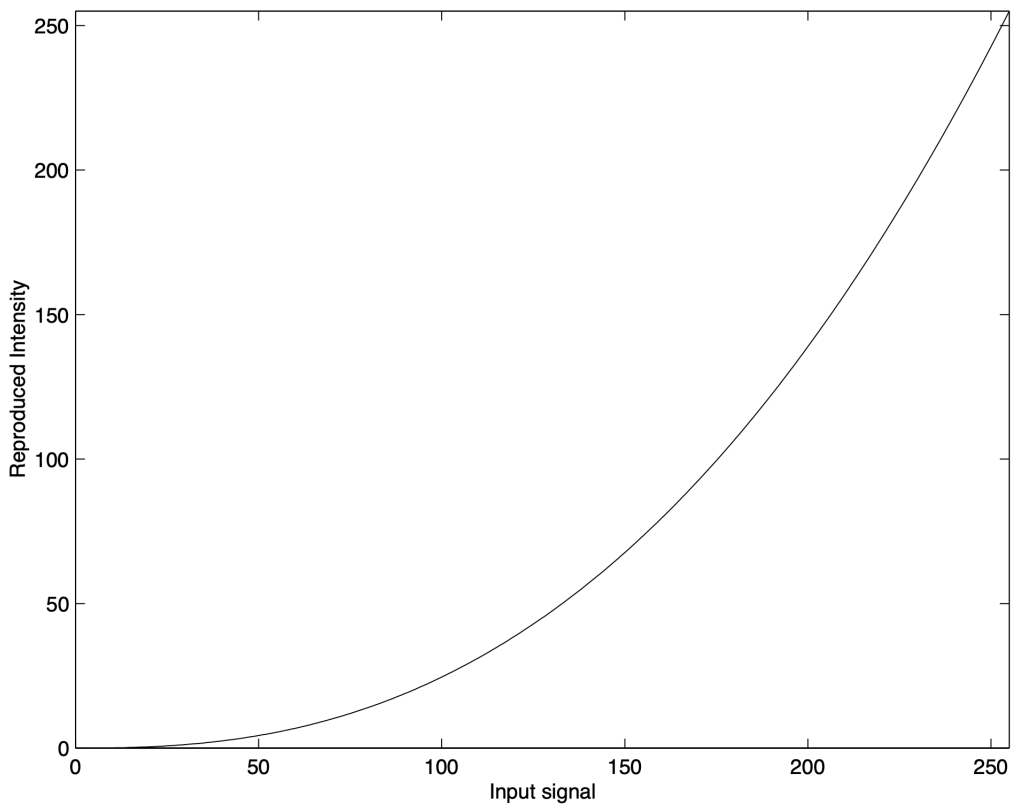


Figure 4: Nonlinear behavior of a display device having a γ of 2.2.

Exercise 4.1: Gamma(γ) Correction

1. Complete the function below that will γ correct an image by applying the inverse of equation (5).

```
In [ ]: def gammCorr(A, gamma):
            """
            Parameters
            ---
            A: the uncorrected image
            gamma: the gamma of the device

            Returns
            ---
            the corrected image
            """

            B = None
            return B
```

2. Load the image file `dark.tif` , which is an image that has not been γ corrected for your monitor. Display it and observe the quality of the image.

In [ ]:
```
# insert your code here
```

3. Assume that the γ for your monitor is 2.2. Use your `gammCorr()` function to correct the image for your monitor, and display the resultant image.

In [ ]:
```
# insert your code here
```

4. How did the correction affect the image? Does this appear to be the correct value for γ?

insert your answer here

5. Image Enhancement Based on Filtering

Sometimes, we need to process images to improve their appearance. In this section, we will discuss two fundamental image enhancement techniques: image *smoothing* and *sharpening*.

Exercise 5.1: 2D Convolution

Filters can be represented as a 2-D convolution of an image $f[i,j]$ with the filter's impulse response $h[i,j]$.

$$\begin{aligned} g[i,j] &= f[i,j] * *h[i,j] \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[i-k, j-l] h[k,l] \end{aligned} \tag{6}$$

1. Complete the function below to convolve the image `image` with the filter `kernel`. Use zero padding to make sure the size of the image is unchanged after it being filtered.

- Assume that the size of the kernel is $s \times s$ where s is odd.
- Try avoid using *for* loops in Equation 6. This can be done by flipping the kernel horizontally and vertically first, and then using element-wise multiplication of matrices and `np.sum()`.
- To zero pad the image, create a new matrix of zeros. The size of this matrix should be $(H+s-1) \times (W+s-1)$, where H, W are the height and the width of the original image, and s is the size of the kernel. Then, assign the correct sub-region of the new matrix with the image.

In [ ]:
```
def convolve2d(image, kernel):
    """
    Parameters
    ---
    image: the input image
    kernel: the filter

    Returns
    ---
    filtered: the filtered image
    """

    filtered = None
    return filtered
```

2. Run the following cell to get the test input image and a 3×3 filter. Use your `convolve2d()` function to get the filtered image.

In [ ]:
```
image = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
                  [0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
                  [0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
                  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
                  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
                  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
                  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
                  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]).astype(float)
kernel = np.array([[-1/8, 1/2, -1/8],
                   [-1/4, 1.0, -1/4],
                   [-1/8, 1/2, -1/8]])
```

In [ ]:
```
# insert your code here
```

3. Run the following cell to check if your `convolve2d()` is correct.

In [ ]:
```
filtered_correct = np.array([[0, 0, 0, 0, -1/8, 1/2, -1/8, 0, 0, 0, 0],
                             [0, 0, 0, -1/8, 1/8, 10/8, 1/8, -1/8, 0, 0, 0],
                             [0, 0, -1/8, 1/8, 7/8, 10/8, 7/8, 1/8, -1/8, 0, 0],
                             [0, -1/8, 1/8, 7/8, 9/8, 1, 9/8, 7/8, 1/8, -1/8, 0],
                             [-1/8, 1/8, 7/8, 9/8, 1, 1, 1, 9/8, 7/8, 1/8, -1/8],
                             [-3/8, 1, 9/8, 1, 1, 1, 1, 1, 9/8, 1, -3/8],
                             [-1/2, 3/2, 1, 1, 1, 1, 1, 1, 1, 3/2, -1/2],
                             [-1/2, 3/2, 1, 1, 1, 1, 1, 1, 1, 3/2, -1/2],
                             [-1/2, 3/2, 1, 1, 1, 1, 1, 1, 1, 3/2, -1/2],
                             [-3/8, 9/8, 6/8, 6/8, 6/8, 6/8, 6/8, 6/8, 6/8, 9/8, -3/8],
                             [-1/8, 3/8, 2/8, 2/8, 2/8, 2/8, 2/8, 2/8, 2/8, 3/8, -1/8]])

np.testing.assert_allclose(filtered, filtered_correct, atol=1e-10, rtol=1e-10)
```

5.2. Image Smoothing

Smoothing operations are used primarily for diminishing spurious effects that may be present in a digital image, possibly as a result of a poor sampling system or a noisy transmission channel. Lowpass filtering is a popular technique of image smoothing.

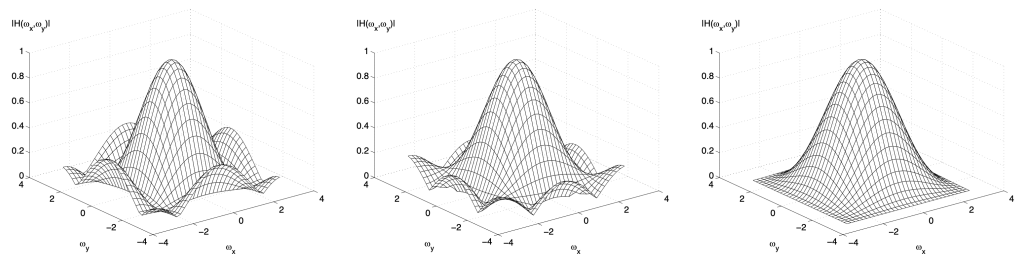
Some typical lowpass filter impulse responses are shown in Figure 5, where the center element corresponds to $h[0,0]$. Notice that the terms of each filter sum to one. This prevents amplification of the DC component of the original image. The frequency response of each of these filters is shown in Figure 6.

$$\frac{1}{9}\cdot\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}\qquad \frac{1}{10}\cdot\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&2&1\\\hline 1&1&1\\\hline\end{array}\qquad \frac{1}{16}\cdot\begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&4&2\\\hline 1&2&1\\\hline\end{array}$$

(a) (b) (c)

Figure 5: Impulse responses of lowpass filters useful for image smoothing.

An example of image smoothing is shown in Figure 7, where the degraded image is processed by the filter shown in Figure 5(c). It can be seen that lowpass filtering clearly reduces the additive noise, but at the same time it blurs the image. Hence, blurring is a major limitation of lowpass filtering.



(a) (b) (c)

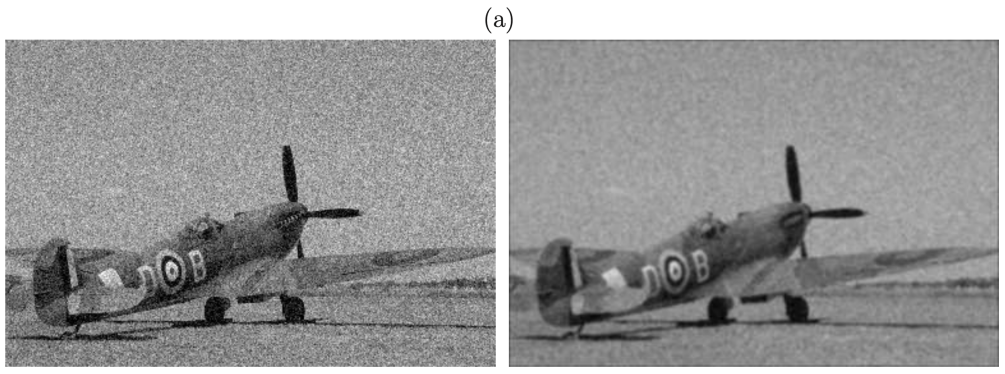
Figure 6: Frequency responses of the lowpass filters shown in Figure 5.

In addition to the above linear filtering techniques, images can be smoothed by nonlinear filtering, such as mathematical morphological processing. **Median filtering** is one of the simplest morphological techniques, and is useful in the reduction of impulsive noise. The main advantage of this type of filter is that it can reduce noise while preserving the detail of the original image. In a median filter, each input pixel is replaced by the median of the pixels contained in a surrounding window. This can be expressed by

$$g[i,j] = median\{f[i-k, j-l]\}, \quad (k,l) \in W \tag{7}$$

where W is a suitably chosen window. Figure 8 shows the performance of the median filter in reducing so-called “salt and pepper” noise.

(a)



(b) (c)

Figure 7: (a) Original gray scale image. (b) Original image degraded by additive white Gaussian noise, $\mathcal{N}(0, 0.01)$. (c) Result of processing the degraded image with a lowpass filter.

(a) (b)

Figure 8: (a) Image degraded by "salt and pepper" noise with 0.05 noise density. (b) Result of 3×3 median filtering.

Exercise 5.3: Image Smoothing

Among the many spatial lowpass filters, the Gaussian filter is of particular importance. This is because it results in very good spatial and spectral localization characteristics. The Gaussian filter has the form

$$h[i, j] = C \cdot \exp\left\{-\frac{i^2 + j^2}{2\sigma^2}\right\} \tag{8}$$

where σ^2, known as the variance, determines the size of passband area. Usually the Gaussian filter is normalized by a scaling constant C such that the sum of the filter coefficient magnitudes is one, allowing the average intensity of the image to be preserved.

$$\sum_{i,j} h[i, j] = 1$$

1. Complete the function below that will create a normalized Gaussian filter that is centered around the origin (the center element of your matrix should be $h[0, 0]$).

- Note that this filter is both separable and symmetric, meaning $h[i, j] = h[i]h[j]$ and $h[i] = h[-i]$.
- Notice that for this filter to be symmetrically centered around zero, `N` will need to be an odd number.
- Make sure the sum of the filter coefficient magnitudes is 1.

In [ ]:
```
def gaussFilter(N, var):
    """
    Parameters
    ---
    N: the size of filter
    var: the variance

    Returns
    ---
    h: the NxN filter
    """

    h = None
    return h
```

2. Compute the frequency response of a 7×7 Gaussian filter with $\sigma^2 = 1$.

- You may use the following command to get a 32×32 DFT.

```
H = np.fft.fftshift(np.fft.fft2(h, (32, 32)))
```

In [ ]:
```
# insert your code here
```

3. Plot the magnitude of the frequency response of the Gaussian filter, $|H_{\text{Gauss}}(\omega_1, \omega_2)|$, using the provided `mesh_plot()` function below. Plot it over the region $[-\pi, \pi] \times [-\pi, \pi]$, and label the axes.

In [ ]:
```
# refer to https://matplotlib.org/3.1.0/gallery/mplot3d/surface3d.html

def mesh_plot(X, Y, Z, title, xlabel, ylabel):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=2, antialiased=True)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    # Add a color bar which maps values to colors.
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()
```

In [ ]:
```
# insert your code here
```

4. Load the image file `race.tif` and filter it with a 7×7 Gaussian filter, with $\sigma^2 = 1$. Display the original and the filtered images.

In [ ]:
```
# insert your code here
```

5. Load the image files `noise1.tif` and `noise2.tif`, and display them. These images are versions of `race.tif` that have been degraded by additive white Gaussian noise and "salt and pepper" noise, respectively.

In [ ]:
```
# insert your code here
```

6. Complete the function below to implement a 3×3 median filter (without using the `signal.medfilt2d()` function).

- For convenience, you do not have to alter the pixels on the border of `image`.
- Use `np.median()` to find the median value of a subarea of the image, i.e., a 3×3 window surrounding each pixel.

In [ ]:
```
def medianFilter(image):
    """
    Parameters
    ---
    image: the input image

    Returns
    ---
    filtered: the output filtered image
    """
    filtered = None
    return filtered
```

7. Filter each of the noisy images with both the 7×7 Gaussian filter ($\sigma^2 = 1$) and the 3×3 median filter. Display the 4 results and place a title indicating the type of noise and the filter on each image.

In [ ]:
```
# insert your code here
```

8. Discuss the effectiveness of each filter for the case of additive white Gaussian noise. Discuss both positive and negative effects that you observe for each filter.

insert your answer here

9. Discuss the effectiveness of each filter for the case of "salt & pepper" noise. Again, discuss both positive and negative effects that you observe for each filter.

insert your answer here

5.4. Image Sharpening

Image sharpening techniques are used primarily to enhance an image by highlighting details. Since fine details of an image are the main contributors to its high frequency content, highpass filtering often increases the local contrast and sharpens the image. Some typical highpass filter impulse responses used for contrast enhancement are shown in Figure 9. The frequency response of each of these filters is shown in Figure 10.

0	1	0
1	-4	1
0	1	0

(a)

1	1	1
1	-8	1
1	1	1

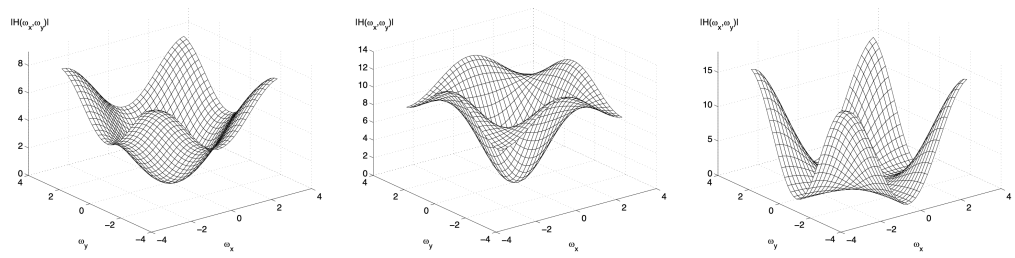
(b)

-1	2	-1
2	-4	2
-1	2	-1

(c)

Figure 9: Impulse responses of highpass filters useful for image sharpening.

An example of highpass filtering is illustrated in Figure 11. It should be noted from this example that the processed image has enhanced contrast, however it appears more noisy than the original image. Since noise will usually contribute to the high frequency content of an image, highpass filtering has the undesirable effect of accentuating the noise.



(a) (b) (c)

Figure 10: Frequency responses of the highpass filters shown in Figure 9.

(a) (b)

Figure 11: (a) Original gray scale image. (b) Highpass filtered image.

5.5. Exercise: Image Sharpening

In this section, we will introduce a sharpening filter known as an unsharp mask. This type of filter subtracts out the "unsharp" (low frequency) components of the image, and consequently produces an image with a sharper appearance. Thus, the unsharp mask is closely related to highpass filtering. The process of unsharp masking an image $f[i,j]$ can be expressed by

$$g[i,j] = \alpha f[i,j] - \beta\left[f\left[i,j\right] * *h\left[i,j\right]\right] \tag{9}$$

where $h[i,j]$ is a lowpass filter, and α and β are positive constants such that $\alpha - \beta = 1$.

Analytically calculate the frequency response of the unsharp mask filter in terms of α, β, and $h[i,j]$ by finding an expression for

$$\frac{G(\omega_1, \omega_2)}{F(\omega_1, \omega_2)}. \tag{10}$$

1. Using your `gaussFilter()` function, create a 5×5 Gaussian filter with $\sigma^2 = 1$.

In [ ]:
```
# insert your code here
```

2. Derive the frequency response of an unsharp mask filter from equation (10).

insert your answer here

3. Compute the frequency response of the unsharp mask filter, using the Gaussian filter as $h[i,j]$, $\alpha = 5$ and $\beta = 4$. The size of the calculated frequency response should be 32×32. Plot the magnitude of this response in the range $[-\pi, \pi] \times [-\pi, \pi]$ using provided function `mesh_plot()` , and label the axes. Print out this response.

In [ ]:
```
# insert your code here
```

4. Load the image file `blur.tif` and display it.

In [ ]:
```
# insert your code here
```

5. Apply the unsharp mask filter with the parameters specified above to this image, using equation (9). Use $\alpha = 5$ and $\beta = 4$ to display the filtered image.

In [ ]:
```
# insert your code here
```

6. What effect did the filtering have on the image?

insert your answer here

7. Now try applying the filter to `blur.tif` , using $\alpha = 10$ and $\beta = 9$. Label the processed image and display it.

In [ ]:
```
# insert your code here
```

8. Compare this result to the previous one.

insert your anwer here