Introduction to Digital Image Processing

Homework 3

October 28, 2020

Student: Ivan Arevalo
Perm Number: 5613567
Email: ifa@ucsb.com

Department of Electrical and Computer Engineering, UCSB

Problem 1: Linear and shift invariant operators:

Determine for the following operators whether they are linear and whether they are shift invariant. Justify your answers.

a) Mean: $f[m,n] \rightarrow g[m,n] = \mu$, where constant $\mu = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f[m,n]$.

b) Gain: $f[m,n] \rightarrow g[m,n] = \alpha f[m,n]$, where α is a non-zero constant.

c) Gamma correction: $f[m,n] \rightarrow g[m,n] = f[m,n]^{\gamma}$, where γ is a non-zero constant.

d) Affine: $f[m,n] \rightarrow g[m,n] = \alpha f[m,n] + \beta$, where α and β are non-zero constants.

e) Thresholding or binarization: $f[m,n] \rightarrow g[m,n] = u(f[m,n] - T)$, where $u(\cdot)$ is the (continuous) unit step function, and T is a constant.

f) Coordinate flip: $f[m,n] \rightarrow g[m,n] = f[n,m]$

For linearity: $H[\alpha f_1(x) + \beta f_2(x)] = \alpha H[f_1(x)] + \beta H[f_2(x)]$

For S.I. if $H[f(x)] = g(x)$, then $g(x+x_0) = H[f(x+x_0)]$

a) Let $r(x) = \alpha f_1(m,n) + \alpha_2 f_2(m,n)$

$$H[r(m,n)] = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} r(m,n)$$

$$= \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}[\alpha f_1(m,n) + \alpha_2 f_2(m,n)]$$

$$= \frac{\alpha}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_1(m,n) + \frac{\alpha_2}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_2(m,n)$$

$$H[\alpha_i f_i(n,m)] = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\alpha_i f_i(m,n) = \frac{\alpha_i}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_i(m,n)$$

Therefore: $H[\alpha_1 f_1(m,n) + \alpha_2 f_2(m,n)] = \alpha_1 H[f_1(m,n)] + \alpha_2 H[f_2(m,n)]$

Mean operator is linear.

$$g(x+x_0) = \mu = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(x+x_0)$$

given μ is a constant, Mean is shift Invariant

b) Let $r(x) = B_1 f_1(m,n) + B_2 f_2(m,n)$

$$H[r(m,n)] = \alpha(B_1 f_1(m,n) + \beta_2 f_2(m,n))$$
$$= \alpha B_1 f_1(m,n) + \alpha B_2 f_2(m,n)$$

$\beta_i H[f_i(m,n)] = \beta_i \alpha f_i(m,n)$

Therefore $H[B_1 f_1(m,n) + \beta_2 f_2(m,n)] = \alpha B_1 f_1(m,n) + \alpha B_2 f_2(m,n)$

Gain operator is linear

$g(m-m_0, n-n_0) = \alpha f_1(m-m_0, n-n_0)$

$H[f_1(m-m_0, n-n_0)] = \alpha f_1(m-m_0, n-n_0)$

Therefore $g(x+x_0) = H[f(x+x_0)]$ and the Gain operator is shift Invariant

c) Let $r(x) = B_1 f_1(m,n) + B_2 f_2(m,n)$

$$H[r(m,n)] = (B_1 f_1(m,n) + B_2 f_2(m,n))^{\gamma}$$

$\beta_i H[f_i(m,n)] = \beta_i (f_i(m,n))^{\gamma}$

$(B_1 f_1(m,n) + \beta_2 f_2(m,n))^{\gamma} \neq \beta_1 (f_1(m,n))^{\gamma} + \beta_2 (f_2(m,n))^{\gamma}$

Therefore Gamma correction is not linear

$g(m-m_0, n-n_0) = (f(m-m_0, n-n_0))^{\gamma}$

$H[f(m-m_0, n-n_0)] = (f(m-m_0, n-n_0))^{\gamma}$

$g(m-m_0, n-n_0) = H[f(m-m_0, n-n_0)]$

Therefore Gamma correction is shift Invariant.

d) Let $r(x) = \lambda_1 f_1(m,n) + \lambda_2 f_2(m,n)$

$H[r(x)] = \alpha(\lambda_1 f_1(m,n) + \lambda_2 f_2(m,n)) + \beta$

$\lambda_i H[f_i(m,n)] = \lambda_i(\alpha f_i(m,n) + \beta) = \lambda_i \alpha f_i(m,n) + \lambda_i \beta$

$\alpha\lambda_1 f_1(m,n) + \alpha\lambda_2 f_2(m,n) + \beta \neq \lambda_1 \alpha f_1(m,n) + \lambda_2 \alpha f_2(m,n) + \beta(\lambda_1 + \lambda_2)$

Therefore Affine operator is not linear

$g(m-m_0, n-n_0) = \alpha f(m-m_0, n-n_0) + \beta$

$H[f(m-m_0, n-n_0)] = \alpha f(m-m_0, n-n_0) + \beta$

$g(m-m_0, n-n_0) = H[f(m-m_0, n-n_0)]$

Therefore Affine operator is Shift Invariant

e) Let $r(x) = \lambda_1 f_1(m,n) + \lambda_2 f_2(m,n)$

$H[r(x)] = u(\lambda_1 f_1(m,n) + \lambda_2 f_2(m,n) - T)$; $u(\cdot) \rightarrow$ step

$\lambda_i H[f_i(m,n)] = \lambda_i u(f_i(m,n) - T)$

$u(\lambda_1 f_1(m,n) + \lambda_2 f_2(m,n) - T) \neq$
$\lambda_1 u(f_1(m-n) - T) + \lambda_2 u(f_2(m-n) - T)$

Therefore binarization is not linear

$g(m-m_0, n-n_0) = u(f(m-m_0, n-n_0) - T)$

$H[f(m-m_0, n-n_0)] = u(f(m-m_0, n-n_0) - T)$

Thefore binarization is shift invariant

f) Let $r(x) = \lambda_1 f_1(m,n) + \lambda_2 f_2(m,n)$

$H[r(x)] = \lambda_1 f_1(n,m) + \lambda_2 f_2(n,m)$

$\lambda_i H(f_i(m,n)) = \lambda_i f_i(n,m)$

$\lambda_1 f_1(n,m) + \lambda_2 f_2(n,m) = \lambda_1 f_1(n,m) + \lambda_2 f_2(n,m)$.

Therefore Coordinate flip is linear

$g(m-m_0, n-n_0) = f(n-n_0, m-m_0)$

$H[f(m-m_0, n-n_0)] = f(n-n_0, m-m_0)$

Therefore coordinate flip is shift Invariant

Problem 2: Convolution with various boundary conditions, and some non-linear filtering:

This problem is to be solved on paper (not computer). Consider the following image of size M = 4, N = 3. (The boxed pixel represents the [0, 0] position.)

$$f = \begin{array}{cccc} \boxed{3} & 2 & 3 & 5 \\ 4 & 1 & 3 & 4 \\ 3 & 2 & 4 & 4 \end{array}$$

which we would like to convolve with a filter having the familiar impulse response

$$h = \begin{array}{ccc} 1 & 2 & 1 \\ 0 & \boxed{0} & 0 \\ -1 & -2 & -1 \end{array}$$

and obtain an output image **of the same size**.

a) First solve while assuming the image is extended with zero padding.

b) Next, assume periodic extension and convolve. (Note: periodic extension means $f[m,n] = f[m+M, n+N]$) Comment on which pixel locations you knew in advance would show the same outcome in (a) and (b)?

c) Next, apply a 3×3 median filter to f under the zero padding extension assumption.

d) Finally, what is the impulse response h of the median filter? Is it surprising?

For each of the following convolution computations, we find each of the output pixel values by

$$I_{out}[k,\ell] = \sum_{m=-floor(L/2)}^{floor(L/2)} \sum_{n=-floor(L/2)}^{floor(L/2)} f[k-m,\ell-n]h[m,n]$$

Where L is the size of the kernel and the input image is of size $k \times l$.

a) With zero padding:

$$f_{zp} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 3 & 5 & 0 \\ 0 & 4 & 1 & 3 & 4 & 0 \\ 0 & 3 & 2 & 4 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$I_{OUT} = f_{zp} * h = \sum_{m=-1}^{1} \sum_{n=-1}^{1} f_{zp}[k-m, \ell-n]h[m,n]$$

$$I_{OUT} = \begin{bmatrix} 9 & 9 & 11 & 11 \\ 0 & 1 & 1 & -1 \\ -9 & -9 & -11 & -11 \end{bmatrix}$$

Simple arithmetic, used mental math.

For the example above, $I_{out}[0,0] = 0*-1+0*-2+0*-1+0*0+3*0+2*0+0*1+2*4+1*1 = 9$. We slide this kernel and repeat for all pixels.

b) With periodic extension:

$$f_{zp} = \begin{bmatrix} 4 & 3 & 2 & 4 & 4 & 3 \\ 5 & 3 & 2 & 3 & 5 & 3 \\ 4 & 4 & 1 & 3 & 4 & 4 \\ 4 & 3 & 2 & 4 & 4 & 3 \\ 5 & 3 & 2 & 3 & 5 & 3 \end{bmatrix} \qquad h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$I_{OUT} = f_{zp} * h = \sum_{m=-1}^{1} \sum_{n=-1}^{1} f_{zp}[k-m, \ell-n]h[m,n]$$

$$I_{OUT} = \begin{bmatrix} 1 & -2 & -3 & 0 \\ -1 & 1 & 1 & -1 \\ 0 & 1 & 2 & 1 \end{bmatrix}$$

Simple arithmetic, used mental math

We knew output pixels at (1,1) & (1,2) would be the same given that the corresponding input window is the same in both padding techniques.

c) 3 x 3 median fiter w/ zero padding

$$f_{zp} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 3 & 5 & 0 \\ 0 & 4 & 1 & 3 & 4 & 0 \\ 0 & 3 & 2 & 4 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad h = \begin{bmatrix} \text{Sort Values} \\ \text{\& pick median} \end{bmatrix}$$

$$I_{OUT} = \begin{bmatrix} 0 & 2 & 2 & 0 \\ 2 & 3 & 4 & 3 \\ 0 & 2 & 2 & 0 \end{bmatrix}$$

Pick 5th (middle) number from least to greatest.

d) The impulse response of the median filter is 0. This makes sense given that it is a low pass filter that removes sudden spikes, such as in salt & pepper noise.

Problem 3: A hands-on problem. Efficient image blurring using the concept of filter separability:

Consider the following bi-variate Gaussian function (can also be viewed as joint PDF of two independent random variables):

$$f(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-(\frac{(x-\mu_x)^2}{2\sigma_x^2} + \frac{(y-\mu_y)^2}{2\sigma_y^2})}. \tag{1}$$

It is centered at the mean (μ_x, μ_y) and its spread is determined by the respective standard deviations σ_x and σ_y, along the x and y directions. We will use (1) to construct a $2D$ filter kernel that belongs to the class of "moving weighted average" filters we discussed in class. Convolving an image with this kernel would blur it. In this problem, whenever a convolution is performed on an image, assume zero padding extension.

a) First, use the gaussian of (1) to create a discrete $2D$ matrix (also called a discrete 2D gaussian kernel) denoted $h[m, n]$, of size 9×9, with the origin $[0, 0]$ at the top left corner of the matrix, i.e., $m \in \{0, 1, \ldots, 8\}$ and $n \in \{0, 1, \ldots, 8\}$. Set $\mu_x = \mu_y = 4$ so that the gaussian center coincides with the center of the matrix, set $\sigma_x = \sigma_y = 1.5$, and populate the matrix with values $h[m, n] = f(m, n)$ using (1). (*hint:* consider using *meshgrid* or nested for loops). Next, normalize the matrix elements so that they add up to 1.0 (i.e., they represent weights of a moving *average*) to obtain the filter impulse response (gaussian kernel) $h[m, n]$. Visualize $h[m, n]$ (use inbuilt visualization tools).
As an aside, note how many multiplications per pixel are required to convolve an image with $h[m, n]$.

b) Next, take a closer look at the gaussian function of (1) and explain why the filter $h[m, n]$, constructed using $f(x, y)$ in part (a), must in fact be a *separable* filter. This means that $h[m, n]$ can be written as an outer product of two 1D filters denoted $h_x[m]$ and $h_y[n]$. Visualize $h_x[m]$ and $[h_y[n]$ (use inbuilt visualization tools).
As another aside, note how many multiplications per pixel would be needed to blur the image using separable row and column filtering with $h_x[m]$ and $h_y[n]$.

c) Perform convolution on the attached image (*peppers.png*): First using $h[m, n]$, as if we did not realize it was separable. Show the result as well as the runtime (using *tic-toc* in matlab and *time* in python). Convolution with $h[m, n]$ should yield a blurry image. Next, exploit the concept of separable filters to blur the image by convolving with the 1D filters $h_x[m]$ and $h_y[n]$. Show the output and report the runtime. Of course, the output image after blurring with $h[m, n]$, regardless of whether we exploit separability, should be exactly the same - verify this. Also verify that the runtime ratio of the two approaches is consistent with the ratio of number of multiplications per pixel, as observed in parts (a) and (b). (Though not exactly the same, because we neglected additions and general overhead in the programs).

3a)

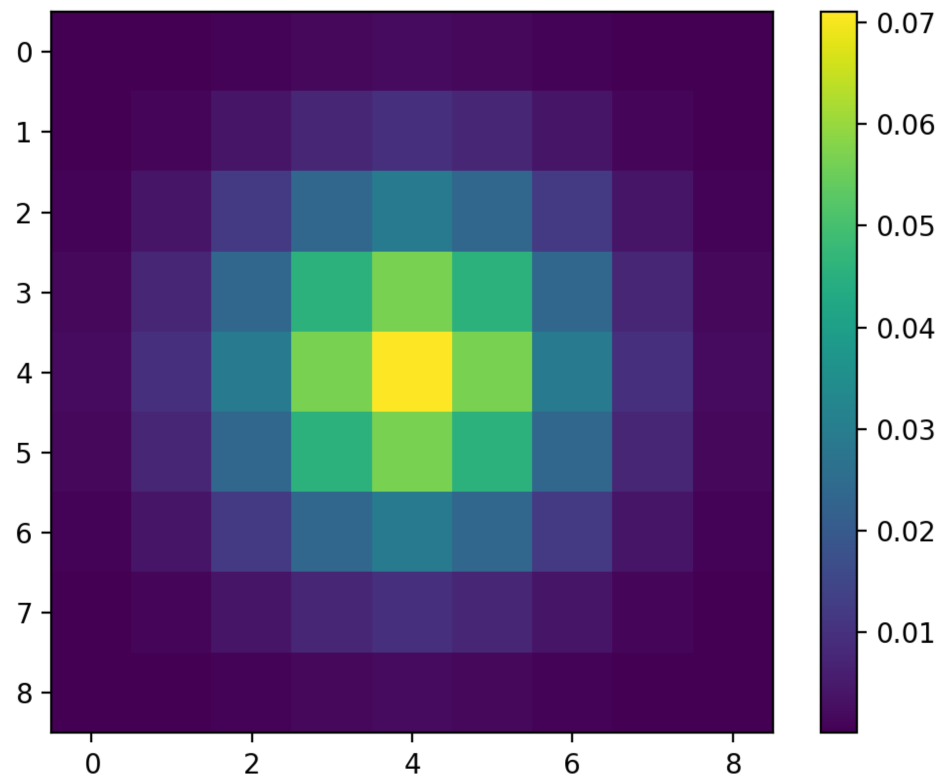

Figure 1: $h[m,n] with kernel size = 9$

Given an image MxN and kernel LxL, it would require (LxL) multiplications for each output pixel and given there are (M - L + 1) * (N - L + 1) output pixels, there is a total of (M - L + 1) * (N - L + 1) * (LxL) multiplications. Given our image is 384×512 and $L = 9$, there are 15,349,824 multiplications.

3b)

We can observe that the 2D Gaussian filter $h[m,n]$ shown in figure 1 is separable by using the inverse exponent product rule to express the pdf as a product of 2 1D Gaussian pdfs.

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-(\frac{(x-\mu)^2}{2\sigma_x^2} + \frac{(y-\mu)^2}{2\sigma_y^2})}$$

$$f(x,y) = \frac{1}{\sqrt{2\pi}\sigma_x} e^{-\frac{1}{2}(\frac{(x-\mu)^2}{\sigma_x})} \frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{1}{2}(\frac{(y-\mu)^2}{\sigma_y})} = f(x)f(y)$$

Figure 2: $h_x[m]$

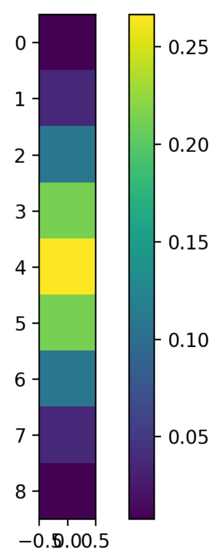

Figure 3: $h_y[n]$

Given an image MxN and kernel size L, it would require 2*L multiplications for each output pixel andgiven there are (M - L + 1) * (N - L + 1) output pixels, there is a total of (M - L + 1) * (N - L + 1) * (2xL) multiplications. Given our image is 384×512 and $L = 9$, there are 3,411,072 multiplications.

3c)

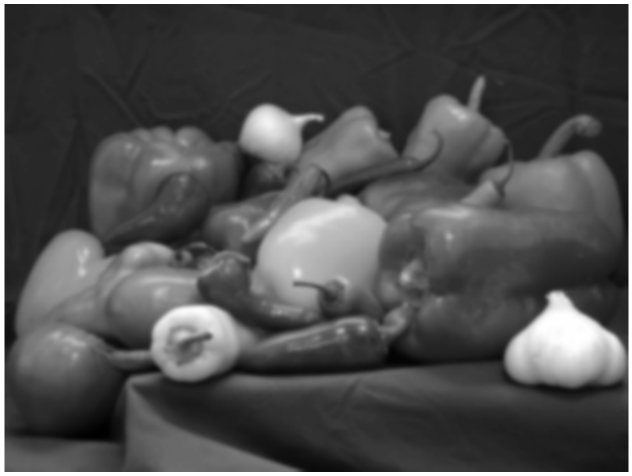

Figure 4: Output image convolved with $h[m, n]$

Performing convolution with $h[m, n]$ took 0.002328157424926758 seconds

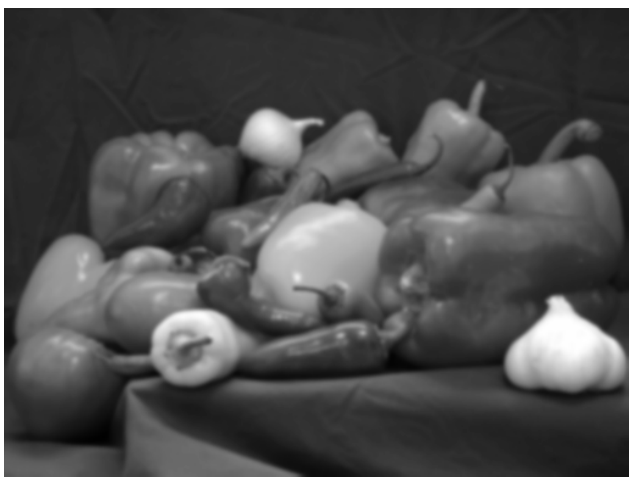

Figure 5: Output image convolved with $h_x[m]$ and $h_y[n]$

Performing convolution with separable kernels $h_x[m]$ and $h_y[n]$ took 0.0004329681396484375 seconds.

Given that filtering with the separable kernels $h_x[m]$ and $h_y[n]$ took 0.0004329681396484375 seconds to run and filtering with the original $h[m, n]$ took 0.002328157424926758 seconds to run, we can conclude that the separable filter ran 5.4 times faster:

$$\frac{0.002328157424926758}{0.0004329681396484375} = 5.37720264315$$

This approximately matches our calculated ratio of the number of multiplications needed with each convolution.

$$\frac{15,349,824}{3,411,072} = 4.5$$

Code

```
import numpy as np
import math
import matplotlib.pyplot as plt
import cv2
import time
from PIL import Image

def create_gaussian_kernel(sigma_size, separable=False):
    """
    Creates a gaussian kernel based on sigma and dimensions of kernel matrix
    :param sigma_size: standard deviation of gaussian
    :param kernel_dimensions: kernel dimensions
    """
    # Can modify kernel_dimension relative to sigma_size
    kernel_dimensions = int(6 * sigma_size)
    gaussian_coefficient = 1/math.sqrt(2*math.pi)*sigma_size if separable else 1 / (2 *
        math.pi * (sigma_size ** 2))
    gaussian_sum = 0
    gaussian_kernel = np.zeros(kernel_dimensions) if separable else
        np.zeros([kernel_dimensions, kernel_dimensions])

    for x in range(0, kernel_dimensions):
        i = x - kernel_dimensions // 2
        if separable:
            gaussian_exponential = math.exp(-1/2*((i/sigma_size) ** 2))
            gaussian_kernel[x] = gaussian_coefficient * gaussian_exponential
            gaussian_sum += gaussian_kernel[x]
        else:
            for y in range(0, kernel_dimensions):
                # i = x - math.floor(kernel_dimensions / 2)
                j = y - kernel_dimensions // 2
                gaussian_exponential = math.exp(-((i**2)+(j**2))/(2*sigma_size**2))
                # print("iteration {}:{}--> i: {} j: {} var = {}".format(x, y, i, j,
                    gaussian_exponential))
                gaussian_kernel[x, y] = gaussian_coefficient*gaussian_exponential
                gaussian_sum += gaussian_kernel[x, y]

    normalized_gaussian_kernel = (1/gaussian_sum)*gaussian_kernel
    return normalized_gaussian_kernel.reshape([1, -1]) if separable else
        normalized_gaussian_kernel

def filter_image(image, kernel, seperable=False):
    start_time = time.time()
    output_im = cv2.filter2D(image, -1, kernel)
    total_time = time.time() - start_time
    if seperable:
        start_time = time.time()
        output_im = cv2.filter2D(output_im, -1, np.transpose(kernel))
        total_time = time.time() - start_time
    return output_im, total_time
```

```python
if __name__ == '__main__':
    separable = True
    peppers_img = Image.open("peppers.png")
    peppers_img_pxl = np.asarray(peppers_img)

    # Part a) Create a discrete 2D gaussian kernel of size 9x9
    # Part b) We can seperate out the exponential into 2 multiplications.
    # Part c) Perform full and separable convolution on peppers.png and time each
        operation

    gaussian_kernel = create_gaussian_kernel(sigma_size=1.5, separable=separable)
    print("Gaussian kernel shape: " + str(gaussian_kernel.shape))

    plt.imsave("IvanArevalo-HW3-P3{}.png".format('B1' if separable else 'A'),
        gaussian_kernel)
    plt.imshow(gaussian_kernel)
    plt.colorbar()
    if separable:
        plt.imsave("IvanArevalo-HW3-P3{}.png".format("B2" if separable else "error"),
            np.transpose(gaussian_kernel))
        plt.figure()
        plt.imshow(np.transpose(gaussian_kernel))
        plt.colorbar()
        print("Given an image MxN and kernel size L, it would require 2*L multiplications
            for each output pixel and"
              "given there are (M - floor(L/2)) * (N - floor(L/2)) output pixels, there
                  is a total of "
              "(M - floor(L/2)) * (N - floor(L/2)) * (2xL) multiplications")
        filtered_im, runtime = filter_image(peppers_img_pxl, gaussian_kernel,
            seperable=separable)
        print("Performing convolution with separable kernels h_x[m] and h_y[n] took {}
            seconds".format(runtime))
        plt.imsave('IvanArevalo-HW3-P3C_Separable.png', filtered_im, cmap='gray')
        plt.figure()
        plt.imshow(filtered_im, cmap='gray')
        plt.title("Convolution with {}".format("h_x[m] and h_y[n]" if separable else
            "h[m, n]"))
    else:
        print("Given an image MxN and kernel LxL, it would require (LxL) multiplications
            for each output pixel and"
              "given there are (M - floor(L/2)) * (N - floor(L/2)) output pixels, there
                  is a total of "
              "(M - floor(L/2)) * (N - floor(L/2)) * (LxL) multiplications")
        filtered_im, runtime = filter_image(peppers_img_pxl, gaussian_kernel,
            seperable=separable)
        print("Performing convolution with h[m, n] took {} seconds".format(runtime))
        plt.imsave('IvanArevalo-HW3-P3C_Not-Separable.png', filtered_im, cmap='gray')
        plt.figure()
        plt.imshow(filtered_im, cmap='gray')
        plt.title("Convolution with separable kernels h_x[m] and h_y[n]")

    plt.show()
```