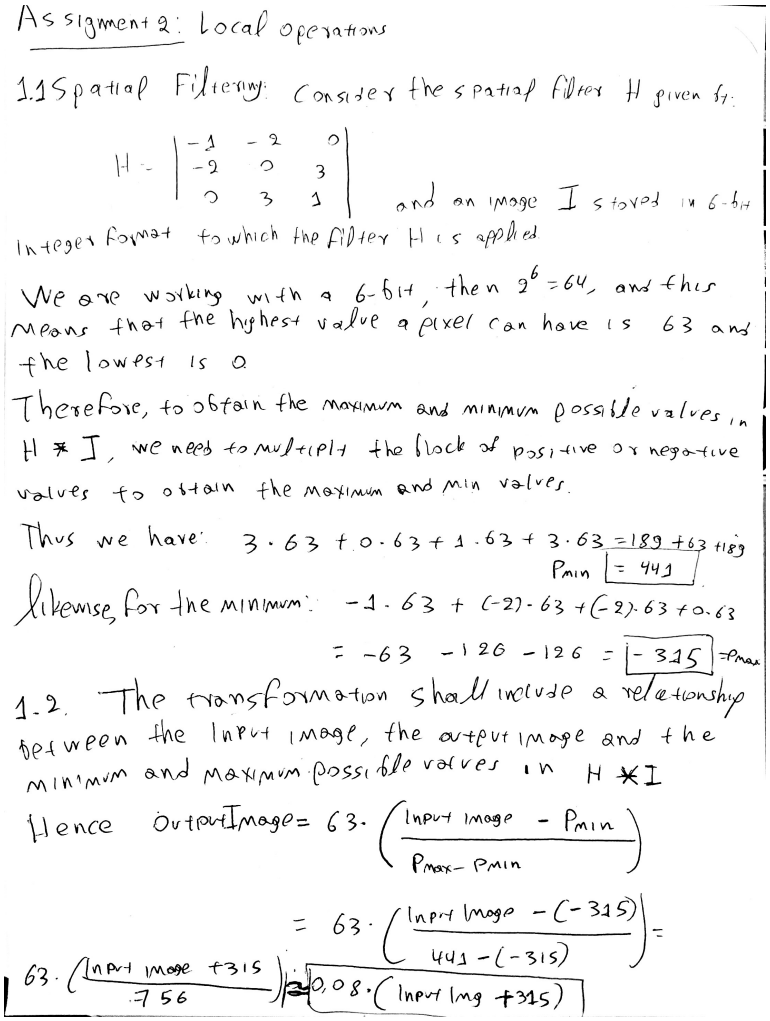
Image Processing - Assignment 2

Kelvin Likollari

1. Spatial Filtering



Assignment 2: Local operations

1.1 Spatial Filtering: Consider the spatial filter H given by:

$$H = \begin{vmatrix} -1 & -2 & 0 \\ -2 & 0 & 3 \\ 0 & 3 & 1 \end{vmatrix}$$

and an image I stored in 6-bit integer format to which the filter H is applied

We are working with a 6-bit, then $2^6 = 64$, and this means that the highest value a pixel can have is 63 and the lowest is 0

Therefore, to obtain the maximum and minimum possible values in $H * I$, we need to multiply the block of positive or negative values to obtain the maximum and min values.

Thus we have: $3 \cdot 63 + 0 \cdot 63 + 1 \cdot 63 + 3 \cdot 63 = 189 + 63 + 189$

$P_{min} = 441$

likewise, for the minimum: $-1 \cdot 63 + (-2) \cdot 63 + (-2) \cdot 63 + 0 \cdot 63$

$= -63 - 126 - 126 = -315 = P_{max}$

1.2. The transformation shall include a relationship between the input image, the output image and the minimum and maximum possible values in $H * I$

Hence $OutputImage = 63 \cdot \left(\frac{Input\ Image - P_{min}}{P_{max} - P_{min}} \right)$

$= 63 \cdot \left(\frac{Input\ Image - (-315)}{441 - (-315)} \right) =$

$63 \cdot \left(\frac{Input\ Image + 315}{756} \right) = 0.08 \cdot (Input\ Img + 315)$

Figure 1: Bonus

BONUS: For a filter to be separable, it must have only one non-zero eigenvalue. Since the filter is non-separable, we can acquire the Singular Value Decomposition method, in order to find an approximation to a filter with the least Frobenius norm error compared to the original filter H. Thus we try to find the lowest Frobenius norm difference between the Frobenius norm of the original filter H, and the approximation to the filter we just derived using the SVD norm. We start off by defining the filter H and computing its SVD decomposition using the **svd()** MATLAB function. Next, we use the rank of the matrix H using the equivalent MATLAB function, to compute the approximation to the filter H. The approximation to the filter is computed by considering the first **r** rows of each matrix. Last but not least, the Frobenius norm error is calculated, which, as mentioned before, is the difference between the Frobenius norm of the original filter H and the approximation we just computed. This way, we have obtained the nearest separable approximation of H, which can be also seen by running the **S**,**V**,**D**, commands in the REPL.

2. Combining linear operations

As seen in class, the unsharp masking procedure involves a plethora of linear operations, which can be represented simply by using a linear filter. For the kernel size and the σ value, as seen in class, the following criterion must hold: **kernelSize = 4 *** σ. The resulting kernel, derived when passed the following σ and γ sizes:

$$\sigma = 1; \text{ kernelSize} = 4 * \sigma; \gamma = 0.29$$

is:

$$\begin{bmatrix} 0.0717 & 0.0510 & 0.0510 & 0.0717 \\ 0.0510 & -0.1739 & -0.1739 & 0.0510 \\ 0.0510 & -0.1739 & -0.1739 & 0.0510 \\ 0.0717 & 0.0510 & 0.0510 & 0.0717 \end{bmatrix}$$

The unsharp masking procedure has been applied to the graz image and the results are clearly depicted below:

Figure 2: Original Graz

Figure 3: Unsharp Masked Graz

3. Morphological Operations A

Given a binary image representing the letter θ, the goal is to get rid of the line in the middle to convert our input image to the desired output, an image representing the number 0. Taking into account the morphological operations we know, we can claim that 2 useful operations that would aid in converting the θ image onto a 0-image, are erosion and dilation. Looking at the definition of those 2 operations, we can see that erosion essentially removes pixels from the borders of an object placed inside an image, and respectively, dilation is used to add some pixels in an object of an image. If we notice closely, we can observe that θ can be converted onto a 0, simply by removing the line in the middle. Therefore, a simple erosion operation is acquired to remove the line in the middle, followed by a dilation operation used to restore the original size. Our goal is to choose a structuring element such that it is the safest structuring element, meaning it does not remove more pixels than required. In our case, we want to remove a line of 1px width, therefore the choice of our structuring element must take into account some parts (borders) of the θ which have a 2px width. So, picking up a structuring element that has a 3px width, will remove pixels (borders) that are necessary to form the θ image. For this reason, we are picking the **c** image, as it does not remove any core pixels (for the design of the θ image). Therefore, we are applying an erosion operation on the original image to remove the line in the middle, and then a dilation operation on the eroded image, so as to restore the original size. However, we can observe that those two operations can be combined into 1, namely the opening operation. The opening

essentially is an erosion followed by a dilation operation, useful for removing relatively small objects from an image while at the same time preserving the shape and the size of the main image object.

4. Morphological Operations B

In this scenario, we are essentially positioning structuring element B over image I. The structuring element has only 1 foreground pixel, placed in some arbitrary position, whereas image I contains both. Suppose we apply erosion to the image I using B. This morphological operation will have some effect on the foreground pixels of I, effects which depend on the structure of B. If a pixel in I is interpreted as a background pixel and it coincides positionally with the unique foreground present in B, then the erosion operation will yield the binary image I to have the foreground pixel set to 1, in the pixel-positions where B and I coincide. In case such a scenario does not hold, and there are positions where they do not coincide, then the pixel value of those positions will be set to 0. It is a fact though that the position of the foreground pixel in B affects the way erosion performs. We have 2 scenarios: 1) In the case where the foreground pixel is located exactly in the center of the structuring element, then the erosion will remove more pixels from the center, while at the same time maintaining more edge pixels. Conversely, if the foreground pixel is located close to the object's edges, then erosion will remove pixels placed around the object's edge.

5. Linear Motion Blur Filter

The anisotropic Gaussian Kernel function has been implemented in a MATLAB file. The code is sufficiently commented on there.

Having implemented the anisotropic Gaussian kernel function, to simulate the motion blur at a 45-degree angle, we need to also define and experiment with some of the parameters of the rotated Gaussian function, such as σ_x, σ_y, and the kernel size. The results are compared below:

Figure 4: Original Graz

Figure 5: Rotated Gaussian Function

The resulting filter is:

$$\begin{bmatrix} 0.0392 & 0.0398 & 0.0400 & 0.0398 & 0.0392 \\ 0.0398 & 0.0404 & 0.0406 & 0.0404 & 0.0398 \\ 0.0400 & 0.0406 & 0.0408 & 0.0406 & 0.0400 \\ 0.0398 & 0.0404 & 0.0406 & 0.0404 & 0.0398 \\ 0.0392 & 0.0398 & 0.0400 & 0.0398 & 0.0392 \end{bmatrix}$$

BONUS: A solution would be to minimize the error rate between the original and the blurred image. The error can be found using the Mean Squared Error applied to the original and blurred images. By definition, the MSE formula contains a relationship between the number of data points, which in our case are the pixels of the image, the observed values, which are the original image, and the predicted values, which are the final blurred image. The formula shall be given by:

$$MSE = \frac{1}{N} \sum_{i=1}^{n} \sum_{j=1}^{m} [O(i,j) - B(i,j)]^2$$

where N is the number of pixels, n is the height of the image, m is the width of the image, O is the original image, and B is the blurred image. The goal is essentially to toggle the kernel size so that this error ratio is minimized. However, we cannot just randomly alter the kernel size, as that might take infinite attempts to minimize the error ratio, and shall be computationally expensive. We can use the Gradient Descent Algorithm (GDA). The GDA algorithm is effectively used to find the minimum value of

a function, which starts from an initial guess and updates the position by going in the negative direction of the function. The algorithm will stop when it reaches a termination criterion, that is the error ratio is minimized and approaches 0, or when the number of max iterations is surpassed. We can take into account the direction of the gradient to change our kernel size accordingly. The least the MSE basically, the closer we are to achieving an optimal solution (an optimal blur that preserves significant image details).

6. Iterative Filtering

As mentioned, applying a Gaussian filter multiple times to an image is the same thing as applying one bigger Gaussian filter. But does the same thing hold for the Bilateral filter? The Bilateral filter takes 2 parameters, namely the degree of smoothing and the spatial sigma. The result of the filter will depend effectively on those 2 parameters.
It is noted that by applying a single big bilateral filter with a low degree of smoothness and spatial sigma, the image does not even get blurred. If we increase those 2 parameters from a value of 2 to a value of 5, we obtain a blurred image, which approximates to a significant extent the blurred image derived using the big Gaussian filter.

However, if we use the same degree of smoothness and spatial sigma in both small and big filter approaches, then the situation is not exactly the same. It is obvious that by doing so, the small Bilateral filter application will generate an even blurrier image, compared to the big filter one. Therefore, if our parameters are set to high values, then our image can get over-blurred, resulting in an aesthetically unpleasing image. If for example, our SpatialSigma and DegreeOfSmoothing are very high, this means that our program takes into account a larger amount of pixels, on which the blur operation will be applied, thereby leading to stronger blur and loss of information detail on many objects present on that image. Concluding, unlike Gaussian filtering, applying a Bilateral filter to an image multiple times is not exactly the same as applying the same filter to the image once. To have almost identical blurring details, the parameters of both techniques need to be toggled and adjusted properly, always depending on the provided image.

Hence this leads us to the conclusion that applying a smaller filter multiple times produces significantly greater blurring results than applying once a bigger bilateral filter.

(P.S: Applying a Bilateral filter to an image (with a greater amount of smoothing and spatial sigma) is computationally more expensive).

Figure 6: Small Filter

Figure 7: Big Filter

To prove that applying two Gaussian filters consecutively to an image is equivalent to applying one Gaussian filter with a bigger σ parameter, the convolution technique will be employed. The proof is shown below:

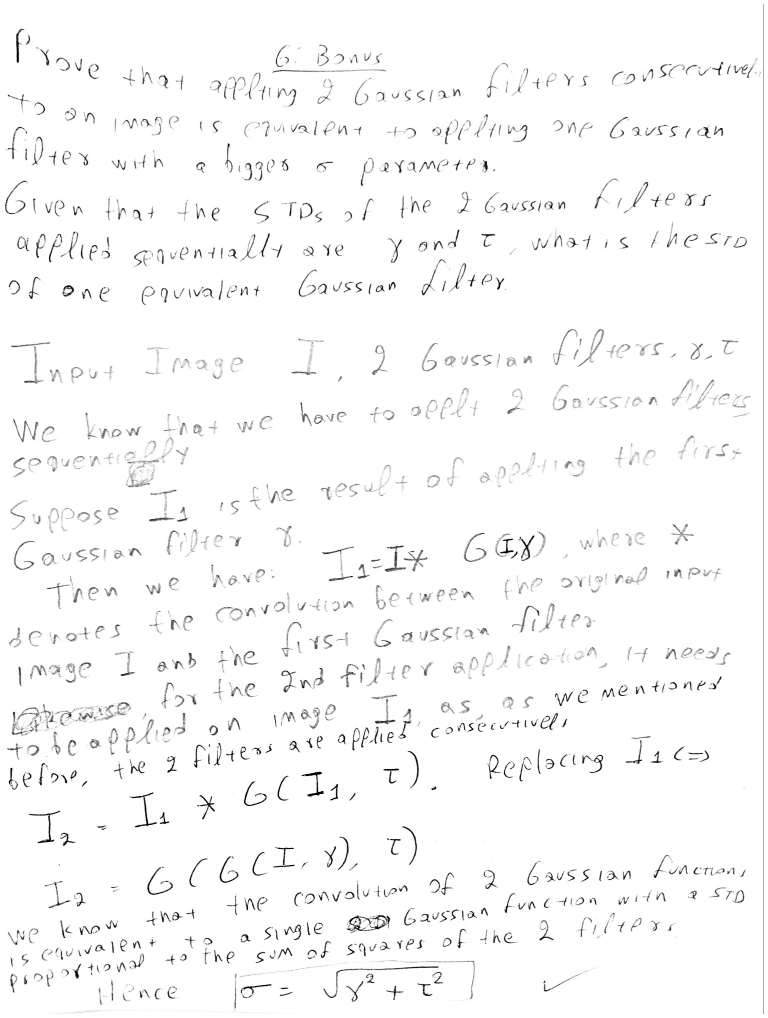


Figure 8: Bonus

A MATLAB script has been written in order to verify that the above scenario is true. The visualization of the two images, one having been applied 2 Gaussian filters and one having been applied a Gaussian filter with a bigger σ is depicted below:

Figure 9: Two Gaussian Filters

Figure 10: Gaussian Filter with bigger σ

7. Image Stylization

Our goal is to effectively implement the two aforementioned techniques. To begin with, the smoothing of the image can be performed using a simple Gaussian Blur. Following, the 2-3 pixel-wide black lines can be added by using the dilation technique, and passing as a structuring element a black line of, as mentioned, 2-3 pixels wide. The results are depicted below:

Figure 11: Original Delicate Arch

Figure 12: Stylized Arch

Effectively, as asked in the exercise description, the script first converts the image onto its grayscale version so that the edge detection is made easier, and applies a Gaussian blur filter onto the image. The blurring is performed in order to smooth the image and remove any possible present noise artifacts, before defining a line as a structural element, which shall be helpful for performing actually the dilation operation which is responsible for making the edges on the image thicker and allowing us to color them black, by assigning them the number 0 (the black color value). In the end, some simple transformation and conversion operations are performed in order to ensure the image is properly outputted, on the right color space.