Eldin Sahbaz

image noise filtering

**The Problem**

The goal of this assignment was to filter images – that contain varying degrees of salt and pepper noise – via mean filtering, Gaussian filtering, and median filtering. In order to accurately evaluate the results of this assignment later in this report, the two images directly below are the initial images in the problem set:



Noisy Image 1

Noisy Image 2

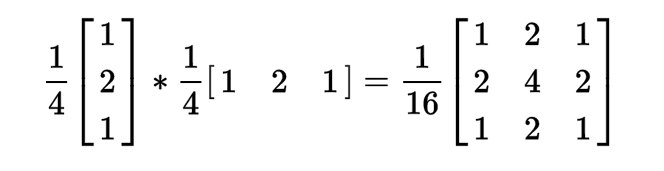
**The Filtering Process**

Mean Filtering

The first and simplest filtering method of the three is mean filtering. The mean filter superimposes a kernel on the image replaces the center pixel with the mean of all the pixels in the neighborhood. The neighborhood is all the pixels in the range of the sliding window. Each pixel in the neighborhood can be assigned different weights. The greater the weight, the greater the contribution of that pixel to the mean. For the weights, I decided to air on the side of simplicity and evenly weigh all pixels in the neighborhood. I accomplished this by creating a kernel of all ones and dividing each element by the sum of all the elements in the kernel. I then used the ‘conv2’ MATLAB function to perform 2-Dimensional convolution with the image and the kernel.

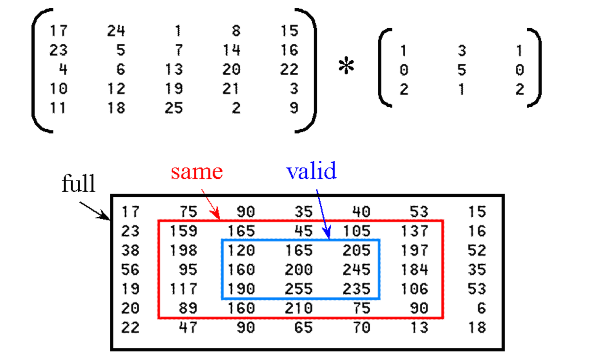
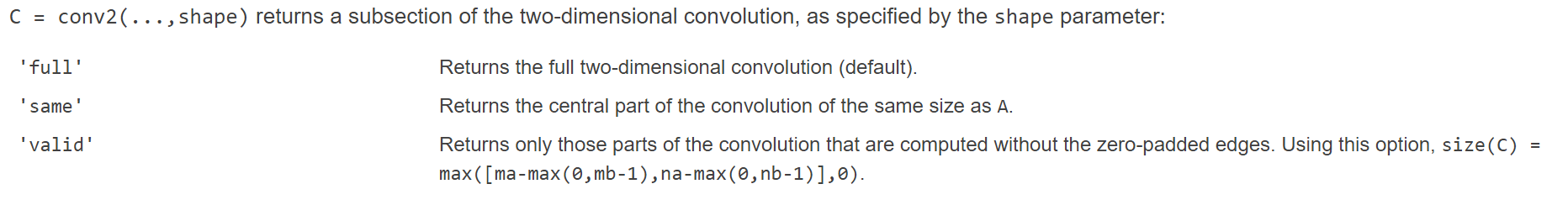
Gaussian Filtering

Gaussian filtering is slightly more complicated than mean filtering. In mean filtering, we have the option to pre-set the values of the kernel, but, in Gaussian filtering, we have to populate our kernel with samples from the normal distribution. Therefore, there are two sub-problems when trying to perform Gaussian filtering. The first sub-problem is to create the Gaussian kernel and the second is to perform the convolution.

Typically, the kernel has N x N dimensions, where N is a non-negative integer. However, to simplify the operation and reduce complexity, the separability of the Gaussian filter is taken advantage of. Therefore, we only need to create a vector which we will later use for convolution. In order to sample points from the normal distribution in MATLAB, we can pass a vector of values to the ‘normpdf’ MATLAB function. This function will sample the normal distribution at each of the points in the given vector. In order to successfully sample from the distribution, we need to specify the mean and standard deviation. For this assignment, I fixed the mean at 0 and left the standard deviation as an input argument to the function that computes the kernel. Once the kernel has been computed, the next step is to then divide each cell by the smallest value in the kernel (so that the smallest values is now one), then – to normalize the kernel – divide the kernel by the sum of all the kernel’s values.

Gaussian Separability

Now we have the kernel with which we can perform convolution. Since we only created a one dimensional kernel, rather than a two dimensional kernel, we have to now make use of the separability feature. In order to make use of this feature, we will convolve the image with the kernel, then convolve the result of the first operation with the transpose of the kernel. Again, using MATLAB’s ‘conv2’ function makes this process simple. All that was required is that we pass the kernel, its transpose, and the image in as arguments.

One thing I was holding out on mentioning until now is that zero padding prior to convolution is not required since MATLAB’s ‘conv2’ function performs this automatically. Looking at the documentation for ‘conv2,’ the ‘full’ returns the output of the convolution (including the areas resulting from zero-padding), ‘same’ only returns the are covered by the size of the original image, and ‘valid’ returns only the sections not computed with zero-padding.

Conv2 example

MATLAB Documentation

Median Filtering

Median filtering is relatively simple and follows intuitively from its name. When performing this type of filtering, we iterate over the image and replace each element’s value with the median value of the mask’s neighborhood. The mask is just a N x N region around a given element, where N is a non-negative integer.

Performing this operation was relatively straight forward. To get all the N x N neighborhoods in the image, I made use of MATLAB’s ‘im2col’ function. According to MATLAB’s documentation, ‘im2col’ “rearranges image blocks into columns.” Therefore, the output of the sliding window ‘im2col’ operation gave me the neighborhood around every cell in the image. Next, we take the transpose of the output in order to make operating on the output easier. To understand why, for an image of 408 x 408 pixels and a mask of 3 x 3 pixels, the output will be 9 x 166,464. From my experience, it’s easier to perform operations on rows than it is to perform operations on columns. Therefore, after taking the transpose, the output is 166,464 x 9. Once the transpose is taken, we can sort each row in descending order and store the medians of the sorted rows in a vector. We can then use this vector to replace their corresponding values in the original image.

As a note, something that I noticed empirically is that if N is an even number in the N x N mask, the image becomes distorted. Therefore, I added logic to increment the size by one if N is an even number. We could have just as easily decremented by one, but then we would have to worry about negative numbers as well.

**The Results**

Mean Filtering

 Looking at the results of the mean filtering operations, it seems that the filtering performance exponentially depreciates as the noise becomes more and more prevalent in the image. This observation makes sense because this filter averages its surrounding to decide the value for each pixel. Therefore, the greater the noise, the more likely it is that the noise will contribute to the calculated pixel value.

Top Row: Noisy Image 1; Bottom Row: Noisy Image 2

Left Column: kernel size is 3; Right Column: kernel size is 5

It also seems to be the case that the greater the kernel size, the more blurred the image becomes. Again, this makes sense because when taking the average, the kernel’s surrounding neighborhood gets summed, meaning the pixels are blended. The greater the area that gets blended, the more blurred the image becomes.

Comparing the four outputs, it seems that the mean filter performs best when the image is not too corrupted with noise and a smaller kernel is used.

Gaussian Filtering

When the image is not too corrupted (i.e. the noise is not too prevalent), the Gaussian filtering operation seems to perform slightly better than the mean filtering operation – but not by much. However, when the noise is stronger and more prevalent, the results are very similar. Therefore, as in the previous section, the Gaussian filtering operation seems to work best when the noise is not too prevalent and the standard deviation is not too high. We can see that as the standard deviation becomes higher, the image becomes blurrier. The only real difference, from what I have observed, between mean filtering and Gaussian filtering is that the weights in the Gaussian kernel follow a normal distribution.

Top Row: Noisy Image 1; Bottom Row: Noisy Image 2

Left Column: σ is 1; Right Column: σ is 5

Choosing a standard deviation for these tests may seem arbitrary. However, I chose a standard deviation of one because the ‘normpdf’ used a default value of one and I chose a standard deviation of five because I wanted to observe the results of a relatively high standard deviation. The important thing to note about choosing a standard deviation is that the chosen width (w) and standard deviation (σ) must satisfy the equation .

Median Filtering

Top Row: Noisy Image 1; Bottom Row: Noisy Image 2

Left Column: kernel size is 3; Right Column: kernel size is 5

 By looking at the outputs of the median filter, it seems that this operation is the most robust and the best at filtering out noise. For the sake of comparison, the kernel sizes of the filters were kept constant across all three filters. With that being said, the most impressive output is the output from Noisy Image 2.

The original Noisy Image 2 is extremely corrupted with noise, but the median filter is able to read through the noise and mostly extract the original image when the kernel has a size of three. When the kernel was increased to a size of five, the noise was virtually non-existent. The only downside to this operation is that there is some blurring, but even when blurring occurs, the edge strength is still retained. Therefore, there is much more useful information that can be extracted from these images after filtering than in the outputs of the other two filtering methods.

The median filter seems to be able to handle filtering with virtually any noise level and with a reasonable kernel size (depending on the amount of noise). The less corrupt an image, the smaller the kernel size should be. The more corrupt an image, the greater the kernel size should be.

Best Filtering Operation

Not every filtering method should be used in every situation. Some methods are more impervious to outliers whereas other are more sensitive. In this case, the median filter is more robust than the Gaussian filter and mean filter when it comes to handling heavy noise. Therefore, of the three filtering operations explored in this assignment, the most effective method, empirically speaking, must be median filtering. Median filtering performed best in both low and high noise environments and was able to keep the edges the most sharply defined – even when blurring occurred.

**Sources**

* <https://www.mathworks.com/help/images/ref/im2col.html>
* <http://www.johnloomis.org/ece563/notes/filter/conv/convolution.html>
* <https://en.wikipedia.org/wiki/Separable_filter>
* <https://www.mathworks.com/help/matlab/ref/conv2.html>