Pouyan Pourmirjafari

Spourm2

Machine Problem #1

1.

To approach the problem, I programmed the functions based on their formulas using matrices and making default kernels. First the kernels were made, then I used a 2d array to test matrix manipulations such as traversing and multiplications. Afterwards, using familiar libraries, I uploaded the images to perform the functions on. Using a python notebook made it simpler to output testing results. Also the Jupyter IDE gives you the freedom to work on different blocks without the need to run all the program. I used for loops to mimic the matrix manipulation.

The functions can take kernel of any size and output will be in the same size as input image.

Convolution: Flipped the kernel horizontally and vertically then traversed it through the image based on the kernel size to get new values for every 3 layers of the image since the shape of all images were (x,x,3). By using a copy of the image with the right padding the kernel makes an output image with the same size as the input image.

Correlation: Same as the Convolution but without flipping the kernel.

Median Filter: Based on the kernel_size we choose a section of the padded image with the same shape as kernel. Then by using flatten and sort function we find the median in the kernel then replace the pixel in the image. I used both odd number kernels and even number kernels.

2. Packages

OPEN_CV was used to upload images to the program and change their color formation. However, two methods were used to display the images on the screen.

The file named "Matplot_MP#1" shows images using the MATPLOT library. To be able to use this library, CVT_COLOR function was used to transfer the BGR order to RGB for the plot. Matplot was initially used because it didn't need any popup windows.

The file named "CV2_MP#1" shows images using the CV2 library. Although the user has to press 'Space' to skip over every image files, this is easier to detect differences.

NUMPY is used to do all matrix manipulation and multiplications. Using the sum() and Flatten() method in Conv and Median functions respectively. Numpy is also used to create the 3x3 and 4x4 kernels. The MATH library was used to create the gaussian value.

3. **Difficulties**:
    a. The sharpening kernel resulted in excessive amounts over 255 for different colors. To overcome the problem, I used the **method numpy.clip(x, min, max)**. This method automatically puts a range on the values. If a value exceeds 255 it is set to 255 and if it goes below zero it is set to 0. Overall the value cannot go beyond the min & max range. This solved the problem for excessive discoloration.
    b. The Gaussian function had to form from the middle of a matrix, meaning the center of the matrix is considered (0,0). To overcome this issue, I used nested for loops to iterate over any given size for gaussian function populating the matrix from -x/2 to +x/2.
    c. Using MatPlot as my output library resulted in discolored images. This is due to the color orientation in OPEN_CV library which is BGR. Therefore the cvt.Color() method was used to change the image Color Space. However, in order to save the image the color space had to be changed back to original for every single image.
    d. Made generic functions to make Gaussian and Mean kernels of any size. Also, the convolution and correlation and median methods are all adjusted to work with any square kernel of add sizes. Since even size kernel was not discussed during lecture the results of even kernels do not satisfy requirements.
    e. The code for matrix multiplication and traversal is generic for both colored and grayscale images since all images had three channels when uploaded through OPEN_CV. Therefore, three nested loops had to be used to traverse the matrix. Same as Kernel functions, Correlation and Convolution perform better using a odd numbered matrix.

.
.
.
.
.
.
.
.

4. **Results**:

   Using Convolution with Mean Kernel 3x3:

   Using Convolution with Mean Kernel 5x5 : results in more fading and slightly less noise

Using Correlation with Mean Kernel 3x3

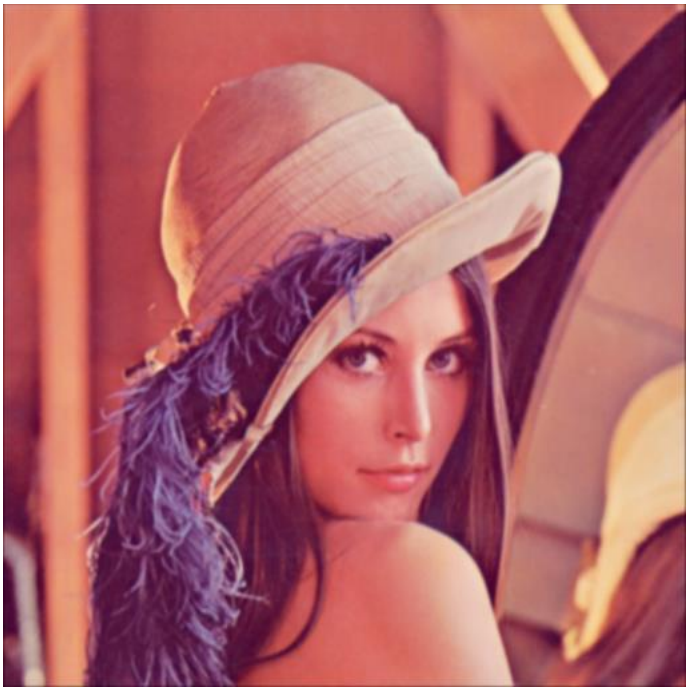
Using Correlation with Mean Kernel 5x5 : results in more fading and slightly less noise

Using Median Filter 3x3

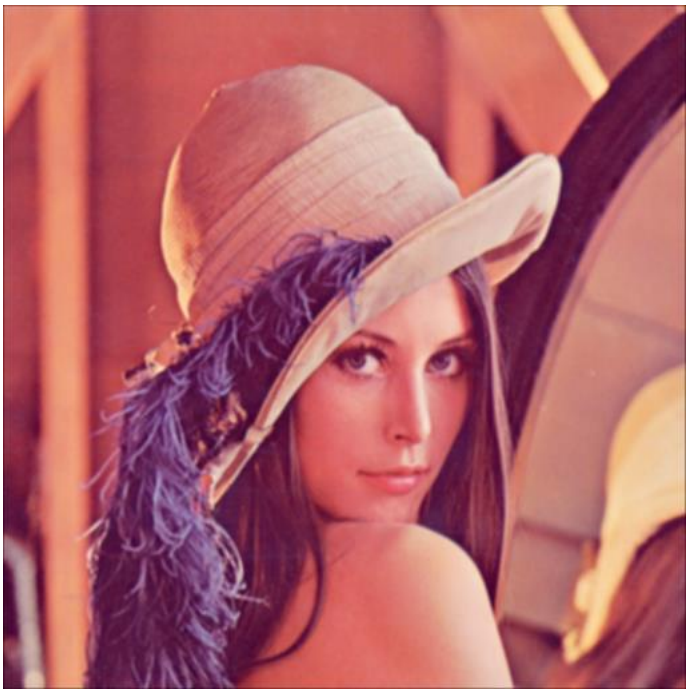
Using Median Filter 5x5 : results in more fading

Using Gaussian 3x3 with Sigma = 0.5

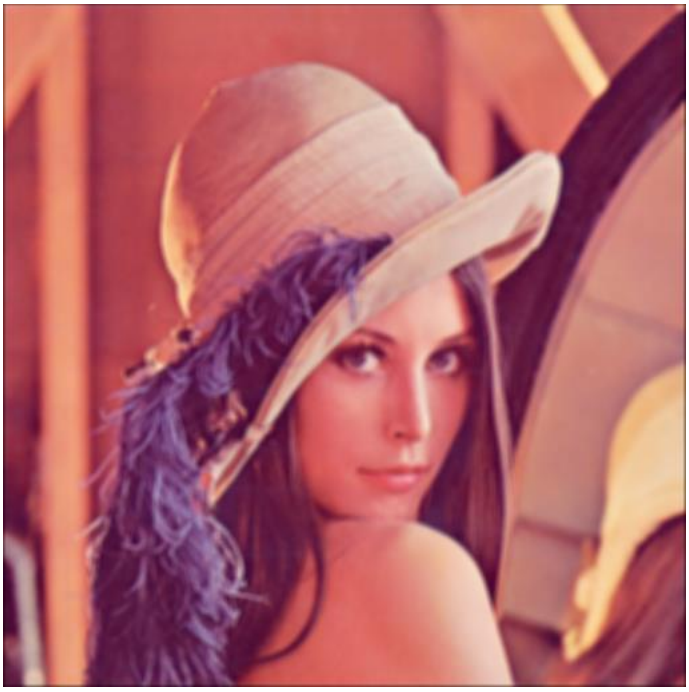
Using Gaussian 5x5 with Sigma = 1.0 : Results in better fading with bigger radius slight darkness.



Using Mean Kernel 3x3:



Using Mean kernel 5x5 : results in more fading effect. There is no noise.



Using Sharpening Kernel 3x3 [0,2,0] – 1/9[1,1,1] :

Using Sharpening Kernel 5x5 [0,0,2,0,0] – 1/25[1,1,1,1,1] : More sharp edges and colors are more distinct. Some parts might be lighter or darker than original due to range.

Expectations:

Throughout sharpening matrix manipulation I used (mod) %255 to map numbers to 0-255 however, the extra green pixels still pop up.

Reality:

I had to make sure the number doesn't exceed 255 or go below 0. This results in no excessive amount of green or any other color in specific pixels. Since mod() would lower other values not needed to be lowered, range was the better option.

Expectations:

Mean Kernel results in less noise and blurring. Gaussian would blur better than Mean.

Reality:

Mean Kernel only blurs the noise however the noise is still visible in the faded, blurred image. Gaussian 3x3 blurred the image though not as much as mean kernel 3x3 did. However, the color was also a bit darker than expected in Gaussian filter. By altering Sigma and the kernel size Gaussian gave better results in fading