Convolution Algorithm:

def convoFilter(img\_in, kernel):

    data = np.array(img\_in) #This converts img to data array

    #Height = shape[0]; Width = shape[1]

    imgHeight = data.shape[0] #Height of original Image

    imgWidth = data.shape[1] #Width of original Image

    kernHeight = kernel.shape[0]

    kernWidth = kernel.shape[1]

    modified = np.zeros\_like(data) #Use zero\_like since we want the same size matrix as image. This is the output array

    #Algorithm:

        #Flip the kernel 180

    horiFlip = np.fliplr(kernel) #Flip Horizontally

    kernel = np.flipud(horiFlip) #Flip Vertically

    #Zero Padding

zeroPad = np.zeros((imgHeight+2, imgWidth+2)) #Pad the array with -> Make a NxN into N+2xN+2

    zeroPad[1:-1, 1:-1] = data

#Put the data of the image in the center of the padded array

        #Traverse through Pixel

    for i in range(imgHeight):

        for j in range(imgWidth):

            modified[i, j] = np.sum(kernel \* zeroPad[i: i+kernHeight, j: j+kernWidth])

    #print(modified) #Prints data matrices to console

    return modified

This algorithm takes in one of the image channels and alters it. First it converts is into data array then it gets the shape of the array. By definition of convolution, I flipped the kernel 180 degrees and then zero padded the array (zeroPad). I added 2 to the height of the image and 2 to the width of the image, this a 3x3 shape will become a 5x5 with padding. After that, I traverse through the image pixel and applied the equation by doing the sum of kernel multiplied the zero padded height and width array. I set it equal to the “modified” array where it is set the same dimension as original image. This in result is similar doing sum of dot product.

Correlation Algorithm:

def correFilter(img\_in, kernel):

    data = np.array(img\_in) #Converts img to darray

    imgHeight = data.shape[0] #Height of original Image

    imgWidth = data.shape[1] #Width of original Image

    kernHeight = kernel.shape[0]

    kernWidth = kernel.shape[1]

    modified = np.zeros\_like(data)

    #Algorithm:

        #Zero Padding

    zeroPad = np.zeros((imgHeight+2, imgWidth+2))

    zeroPad[1:-1, 1:-1] = data

        #Traverse through Pixel

    for i in range(imgHeight):

        for j in range(imgWidth):

            modified[i, j] = np.sum(kernel \* zeroPad[i: i+kernHeight, j: j+kernWidth])

    return modified

This is very similar to the convolution algorithm. The only thing that is different is that I did not do a 180 degrees rotation on the kernel. Everything else is the same as convolution algorithm (above).

Median Algorithm:

def medFilter(img\_in, k\_size):

    data = np.array(img\_in)

    imgHeight = data.shape[0]

    imgWidth = data.shape[1]

    window = []

    modified = np.zeros\_like(data)

    pointer = m.floor(k\_size/2)

    #Algorithm:

    for x in range(imgHeight):

        for y in range(imgWidth):

            for z in range(k\_size):

                if (x + z - pointer) > (imgHeight - 1) or (x + z - pointer) < 0:

                    for i in range(k\_size):

                        window.insert(0,0) #Insert Zero into the Window

                elif (y + z + pointer) > (imgWidth - 1) or (y + z - pointer) < 0:

                    window.insert(0,0) #Insert Zero into the Window

                else:

                    for j in range(k\_size):

                        window.insert(0, data[x + z - pointer][y + j - pointer])

            #Median

            window.sort()

            modified[x][y] = window[m.floor(len(window)/2)]

            window = []

    return modified

This was a bit complicated, so I had look at some guide from wiki and the textbook. The resources for the wiki I used are at the last page of this report.

Basically, I first converted the image into data array like the other two then I created the window array that will allow me to get the median of that window. I also had to change how I did the zero padding from the previous since numpy did not allow me to insert the way I wanted it. In general, I traverse through the image pixel and each time I did a sort on the window array and find the median of that window array.

Splitting Channels:

def blueChannel(img\_in):

    image = cv2.imread(img\_in)

    b, g, r = cv2.split(image) #Splits image into 3 channels

    return b

def convolution(img\_in, kernel):

    blueImage = blueChannel(img\_in)

    blueConv = convoFilter(blueImage, kernel)

    greenImage = greenChannel(img\_in)

    greenConv = convoFilter(greenImage, kernel)

    redImage = redChannel(img\_in)

    redConv = convoFilter(redImage, kernel)

    result = cv2.merge((blueConv, greenConv, redConv))

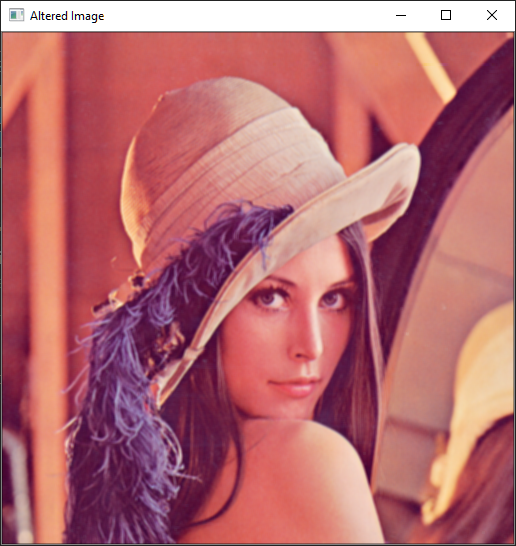
    return result

To save space I only showed the blue channel and the convolution in this report. Other two channels are the same as the blue channel, I just return green and red instead. The other two filter (correlation/median) are the same as the convolution. I just renamed it accordingly.

This is how I did split RGB into 3 different channels. I first split it up into 3 channels and run each channel through the filter algorithm individually after that I merge the 3 channels together and return the result.

Convolution [Lena]:

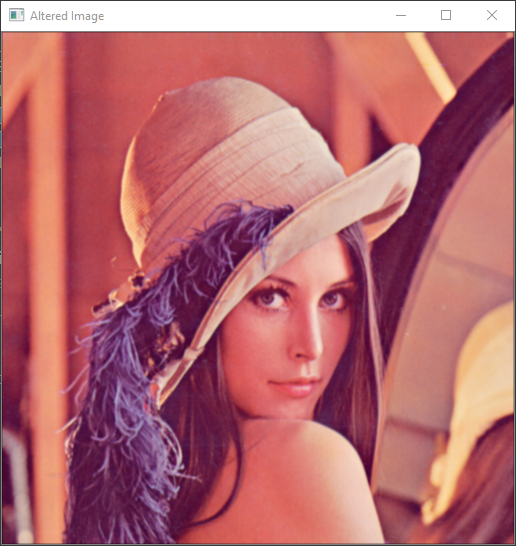
Mean 3x3: ([[1, 1, 1], [1, 1, 1], [1, 1, 1]])/9



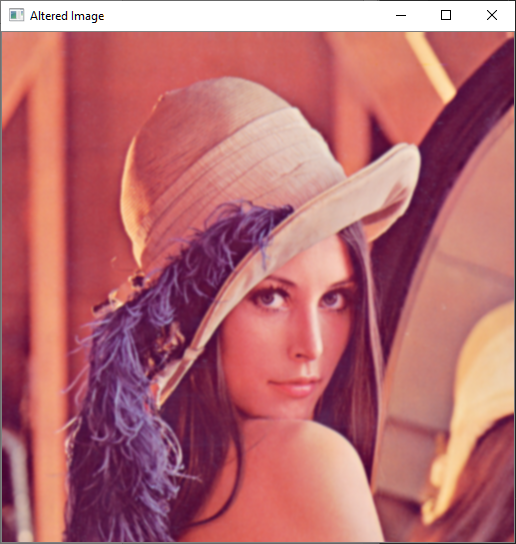
Mean 5x5: ([[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]])/25



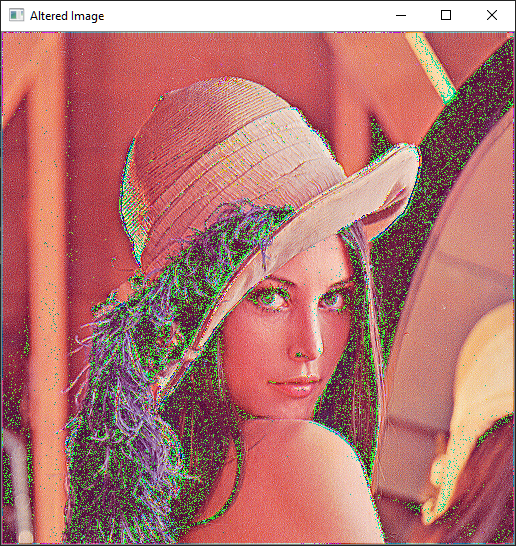
Gaussian 3x3: ([[1, 2, 1], [2, 4, 2], [1, 2, 1]])/16



Gaussian 5x5: ([[1, 4, 6, 4, 1], [4, 16, 24, 16, 4], [6, 24, 36, 24, 6], [4, 16, 24, 16, 4], [1, 4, 6, 4, 1]])/256



Sharpening: ([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])



Convolution [Art]:

Mean 3x3: ([[1, 1, 1], [1, 1, 1], [1, 1, 1]])/9



Mean 5x5: ([[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]])/25



Correlation [Art]:

Mean 3x3: ([[1, 1, 1], [1, 1, 1], [1, 1, 1]])/9



Mean 5x5: ([[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]])/25



Median Filter:

Kernel Size: 2



Kernel Size: 3



Kernel Size: 4



Resources Used:

<https://en.wikipedia.org/wiki/Kernel_(image_processing)>

<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html>

<https://en.wikipedia.org/wiki/Median_filter>