ECE 558 - Project 1
Alex Carter
agcarte4

Section 1: Two-dimensional Convolution

1. Implementation

My two dimensional convolution function (conv2) follows a few steps to perform a two dimensional convolution between a provided image and kernel with a particular padding type. To begin with the image is padded by utilizing NumPy padding routines. Once the image is padded, the function iterates over all pixels and performs the convolution between the section and provided kernel. If the image is RGB, then this step is extrapolated to the three channels and the results are recombined into the final image.

To call this function (please see the provided python source file), it is necessary to pass in an image, a kernel of class Kernel (see Kernel in source), and a padding type ('clip', 'wrap around', 'copy edge', 'reflect'): `result = conv2(lena_grey, w=BOX_3, pad='clip')`. It will return an image that is the same size as the original image.

I must note that I implemented a custom class to handle the filters, which I called class Kernel. This class creates an object from filter matrix and scalar. I created several, including the ones required for the project, here is an example: `ID_FILTER_3 = Kernel(scalar= 1, matrix= np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]], dtype= int))`
This is the identity kernel, when applied to an image, nothing should change. Here are the others that I created:

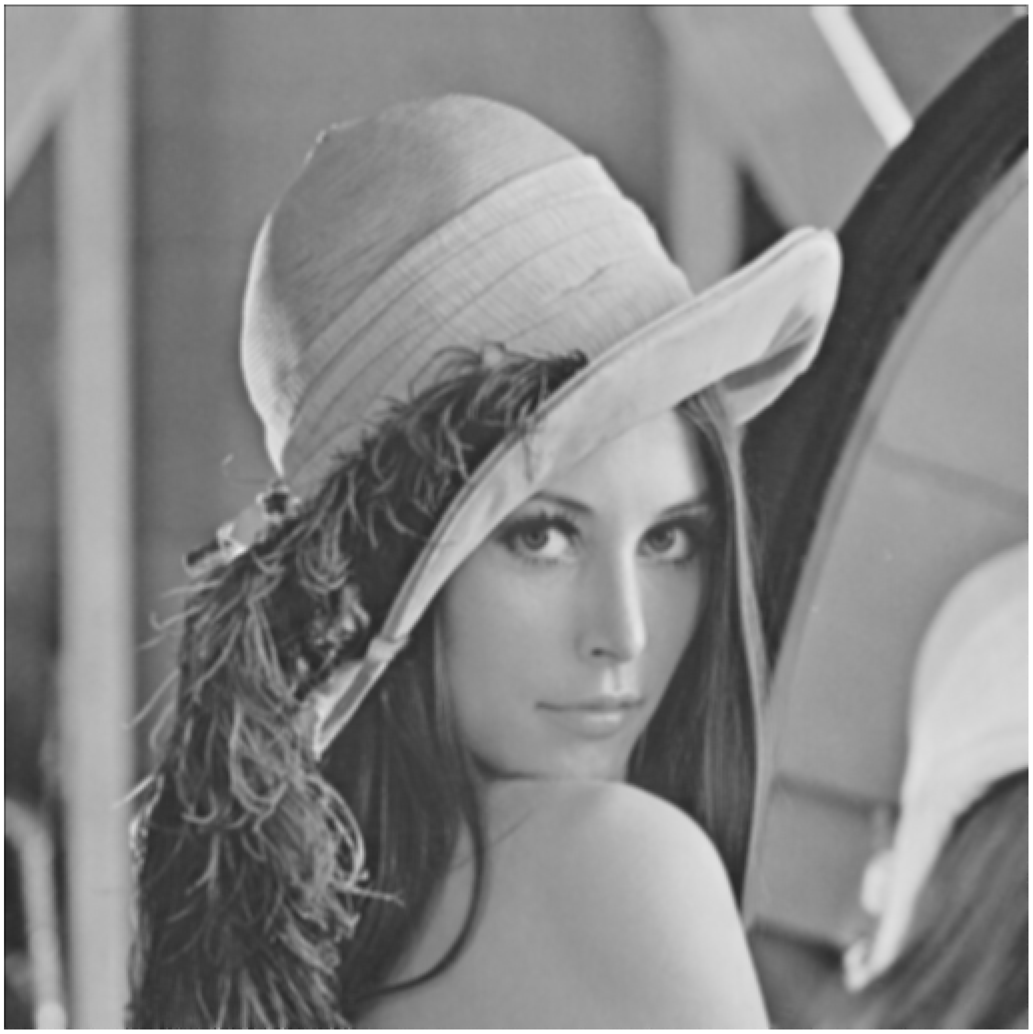
```
# # # Filters
# ID_FILTER_3   - 3x3 Identity Filter
# BOX_3         - 3x3 Box Filter
# FO_DX         - 2x1 First-order X Derivative Filter
# FO_DY         - 1x2 First-order Y Derivative Filter
# PREWITT_X     - 3x3 Prewitt X Filter
# PREWITT_Y     - 3x3 Prewitt Y Filter
# SOBEL_X       - 3x3 Sobel X Filter
# SOBEL_Y       - 3x3 Sobel Y Filter
# ROBERTS_X     - 2x2 Roberts X Filter
# ROBERTS_Y     - 2x2 Roberts Y Filter
# SHARP_3       - 3x3 Sharpening Filter
# GAUSS_BLUR_3  - 3x3 Gaussian Blurring Filter
# GAUSS_BLUR_5  - 5x5 Gaussian Blurring Filter
```

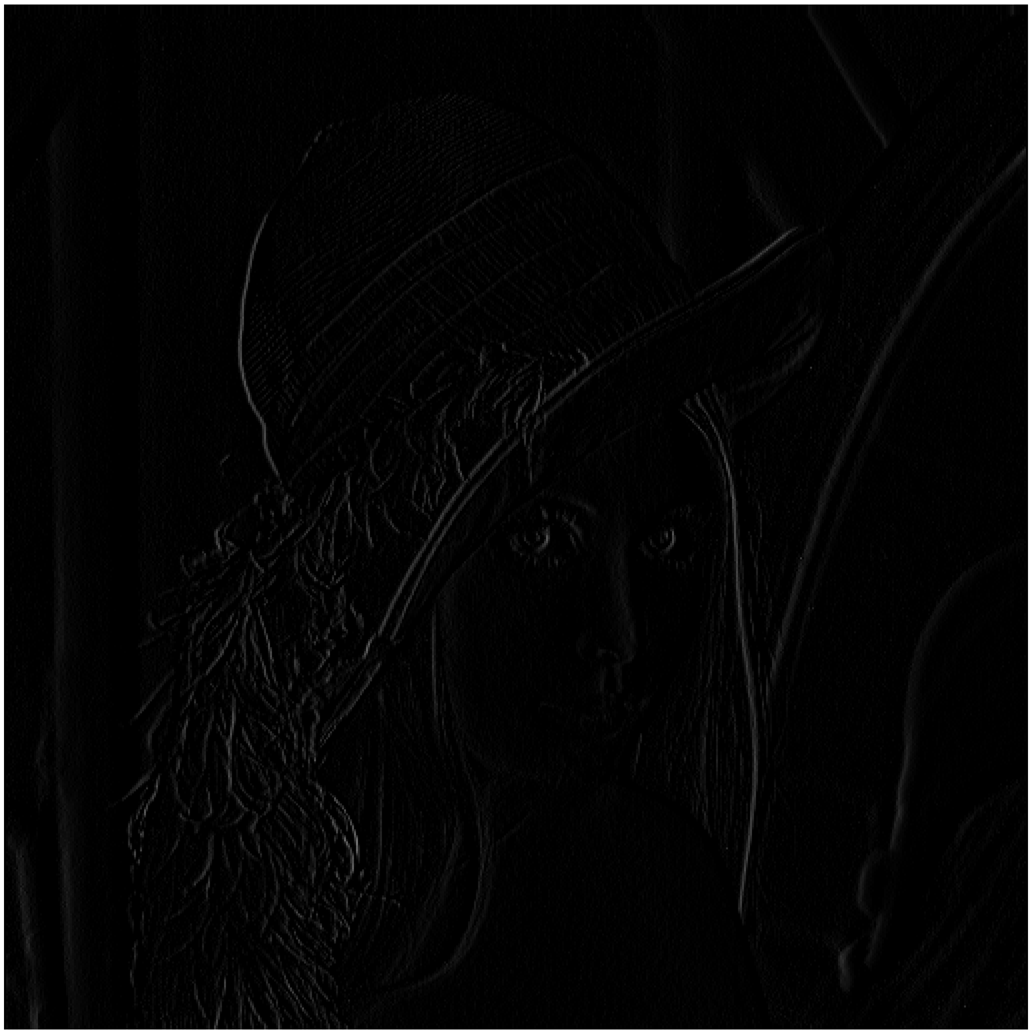
For more examples on how to use my implementation, I created a function called `generate_filtered_images()` that generates all possible filter and padding combinations for the 'Lena' image.

2. Different Filters and Results

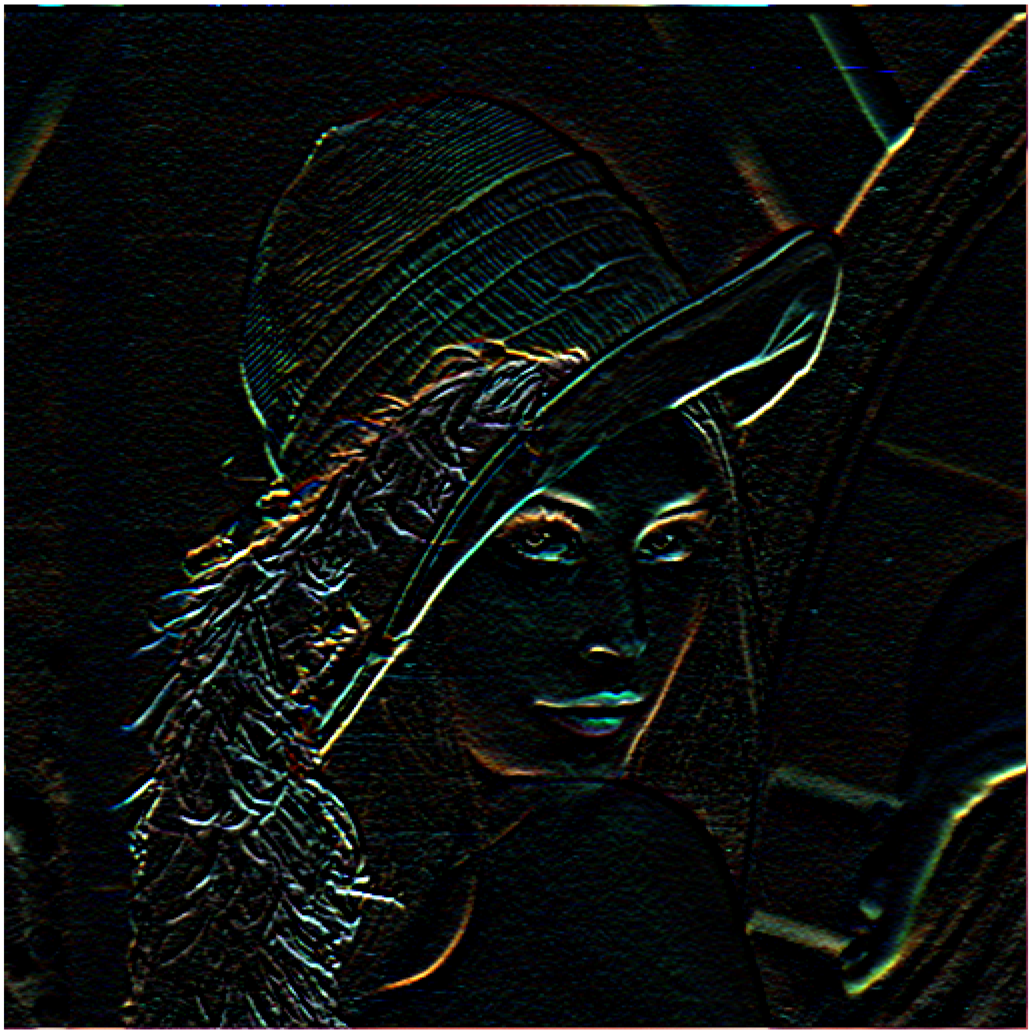
Here are some of the results:

ECE 558 - Project 1
Alex Carter
agcarte4

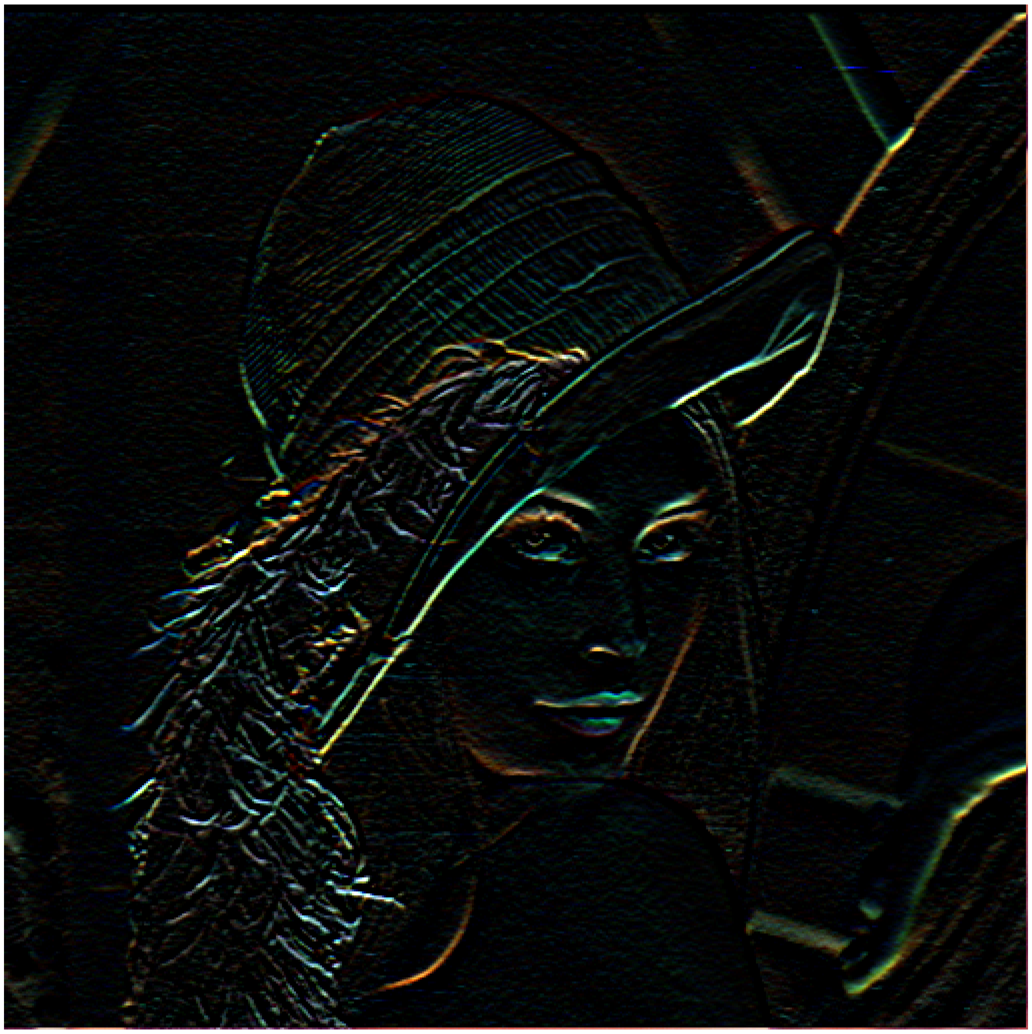

Lena in grayscale with ‘clip’ padding and box filter



Lena in grayscale with ‘copy edge’ padding and first-order Dx filter



Lena in RGB with ‘wrap around’ padding and Sobel Y filter


Lena in RGB with ‘copy edge’ padding and Prewitt Y filter

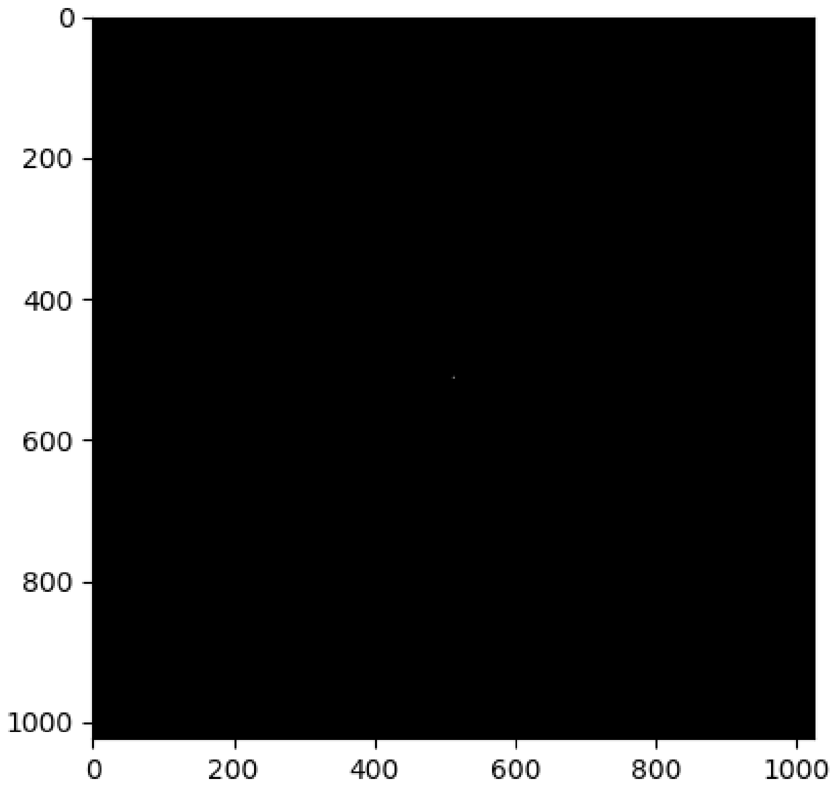
In total there are 80 images for Lena alone. It is important to note that it is hard to tell the differences that the padding types cause due to the small size of the filters when compared to the overall image size. For most filters, the padding is 1 pixel in any given direction which has minimal effect on the edge pixels. The results are separated into folders called ‘lena_grey_convs’ and ‘lena_rgb_convs.’

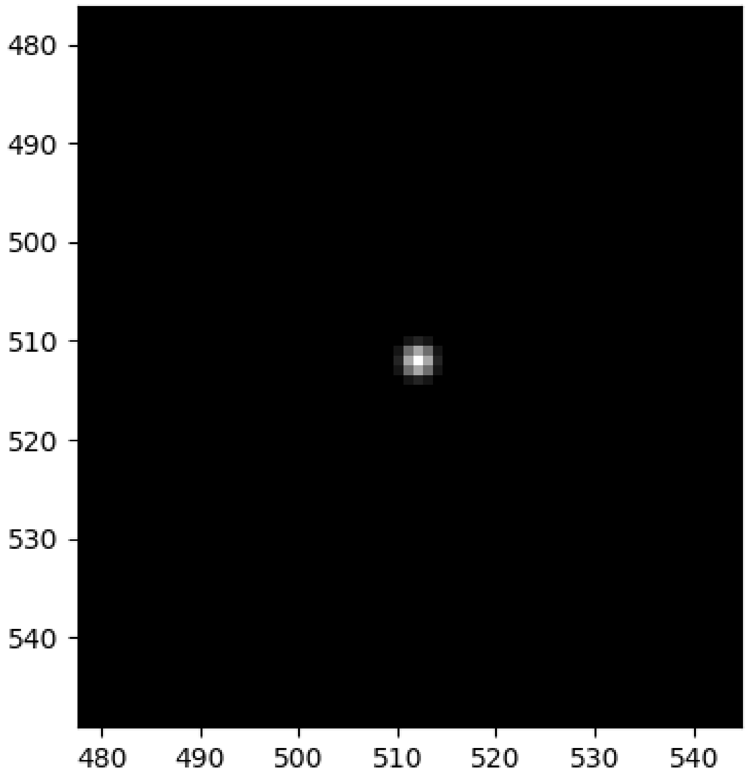
3. Unit Impulse Result

This is a fairly simple task, I created a few functions to accomplish this task:

```
def unit_impulse_image(shape = (1024,1024)) -> ndarray:
    unit_impulse = np.zeros(shape)

    x_cen = int(len(unit_impulse)/2)
    y_cen = int(len(unit_impulse[0])/2)

    unit_impulse[x_cen, y_cen] = 255
    return unit_impulse

def unit_impulse_conv2(impulse, w) -> ndarray:
    out = conv2(impulse, w, pad='clip')
    cv.imwrite(filename='C:/Users/bravo/Desktop/558/proj1/unit_impulse_conv.png', img= out)
    return out

def unit_impulse_test() -> None:
    unit = unit_impulse_image()
    result = unit_impulse_conv2(unit, w=GAUSS_BLUR_5)
    plt.show(imshow(result, cmap='gray'))
```

I used my own 5x5 Gaussian blurring kernel since this was the largest kernel and would be the easiest to see in the output.





Impulse image convolved with 5x5 Gaussian kernel

It is hard to see, but the second image is zoomed in and it is much more distinguishable. This shows that my two dimensional convolution function works as the output of the unit impulse image with this kernel is the kernel itself in the center of the image. Since the image is a unit impulse, there is only one pixel value that the kernel can be applied and that is the center pixel.

Section 2: Two-dimensional Discrete Fourier Transform

1. Implementation

Although I have a working function, the output is not 100% correct when compared to NumPy's fft2 routine. I am unsure why my implementation doesn't produce the same results, I figure that I am missing a step somewhere. Anyways, I decided to use NumPy's built-in 1-Dimensional FFT function to create my own DFT2 function. NumPy's function performs the Fast Fourier Transform on a provided sequence and returns the result. To create my own DFT2 implementation, the algorithm is as follows:

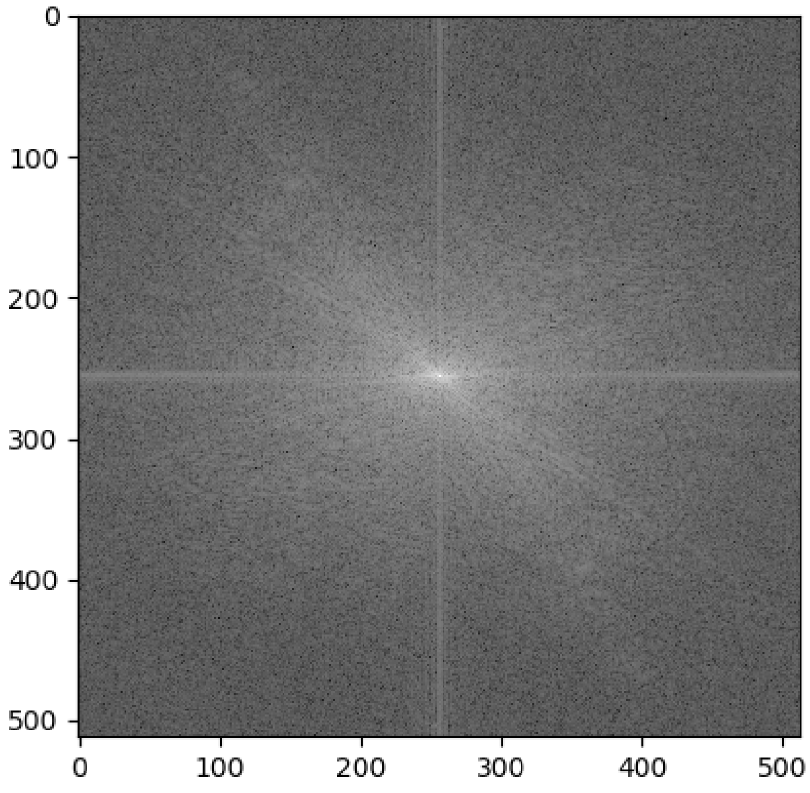
   i. Perform FFT on each image row
   ii. Perform FFT on the each column of previous result

Here is my own implementation as found within my provided source code:

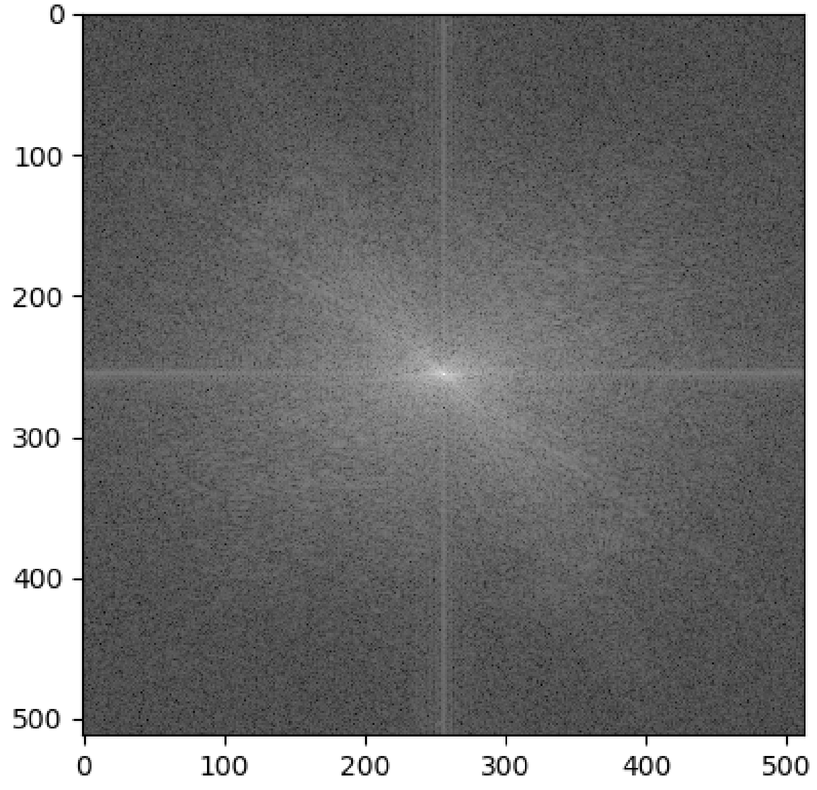
```
def DFT2(img) -> ndarray:
    fft_img = np.zeros(img.shape, dtype=complex)

    # Perfrom 1d DFT of each row
    for r in range(0, len(img)):
        fft_img[r,:] = fft(img[r, :])

    # Perfrom 1d DFT on each column using previus results
    for c in range(0, len(img[0])):
        fft_img[:,c] = fft(fft_img[:,c])

    return fft_img
```

As stated, I may be missing a step as my results were never exactly the same as the results returned from NumPy's fft2 function.

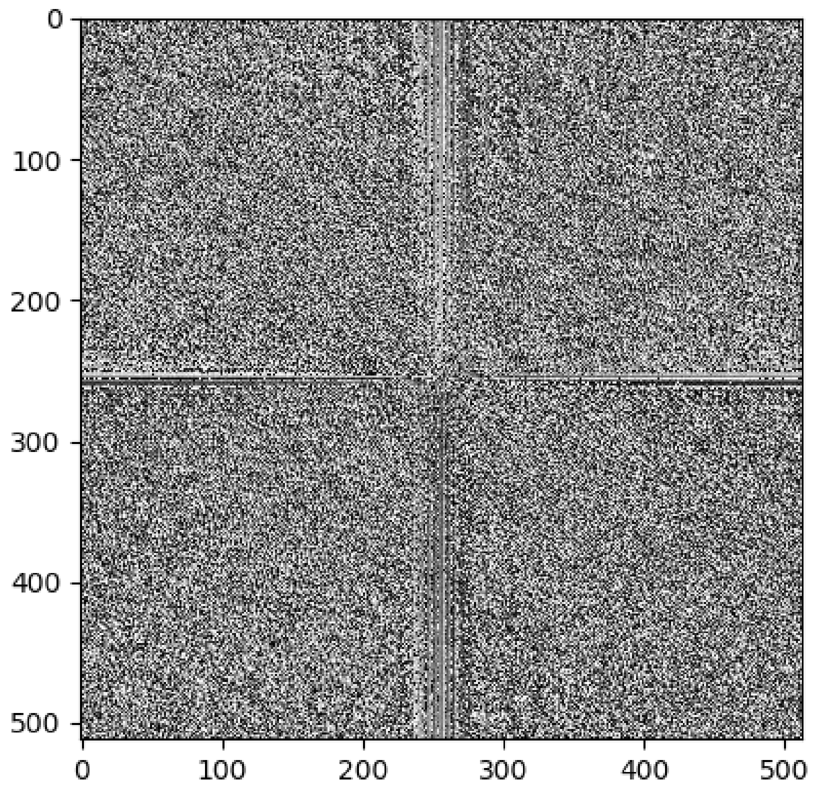
2. DFT2 Results
   I followed the steps that were outlined in the project to obtain the following results:



My DFT2 of Lena - amplitude



NumPy’s fft2 of Lena - amplitude



My DFT2 of Lena - angle

Although the amplitude images look almost identical, my result appears to be whitewashed. Like I said, I am unsure as to why this is the case.

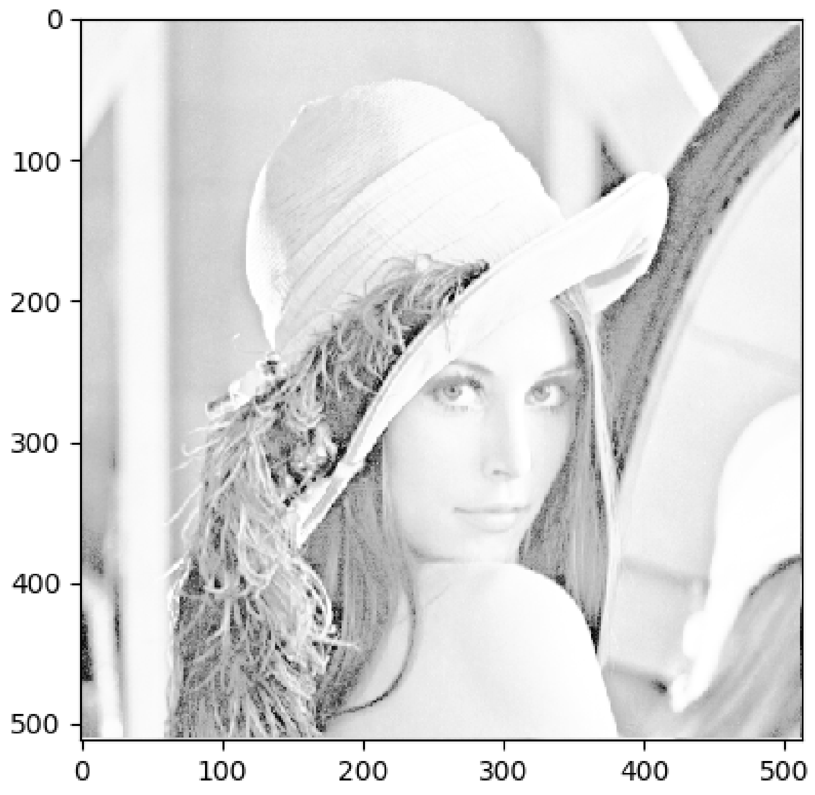
3. Inverse DFT2 and Results

Theoretically, the inverse DFT is just the DFT of the DFT multiplied by a scalar.

```
def IDFT2() -> ndarray:
    lena_fft, = generate_fft(False)
    lena_ifft = (1 / len(lena_fft)) * (DFT2(lena_fft))
    return np.rot90(np.rot90(np.log(1+np.abs(lena_ifft))))
```

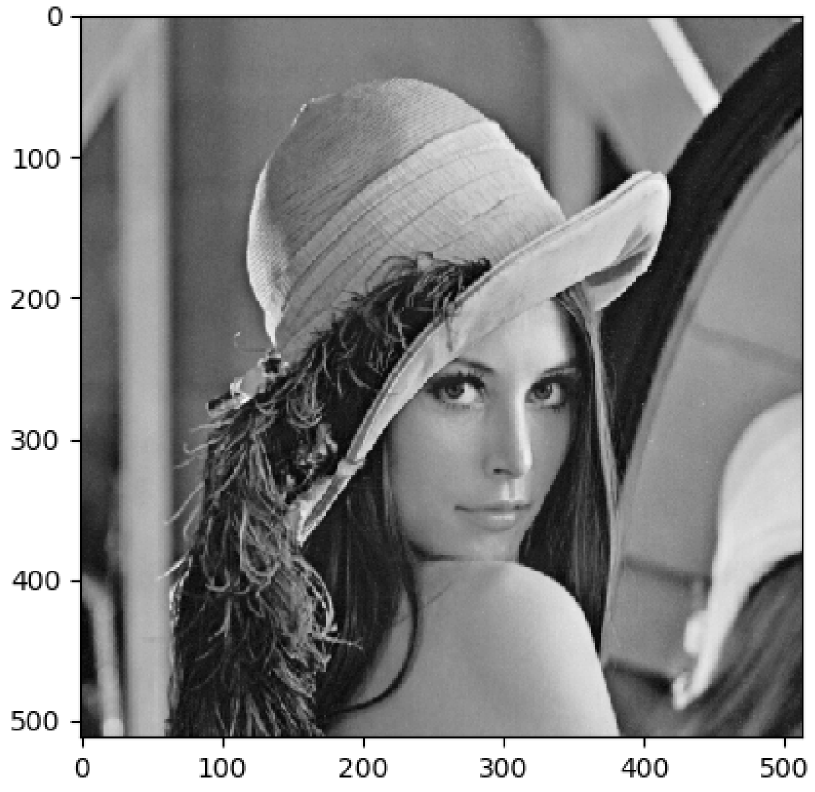
ECE 558 - Project 1
Alex Carter
agcarte4

Above is my function, please note that my result was rotated 180 degrees when compared to the original image. This might stem from the fact that my implementation is not perfect.



My IDFT2 result

ECE 558 - Project 1
Alex Carter
agcarte4



Difference of Original and my IDFT2 result

As seen above, my IDFT2 result is not correct. It is whitewashed when compared to the actual result. Interestingly, when the difference is taken, the resulting image appears to be sharpened. I created a function below to compare the results:

```
def compare_fft_ifft() -> None:
    # True image
    lena_grey = cv.imread("C:/Users/bravo/Desktop/558/proj1/lena.png",
cv.IMREAD_GRAYSCALE)
    lena_ifft = IDFT2()

    plt.show(imshow(lena_ifft, cmap='gray'))

    plt.show(imshow(lena_grey-lena_ifft, cmap='gray'))
```

Note: I was unsure how to organize the structure of my code, if there are any questions please reach out to me.