ECE-558 Project 1 Report

Derik Fernando Muñoz Solis

The purpose of this project was to implement some common image processing techniques including image padding, convolution with different kernels as well as some image manipulation in the frequency domain by taking the 2d Fourier transform and the 2d inverse Fourier transform of an image.

1 Padding and Convolution

1.a Creating and implementing a 2d convolution function

For this problem the task was to create and implement a $g = conv2(f, w, pad)$ this function also needed a *pad* function to be created and implemented from scratch so that it could help us do the actual convolution function.

The padding function was implemented by creating the appropriate "chunk" that will pad the respective side of the image and then stacking the chunks together with the original image to produce the padded image. The padding function was then checked against the numpy.pad function from the numpy python module for correctness.
The types of padding that were implemented were:

- Zero padding
- Copy edge
- Wrap around
- Reflect across edge

Once the padding function was made the next step was to create the convolution function this function was created by first padding the image according to the type specified by the third parameter to the *conv*2 function. Once the image was padded the next step was to slide the kernel across the padded image and computing the sum of the kernel and the windowed values of the padded image. Once this convolution function was completed the function returned the convolved image.

For graycale images, the convolution was simply applied to the only channel in the image but for color images the padding and convolution is done for each of the three channels independently and then at the end the image is recreated by stacking the 3 channels depth-wise.

These functions were then tested with the *lenna.png* and *wolves.png* images loaded as both grayscale and color (BGR) images and with the 9 different kernel values specified by the project description.

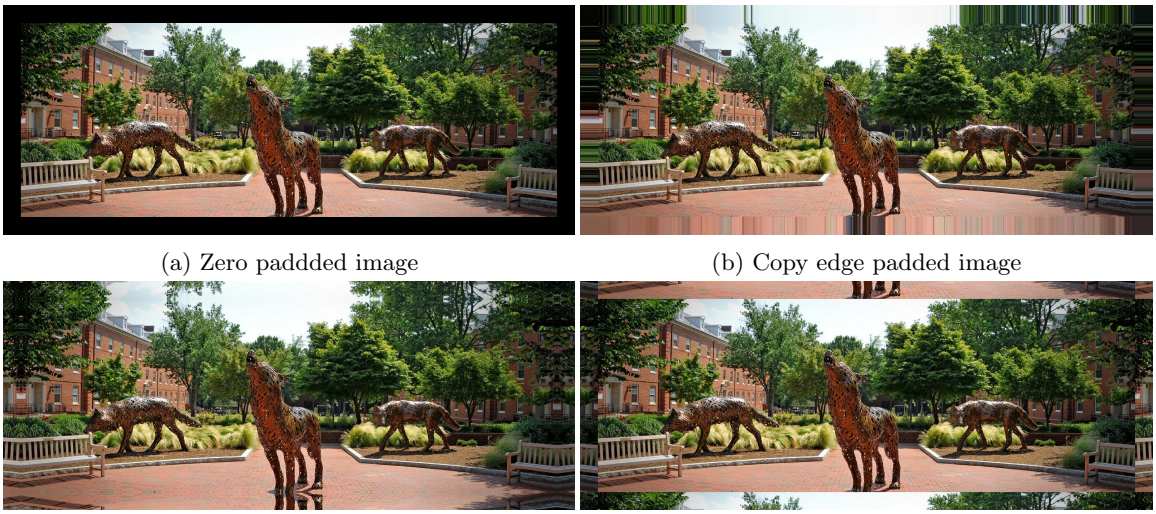
1.a.i Testing process

In order to test the function a main driver function was created that takes in parameters from the command line to specify the input image, kernel, wheter to load the image as a grayscale or color

image and where to save the output. The process was then to test with two different images to make sure the convolution works for any type of image with any dimensions and with both graycale and color images. The actual analysis was done with the *lenna.png* image for the different kernel types. The main function was given different kernels to try for the same padding type and then different padding types for the same kernel and the results were written back and then analyzed.

1.a.i Results

Padding Function Results The padding function was tested with the *wolves.png* image to make sure that the results were correct and can be summarized in the following images in figure 1 below:



(a) Zero paddded image

(b) Copy edge padded image

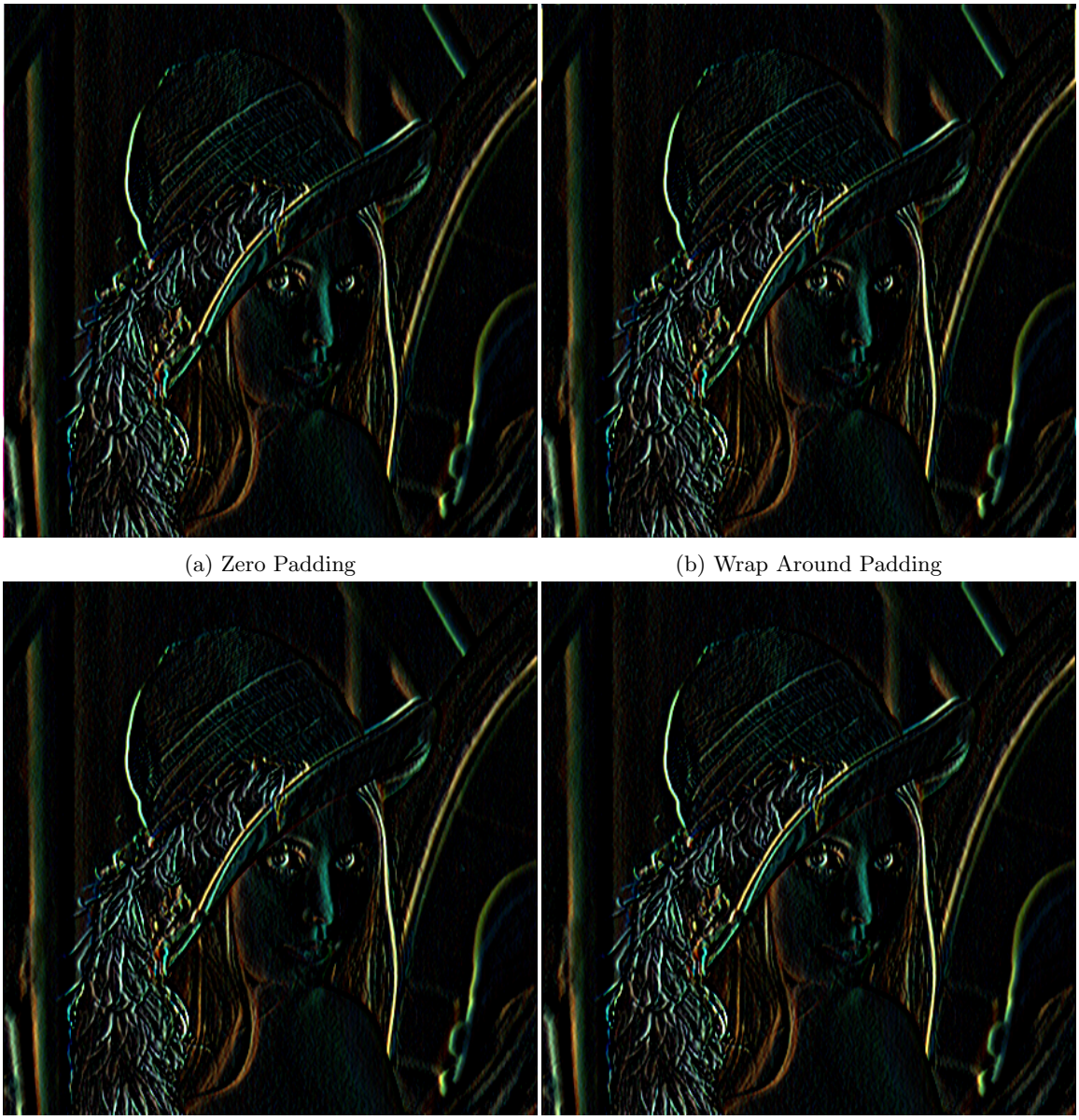
(c) Reflect across edge padded image

(d) Wrap around padded image

Figure 1: *wolves.png* with a uniform padding width of 50 applied for every one of the 4 padding types implemented into the padding function to illustrate the padding types

As mentioned before, the padding function was tested on the *wolves.png* image to verify that it was working as intended before this function was implemented into the *conv*2 function

Convolution Results In order to test the convolution function the image *lena.png* was used and first the filter used was kept constant and the padding type was varied and the results can be summarized in figure 2 below



(a) Zero Padding

(b) Wrap Around Padding

(c) Copy Edge Padding

(d) Reflect Across Edge Padding

Figure 2: Demonstration Of the Different Padding types with the Prewitt Kernel applied to every image through convolution

To test the different kernel the padding type was kept the same for all of the convoluttions as well as the image and only the kernel was varied and 9 different kernels were tested including:

- Box Filter: $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

- Simple first order derivative (row):

-1	1

- Simple first order derivative (column):

1
-1

- Prewitt $M_x =$

-1	0	1
-1	0	1
-1	0	1

- Prewitt $M_y =$

1	1	1
0	0	0
-1	-1	-1

- Sobel $M_x =$

-1	0	1
-2	0	2
-1	0	1

- Sobel $M_y =$

1	2	1
0	0	0
-1	-2	-1

- Roberts $M_x =$

0	1
-1	0

- Roberts $M_y =$

1	0
0	-1

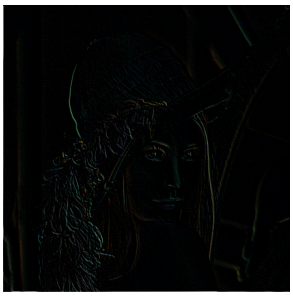
(a) Box Filter

(b) Column Derivative

(c) Row Derivative

(d) Prewitt M_x

(e) Prewitt M_y



(f) Roberts M_x

(g) Roberts M_y

(h) Sobel M_x

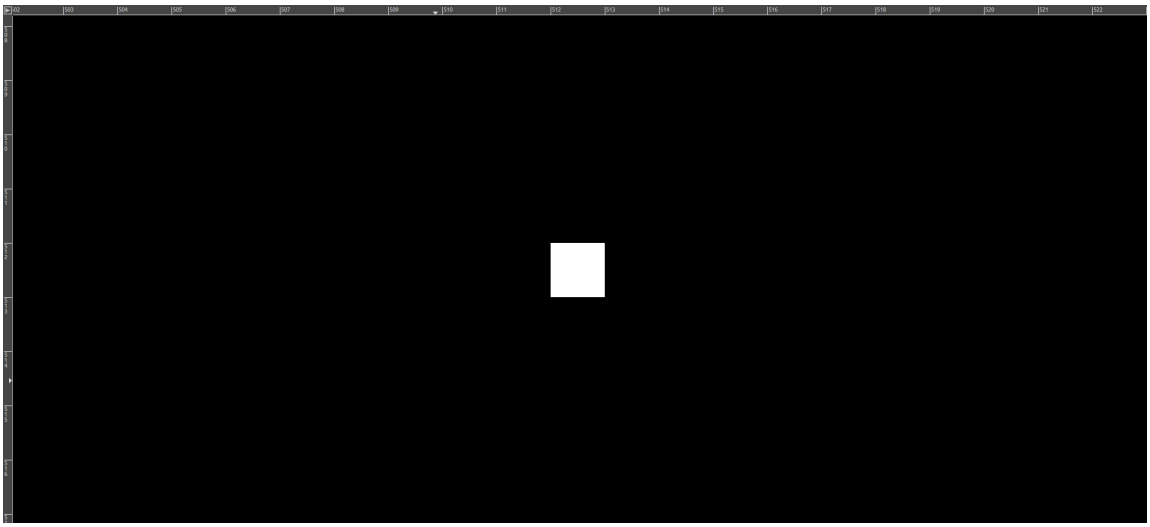
(i) Sobel M_y

Figure 3: Examples of the image *lena.png* convolved with the different kernels with a constant padding type (image was loaded and convolved as a color image for all kernels

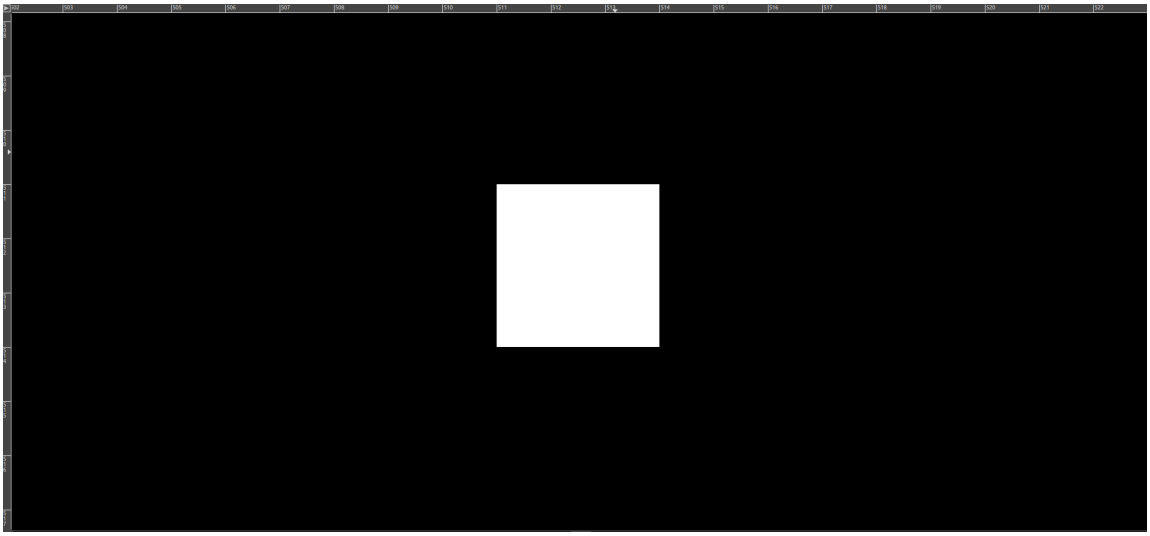
1.b Confirming that the Function is indeed performing convolution

In order to confirm that the function is indeed performing convolution, a unit impulse image with dimensions 1024×1024 was created with a unit impulse of magnitude 255 in the center of the image at location (512,512) and zero everywhere else. the results of this convolution can be seen in figure 4 below

The function can be shown that it is indeed performing convolution because the center of the image (the unit impulse) extracted the filter at that location as can be seen from the results, thus showing that the function is indeed performing convolution.



(a) Original impulse image at the impulse location



(b) Convolved image at the impulse location

Figure 4: Comparison between the original impulse image and the resuts from the convolution with a box kernel (a 3×3 kernel with a value of $\frac{1}{9}$ for every element

2 2D Fourier Transform and 2D Inverse Fourier Transform

2.a Implementing the 2D Fourier Transform using the built-in Matlab fft function

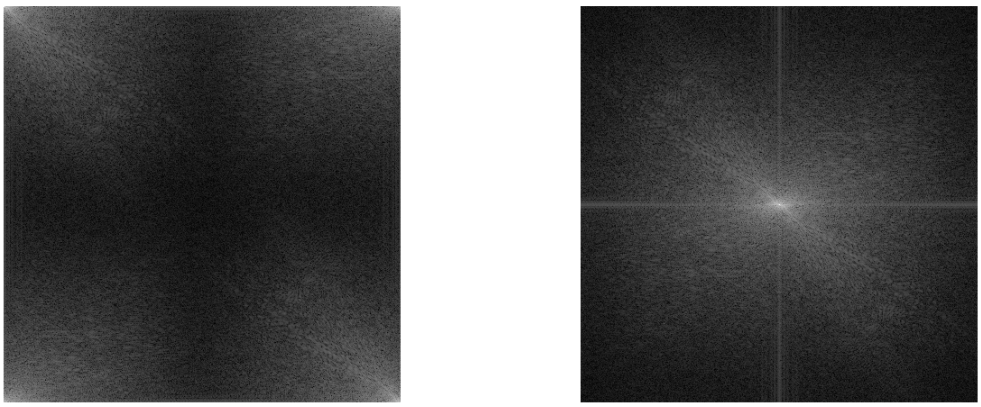
This portion of the project was done in Matlab and the way that the 2D FFT was implemented was using the following process:

1. Compute the FFT for each row in the image
2. Compute the FFT for each column in the image

These two steps were done one after the other and resulted in the final transformed image.

2.a.i Results

The results of applying the 2D FFT to the image after scaling the values in the image from 0-1 were visualized by showing the spectrum and phase angle image for the transformed image after applying the transformation $s = log(1 + abs(fftshift(F)))$ the shift was done so that it would be centered in the image instead of in the corners. In order to visualize the phase angle image the transformed image was passed into the *angle* Matlab function. The image that was used was *lena.png* The results of scaling the image and applying the 2DDFT can be seen in figure 5 below:



(a) FFT with no shift applied

(b) FFT with the fftshift applied

Figure 5: FFT applied to *lena.png* with (b) and without (a) doing the fft shift

The phase angles of the transformed image were also visualized and can be seen in figure 6 below

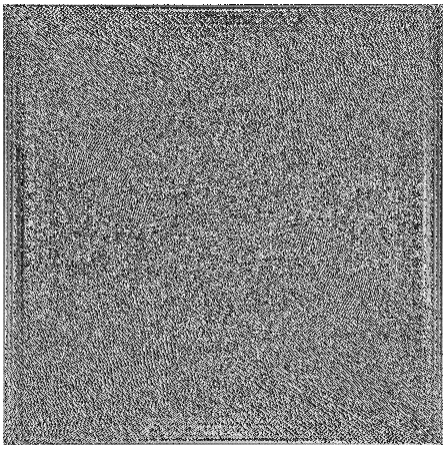


Figure 6: Phase angles for the image *lena.png*

2.b Implementing the 2D IDFT using the built in Matlab ifft function

For the 2D inverse DFT the process was very similar to that of the 2D DFT except instead of doing the FFT the inverse FFT was done. The steps were as follows:

1. Compute the 1DIFFT for each row in the image
2. Compute the 1DIFFT for each column in the image

In order to verify that the function was indeed working as intended, the results were compared to those of the built-in $fft2$ function and the built in $ifft2$ function from Matlab.

Once the image had been brought into the frequency domain by applying the 2D DFT it was then brought back into the image domain by applying the IDFT to the image and this was then visualized and compared to the original image

(a) Original image f

(b) Image resulting from applying the IDFT to the FFT image

Figure 7: Comparison of the Original image and the image resulting from the IDFT function, as can be clearly seen they are identical, as expected

Finally the two images above, f and g were subtracted and the result was a zero image or a completely black image as expected.

Figure 8: Results from subtracting the image in figure 7b from the image in figure 7a