Frequency Filtering

Srinidhi Patil
Department of Computer Science
University of Houston
Houston, Texas, USA
sppatil2@uh.edu

Rajath Puttaswamy Gowda
Department of Computer Science
University of Houston
Houston, Texas, USA
rpgowda@uh.edu

Rahul Tandon
Department of Computer Science
University of Houston
Houston, Texas, USA
rtandon@uh.edu

Srihitha Chebiyyam
Department of Computer Science
University of Houston
Houston, Texas, USA
srihitha.vfn12@gmail.com

Naveen Kumar Vangapandu
Department of Computer Science
University of Houston
Houston, Texas, USA
nvangapandu@uh.edu

Kishore Sabbavarapu
Department of Computer Science
University of Houston
Houston, Texas, USA
ksabbavarapu@uh.edu

I. Objectives

Our project objectives are to implement various transformation and filtering algorithms on the images and provide the corresponding results to the end user of our application. Applying frequency filtering techniques to the provided input images and displays the filtering output.

Keywords—Fourier, Transformation, DFT, Filtering, Convolution, Ideal, Butterworth, Gaussian, Laplacian, Masking, Cooley-Tukey, FFT

II. Introduction

We have implemented and completed multiple requirements in this project - 1) Computing a fast Fourier transformation of the matrix. 2) Use of convolution theorem to apply filtering in frequency domain. 3) Implementation of Low and High pass filters with varying parameters. 4) Implementation of Sharpening filters with varying parameters. We plan to implement a GUI to view the output of all our implementations.

III. Methods and Implementation

We have implemented and completed multiple requirements in this project - 1) Computing a fast Fourier transformation of the matrix. 2) Use of convolution theorem to apply filtering in frequency domain. 3) Implementation of Low and High pass filters with varying parameters. 4) Implementation of Sharpening filters with varying parameters. We plan to implement a GUI to view the output of all our implementations.

A. Fourier Transformation

We implemented the Fourier transformation. An important image processing tool used to represent an image into its sine and cosine components. The output of the transformation represents image in frequency/Fourier domain, while the input image matrix is in the spatial domain. After the transformation, each point in Fourier domain represents frequency contained in spatial domain range. This Fourier Transformation is used in image analysis, image filtering and image compression among other applications. We implemented this formation using three methods – naïve implementation of DFT, using DFT symmetric property and using Cooley-Tukey FFT algorithm.

<u>Naïve implementation</u>: For a square image of size N×N, the two-dimensional DFT is given by: F(u, v) = Σ (i = 0, i = N-1) Σ (j = 0, j = N-1) f(i , j) e ^ (-i* 2π* (ui/N + vj/N)) for every f(i, j) frequency sample from the input image matrix, we computed F(u, v) output in the frequency/Fourier domain. We implemented this using the brute force approach of applying four for loops with complexity O(N^4). This was a simple brute force implementation of DFT transformation.

<u>Implementation using symmetry property of DFT</u>: One of the most important tools of an algorithm is to exploit symmetries of a problem i.e., one piece of a problem is simply related to another, hence we can compute sub result only once and saved on the computational cost. The DFT of a real-valued discrete-time signal has a special symmetry, in which the real part of the transform values are DFT even symmetric and the imaginary part is DFT odd symmetric. Hence instead of computing the DFT for whole matrix of size NxN, we will only be doing the actual computation till M = N/2 and using the computation done for each F(u, v) for F(N-u, N-v). Due to this, our overall complexity instead of being O(N^4), will instead be O(M^4) where M = N/2.

<u>Cooley-Tukey FFT implementation</u>: This is where we faced problem in implementation. We read materials online and took help from many places in understanding the concept properly and its implementation. We were unable to implement it, so for comparison purpose we took help from online code. Cooley-Tukey implemented Fast Fourier Transformation by exploiting symmetry. Here instead of NxN matrix, we have a 1D array (1xN). Spilt the single DFT into two terms which themselves look very similar to smaller DFT, one on the odd-numbered values and one on even-numbered values. The trick comes in

making use of symmetries in each of these terms. We see from symmetry property that we need only perform half the computations for each sub-problem. Hence O(N^2) computation has become O(M^2) with M half the size of N. But it does not stop here, if the smaller Fourier transforms have even numbered valued M. Hence reapplied this divide-and-conquer approach, halving the computation cost each time, until arrays got small enough that no further divide-and-conquer could be applied. Hence this recursive approach scales as O(NlogN). On comparison we concluded that the calculation done with Cooley-Tukey FFT is faster than calculation done with naïve and symmetric DFT implementation. Note that we still haven't come close to the speed of the built-in FFT algorithm in NumPy. On further findings we realized that this is to be expected. The FFTPACK algorithm behind NumPy’s FFT is a Fortran implementation which has received years of tweaks and optimizations.

B. *Convolution Theorem*

Convolution is helpful for blurring, sharpening, edge detection and more. It is done between an original image and a kernel (multiple kernels available for various purposes). The general expression of a convolution is g(x,y) = Σ(dx=-a, dx=a) Σ(dy=-b,dy=b) w(dx,dy)f(x+dx,y+dx), where g(x,y) is a filtered image , f(x,y) is the original image, w is the filter kernel. In the implementation process, we’ve faced a challenge while calculating weighted sum. It is producing an invalid output with calculated weighted sum. Even after exploring various materials and understanding the concept comprehensively, we were unable to fix the code in the given time. So, finally we’ve used the in-built function in our implementation to achieve the resultant output.

C. *Low Pass and High Pass Filters*

The Spatial Filtering technique is applied to individual pixels in an image. A mask is typically thought to be increased in size so that it has a distinct center pixel. This mask is positioned on the image so that the mask's center traverses all of the image's pixels.
Based on general classification, there are two main components that comes under Spatial Filtering. That are Smoothing and Sharpening.

i. Smoothing Spatial Filter: Smoothing is a filtering process that removes high-frequency noise from a digital image and preserves low frequency components. This is used for blurring which in turn accomplishes noise reduction. Image smoothing can be achieved with low pass filters such as Ideal, Butterworth, Gaussian.

ii. Sharpening Spatial Filter: Image Sharpening is a technique to enhance the fine details and highlight the edges in a digital image. It removes low-frequency components from an image and preserves high-frequency components.

Low Pass Filter: The low pass filters are used for smoothing an image. A low pass filter allows the frequency below a specified cut-off value to pass through it as it attenuates the high frequency. Hence, it helps in the removal of the aliasing effect.

$$G(u, v) = H(u, v) \cdot F(u, v)$$

Where G is the enhanced image obtained by multiplying H (Mask) and F (Fourier transformed image).

High Pass Filter: The high pass filters are used for sharpening an image. A high pass filter allows the frequencies above a specified cut-off value to pass through it. Hence, it helps in the removal of noise.

$$H(u, v) = 1 - H'(u, v)$$

The approach followed for all 6 filtering techniques is the same and it is articulated in the following steps.
Reading an input image and converting it into a NumPy array.
Getting the shape of the image in the rows and columns format.
Assign inputted cut-off frequency and filter order (Only for Butterworth filters).
Applying filter-specific transfer function to get a mask.
The mask obtained is then multiplied with the Fourier transformed image to get the convoluted image as the output.
Apply inverse Fourier transform on the convoluted image.
Obtained is the final enhanced image that is sent to the output folder.
The transfer functions of the 6 different filters are -

i. Ideal Low Pass Filter: For implementing an Ideal Low Pass Filter, we followed the lecture and implemented the following formula.

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

ii. Butterworth Low Pass Filter: For implementing the Butterworth Low Pass Filter, we have implemented the transfer function with an order n as follows.

$$H(u,v) = \frac{1}{1 + \left[D(u,v) / D_0\right]^{2n}}$$

For a specific cut-off frequency, the ringing effect will increase with an increase in the filter order.

iii. Gaussian Low Pass Filter: The transfer function for the Gaussian Low Pass filter is,

$$H(u,v) = e^{-D^2(u,v)/2D_0^{\,2}}$$

iv. Ideal High Pass Filter: It is obtained by applying a reverse operation on the Ideal Low Pass filter.

$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$

v. Butterworth High Pass Filter: The transform function is as follows,

$$H(u,v) = \frac{1}{1 + \left[D_0 / D(u,v)\right]^{2n}}$$

vi. Gaussian High Pass Filter: It is obtained by applying a reverse operation on the Ideal Low Pass filter.

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

D0 is the cut-off frequency provided as input. The Euclidian distance is

D(u, v) = math.sqrt((u- R//2)*2 + (v - C//2)*2)

D. Image Sharpening

We implement the concept of Image Sharpening using two filters –

Laplacian Filtering: It is used to restore fine detail of an image which has been smoothed while applying smoothing filters to reduce noise. The Laplacian operator is an example of second order derivate will be used to enhance any feature with sharp discontinuity like noise. In the implementation, we considered the mask [[0,-1,0][[-1,4,-1],[0,-1,0]]. Next step is to create a padding matrix from the given image. We applied the convolution between the kernel and image (pixel values are replaced by the sum of the kernel multiplied by the image values). It gives us the resultant Laplacian filtered image.

Unsharp Masking: The Unsharp mask operation actually consists of performing several operations in series on the original image. Initial step is to blur the original image using Laplacian filter. Here we faced a challenge while trying to use the Laplacian function implemented in the previous task. With this function, the Unsharp filter was generating an inappropriate output. So, what we've decide is, to make use of in-built Laplacian function, which is then used to produce a desired output. The blurred image generated in the previous step is subtracted from the original image to create a mask. The original image and the mask are now be added to produce a resultant Unsharp image.

IV. GUI

Our implementation of GUI is done via tkinter package, the standard Python interface to the Tcl/Tk GUI toolkit. We choose this GUI implementation as we wanted to learn something new on the python language. The idea of tkinter helped us implement the user interface with the ease of integration on the backend code which is completely written in python language. A lot of material on the tkinter was available online which made it easy at our end to learn and implement smoothly. Attached are the screenshots of our GUI.

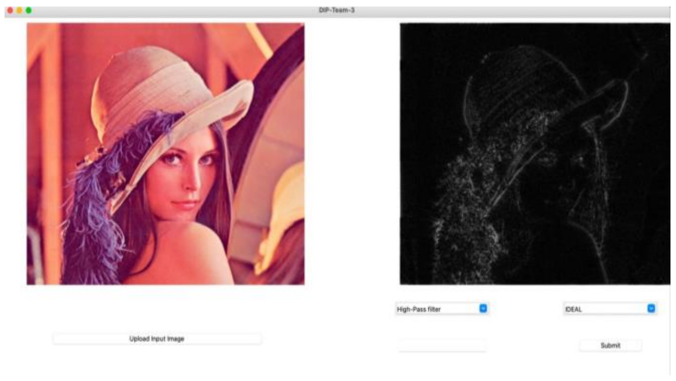

Image 1

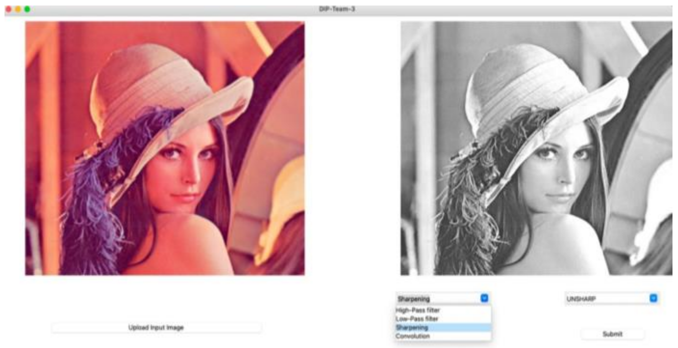

Image 2

V. Responsibilities

Topic	Member
Frequency Filtering: Symmetry	Rajath Puttaswamy Gowda
Cooley-Tukey FFT	Rahul Tandon
Naïve Implementation	Srinidhi Patil
Computational times of a naïve implementation	Rahul Tandon
Convolution theorem	Srihitha Chebiyyam
Low pass filters	Naveen Kumar Vangapandu
High pass filters	Naveen Kumar Vangapandu
Laplacian and Unsharp masking	Kishore Sabbavarapu
GUI	Rajath Puttaswamy Gowda
Code Integration with GUI	Srinidhi Patil
Final Report	Srihitha Chebiyyam, Srinidhi Patil
Presentation	Kishore Sabbavarapu

Table 1

VI. Summary

In conclusion, we have been successful in implementing almost all the tasks. The key highlights of this project are the GUI implementation, the Fast Fourier Transformation algorithms implementation & its computational comparisons, Low – High Pass Filters and Image Sharpening algorithms. We couldn’t implement the Convolution theorem and had little hiccups in the implementation of few other algorithms (Cooley-Tukey FFT). In spite of all these challenges, in this project we acquired in-depth knowledge of frequency filtering concepts and are able to learn its significance and applications in various fields.

VII. ACKNOWLEDGEMENT

We thank our professor Dr. Pranav Mantini for approving and supporting our project. We would like to express our special thanks and gratitude to all our TA. Finally, we thank University of Houston for providing us this opportunity. We also thank all the references and suggestions used while developing this project including the lecture slides provided the professor for this course.

VIII. Links to uploaded video

YouTube:
https://www.youtube.com/watch?v=XmJv6etcWN8

Google Drive:
https://drive.google.com/file/d/14ySU6SfCqgWLFNSz-j9hNM31JWoJh4Rk/view?usp=sharing