**CPEG 585 – COMPUTER VISION**

**HOMEWORK 2**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** February 20, 2019

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

[**SOURCE CODE:**](#_Toc446970374) 16

**[CONCLUSION:](#_Toc446970375)**20

**INTRODUCTION**

The purpose of this assignment is to get familiar and understand better all the Kernels shown in class. The first part of the assignment will help us understand better how each Kernel is created and what is the math behind it.

The second part of the assignment will help us understand the purpose of each Kernel by looking at the result when the different Kernels are applied to an image.

**SCREEN SHOTS:**

**Problem 1:**

It is important to understand that in order to obtain all the following kernels, they are all based on this starting Matrix:

|  |  |  |
| --- | --- | --- |
| x - Δx, y + Δy | x, y + Δy | x + Δx, y + Δy |
| x - Δx, y | x, y | x + Δx, y |
| x - Δx, y - Δy | x, y - Δy | x + Δx, y - Δy |

IDENTITY:

The basic propertie about the Identity Kernel is that the midle pixel from an image must remain the same (different from an identity Matrix). That means that the sum of all the points must be equal to the point P(x,y).

Sum = P(x-Δx, y+Δy) + P(x-Δx, y) + P(x-Δx, y-Δy) + P(x, y+Δy) + P(x, y)+ P(x, y-Δy) + P(x+Δx, y+Δy) + P(x+Δx, y) + P(x+Δx, y-Δy)

Since Sum mus be equal to P(x, y), the identity matrix will look like:

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

AVERAGE (LOW PASS):

The goal for an average Kernel is to remove noise.

As the name says, it is based on an average value. The average Kernel is created by adding up all the numbers in the kernel and dividing each number by the sum of all the elements:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

Average Kernel:

**/9 (since the sum of all the 1 is 9)**

HIGH PASS:

The goal of a high pass filter is to enhace the high frequancies. In order to compute this Kernel, we will use the identity and average Kernel. Since the goal of the high pass is to enhace the high frecuencies, we will substract the average kernel to the identity and obtain a high pass kernel. When applying the final kernel obtained to the image, we will enhace the high frequencies.

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

- /9

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

High Pass= /9

SHARPENING:

In order to sharpen an image, we need a parameter “p” which is the sharpening parameter that goes from 0-1. Once we decided the parameter p, we will apply the following formula:

If we translate that formula to matrix form:

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

S = (1-p) \* + p \* /9

GAUSSIAN:

In order to obtain the gaussian Kernel, we need to apply the gaussian formula to the starting kernel.

|  |  |  |
| --- | --- | --- |
| -1,1 | 0,1 | 1,1 |
| -1,0 | 0,0 | 1,0 |
| -1,-1 | 0,-1 | 1,-1 |

Gaussian (x,y): Starting Matrix:

We need to apply Gaussian to each of the (x, y), we will be using a for this example.

P(-1,1) = = 0.155

P(0,1) = = 0.176

P(1,1) = =0.155

P(-1,0) = = 0.176

P(0,0) = = 0.199

P(1,0) ==0.176

P(-1,-1) = = 0.155

P(0,-1) = =0.176

P(1,-1) = =0.155

|  |  |  |
| --- | --- | --- |
| 0.155 | 0.176 | 0.155 |
| 0.176 | 0.199 | 0.176 |
| 0.155 | 0.176 | 0.155 |

Gaussian Kernel: /1.523 (sum of cells cannot be more than 1)

FIRST DERIVATIVE (GRADIENT):

Formula for First derivative (gradient) filter:

|  |  |  |
| --- | --- | --- |
| -1,1 | 0,1 | 1,1 |
| -1,0 | 0,0 | 1,0 |
| -1,-1 | 0,-1 | 1,-1 |

In order to take an accurate first derivative, we will use Taylor series. Taking the starting Matrix as a reference:

|  |  |  |
| --- | --- | --- |
| x-Δx,y+Δy | x,y+Δy | x+Δx, y+Δy |
| x-Δx, y | x, y | x+Δx, y |
| x-Δx, y-Δy | x, y-Δy | x+Δx, y-Δy |

LEFT COLUMN:

P(x-1, y+1)= P(x, y) - 1 + 1

P(x-1, y)= P(x, y) - 1

P(x-1, y-1)= P(x, y) - 1 - 1

After adding the three equations:

RIGHT COLUMN

P(x+1, y+1)= P(x, y) + 1 + 1

P(x+1, y)= P(x, y) + 1

P(x+1, y-1)= P(x, y) + 1 - 1

After adding 3 equations:

MIDDLE COLUMN:

P(x, y+1)= P(x, y) + 1

P(x, y) = 0

P(x, y-1)= P(x, y) - 1

After adding 3 equations:

In order to find we will do (sum of right col pixel) – (sum of left column pixels):

)

Sobel Kernel to enhance X pixels:

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

This will enhance the X pixels

Now we have to repeat the process for the Y pixels (do derivatives by rows now):

Sobbel Kernel to enhance Y pixels:

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

Once we find both Kernels (for X and Y pixels), we need to square the obtained pixels, add both matrixs and apply the square root as show in the gradient formula:

SECOND DERIVATIVE (LAPLACIAN):

In order to find Laplacian filter we need to focus on the neighboors pixels

|  |  |  |
| --- | --- | --- |
|  | x,y+Δy |  |
| x-Δx, y | x, y | x+Δx, y |
|  | x, y-Δy |  |

|  |  |  |
| --- | --- | --- |
|  | 0,1 |  |
| -1,0 | 0,0 | 1,0 |
|  | 0,-1 |  |

P(x-1, y)= P(x, y) - 1 +

P(x, y+1)= P(x, y) + 1 +

P(x, y-1)= P(x, y) - 1

P(x+1, y)= P(x, y) + 1 +

Add for equations:

Example of Laplacian Kernel:

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

DIFFERENCE OF GRADIENTS (Bandpass):

In order to obtain a Kernel based on the difference of gradients we need to compute the difference of two LPF. The most popular choice is the difference of Gaussians.

How to obtain the Gaussian filters is shown on the examples above.

Gaussian for

|  |  |  |
| --- | --- | --- |
| 0.170 | 0.461 | 0.170 |
| 0.461 | 0.798 | 0.461 |
| 0.170 | 0.461 | 0.170 |

/3.322

Gaussian for :

|  |  |  |
| --- | --- | --- |
| 0.155 | 0.176 | 0.155 |
| 0.176 | 0.199 | 0.176 |
| 0.155 | 0.176 | 0.155 |

/1.523

Bandpass Kernel:

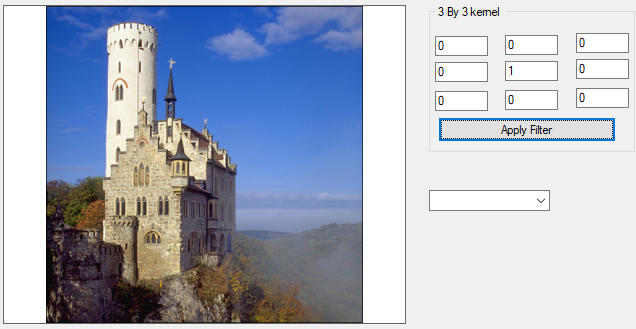
|  |  |  |
| --- | --- | --- |
| 0.155 | 0.176 | 0.155 |
| 0.176 | 0.199 | 0.176 |
| 0.155 | 0.176 | 0.155 |

|  |  |  |
| --- | --- | --- |
| 0.170 | 0.461 | 0.170 |
| 0.461 | 0.798 | 0.461 |
| 0.170 | 0.461 | 0.170 |

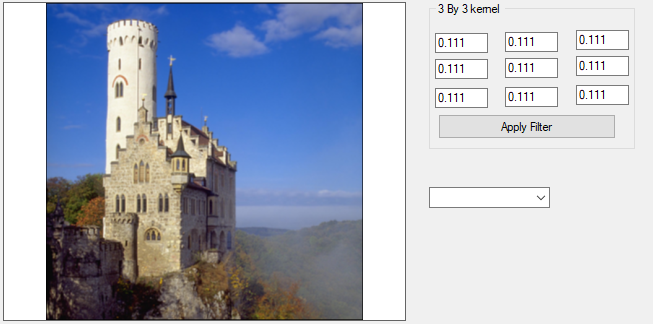
/1.523 - /3.322

**Problem 2:**

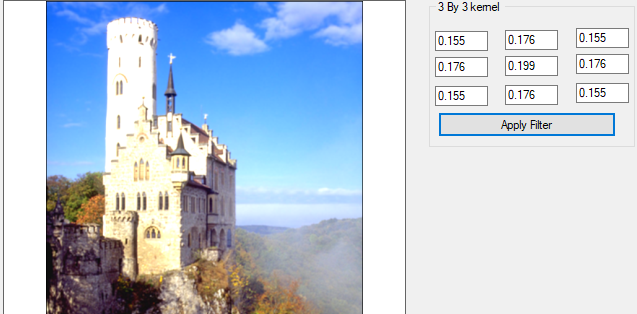
After applying the identity kernel the image remains the same:



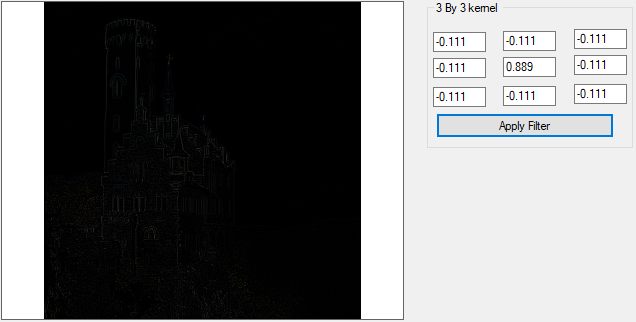
Output after applying an acerage kernel of 1/9:



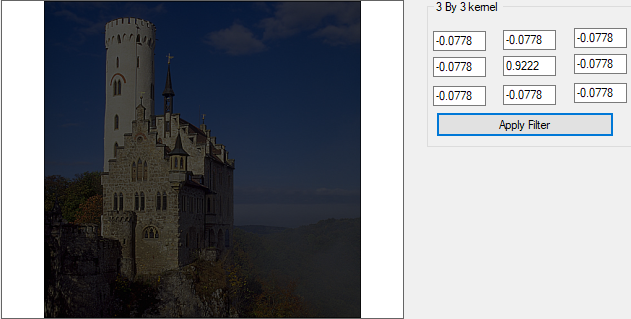
Gaussian filter of :



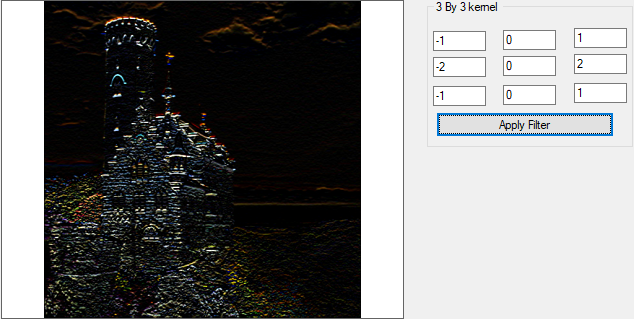
Results after applying a high pass filter:



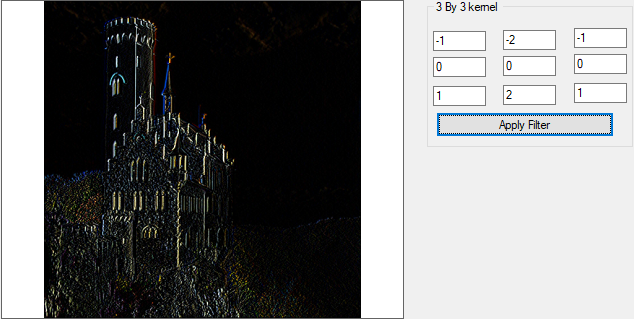
Sharpening filter of p=0.7:



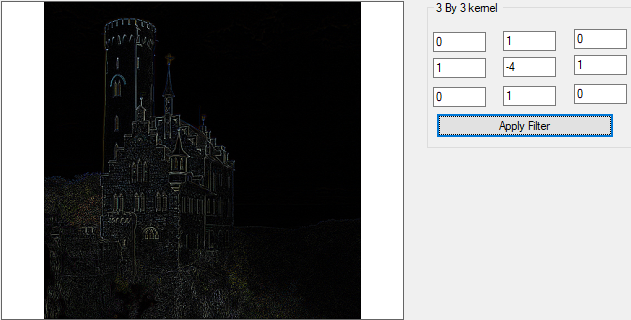
First derivative filter on the X pixels:



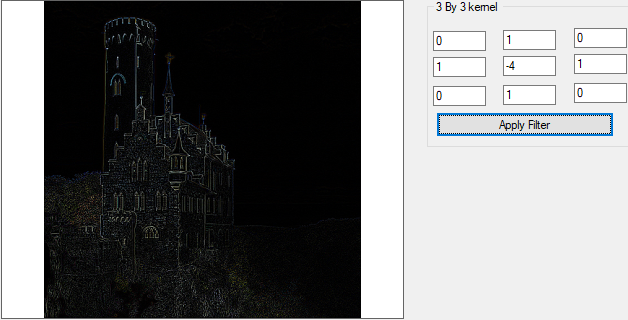
First derivative filter on the Y axis:



Second derivative filter (Laplacian):



Difference of Gaussians filters ( ):



**SOURCE CODE:**

**Problem 2:**

Form:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Drawing.Imaging;

namespace Image\_Convolution

{

public partial class Form1 : Form

{

// declare classes and variables

convolution convolution = new convolution();

public Bitmap origina\_Image = null;

public Bitmap resultImg = null;

public Form1()

{

InitializeComponent();

}

private void LoadImgBtn\_Click(object sender, EventArgs e)

{

// load img

OpenFileDialog ofd = new OpenFileDialog();

ofd.Title = "Select an image file.";

ofd.Filter = "Png Images(\*.png)|\*.png|Jpeg Images(\*.jpg)|\*.jpg";

ofd.Filter += "|Bitmap Images(\*.bmp)|\*.bmp";

if (ofd.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

StreamReader streamReader = new StreamReader(ofd.FileName);

origina\_Image = (Bitmap)Bitmap.FromStream(streamReader.BaseStream);

streamReader.Close();

pictureBox1.Image = origina\_Image;

ApplyMaskBtn.Enabled = true;

textBox1.Text = String.Empty;

textBox2.Text = String.Empty;

textBox3.Text = String.Empty;

textBox4.Text = String.Empty;

textBox5.Text = String.Empty;

textBox6.Text = String.Empty;

textBox7.Text = String.Empty;

textBox8.Text = String.Empty;

textBox9.Text = String.Empty;

}

}

private void ApplyMaskBtn\_Click(object sender, EventArgs e)

{

double[,] MyMask = new double[3, 3];

MyMask[0, 0] = Convert.ToDouble( textBox1.Text);

MyMask[0, 1] = Convert.ToDouble(textBox2.Text);

MyMask[0, 2] = Convert.ToDouble(textBox3.Text);

MyMask[1, 0] = Convert.ToDouble(textBox4.Text);

MyMask[1, 1] = Convert.ToDouble(textBox5.Text);

MyMask[1, 2] = Convert.ToDouble(textBox6.Text);

MyMask[2, 0] = Convert.ToDouble(textBox7.Text);

MyMask[2, 1] = Convert.ToDouble(textBox8.Text);

MyMask[2, 2] = Convert.ToDouble(textBox9.Text);

resultImg = convolution.Convolution(origina\_Image, MyMask);

pictureBox1.Image = resultImg;

}

private void Savebtn\_Click(object sender, EventArgs e)

{

SaveFileDialog dlg = new SaveFileDialog();

dlg.Filter = "jpeg files (\*.jpg)|\*.jpg";

if (DialogResult.OK == dlg.ShowDialog())

this.pictureBox1.Image.Save(dlg.FileName, ImageFormat.Jpeg);

}

}

}

Convolution:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Image\_Convolution

{

class convolution

{

public Bitmap Convolution(Bitmap Original\_Image, double[,] Mask\_Matrix)

{

// get the values of pixels of the original image and store them in one D array

double[] pixelArray = ReadPixels(Original\_Image);

// declare a new 1 D array to handel the new values of pixels after manipulation:

double[] resultArray = new double[(Original\_Image.Width \* 3) \* Original\_Image.Height];

// loop through image pixels and do the convultion opration

for (int ImgY = 1; ImgY < Original\_Image.Height - 1; ImgY++)

{

for (int ImgX = 1; ImgX < Original\_Image.Width - 1; ImgX++)

{

// declare colors

double B = 0;

double G = 0;

double R = 0;

int imgCo = ImgY \* (Original\_Image.Width \* 3) + ImgX \* 3;

for (int Msk\_Y = -1; Msk\_Y <= 1; Msk\_Y++)

{

for (int Msk\_X = -1; Msk\_X <= 1; Msk\_X++)

{

int MskC = imgCo + (Msk\_X \* 3) + (Msk\_Y \* (Original\_Image.Width \* 3));

R += (double)(pixelArray[MskC]) \* Mask\_Matrix[Msk\_Y + 1, Msk\_X + 1];

G += (double)(pixelArray[MskC + 1]) \* Mask\_Matrix[Msk\_Y + 1, Msk\_X + 1];

B += (double)(pixelArray[MskC + 2]) \* Mask\_Matrix[Msk\_Y + 1, Msk\_X + 1];

}

}

// scale the value between 0 - 255

if (R > 255)

R = 255;

else if (R < 0)

R = 0;

if (B > 255)

B = 255;

else if (B < 0)

B = 0;

if (G > 255)

G = 255;

else if (G < 0)

G = 0;

resultArray[imgCo] = (byte)(R);

resultArray[imgCo + 1] = (byte)(G);

resultArray[imgCo + 2] = (byte)(B);

}

}

Bitmap resultImage = new Bitmap(Original\_Image.Width, Original\_Image.Height);

// create a new Bitmap & assign the new pixel values to it.

int Size = 0;

for (int y = 0; y < Original\_Image.Height; y++)

{

for (int x = 0; x < Original\_Image.Width; x++)

{

resultImage.SetPixel(x, y, Color.FromArgb(Convert.ToInt32(resultArray[Size]), Convert.ToInt32(resultArray[Size + 1]), Convert.ToInt32( resultArray[Size + 2]) ));

Size = Size + 3;

}

}

return resultImage;

}

double[] ReadPixels(Bitmap Original\_Image)

{

// retrieve the pixel values and store it in vector

Color c;

double[] xx = new double[Original\_Image.Width \* Original\_Image.Height \* 3];

int i = 0;

for (int y = 0; y < Original\_Image.Height; y++)

{

for (int x = 0; x < Original\_Image.Width; x++)

{

c = Original\_Image.GetPixel(x, y);

xx[i] = (double)(c.R);

xx[i + 1] = (double)(c.G);

xx[i + 2] = (double)(c.B);

i = i + 3;

}

}

return xx;

}

}

}

**Conclusion:**

After this assignment I have to say I understand better how the different Kernels are created and what is the math behind them. Also, this assignment helped me realize what is the function or purpose of each kernel.