# Challenge 2

ECE 131L - Fall 2022

## Image

In our daily life, we see digital images on social medias or we can also take some pictures with our phones. We can define an image as a 2D array because it has a certain number of rows and columns. However, color images has an extra dimension, the number of channels. For us, this dimension is equal to 3 and it is constant. One example of this type of images is the RGB images that has one channel for red, one for blue, one for green. For this challenge, we are going to define a gray scale $\mathcal{I}$ as:

$$\mathcal{I} = \frac{R + G + B}{3} \tag{1}$$

where are R, G, and B are the channels of a RGB image.

## Zero Padding and convolution

Convolution is the most basic operation in image processing. It is defined as:

$$\mathcal{I}'(x, y) = \sum_i \sum_j \mathcal{I}(i, j) F(x - i, y - j) \tag{2}$$

where $\mathcal{I}'(x, y)$, and $(x, y)$ are the convoluted image and the pixel coordinates. For this assignment, we will consider an average filter whose dimensions (rows and columns) are the same. So we can reduce equation 2 as:

$$\mathcal{I}'(x, y) = \frac{1}{N^2} \sum_i \sum_j \mathcal{I}(i, j) \tag{3}$$

where $N$ is the size of the filter.

### Example

$$\mathcal{I} * F = \mathcal{I}'$$

$$\begin{bmatrix} 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

As we can observe, the dimensions of $\mathcal{I}$ and $\mathcal{I}'$are $5 \times 5$ and $3 \times 3$. So we can establish the next relationship

$$\dim(\mathcal{I}) - N/2 = \dim(\mathcal{I}') \tag{4}$$

As we can observe from equation 4, the dimension of the output will always be smaller than the original image. To solve these problem, we use zero padding technique. In the following example, we will use the same input image but with zero padding.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 3 & 3 & 0 \\ 0 & 3 & 3 & 3 & 3 & 3 & 0 \\ 0 & 3 & 3 & 3 & 3 & 3 & 0 \\ 0 & 3 & 3 & 3 & 3 & 3 & 0 \\ 0 & 3 & 3 & 3 & 3 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} 4/3 & 2 & 2 & 2 & 4/3 \\ 2 & 3 & 3 & 3 & 2 \\ 2 & 3 & 3 & 3 & 2 \\ 2 & 3 & 3 & 3 & 2 \\ 4/3 & 2 & 2 & 2 & 4/3 \end{bmatrix}$$

# 1 Challenge explanation (50pts)

For this challenge, you have to create a code that loads an image, does the convolution operation with zero padding using an average filter whose size is given by the user, and saves the convoluted image.

**You are NOT ALLOWED of using any BUILT-IN FUNCTION**. Except for the ones you have on the template.

## 1.1 Input explanation

- **file name** → the name of the file. It should be in the same folder than the code.
- **filter size** → the size for our square filter. It must be an odd number.

## 1.2 Output

The out must be a jpg file that contains the filtered image.

## 1.3 Built-in functions

In the zip file, you will find to h files: stb_image.h and stb_image_write.h. They must be in the same folder as the C file. The two built-in function you will use are: stbi_load() and stbi_write_jpg().

The arguments for **stbi_load()** are:

- filename → The name of the file we want to load.
- &w → the pointer of the width of the input image

- &h → the pointer of the height of the input image
- &c → the pointer of the channels of the input image

The way to call the function is: stbi_load(filename, &w, &h, &c, 0) . This function return an unsigned char* which is explained in the subsection.

The arguments for **stbi_write_jpg()** are:

- filename → The name of the file we want to create. It must finish with ".jpg"
- w → the width of the input image
- h → the height of the input image
- c → the channels of the input image
- img → pointer of a 1D unsigned char array

The way to call the function is: stbi_write_jpg(filename, w, h, c, img, w×c);

## 1.4 Unsigned char

As we learn in class, *char* variables has values from -128 to 127. In the case of *unsigned char*, the value goes from 0 to 255. We use this type of variable because images also usually goes from 0 to 255. So we can consider *unsigned char* as integer whose value goes from 0 to 255.

## 1.5 Output of stbi_load()

As it was mentioned before, this function returns an *unsigned char**, however when we load images, they usually are a 3D matrix because we load an RGB image. The next example will give you a better understanding:

*unsigned char** *a*:

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$

3D matrix :

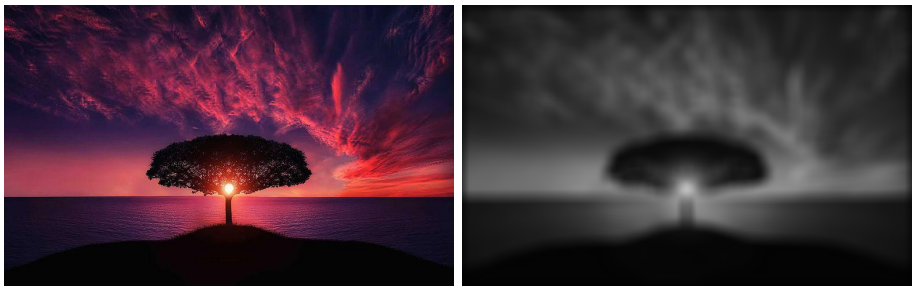
R channel → $[1, 4, 7, 10, 13]$

G channel → $[2, 5, 8, 11, 14]$

B channel → $[3, 6, 9, 13, 15]$

So basically, the first 3 elements correspond to the $(0, 0)$ pixel, the next 3 elements are the values for the pixel in the position$(0, 1)$.

### 1.6 Example



(a) (b)

Figure 1: (a) original image (b) filtered image

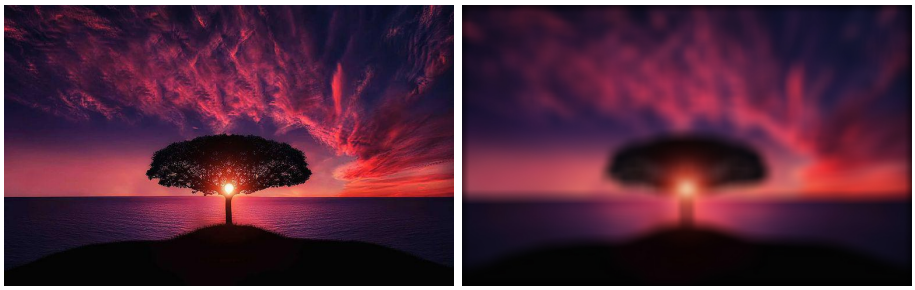
### 1.7 Compiling

To compile your code you need to use the following command:
gcc -std=c17 filename -lm
where filename must replace it by the name of your C file

## 2 Extra credit(10pts)

You have to do the same as before but now you will perform the convolution for an RGB image. Basically, you have to do the convolution for each channel.

### 2.1 Example



(a) (b)

Figure 2: (a) original image (b) filtered image