IMAGE MANIPULATION USING CONVOLUTIONAL METHODS

PROBLEM SOLVING AND C PROGRAMMING

PRESENTED BY:
A KATHIR
VAMSHIDHARREDDY.K. V
ARIVANANTHAN M
SURYA SANTHOSH KUMAR

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF THE UGC ACT, 1956

DEPARTMENT OF CEN
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE CAMPUS (INDIA)
FEBRUARY 2023

DECLARATION

We hereby declare that the project work entitled, “IMAGE MANIPULATION USING CONVOLUTIONAL METHODS” submitted to the Department of CEN is a record of the original work done by us under the guidance of **Ms. Aswathy** , Assistant Professor of the Department of CEN, Amrita Vishwa Vidyapeetham and that it has not been performed for the award of any degree/diploma/Associate fellowship and similar titles if any.

Signature of the Faculty :

Names:	**Roll number:**
A KATHIR	CB.EN.U4AIE22001
ARIVANANTHAN M	CB.EN.U4AIE22007
SURYA SANTHOSH KUMAR	CB.EN.U4AIE22049
VAMSHI KV	CB.EN.U4AIE22028

Place: Ettimadai
Date: 05-FEB-2023

CONTENTS

ACKNOWLEDGEMENT

This project has been possible due to the sincere and dedicated efforts of many. First of all, we would like to thank the dean of our college for allowing us to get involved in a project and express our skills. We thank our C Programming teacher, **Ms. Aswathy** for her guidance and support without which this project would have been impossible. We also would like to thank all Authors and Sources we referred for the making of this project. Last but not least; we thank our parents and our classmates who encouraged us throughout the project.

ABSTRACT

The central theme of this Project will be Image Blurring, However, We will also look at other Image Manipulation techniques using Convolutional Methods. Convolution is a mathematical tool to combine two signals to form a new signal. We will study the dependency of Intensity of Blur on Kernel Size, and Image Size. And, once a 'Convolutional' Framework and Algorithm for our program has been set up, we will try changing the Kernel values and observe changes in our output image, and see if something Interesting pop's up. We will be doing all of this in an Image Library called stb_image which was created by a team of people in github. We will also document the outputs that we get, so that people reading this can view our outputs without having to run the code.

CHAPTER-1
INTRODUCTION & LITERATURE

The field of Image Processing is vast and diverse. It is developing day by day at a very rapid pace. According to YouTube statistics published in 2023, over 2.6 Billion people around the globe use the video streaming service. And there are over 40 Billion posts on Instagram. We come across Images on a daily basis, but we are never curious on how these services ‘magically’ make images more beautiful, or how different effects are created. All the effects that we see on Images have an Interesting story behind them. They, all have mathematical equations driving them. In this simple Project we will look at how a common person with very little experience, can start manipulating Image pixels, to create interesting outputs.

What is an Image?

How can you capture something that you see in front of you right now, and save it so that you can view it later? When we take a photo on our camera, the camera does exactly this job. It reads the light that falls on its sensor that has been reflected off of different objects. The sensor then distinguishes between different colours of light that fall on it, then remembers this information. The information is stored as pixel values which is basically binary. And that’s how the Images we see around us are actually captured.

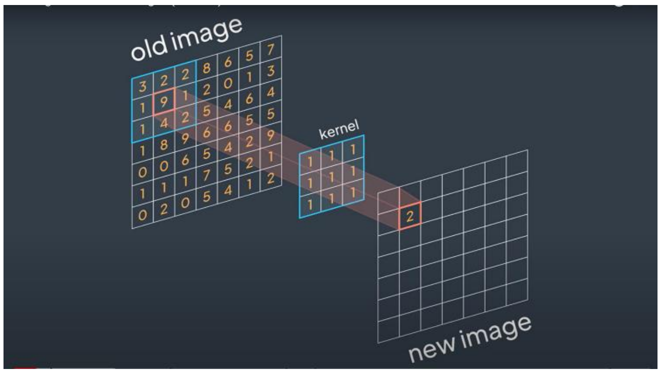
Basics of Images (Bitmap)

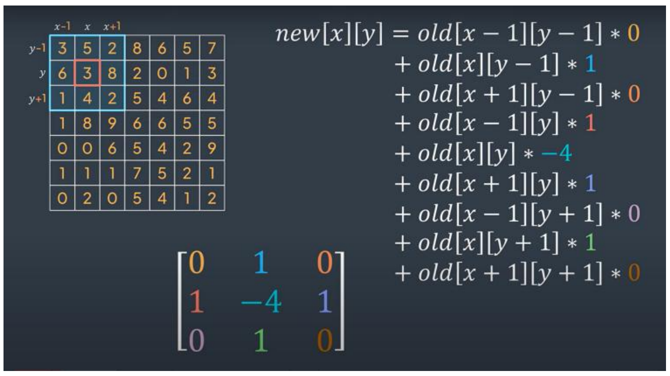
Modern day computers use 8bit colour, which means it takes 1byte/8bits of data to store the R value of a single pixel. So for a single RGB pixel, it takes 3 bytes of data. It becomes clear that an Image of resolution width x height(say, 4000x3000) has 12,000,000 pixels, or 36,000,000 bytes which is equivalent to 35156.25KB or 34.33MB.
We might wonder, how Computers can store 4K movies within the size of 6GB, because it seems impossible at the rate of 34.3MB for each image/video-frame, at a 60Hz refresh rate for a movie that is 2Hours Long. Modern day Computers tackle this problem by using Image Compression tactics. Video Compression Codec’s such as H.265 also known as High efficiency video codec are the latest Codecs for Movie Compression. Our Media players will be equipped with the technology to lets say ‘decrypt’ this codec on the movie.

Convolution

Convolution is a mathematical tool to combining two signals to form a third signal. Therefore, in signals and systems, the convolution is very important because it relates the input signal and the impulse response of the system to produce the output signal from the system.

Convolution is a general purpose filter effect for images. It Is a matrix applied to an image and a mathematical operation. comprised of integers. It works by determining the value of a central pixel by adding the. weighted values of all its neighbors together.





	x-1	x	x+1				
y-1	3	5	2	8	6	5	7
y	6	3	8	2	0	1	3
y+1	1	4	2	5	4	6	4
	1	8	9	6	6	5	5
	0	0	6	5	4	2	9
	1	1	1	7	5	2	1
	0	2	0	5	4	1	2

$$\begin{aligned} new[x][y] = {} & old[x-1][y-1]*0 \\ & + old[x][y-1]*1 \\ & + old[x+1][y-1]*0 \\ & + old[x-1][y]*1 \\ & + old[x][y]*-4 \\ & + old[x+1][y]*1 \\ & + old[x-1][y+1]*0 \\ & + old[x][y+1]*1 \\ & + old[x+1][y+1]*0 \end{aligned}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

As, we can see, in the Image Processing context of Convolution, it is just a mathematical tool for multiplying and adding values. In particular, we loop through the image as a 2D plane, then for each pixel we create a Image Matrix that contains pixel values of all pixels nearby it, and then these pixel values are multiplied with corresponding kernel values and are added up, to obtain the pixel value of the center pixel.

CHAPTER-2
METHODOLOGY

Before we begin, we need to download and extract a popular image manipulation themed library called 'stb_image' from GitHub, into our Project directory. The library can be accessed here:

https://github.com/nothings/stb/

Once that's done, We can collect a few Images to work with from the internet. We will collect Images of Different resolutions to make our Image set diverse. Some sources have been mentioned in the reference page.

We need to define STB_IMAGE_IMPLEMENTATION before including the <stb_image.h> header, and STB_IMAGE_WRITE_IMPLEMENTATION before including the <stb_image_write> header file.

We will import an image in stb_image library, using the stbi_load() function. The image can only be imported as an unsigned char pointer. The loading function takes the declared width, height, channels variables addresses as inputs, and stores the width height and channels of the image we are trying to import into our corresponding variable addresses.

unsigned char **img=stbi_load('city.jpg',&width,&height,&channels,3);*

The image is imported as a 1D array of pixel values. The pixel values are stored in the following format. If R1 is the Red of 1st pixel , G1 is the green and so on… then the image array will look like

[R1 G1 B1 R2 G2 B2 R3 G3 B3……………Rn Gn Bn]

The pixel values are stored Row wise, that is, the first row of the image is stored first, from left to right, and then once the end pixel has reached, it goes to the second row and start from the left most pixel to the right most pixel, and like wise the entire Image is covered.

We find the size in bytes of the loaded image using the formula: width*height*channels. We allocate some memory for the output image using malloc() function. For example, if we are doing RGB to grayscale conversion then the Memory to be allocated will not be same as that of the input image. Since grayscale images have only 1 channel, the size would be width*height*1 bytes. A simple for loop can give us access to each and every pixel of the loaded image, through a pixel pointer.

Things to be defined
unsigned char **img=stbi_load('city.jpg',&width,&height,&channels,3);*
unsigned char **bimg=malloc(imagebytes);*
unsigned char **p=img;   // pixel pointer of 1st image.*
unsigned char **pg=bimg;   // pixel pointer of 2nd image.*

int *kernel[3][3]=*`{{1,1,1},{1,1,1},{1,1,1}};`

The pixel pointer to the input image (*p) and the pixel pointer to the output image (*pg) should loop through the image at the same pace. For any *p, which is the Red value of a Pixel, *(p+1) and *(p+2) will give the Green value and Blue value of that pixel respectively. p+=channels, will go to the red value of the next pixel.

For the sake of Convolution, it becomes logical to make the 1D *img array into a 2D one for better visualization(just like how our computers present an Image(2D)). So, now instead of looping through the *img array, we loop through all coordinates of all pixels on the image row wise. We then make a function to return the address of the pixel in the *img array by taking the (x,y) coordinates of that particular pixels. Now we can get the pixel values just by using the coordinates f the pixel. And there we go! 1D is now 2D.

The function we are talking about is the pix(x,y,w,h,c) function it returns x*c+y*w*c, and it is just 1 line of code.

In the for loop we mentioned earlier where we loop through coordinates, we make an image array ie; an array of pixel values. However our task gets simpler because, if we want to access a pixel above our current pixel, the pix(x,y-1,w,h) function will just do it. We have to make image array for R G and B values in total 3 matrices, for one pixel.

The convolve function we made earlier will convolve these image arrays with the kernel, to give the sum. This can be made into average and the value can be stored back into the final image using the pixel pointer of the final image that is *pg = convolved_valueR and *(pg+1) =convolved_valueG and so on…

Now our image is ready. It can be written back to our folder for viewing using the stbi_write_jpg() function. There is also a write_png, and write_bmp function available.

CHAPTER-3
SOURCE CODE

File: boxblur.c

```
#include<stdio.h>
#include<stdlib.h>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image/stb_image_write.h"

uint8_t mmult( int s1m, int s1n, int s2m, int s2n, uint8_t (*a)[s1m][s1n], uint8_t
(*b)[s2m][s2n],uint8_t (*save)[s1m][s2n]){
    if(s1n!=s2m){
        return 0;
    }
    else{
        uint8_t sum;
        for(int i=0;i<s1m;i++){
            for(int l=0;l<s2n;l++){
                sum=0;
                for(int k=0;k<s2m;k++){
                    sum+=((*a)[i][k]*(*b)[k][l]);
                }
                (*save)[i][l]=sum;
            }
        }
    }

}
void disp(int m,int n, uint8_t (*array)[m][n]){
    printf("\n\n");
    for(int i=0;i<m;i++){printf("\n");
        for(int k=0;k<n;k++){
            printf(" %u ",(*array)[i][k]);
        }
    }
    printf("\n\n");
}

int convolve(int s1m,int s1n,int s2m,int s2n,uint8_t (*a)[s1m][s1n],int (*b)[s2m][s2n]){
    if(s1m!=s2m||s1n!=s2n){
        return 0;
    }
    int sum=0;
    int pop;
    for(int i=0;i<s1m;i++){
        for(int j=0;j<s1n;j++){
            pop=((int)(*a)[i][j])*(*b)[i][j];
            sum+=(pop);
        }
    }

    return sum;
}


void dispi(int m,int n, int (*array)[m][n]){
    printf("\n\n");
    for(int i=0;i<m;i++){printf("\n");
        for(int k=0;k<n;k++){
            printf(" %i ",(*array)[i][k]);
        }
    }
    printf("\n\n");
}
int pix(unsigned char *im,int x, int y, int w,int h,int c){return x*c+y*w*c;}

void boxblur(unsigned char *image,int w,int h,int c, unsigned char *save){

    int kernel[3][3]={{0,1,0},{1,-4,1},{0,1,0}};

    dispi(3,3,&kernel);

    //SomethingRandom{{-1,-1,-1},{-1,8,-1},{-1,-1,-1}}
    //BowxBluwr{{1,1,1},{1,1,1},{1,1,1}}
    //Image Sharpenign{{0,-1,0},{-1,5,-1},{0,-1,0}}
    //GaussianItseems{{1,2,1},{2,4,2},{1,2,1}}
    //EMBOSSS{{-2,-1,0},{-1,1,1},{0,1,2}}
    //EdgeDetection{{0,1,0},{1,-4,1},{0,1,0}}

    int image_size=w*h*c;
    int gb=(c==4)?4:3;
    int bytes=w*h*gb;
    unsigned char *bimg=malloc(bytes);

    unsigned char *p=image,*pg=bimg;
    for(int i=0;i<h;i++){
        for(int j=0;j<w;j++){

            if(i>=1&&j>=1&&i<h-1&&j<w-1){
                uint8_t imarr[3][3]={
                {*(p+pix(image,j-1,i-1,w,h,c)),*(p+pix(image,j,i-
1,w,h,c)),*(p+pix(image,j+1,i-1,w,h,c))},
                {*(p+pix(image,j-
1,i,w,h,c)),*(p+pix(image,j,i,w,h,c)),*(p+pix(image,j+1,i,w,h,c))},
                {*(p+pix(image,j-
1,i+1,w,h,c)),*(p+pix(image,j,i+1,w,h,c)),*(p+pix(image,j+1,i+1,w,h,c))},
                };

                uint8_t imarg[3][3]={
                {*(p+pix(image,j-1,i-1,w,h,c)+1),*(p+pix(image,j,i-
1,w,h,c)+1),*(p+pix(image,j+1,i-1,w,h,c)+1)},
                {*(p+pix(image,j-
1,i,w,h,c)+1),*(p+pix(image,j,i,w,h,c)+1),*(p+pix(image,j+1,i,w,h,c)+1)},
                {*(p+pix(image,j-
1,i+1,w,h,c)+1),*(p+pix(image,j,i+1,w,h,c)+1),*(p+pix(image,j+1,i+1,w,h,c)+1)},
                };

                uint8_t imarb[3][3]={
                {*(p+pix(image,j-1,i-1,w,h,c)+2),*(p+pix(image,j,i-
1,w,h,c)+2),*(p+pix(image,j+1,i-1,w,h,c)+2)},
                {*(p+pix(image,j-
1,i,w,h,c)+2),*(p+pix(image,j,i,w,h,c)+2),*(p+pix(image,j+1,i,w,h,c)+2)},
                {*(p+pix(image,j-
1,i+1,w,h,c)+2),*(p+pix(image,j,i+1,w,h,c)+2),*(p+pix(image,j+1,i+1,w,h,c)+2)},
                };
            int
ft1=(convolve(3,3,3,3,&imarr,&kernel)),ft2=(convolve(3,3,3,3,&imarg,&kernel)),ft3=(convo
lve(3,3,3,3,&imarb,&kernel));

            uint8_t f1=(ft1),f2=(ft2),f3=(ft3);

            int lo=pix(image,j,i,w,h,c);

            *(pg+lo)=f1;
            *(pg+lo+1)=f2;
            *(pg+lo+2)=f3;

            if(c==4){
                *(pg+lo+3)=*(p+lo+3);
            }
            }
        }
    }
    stbi_write_jpg("boxblur.jpg",w,h,c,bimg,100);
}

void main(){
    int w,h,c;
    unsigned char *img=stbi_load("grays.jpg",&w,&h,&c,0);
    boxblur(img,w,h,c,img);
    stbi_image_free(img);
}
```

File: boxblur5.c

```
#include<stdio.h>
#include<stdlib.h>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image/stb_image_write.h"

uint8_t mmult( int s1m, int s1n, int s2m, int s2n, uint8_t (*a)[s1m][s1n], uint8_t
(*b)[s2m][s2n],uint8_t (*save)[s1m][s2n]){
    if(s1n!=s2m){
        return 0;
    }
    else{
        uint8_t sum;
        for(int i=0;i<s1m;i++){
            for(int l=0;l<s2n;l++){
                sum=0;
                for(int k=0;k<s2m;k++){
                    sum+=((*a)[i][k]*(*b)[k][l]);
                }
                (*save)[i][l]=sum;
            }
        }
    }
    int convolve(int s1m,int s1n,int s2m,int s2n,uint8_t (*a)[s1m][s1n],int
(*b)[s2m][s2n]){
    if(s1m!=s2m||s1n!=s2n){
        return 0;
    }
    int sum=0;
    int pop;
    for(int i=0;i<s1m;i++){
        for(int j=0;j<s1n;j++){
            pop=(int)(*a)[i][j]*(*b)[i][j];
            sum+=(pop);
        }
    }


    return sum;
}
void disp(int m,int n, uint8_t (*array)[m][n]){
    printf("\n\n");
    for(int i=0;i<m;i++){printf("\n");
        for(int k=0;k<n;k++){
            printf(" %i ",(*array)[i][k]);
        }
    }
    printf("\n\n");
}


void dispi(int m,int n, int (*array)[m][n]){
    printf("\n\n");
    for(int i=0;i<m;i++){printf("\n");
```

```
        for(int k=0;k<n;k++){
            printf(" %i ",(*array)[i][k]);
        }
    }
    printf("\n\n");
}
int pix(unsigned char *im,int x, int y, int w,int h,int c){


    int xpix=x*c,ypix=y*w*c,k=0;
    k+=(xpix+ypix);
    return k;
}

void boxblur(unsigned char *image,int w,int h,int c, unsigned char *save){

   int
kernel[5][5]={{1,4,7,4,1},{4,16,26,16,4},{7,26,41,26,7},{4,16,26,16,4},{1,4,7,4,1}},
(*kern)[5][5]=&kernel;
// Gaussian
Kernel{{1,4,7,4,1},{4,16,26,16,4},{7,26,41,26,7},{4,16,26,16,4},{1,4,7,
4,1}}

    int image_size=w*h*c;
    int gb=(c==4)?4:3;
    int bytes=w*h*gb;
    unsigned char *bimg=malloc(bytes);

    unsigned char *p=image,*pg=bimg;


    for(int i=0;i<h;i++){
        for(int j=0;j<w;j++){


            if(i>1&&j>1&&i<h-2&&j<w-2){



            uint8_t imarr[5][5]={
                {*(p+pix(image,j-2,i-2,w,h,c)),*(p+pix(image,j-1,i-
2,w,h,c)),*(p+pix(image,j,i-2,w,h,c)),*(p+pix(image,j+1,i-
2,w,h,c)),*(p+pix(image,j+2,i-2,w,h,c))},
                {*(p+pix(image,j-2,i-1,w,h,c)),*(p+pix(image,j-1,i-
1,w,h,c)),*(p+pix(image,j,i-1,w,h,c)),*(p+pix(image,j+1,i-
1,w,h,c)),*(p+pix(image,j+2,i-1,w,h,c))},
                {*(p+pix(image,j-2,i,w,h,c)),*(p+pix(image,j-
1,i,w,h,c)),*(p+pix(image,j,i,w,h,c)),*(p+pix(image,j+1,i,w,h,c)),*(p+pix(image,j+2,
i,w,h,c))},
                {*(p+pix(image,j-2,i+1,w,h,c)),*(p+pix(image,j-
1,i+1,w,h,c)),*(p+pix(image,j,i+1,w,h,c)),*(p+pix(image,j+1,i+1,w,h,c)),*(p+pix(ima
ge,j+2,i+1,w,h,c))},
                {*(p+pix(image,j-2,i+2,w,h,c)),*(p+pix(image,j-
1,i+2,w,h,c)),*(p+pix(image,j,i+2,w,h,c)),*(p+pix(image,j+1,i+2,w,h,c)),*(p+pix(ima
ge,j+2,i+2,w,h,c))}


            };
            uint8_t imarg[5][5]={
                {*(p+pix(image,j-2,i-2,w,h,c)+1),*(p+pix(image,j-1,i-
2,w,h,c)+1),*(p+pix(image,j,i-2,w,h,c)+1),*(p+pix(image,j+1,i-
2,w,h,c)+1),*(p+pix(image,j+2,i-2,w,h,c)+1)},
                {*(p+pix(image,j-2,i-1,w,h,c)+1),*(p+pix(image,j-1,i-
1,w,h,c)+1),*(p+pix(image,j,i-1,w,h,c)+1),*(p+pix(image,j+1,i-
1,w,h,c)+1),*(p+pix(image,j+2,i-1,w,h,c)+1)},
                {*(p+pix(image,j-2,i,w,h,c)+1),*(p+pix(image,j-
1,i,w,h,c)+1),*(p+pix(image,j,i,w,h,c)+1),*(p+pix(image,j+1,i,w,h,c)+1),*(p+pix(ima
ge,j+2,i,w,h,c)+1)},
                {*(p+pix(image,j-2,i+1,w,h,c)+1),*(p+pix(image,j-
1,i+1,w,h,c)+1),*(p+pix(image,j,i+1,w,h,c)+1),*(p+pix(image,j+1,i+1,w,h,c)+1),*(p+p
ix(image,j+2,i+1,w,h,c)+1)},
                {*(p+pix(image,j-2,i+2,w,h,c)+1),*(p+pix(image,j-
1,i+2,w,h,c)+1),*(p+pix(image,j,i+2,w,h,c)+1),*(p+pix(image,j+1,i+2,w,h,c)+1),*(p+p
ix(image,j+2,i+2,w,h,c)+1)}


            };
            uint8_t imarb[5][5]={
                {*(p+pix(image,j-2,i-2,w,h,c)),*(p+pix(image,j-1,i-
2,w,h,c)),*(p+pix(image,j,i-2,w,h,c)+2),*(p+pix(image,j+1,i-
2,w,h,c)+2),*(p+pix(image,j+2,i-2,w,h,c)+2)},
                {*(p+pix(image,j-2,i-1,w,h,c)+2),*(p+pix(image,j-1,i-
1,w,h,c)+2),*(p+pix(image,j,i-1,w,h,c)+2),*(p+pix(image,j+1,i-
1,w,h,c)+2),*(p+pix(image,j+2,i-1,w,h,c)+2)},
                {*(p+pix(image,j-2,i,w,h,c)+2),*(p+pix(image,j-
1,i,w,h,c)+2),*(p+pix(image,j,i,w,h,c)+2),*(p+pix(image,j+1,i,w,h,c)+2),*(p+pix(ima
ge,j+2,i,w,h,c)+2)},
                {*(p+pix(image,j-2,i+1,w,h,c)+2),*(p+pix(image,j-
1,i+1,w,h,c)+2),*(p+pix(image,j,i+1,w,h,c)+2),*(p+pix(image,j+1,i+1,w,h,c)+2),*(p+p
ix(image,j+2,i+1,w,h,c)+2)},
                {*(p+pix(image,j-2,i+2,w,h,c)+2),*(p+pix(image,j-
1,i+2,w,h,c)+2),*(p+pix(image,j,i+2,w,h,c)+2),*(p+pix(image,j+1,i+2,w,h,c)+2),*(p+p
ix(image,j+2,i+2,w,h,c)+2)}


            };

            int
ft1=convolve(5,5,5,5,&imarr,&kernel),ft2=convolve(5,5,5,5,&imarg,&kernel),ft3=convo
lve(5,5,5,5,&imarb,&kernel);
            uint8_t f1=(ft1/25),f2=(ft2/25),f3=(ft3/25);

            int lo=pix(bimg,j,i,w,h,gb);
            *(pg+lo)=f1;
            *(pg+lo+1)=f2;
            *(pg+lo+2)=f3;

            if(c==4){

                *(pg+lo+3)=*(p+lo+3);
            }

            }


        }
```

```
    }
    stbi_write_jpg("boxblur5.jpg",w,h,c,bimg,100);



}



void main(){
    int w,h,c;
    unsigned char *img=stbi_load("island.jpg",&w,&h,&c,3);

    boxblur(img,w,h,c,img);

    stbi_image_free(img);
}
```

File: boxblurn.c

```c
#include<stdio.h>
#include<stdlib.h>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image/stb_image_write.h"

int convolve(int s1m,int s1n,int s2m,int s2n,uint8_t (*a)[s1m][s1n],uint8_t
(*b)[s2m][s2n],uint8_t (*save)[s1m][s2n]){
    if(s1m!=s2m||s1n!=s2n){
        return 0;
    }
    int sum=0;
    uint8_t pop;
    for(int i=0;i<s1m;i++){
        for(int j=0;j<s1n;j++){
            pop=(*a)[i][j]*(*b)[i][j];
            sum+=(int)(pop);
            (*save)[i][j]=pop;

        }
    }


    return sum;
}


int pix(unsigned char *img,int x,int y,int w,int h,int c){return x*c+y*w*c;}

int boxblur(unsigned char *img,int w,int h,int c,int radius){

    int bg=(c==4)?4:3,imagebytes=w*h*bg;
    unsigned char *bimg =malloc(imagebytes),*p=img,*pg=bimg;
    int size=2*radius+1,k=1/(radius*radius);
    uint8_t kernel[size][size],one1=(uint8_t)1;
    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
            kernel[i][j]=one1;
        }
    }
    int total=(h)*(w),count=0;
    float prc=0;
    for(int i=0;i<h;i++){
        for(int j=0;j<w;j++){

            count+=1;
            if(count%10000==0){
                prc=(count/(float)total)*100;
                printf("\n%.2f",prc);
            }

            if(i>=radius&&j>=radius&&i<h-radius&&j<w-radius){
            uint8_t imarr[size][size];
            uint8_t imarg[size][size];
            uint8_t imarb[size][size];

            int r1=0,c1=0;
            for(int t=-radius;t<=radius;t++){c1=0;
                for(int r=-radius;r<=radius;r++){

                    imarr[r1][c1]=(uint8_t)*(p+pix(img,j+t,i+r,w,h,c));
                    imarg[r1][c1]=(uint8_t)*(p+pix(img,j+t,i+r,w,h,c)+1);
                    imarb[r1][c1]=(uint8_t)*(p+pix(img,j+t,i+r,w,h,c)+2);

                    c1++;


                }
                r1++;
            }

            int f1=convolve(size,size,size,size,&imarr,&kernel,&imarr)/(size*size);
            int f2=convolve(size,size,size,size,&imarg,&kernel,&imarg)/(size*size);
            int f3=convolve(size,size,size,size,&imarb,&kernel,&imarb)/(size*size);

            uint8_t t1=f1,t2=f2,t3=f3;
            int lo=pix(img,j,i,w,h,c);
            *(pg+lo)=t1;
            *(pg+lo+1)=t2;
            *(pg+lo+2)=t3;

            if(c==4){
                int k=pix(img,j,i,w,h,c);
                *(pg+lo+3)=*(p+k);
            }
            }
            else{
                int lo=pix(img,j,i,w,h,c);
                *(pg+lo)=*(p+pix(img,j,i,w,h,c));
                *(pg+lo+1)=*(p+pix(img,j,i,w,h,c)+1);
                *(pg+lo+2)=*(p+pix(img,j,i,w,h,c)+2);

                if(c==4){
                    *(pg+lo+3)=*(p+pix(img,j,i,w,h,c));
                }

            }
        }
    }
    printf("\n100.00");
    stbi_write_jpg("boxblurn.jpg",w,h,bg,bimg,100);
}
void main(){
    int w,h,c;
    unsigned char *img=stbi_load("snow.jpg",&w,&h,&c,3);
    boxblur(img,w,h,c,10);




}
```

CHAPTER-4
RESULTS

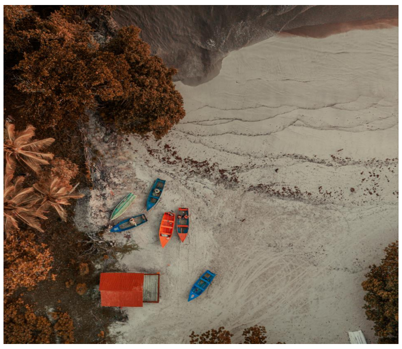
1. BoxBlur (radius=1)(size=3)

KERNELS

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

In:

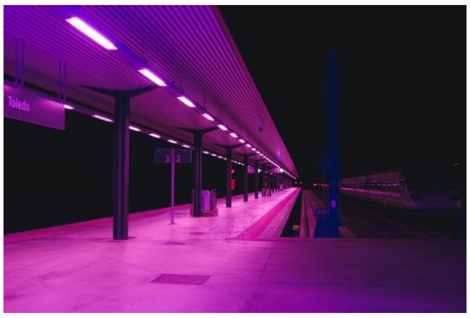
out:



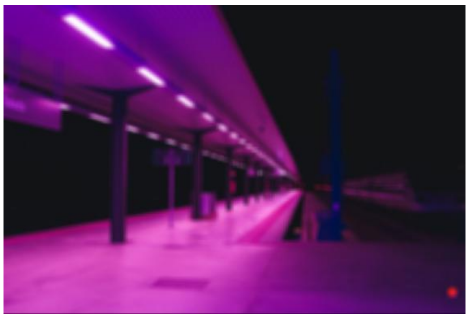
2. BoxBlur(radius=2)(size=5)

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

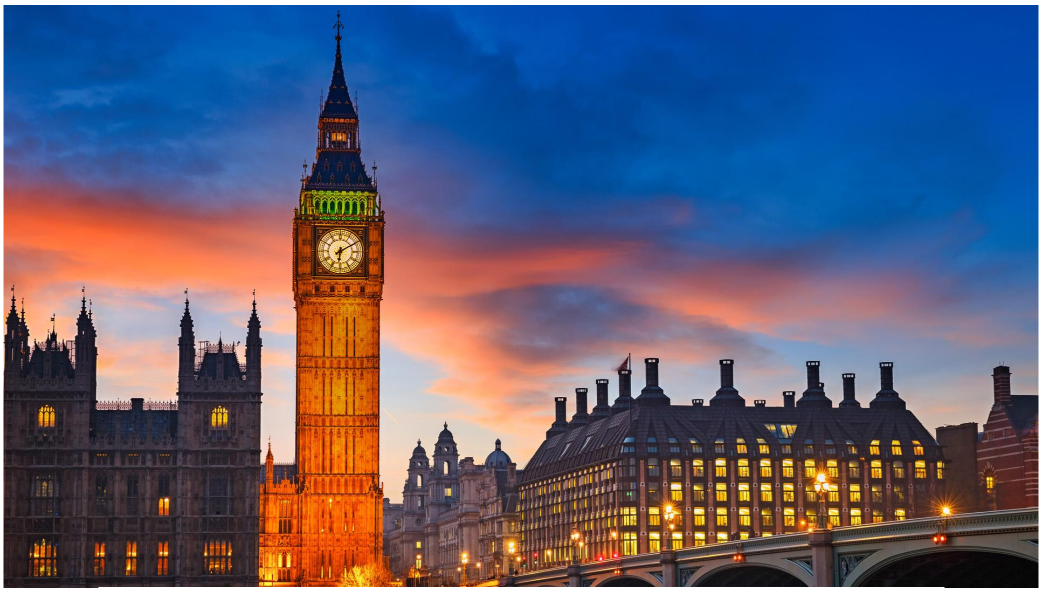
in:



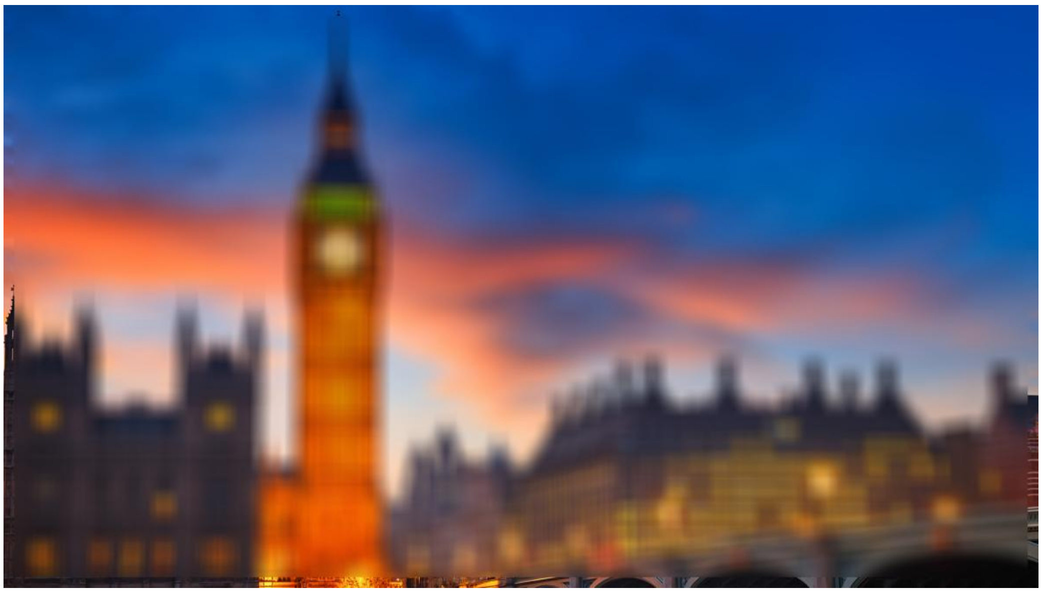
out:



3. BoxBlurn(radius=20)(size=41)

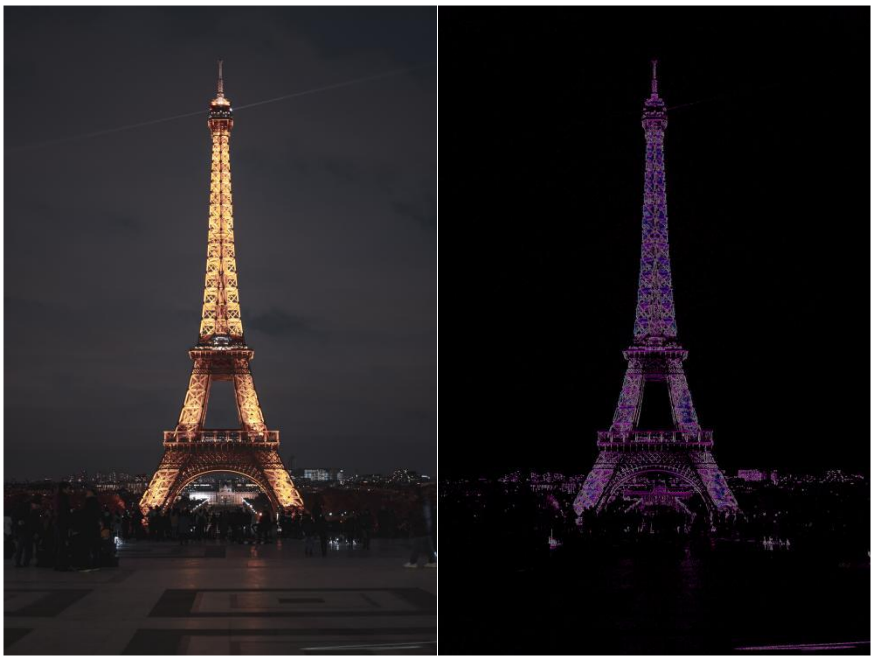


1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1



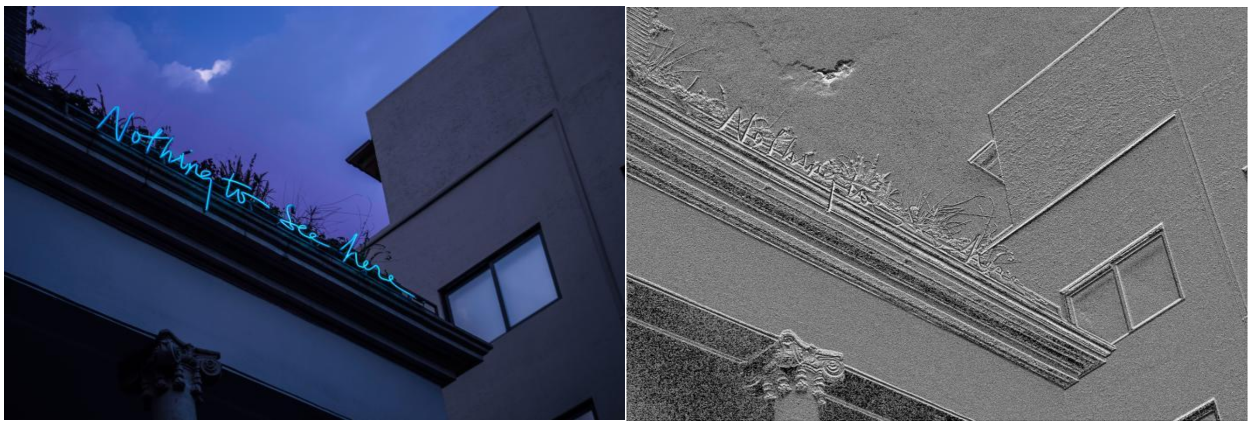
Edge Detection Kernel

```
0  1  0
1  -4  1
0  1  0
```

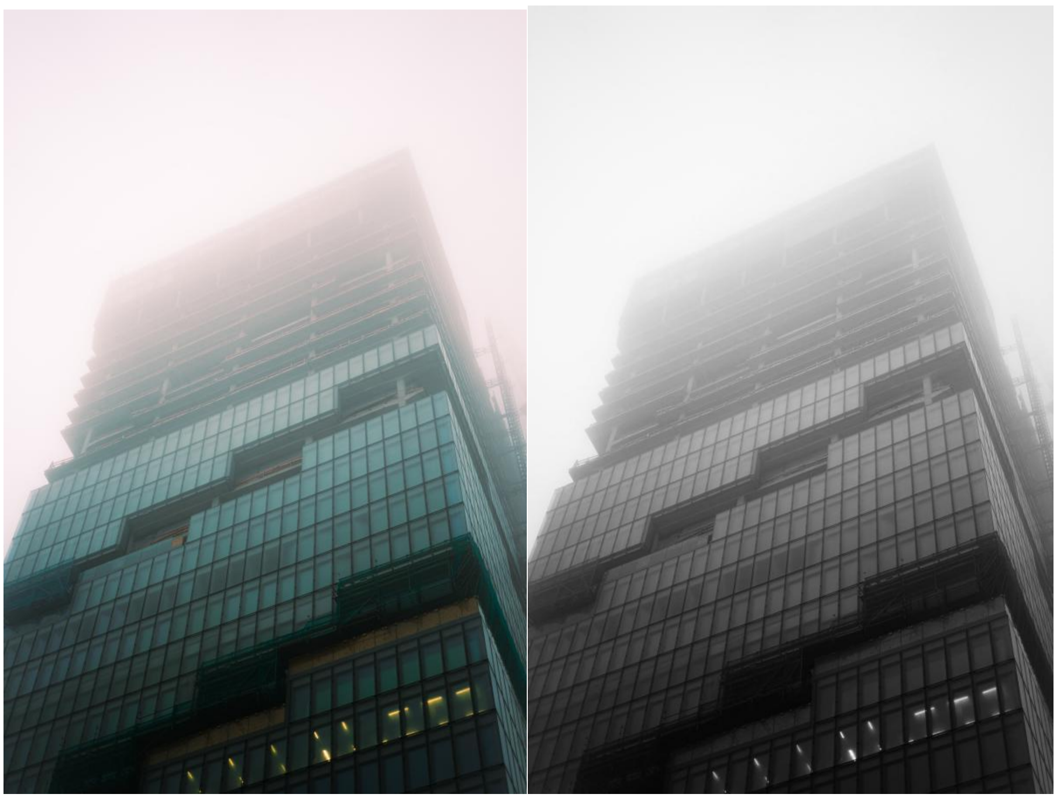


Emboss Kernel

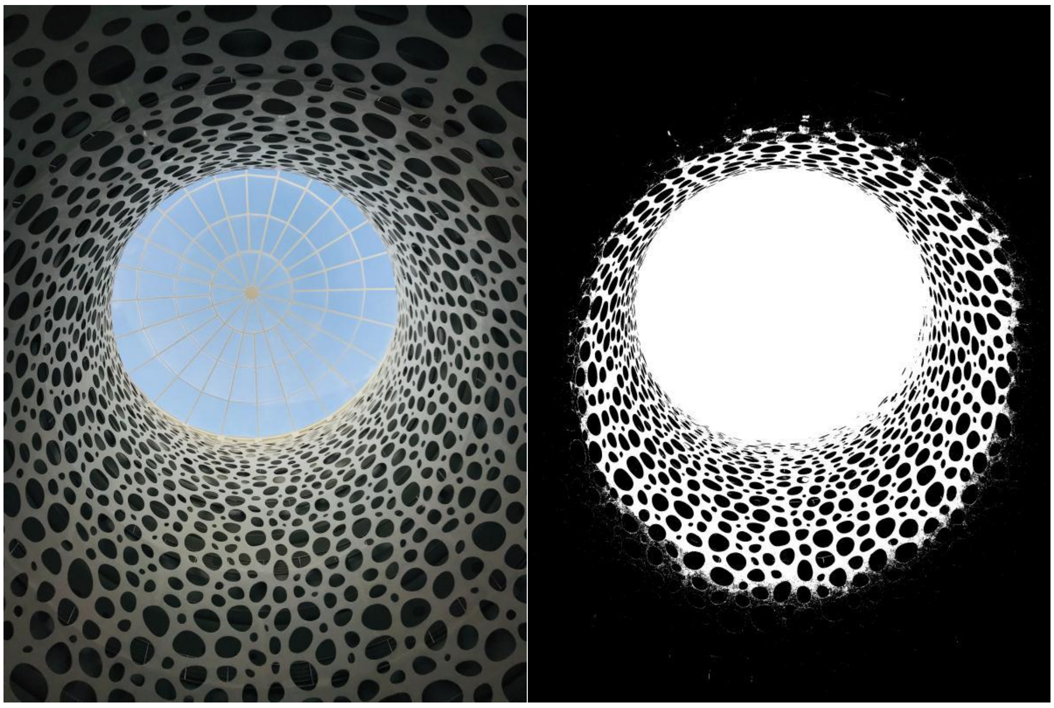
```
-2  -1  0
-1  1  1
0  1  2
```



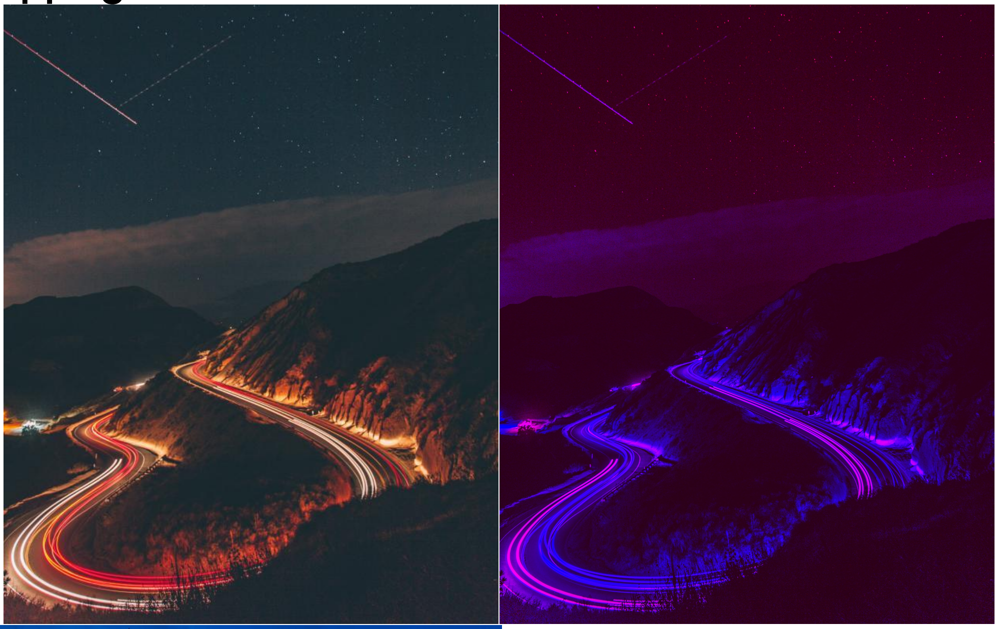
Grayscale

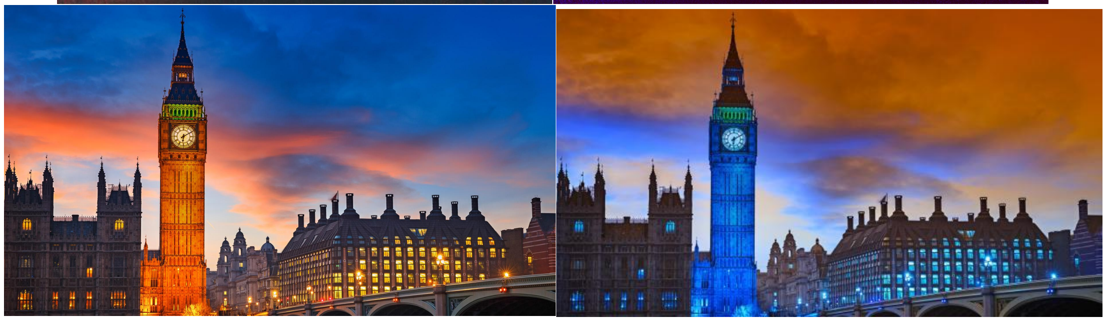


Black’nWhite

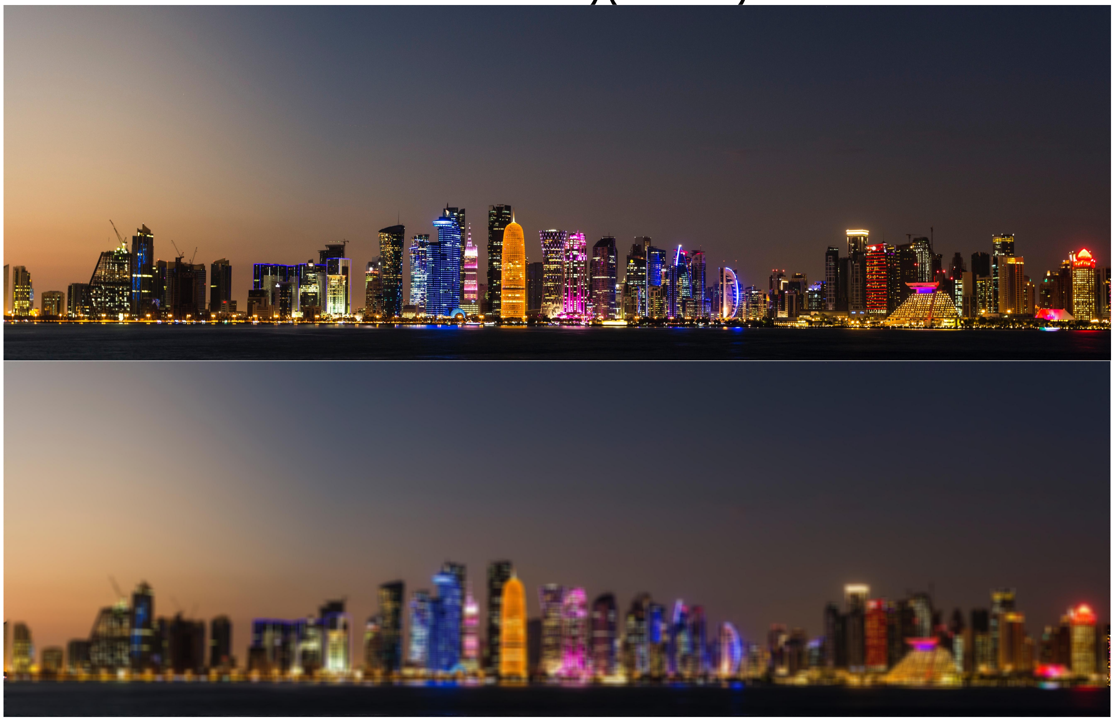


Swapping Channels





MORE BLURRING :) (LAST ONE)



CHAPTER-5
CONCLUSION

The field of Image Processing is very vast, but it is very interesting. Every single little change that you make in your kernel can give us a new Image. Its like a new discovery every time you run the program, but all we did was, change a few values. Getting this opportunity to work on this Project, has allowed us to explore this vast field and use our creativity to make something beautiful. We started by defining an Image, and then various properties of Images, like its width, height, channels etc.. How modern computers deal with large image data and so on. We also Learned how to use the stb_image Library, which allowed us to perform this Image Manipulation in the C Language, without which it might have been very difficult. We have also seen several other outputs other than Image Blurring, such as Emboss Filter, and Edge Detection, which were just minor editings done to the kernel. Now after doing all this, We feel inspired to continue this Project, and maybe explore more of Image Processing area, or maybe move on to Video Processing. So we can say, that since we have done the Project, and have obtained outputs, the Project can be concluded. But really, there is no conclusion, We are still explorers in the vast Sea.

CHAPTER-6
REFERENCES

All sources referred for making this Project are hereby accredited.
We Thank all Authors and Sources, for sharing such knowledge to the public open source.

1. Nothings(stb_image Library Creators)
https://github.com/nothings/stb

2. Solarian Programmer (Website+Tutorial)
https://solarianprogrammer.com/2019/06/10/c-programming-reading-writing-images-stb_image-libraries

3. Unsplash(Great Images for Image Processing)
https://www.unsplash.com

4. Stetosa.io(Website)
https://setosa.io/ev/image-kernels/

5. Medium(Article on Image Processing)
https://medium.com/practical-data-science-and-engineering/image-kernels-88162cb6585d

6. Python Tutorial(Explanation)
https://www.codingame.com/playgrounds/2524/basic-image-manipulation/filtering

7. Youtube Video(Reading and Writing Images in stb)
https://www.youtube.com/watch?v=1OyQoPCp46o&t=22s