CS205 C/C++ Program Design – Project 3

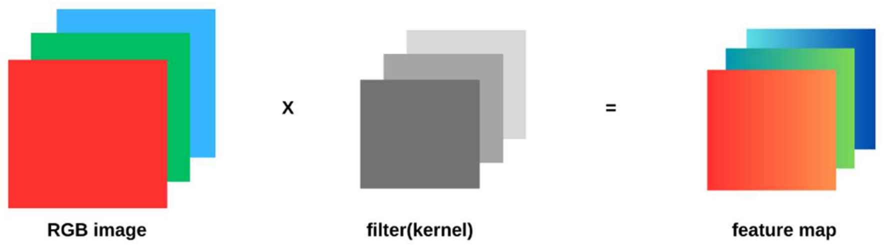
Name : Fitria Zusni Farida
SID : 12112351

Part 1 – Analysis

In this problem asking to implement the convolution layer in deep learning using C programming language. The basic idea behind a Convolutional Neural Networks (CNNs) is to apply a series of filters (kernels) to an input image with each filter looking for specific features or patterns within the image. In this problem, the data is RGB image which is represented by 3-dimensional array of floats (width, height, and number of channels). In the codes, the 3-dimensionel matrix is defined using struct. The project would perform following features:

```
typedef struct
{
    int width ;
    int height ;
    int channel ;
    float *** elements ;
} MATRIX;

static MATRIX * create_matrix3D(int channel, int width, int height);
static MATRIX * convo_operation(MATRIX* image, MATRIX* filter, int strides) ;
static MATRIX * padding(MATRIX* image, int channel, int width, int height, int pad_width, int pad_height) ;
static MATRIX * ReLU(MATRIX* image) ;
static void print_matrix(MATRIX* matrix) ;
```

1. Convolution Operation
   This operation is used to extract features from input images using filters or kernels. The filter size is smaller than the input image. The result of this multiplication with the corresponding patch of the of the input image. The result of this multiplication is then summed up to produce a single output value. This process is repeated for every patch of the input image, producing new output image, called feature map.



2. Strides
   Strides is step size in which the convolutional filter moves across the input image. Strides size determine the amount of overlap between adjacent patches of the input image. Suppose the stride size is set to n, the convolutional filter moves n pixel at a time which can reduce the amount of overlap. This provides a high level of detail in the output feature map, but also can result in very large output feature maps. Using large size of stride can reduce the computational complexity and make the

operation faster.

3. Padding

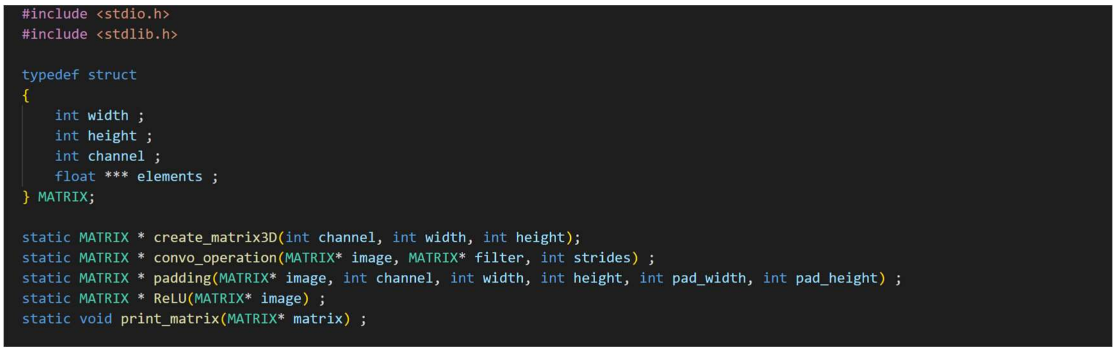
   Padding is adding extra border pixels around the input image before applying convolution operation. In this work, the extra padding is set as “0”. Padding can help to preserve the spatial dimensions of the input image and ensure that the output feature map has the same spatial dimensions as the input image. Adding (k-1)/2 pixels of padding to each side of the input image, where k is the size of convolutional filter will output feature map with spatial dimension as the input image. The choice of padding depends on the specific application and desire.

4. ReLU (Rectified Linear Unit)

   This function introduce non-linearity into the network defined as f(x)=max(0,x) which mean the output is zero for negative values of x. By zeroing out negative values, ReLU helps to prevent the vanishing gradient problem and allows the network to learn more effectively.

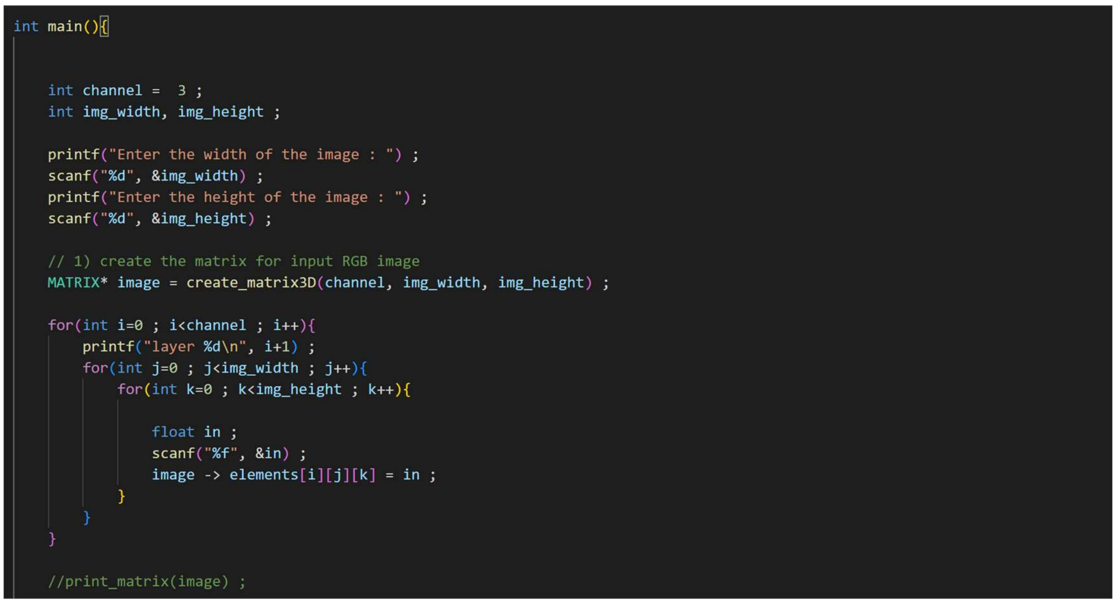
   Applying

Part 2 – Code

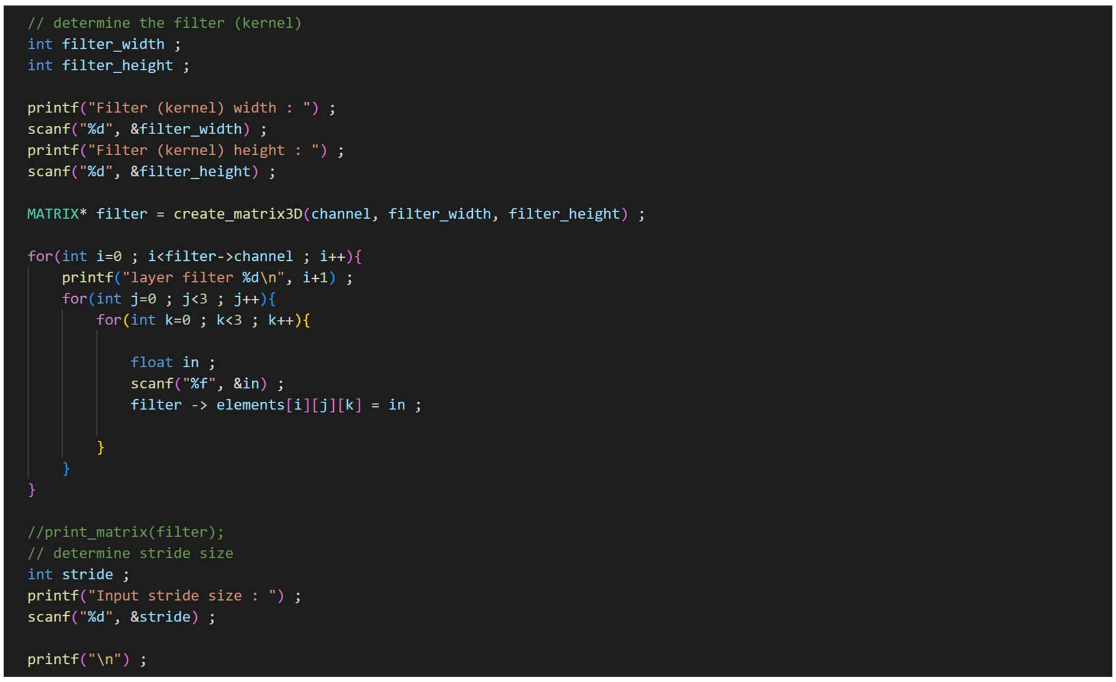


```c
#include <stdio.h>
#include <stdlib.h>

typedef struct
{
    int width ;
    int height ;
    int channel ;
    float *** elements ;
} MATRIX;

static MATRIX * create_matrix3D(int channel, int width, int height);
static MATRIX * convo_operation(MATRIX* image, MATRIX* filter, int strides) ;
static MATRIX * padding(MATRIX* image, int channel, int width, int height, int pad_width, int pad_height) ;
static MATRIX * ReLU(MATRIX* image) ;
static void print_matrix(MATRIX* matrix) ;
```

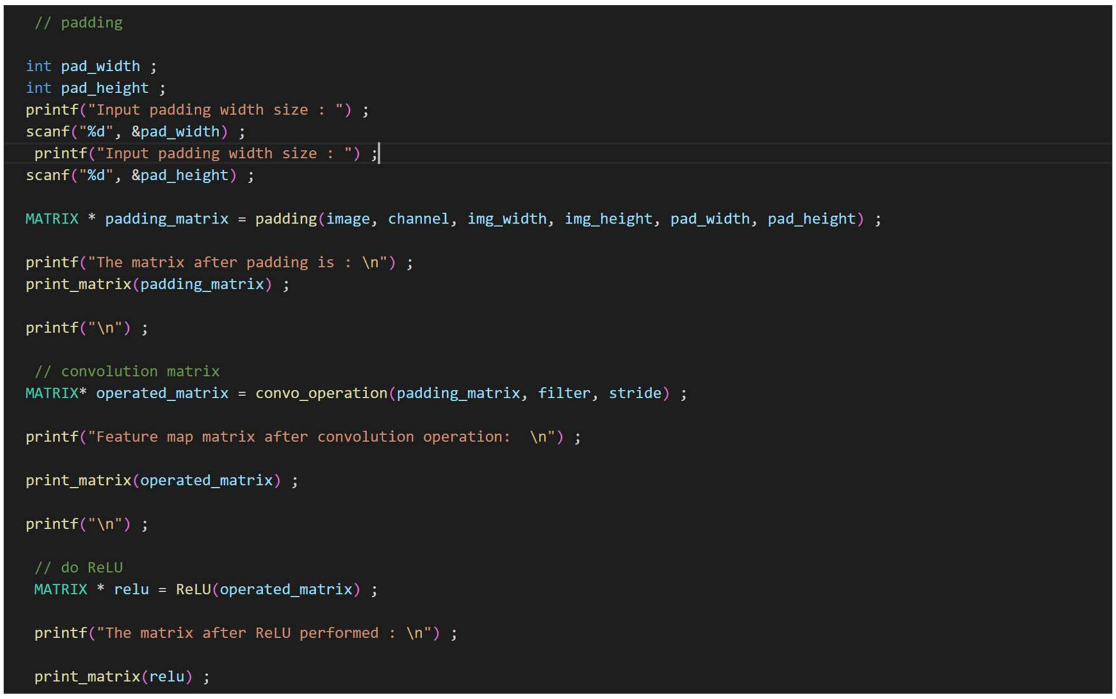
1. Main function



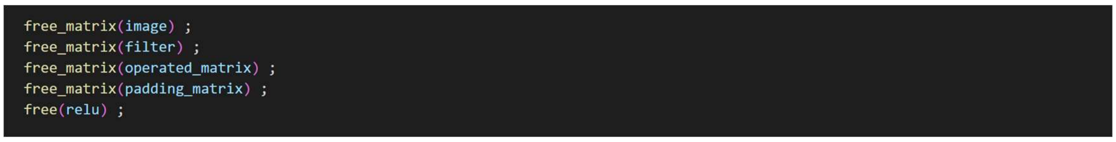
```c
int main(){


    int channel =  3 ;
    int img_width, img_height ;

    printf("Enter the width of the image : ") ;
    scanf("%d", &img_width) ;
    printf("Enter the height of the image : ") ;
    scanf("%d", &img_height) ;

    // 1) create the matrix for input RGB image
    MATRIX* image = create_matrix3D(channel, img_width, img_height) ;

    for(int i=0 ; i<channel ; i++){
        printf("layer %d\n", i+1) ;
        for(int j=0 ; j<img_width ; j++){
            for(int k=0 ; k<img_height ; k++){

                float in ;
                scanf("%f", &in) ;
                image -> elements[i][j][k] = in ;
            }
        }
    }

    //print_matrix(image) ;
```



```c
// determine the filter (kernel)
int filter_width ;
int filter_height ;

printf("Filter (kernel) width : ") ;
scanf("%d", &filter_width) ;
printf("Filter (kernel) height : ") ;
scanf("%d", &filter_height) ;

MATRIX* filter = create_matrix3D(channel, filter_width, filter_height) ;

for(int i=0 ; i<filter->channel ; i++){
    printf("layer filter %d\n", i+1) ;
    for(int j=0 ; j<3 ; j++){
        for(int k=0 ; k<3 ; k++){

            float in ;
            scanf("%f", &in) ;
            filter -> elements[i][j][k] = in ;

        }
    }
}

//print_matrix(filter);
// determine stride size
int stride ;
printf("Input stride size : ") ;
scanf("%d", &stride) ;

printf("\n") ;
```

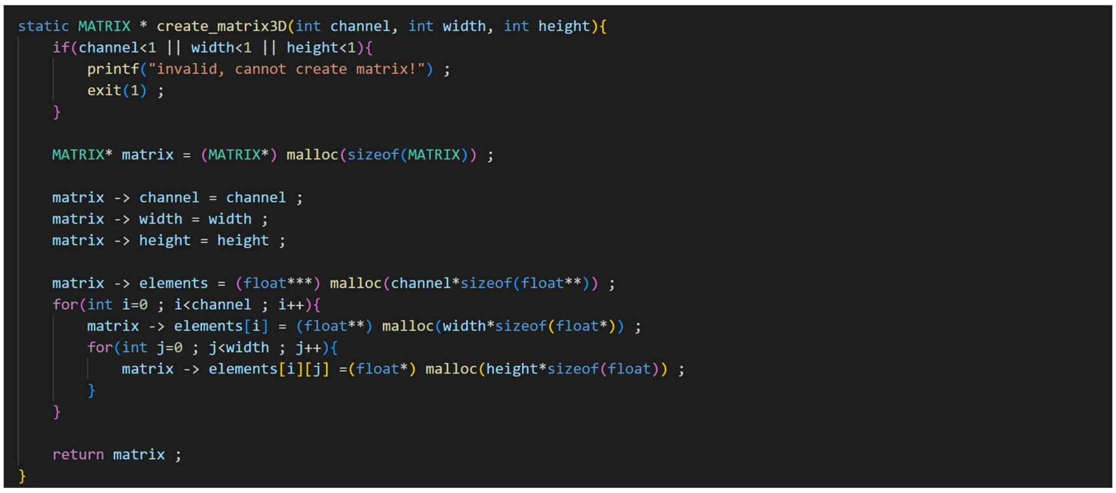


```c
 // padding

int pad_width ;
int pad_height ;
printf("Input padding width size : ") ;
scanf("%d", &pad_width) ;
 printf("Input padding width size : ") ;
scanf("%d", &pad_height) ;

MATRIX * padding_matrix = padding(image, channel, img_width, img_height, pad_width, pad_height) ;

printf("The matrix after padding is : \n") ;
print_matrix(padding_matrix) ;

printf("\n") ;

 // convolution matrix
MATRIX* operated_matrix = convo_operation(padding_matrix, filter, stride) ;

printf("Feature map matrix after convolution operation:  \n") ;

print_matrix(operated_matrix) ;

printf("\n") ;

 // do ReLU
 MATRIX * relu = ReLU(operated_matrix) ;

 printf("The matrix after ReLU performed : \n") ;

 print_matrix(relu) ;
```



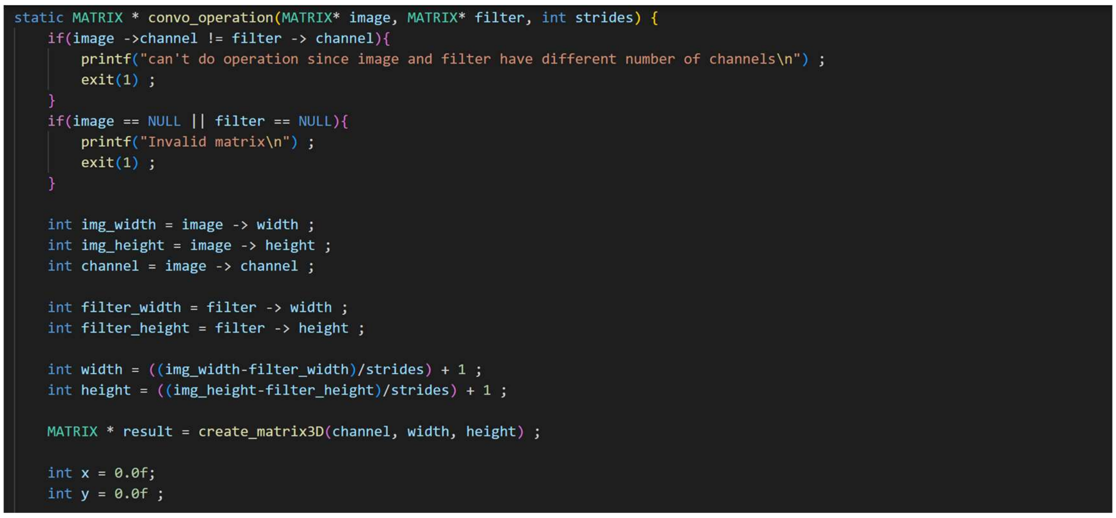
```c
free_matrix(image) ;
free_matrix(filter) ;
free_matrix(operated_matrix) ;
free_matrix(padding_matrix) ;
free(relu) ;
```

2. Create the 3D matrix

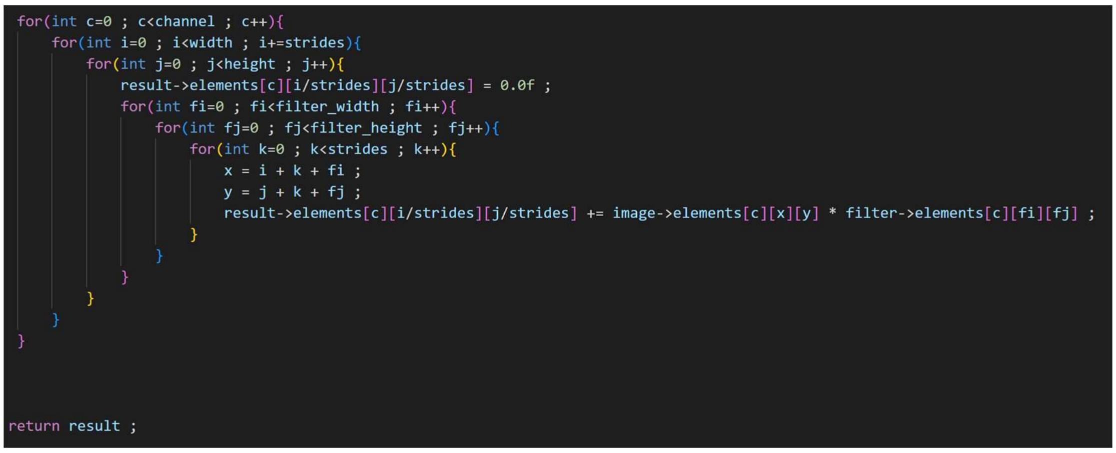


```c
static MATRIX * create_matrix3D(int channel, int width, int height){
    if(channel<1 || width<1 || height<1){
        printf("invalid, cannot create matrix!") ;
        exit(1) ;
    }

    MATRIX* matrix = (MATRIX*) malloc(sizeof(MATRIX)) ;

    matrix -> channel = channel ;
    matrix -> width = width ;
    matrix -> height = height ;

    matrix -> elements = (float***) malloc(channel*sizeof(float**)) ;
    for(int i=0 ; i<channel ; i++){
        matrix -> elements[i] = (float**) malloc(width*sizeof(float*)) ;
        for(int j=0 ; j<width ; j++){
            matrix -> elements[i][j] =(float*) malloc(height*sizeof(float)) ;
        }
    }

    return matrix ;
}
```

3. Convolution matrix operation function

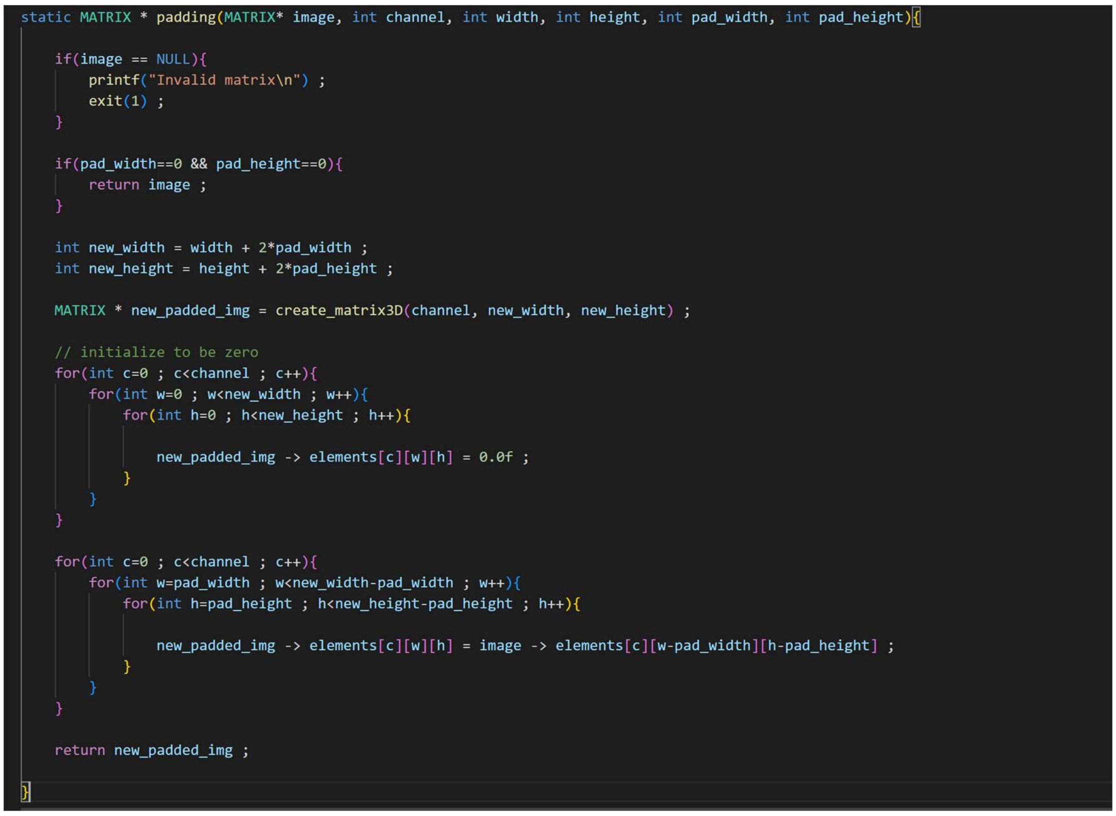


```c
static MATRIX * convo_operation(MATRIX* image, MATRIX* filter, int strides) {
    if(image ->channel != filter -> channel){
        printf("can't do operation since image and filter have different number of channels\n") ;
        exit(1) ;
    }
    if(image == NULL || filter == NULL){
        printf("Invalid matrix\n") ;
        exit(1) ;
    }

    int img_width = image -> width ;
    int img_height = image -> height ;
    int channel = image -> channel ;

    int filter_width = filter -> width ;
    int filter_height = filter -> height ;

    int width = ((img_width-filter_width)/strides) + 1 ;
    int height = ((img_height-filter_height)/strides) + 1 ;

    MATRIX * result = create_matrix3D(channel, width, height) ;

    int x = 0.0f;
    int y = 0.0f ;
```



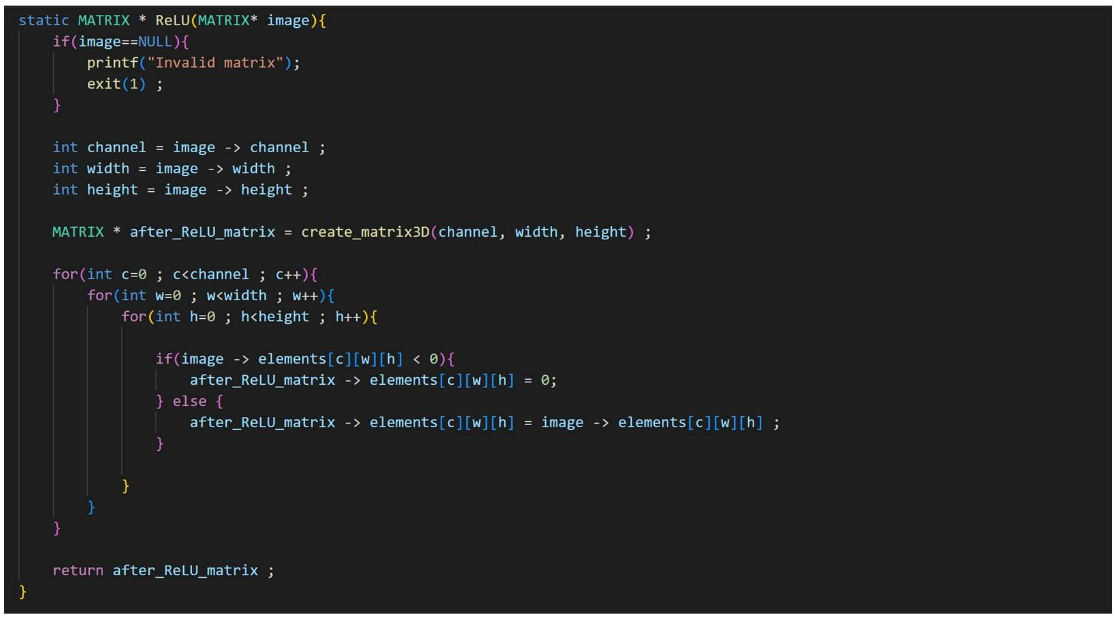
```c
for(int c=0 ; c<channel ; c++){
    for(int i=0 ; i<width ; i+=strides){
        for(int j=0 ; j<height ; j++){
            result->elements[c][i/strides][j/strides] = 0.0f ;
            for(int fi=0 ; fi<filter_width ; fi++){
                for(int fj=0 ; fj<filter_height ; fj++){
                    for(int k=0 ; k<strides ; k++){
                        x = i + k + fi ;
                        y = j + k + fj ;
                        result->elements[c][i/strides][j/strides] += image->elements[c][x][y] * filter->elements[c][fi][fj] ;
                    }
                }
            }
        }
    }
}


return result ;
```

4. Padding function



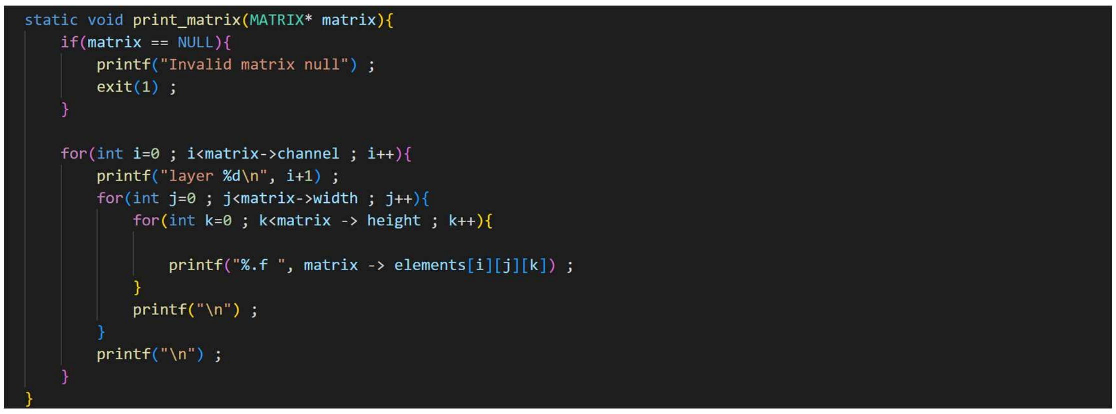
```c
static MATRIX * padding(MATRIX* image, int channel, int width, int height, int pad_width, int pad_height){

    if(image == NULL){
        printf("Invalid matrix\n") ;
        exit(1) ;
    }

    if(pad_width==0 && pad_height==0){
        return image ;
    }

    int new_width = width + 2*pad_width ;
    int new_height = height + 2*pad_height ;

    MATRIX * new_padded_img = create_matrix3D(channel, new_width, new_height) ;

    // initialize to be zero
    for(int c=0 ; c<channel ; c++){
        for(int w=0 ; w<new_width ; w++){
            for(int h=0 ; h<new_height ; h++){

                new_padded_img -> elements[c][w][h] = 0.0f ;
            }
        }
    }

    for(int c=0 ; c<channel ; c++){
        for(int w=pad_width ; w<new_width-pad_width ; w++){
            for(int h=pad_height ; h<new_height-pad_height ; h++){

                new_padded_img -> elements[c][w][h] = image -> elements[c][w-pad_width][h-pad_height] ;
            }
        }
    }

    return new_padded_img ;

}
```

5. ReLU function



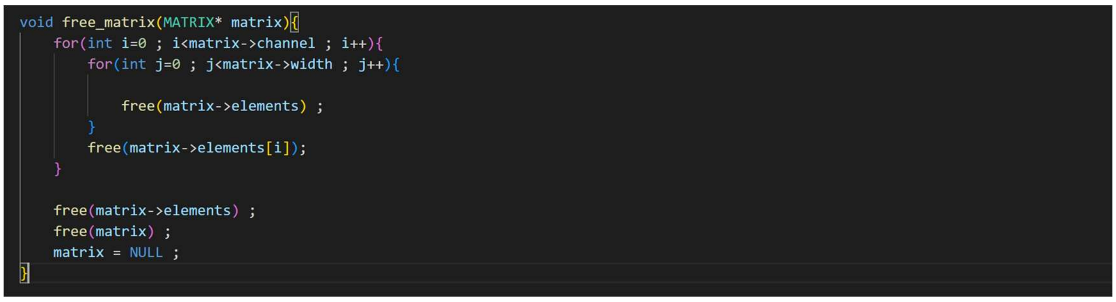
```c
static MATRIX * ReLU(MATRIX* image){
    if(image==NULL){
        printf("Invalid matrix");
        exit(1) ;
    }

    int channel = image -> channel ;
    int width = image -> width ;
    int height = image -> height ;

    MATRIX * after_ReLU_matrix = create_matrix3D(channel, width, height) ;

    for(int c=0 ; c<channel ; c++){
        for(int w=0 ; w<width ; w++){
            for(int h=0 ; h<height ; h++){

                if(image -> elements[c][w][h] < 0){
                    after_ReLU_matrix -> elements[c][w][h] = 0;
                } else {
                    after_ReLU_matrix -> elements[c][w][h] = image -> elements[c][w][h] ;
                }

            }
        }
    }

    return after_ReLU_matrix ;
}
```

6. Print the matrix



```c
static void print_matrix(MATRIX* matrix){
    if(matrix == NULL){
        printf("Invalid matrix null") ;
        exit(1) ;
    }

    for(int i=0 ; i<matrix->channel ; i++){
        printf("layer %d\n", i+1) ;
        for(int j=0 ; j<matrix->width ; j++){
            for(int k=0 ; k<matrix -> height ; k++){

                printf("%.f ", matrix -> elements[i][j][k]) ;
            }
            printf("\n") ;
        }
        printf("\n") ;
    }
}
```

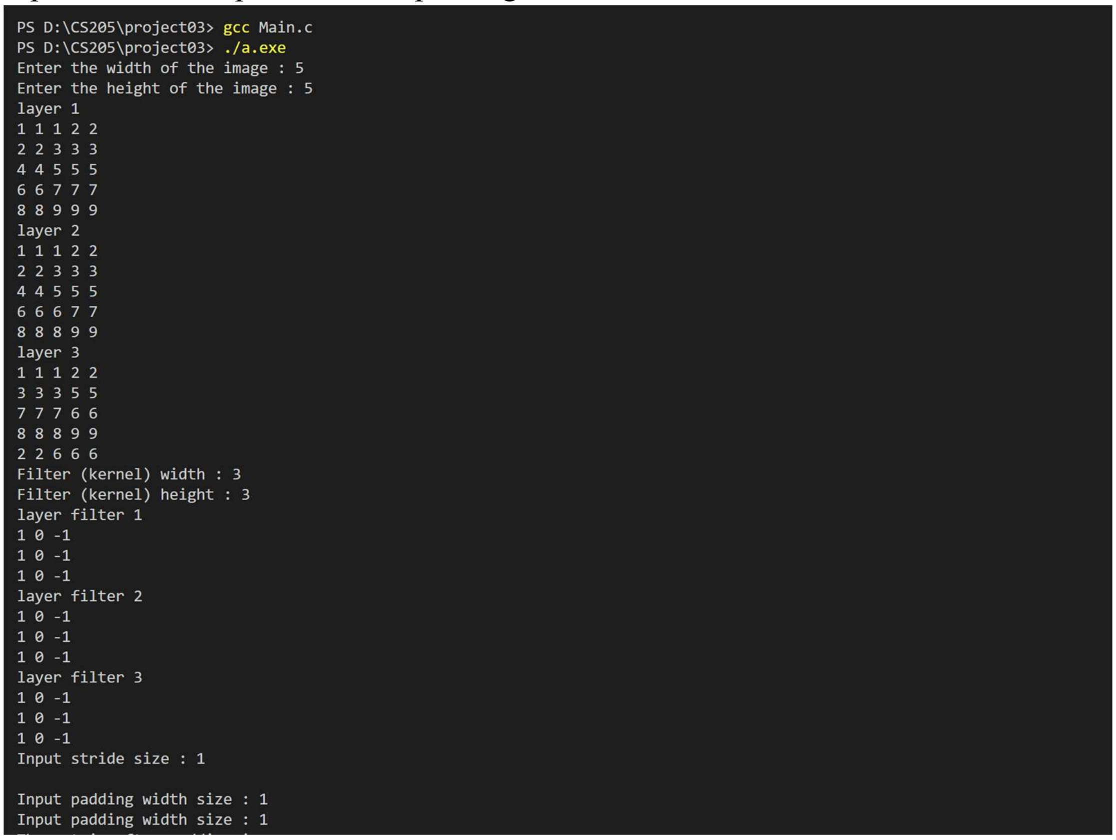
7. Free the memory



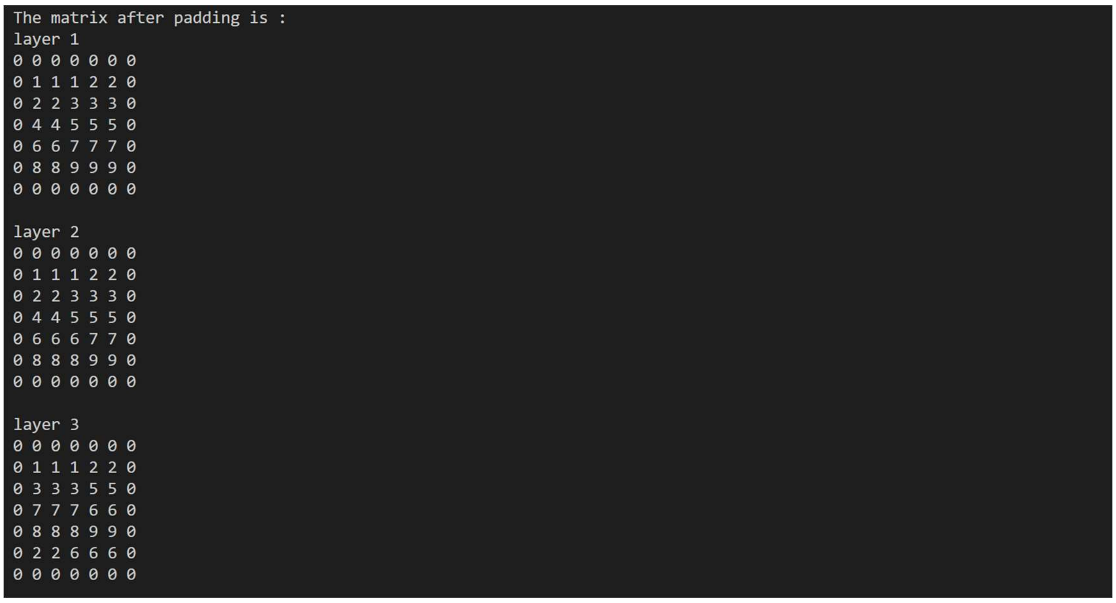
```c
void free_matrix(MATRIX* matrix){
    for(int i=0 ; i<matrix->channel ; i++){
        for(int j=0 ; j<matrix->width ; j++){

            free(matrix->elements) ;
        }
        free(matrix->elements[i]);
    }

    free(matrix->elements) ;
    free(matrix) ;
    matrix = NULL ;
}
```

Part 3 – Result and Verification

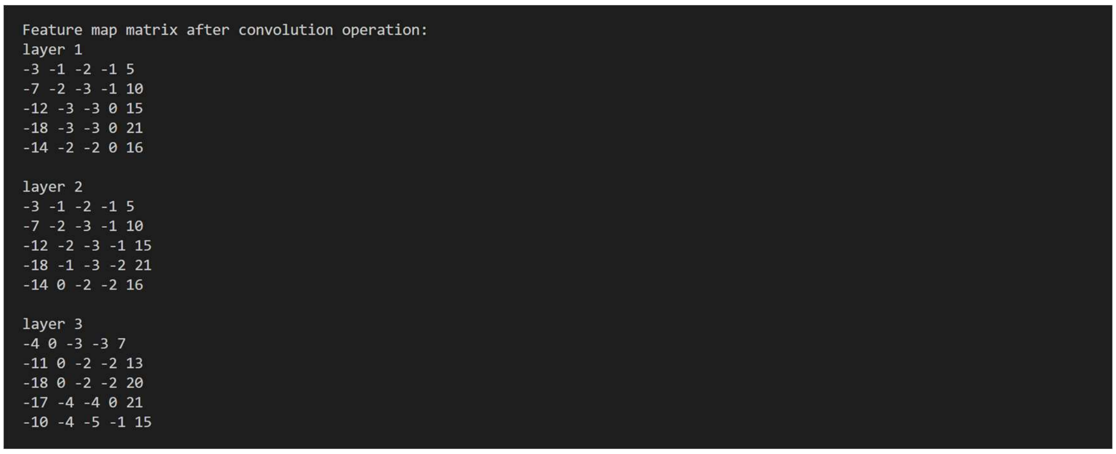
Input matrix → input strides → padding → Convolution matrix with strides → ReLU



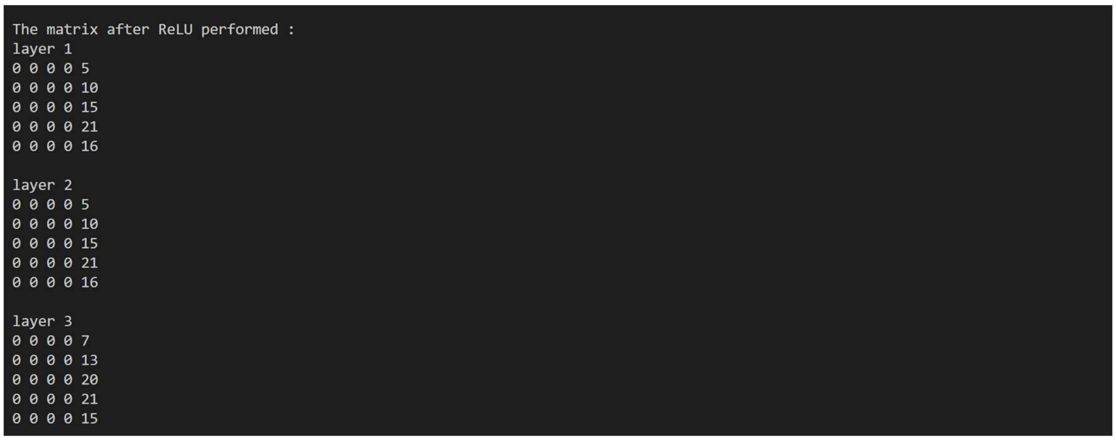
```
PS D:\CS205\project03> gcc Main.c
PS D:\CS205\project03> ./a.exe
Enter the width of the image : 5
Enter the height of the image : 5
layer 1
1 1 1 2 2
2 2 3 3 3
4 4 5 5 5
6 6 7 7 7
8 8 9 9 9
layer 2
1 1 1 2 2
2 2 3 3 3
4 4 5 5 5
6 6 6 7 7
8 8 8 9 9
layer 3
1 1 1 2 2
3 3 3 5 5
7 7 7 6 6
8 8 8 9 9
2 2 6 6 6
Filter (kernel) width : 3
Filter (kernel) height : 3
layer filter 1
1 0 -1
1 0 -1
1 0 -1
layer filter 2
1 0 -1
1 0 -1
1 0 -1
layer filter 3
1 0 -1
1 0 -1
1 0 -1
Input stride size : 1

Input padding width size : 1
Input padding width size : 1
```



```
The matrix after padding is :
layer 1
0 0 0 0 0 0 0
0 1 1 1 2 2 0
0 2 2 3 3 3 0
0 4 4 5 5 5 0
0 6 6 7 7 7 0
0 8 8 9 9 9 0
0 0 0 0 0 0 0

layer 2
0 0 0 0 0 0 0
0 1 1 1 2 2 0
0 2 2 3 3 3 0
0 4 4 5 5 5 0
0 6 6 6 7 7 0
0 8 8 8 9 9 0
0 0 0 0 0 0 0

layer 3
0 0 0 0 0 0 0
0 1 1 1 2 2 0
0 3 3 3 5 5 0
0 7 7 7 6 6 0
0 8 8 8 9 9 0
0 2 2 6 6 6 0
0 0 0 0 0 0 0
```



```
Feature map matrix after convolution operation:
layer 1
-3 -1 -2 -1 5
-7 -2 -3 -1 10
-12 -3 -3 0 15
-18 -3 -3 0 21
-14 -2 -2 0 16

layer 2
-3 -1 -2 -1 5
-7 -2 -3 -1 10
-12 -2 -3 -1 15
-18 -1 -3 -2 21
-14 0 -2 -2 16

layer 3
-4 0 -3 -3 7
-11 0 -2 -2 13
-18 0 -2 -2 20
-17 -4 -4 0 21
-10 -4 -5 -1 15
```



```
The matrix after ReLU performed :
layer 1
0 0 0 0 5
0 0 0 0 10
0 0 0 0 15
0 0 0 0 21
0 0 0 0 16

layer 2
0 0 0 0 5
0 0 0 0 10
0 0 0 0 15
0 0 0 0 21
0 0 0 0 16

layer 3
0 0 0 0 7
0 0 0 0 13
0 0 0 0 20
0 0 0 0 21
0 0 0 0 15
```

Part 4 – Difficulties and Solutions

1. The image RDB represented by 3D matrix is inputted by the user randomly, not based on actual image color and pixel size.
2. There are many kinds of filter (kernel) operated in convolution operation, user can input the filter they want. The code does not specify what filter we can use.