# Section One – Vision Task

Step One → we should  understand
1→ threshold
2→ convulsions images

---

step two → so  let's search the problem at google :)

---

step three → one of the sites that I like it
 it is https://learnopencv.com/getting-started-with-opencv/

---

step four → First, we need to implement the threshold process, it is not a difficult process, and it is necessary to check the pix values with two conditions, and then affect the formula of the new values.

```
Mat Image::makebinary(Mat image, int threshold){

    Mat result(image.size(),CV_8UC1);
    for(int i = 0 ; i < image.rows ; i++){
        for(int j = 0 ; j < image.cols ; j++){
            if(image.at<uchar>(i,j)>threshold){
                                result.at<uchar>(i,j)=255;      //Make pixel white
            }
                        else{
                                result.at<uchar>(i,j)=0;        //Make pixel black
        }
        }
    }
    return result;

}
```

One thing that should not be forgotten is that the image should be received in black and white, not in color
Because the color image has three channels, and for this process, we need to create an image that has a color spectrum between black and white

so :

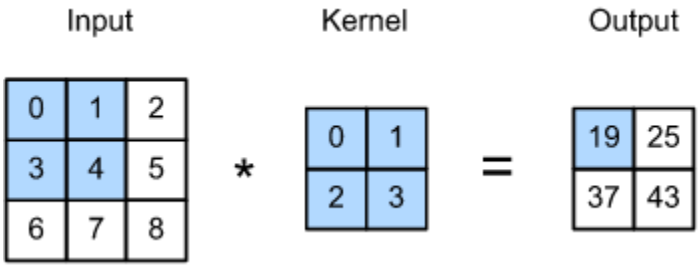
```
int Image::imageThresholding(){

    VideoCapture camera(0);
    Mat image;
    namedWindow("Binary", WINDOW_AUTOSIZE);
    int threshold = 10;
    createTrackbar("Threshold", "Binary", &threshold, 255);
    while (1)
    {
        camera >> image;
        Mat gray;
        cvtColor(image, gray, COLOR_BGR2GRAY);
        Mat binary = makebinary(gray, threshold);
        imshow("Binary", binary);
        char a = waitKey(33);
        if (a == 27)
        {
            break;
        }
    }
   return 0;

}
```

with cvtcolor() image convert rgb to grayscale picture

convulsions images:

Well, this process is very interesting and maybe simple, but I will tell you what problems we encountered!

In image processing, a kernel, convolution matrix is a small matrix used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between the kernel and an image.




For example, in this kernel, we take a 2 x 2 image on the pixel and multiply the corresponding pixels together and add them all together.

We have two type convolution :

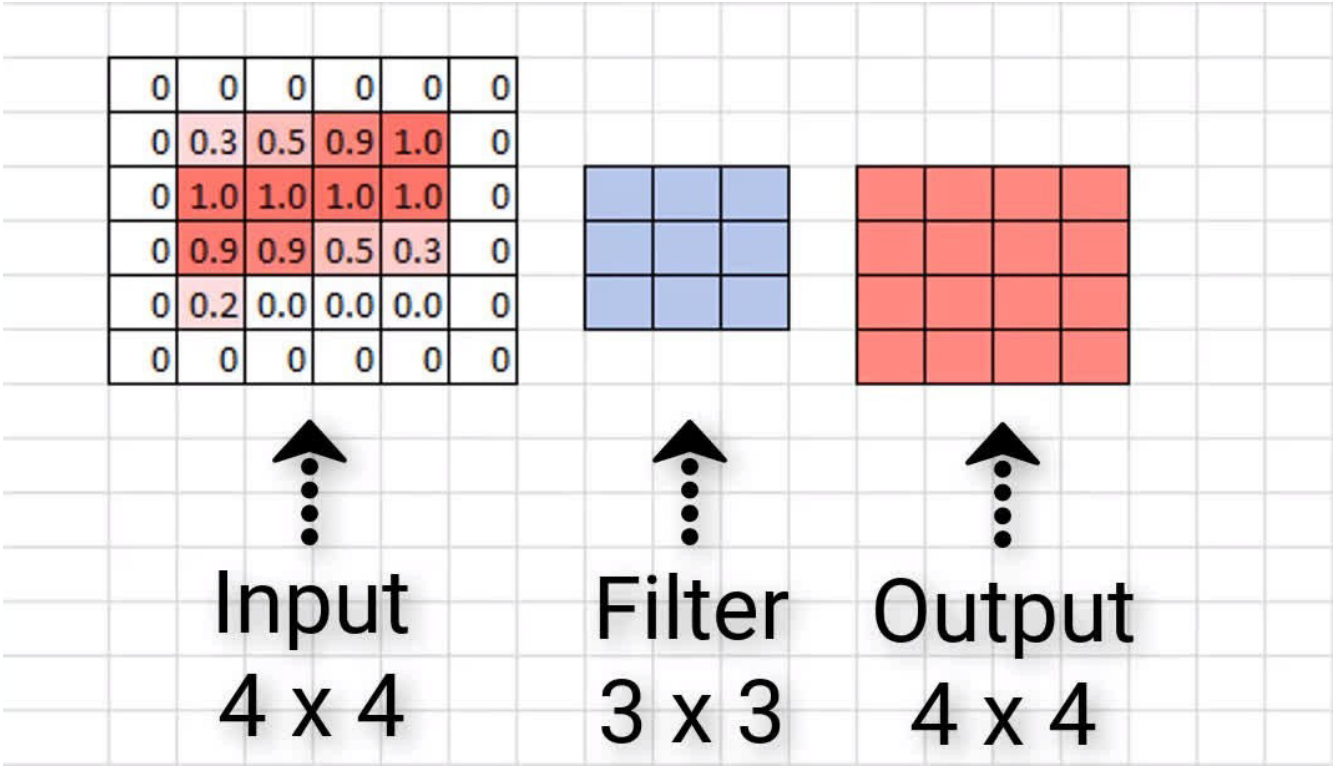
1- with padding
2-without padding

1→ Kernel With padding:
Kernel convolution usually requires values from pixels outside of the image boundaries. There are a variety of methods for handling image edges.

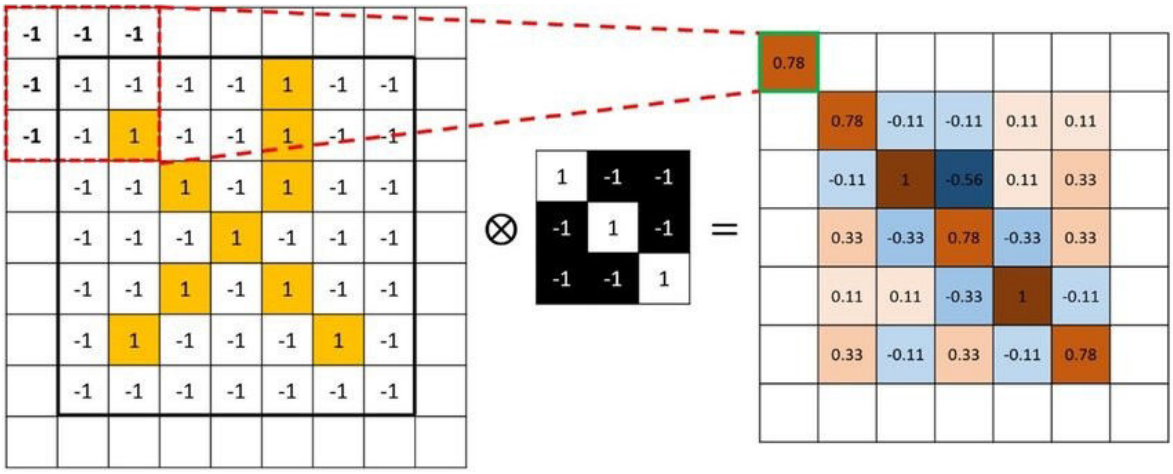
In this method, when we scroll the image and the values are calculated and added together, the output image is exactly the size of the input image.

There are different ways to implement it, such as:

Zero padding :




mirror padding:



reflection padding :

| | | |
|---|---|---|
| 3 | 5 | 1 |
| 3 | 6 | 1 |
| 4 | 7 | 9 |

No padding

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 6 | 3 | 6 | 1 | 6 | 3 |
| 1 | 5 | 3 | 5 | 1 | 5 | 3 |
| 1 | 6 | 3 | 6 | 1 | 6 | 3 |
| 9 | 7 | 4 | 7 | 9 | 7 | 4 |
| 1 | 6 | 3 | 6 | 1 | 6 | 3 |

(1, 2) reflection padding

```
Mat Image::padding(Mat img, int k_width, int k_height, string type){

        Mat scr;
        img.convertTo(scr, CV_64FC1);
        int pad_rows, pad_cols;
        pad_rows = (k_height - 1) / 2;
        pad_cols = (k_width - 1) / 2;
        Mat pad_image(Size(scr.cols + 2 * pad_cols, scr.rows + 2 * pad_rows), CV_64FC1, Scalar(0));
        scr.copyTo(pad_image(Rect(pad_cols, pad_rows, scr.cols, scr.rows)));
        // mirror padding
        if (type == "mirror")
        {
                for (int i = 0; i < pad_rows; i++)
                {
                        scr(Rect(0, pad_rows - i, scr.cols, 1)).copyTo(pad_image(Rect(pad_cols, i, scr.cols, 1)));
                        scr(Rect(0, (scr.rows - 1) - pad_rows + i, scr.cols, 1)).copyTo(pad_image(Rect(pad_cols,
                                (pad_image.rows - 1) - i, scr.cols, 1)));
                }

                for (int j = 0; j < pad_cols; j++)
                {
                        pad_image(Rect(2 * pad_cols - j, 0, 1, pad_image.rows)).copyTo(pad_image(Rect(j, 0, 1, pad_image.rows)));
                        pad_image(Rect((pad_image.cols - 1) - 2 * pad_cols + j, 0, 1, pad_image.rows)).
                                copyTo(pad_image(Rect((pad_image.cols - 1) - j, 0, 1, pad_image.rows)));
                }

                return pad_image;
        }
        // replicate padding
        else if (type == "replicate")
        {
                for (int i = 0; i < pad_rows; i++)
                {
                        scr(Rect(0, 0, scr.cols, 1)).copyTo(pad_image(Rect(pad_cols, i, scr.cols, 1)));
                        scr(Rect(0, (scr.rows - 1), scr.cols, 1)).copyTo(pad_image(Rect(pad_cols,
                                (pad_image.rows - 1) - i, scr.cols, 1)));
                }

                for (int j = 0; j < pad_cols; j++)
                {
                        pad_image(Rect(pad_cols, 0, 1, pad_image.rows)).copyTo(pad_image(Rect(j, 0, 1, pad_image.rows)));
                        pad_image(Rect((pad_image.cols - 1) - pad_cols, 0, 1, pad_image.rows)).
                                copyTo(pad_image(Rect((pad_image.cols - 1) - j, 0, 1, pad_image.rows)));
                }
                // zero padding
                return pad_image;
        }
```

Without padding:

We have to make a condition that the kernel does not go out of the picture and the calculation is done correctly.

```
void Image::applyFilter() {

    Mat src((image.rows - size + 1),(image.cols - size + 1),image.type());
    int dx = size / 2;
    int dy = size / 2;
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            sumIndex = 0;
            for (int k = 0; k < size; k++) {
                for (int l = 0; l < size; l++) {
                    if(ChoiceConvolvemulti == 1 ){
                            int x = j - dx + l;
                            int y = i - dy + k;
                            if (x >= 0 && x < src.cols && y >= 0 && y < src.rows){
                            sumIndex += (uchar) arrKernel[k][l] * image.at<uchar>(y, x) * ratio ;
                    }}
                    if( ChoiceConvolvemulti == -1){
                            int x = j - dx + l;
                            int y = i - dy + k;
                            if (x >= 0 && x < src.cols && y >= 0 && y < src.rows){
                               sumIndex += kernel.at<uchar>(k,l) * image.at<uchar>(y, x) * ratio ;
                           }
                    }
                }
            }
            src.at<uchar>(i, j) = saturate_cast<uchar>(sumIndex) ;
```

We create an empty matrix with the size calculated according to the following formula and put the new pixel values in it

You can find it in two ways: simple method: input_size - (filter_size - 1)

```
W - (K-1)
Here W = Input size
         K = Filter size
         S = Stride
         P = Padding
```

for example Sharpen filter: