DogoRangsang Research Journal
ISSN: 2347-7180

UGC Care Group I Journal
Vol-13 Issue-02 Nov 2023

# A Review of Basic Image Processing Techniques

***Dr. N. Vinaya Kumari[1], Chandra Sekhara Pramod Chadalapaka[2], Praveen Madela[3], Teja Kondaparthy[4], Venkat Sai Sunnam[5]***

*[1]Associate Professor, [2,3,4,5]Students*

*Department of Artificial Intelligence and Machine Learning*

*Malla Reddy Institute of Technology and Science, Hyderabad, India.*

*Email: : vinayakumari@gmail.com, chandrasekharapramod@gmail.com, madelapraveen35@gmail.com, kondaparthiteja33@gmail.com, venkatsai4884@gmail.com*

## Abstract

***Image processing refers to the act of reading an image, and then procedurally extracting information performing certain operations upon it. Image manipulation refers to the process of changing that image to suit a certain use-case. In this article, we review the basic techniques of image processing and manipulation, beginning with the nuances of the actual file formats in which images might be stored, to specific techniques of blurring or detecting edges of an image.***

***In the end, we state our opinions on the several techniques and optimizations thereof that we have seen and include our general opinion on newer methods of image processing and manipulation, including those built on machine learning models and other purpose-built AI models.***

## I. Introduction

Image processing as a field is as old as the camera. Digital image processing is a newer field, and many consider it to have been firmly established as its own field in the 1960s, at AT&T Bell Labs, the JPL, MIT, the University of Maryland, and others. The focus back then was on processing images to be better viewed and understood by humans.

There are three main operations we will be reviewing in this article excluding those that depend upon colour. These are *blurring*, *sharpening*, and *edge detection*.

**Blurring** is the process of manipulating in an image in such a way that pixels next to each other blend in together, leading to a smoother image and most of the time loss of detail. Blurring is commonly used to upscale low-resolution images, and most web browsers implement some version of blurring to scale images to fit larger dimensions.

**Sharpening** is the exact opposite of blurring. Sharpening is the process of manipulating an image so that influences of neighbouring pixels on every pixel is reduced, therefore improving contrast between neighbouring pixels. This process is usually known as *Unsharp Masking*.

**Edge Detection** is the process of identifying *edges*, or curves with high variation in brightness or colour in pixels on either side. Edge detection is often used to digitally mimic lens focus on mobile phones, and is also used in image editing software to recognize the boundaries of objects to be or to not be edited.

## II. The Bitmap File Format

Bitmap refers to a technique of storing image data: literally mapping groups of bits to pixels of the image. There are multiple bitmap image formats, but the most widely known and used is the .bmp file format, developed by Microsoft for use with IBM's OS/2 and Microsoft Windows.[1]

This file format is based on what Microsoft called DIBs, or Device-Independent Bitmaps. This is in contrast with device-dependent bitmaps, which are bitmaps created usually in memory, usually by applications, to store image data. Loaded into memory, the DIB files become a DIB data structure, which is a crucial part of Windows' GDI API.[2] This data structure does not contain the BMP header that defines a Bitmap file and is usually in the form of a packed image, with no gap fields.

### The Structure of the BMP file

A BMP file consists of four main parts: the *BITMAPFILEHEADER*, the *BITMAPINFOHEADER*, the *COLOUR TABLE*, and the actual *PIXEL DATA*.

The 14-byte long **BITMAPFILEHEADER** begins with the ASCII characters 'BM'[3]signifying to any application reading this file that it is a bitmap file. It is then followed by a four-byte *file size* field in which the size of the BMP file must be stored. There are then two two-byte *reserved* fields which can be used by the applications that write the image, and finally, there is the four-byte *Pixel Data Offset* field which stores the location of where the pixel data starts.

The **BITMAPINFOHEADER**[4]is less rigid in its contents. Many variations of the BMP file format have this field in different lengths, starting with 40 in the original Microsoft implementation. To this end, the first field, the four-byte Header Size field is *not* guaranteed to be 40 bytes. Applications must therefore take care to read this field for the header size instead of assuming it to be 40.

There are then two 2 byte fields denoting the width and the height of the image respectively. The next field is the number of colour planes, which is usually 1. The next field signifies *colour depth*, or the number of bits used to represent each colour channel. Typical values are 1, 4, 8, 16, 24, and 32. This is followed by fields for the compression method used, the raw size in bytes of the pixel array, the horizontal resolution and the vertical resolution of the image in *ppm* or *pixels per metre*, the number of colours used, and the number of *important* colours used.

There may be additional fields following this depending on the specific variation of the header.

The **PIXEL DATA** field is the raw pixel data, consisting of the padded bits of data representing each pixel. Each pixel must be $2^n$ bits long, and padding is used in cases this is not directly possible.[4]

**Bitmap File Structure**

| Block | Field | Width | Description |
|---|---|---|---|
| **BITMAPFILEHEADER** Fields: 5 Width: 14 bytes | FileType | 2 bytes | A 2 character string value in ASCII. It must be 'BM' or '0x42 0x4D' |
| | FileSize | 4 bytes | An integer (unsigned) representing entire file size in bytes (number of bytes in a BMP image file ) |
| | Reserved | 2 bytes | To be utilized by an image processing application. Initialized to '0' integer (unsigned) value. |
| | Reserved | 2 bytes | To be utilized by an image processing application. Initialized to '0' integer (unsigned) value. |
| | PixelDataOffset | 4 bytes | An integer (unsigned) representing the offset of actual pixel data in bytes. |
| **BITMAPINFOHEADER** Fields: 11 Width: 40 bytes | HeaderSize | 4 bytes | An integer (unsigned) representing the size of the header in bytes. It should be '40' in decimal. |
| | ImageWidth | 4 bytes | An integer (signed) representing the width of the final image in pixels. |
| | ImageHeight | 4 bytes | An integer (signed) representing the height of the final image in pixels. |
| | Planes | 2 bytes | An integer (unsigned) representing the number of color planes. Should be '1' in decimal. |
| | BitsPerPixel | 2 bytes | An integer (unsigned) representing the number of bits a pixel takes to represent a color. |
| | Compression | 4 bytes | An integer (unsigned) representing the value of compression to use. Should be '0' in decimal. |
| | ImageSize | 4 bytes | An integer (unsigned) representing the final size of the compressed image. Should be '0' in decimal. |
| | XpixelsPerMeter | 4 bytes | An integer (signed). Should be set to '0' in decimal to indicate no preference of the target device. |
| | YpixelsPerMeter | 4 bytes | An integer (signed). Should be set to '0' in decimal to indicate no preference of the target device. |
| | TotalColors | 4 bytes | An integer (unsigned) representing the number of colors in the color pallet. |
| | ImportantColors | 4 bytes | An integer (unsigned) representing the number of important colors. Ignored by setting '0' in decimal. |
| **COLOR TABLE** Fields: 4 x entries Width: 4 x entries | Red | 1 bytes | An integer (unsigned) representing Red color channel intensity. |
| | Green | 1 bytes | An integer (unsigned) representing Green color channel intensity. |
| | Blue | 1 bytes | An integer (unsigned) representing Blue color channel intensity. |
| | Reserved | 1 bytes | An integer (unsigned) reserved for other uses. Should be set to '0' in decimal |
| **PIXEL DATA** | | | An array of pixel values with padding bytes. A pixel value defines the color of the pixel. |

*Figure 1: The structure of the BITMAP file header*

## III. Image Convolution

Image convolution is the process of convolving an image with another image, or more frequently, a *kernel*.[6] A kernel can be defined as some matrix that is the function that governs the relationship between the pixels of the convoluted image and its neighbours.

For some kernel ω, with an image $f(x,y)$, the convolution*g* is defined as

$$g(x,y) = \omega * f(x,y) = \sum_{i=-a}^{a}\sum_{j=-b}^{b} \omega(i,j) f(x-i, y-j)$$

For some specific cases, kernels must be *normalized*, i.e., the elements be divided by the sum of all elements so that the sum of the kernel is unity.

# Edge and corner case handling

The nature of kernel convolution often requires pixels outside of the bounds of the image. There are different known techniques to handle such situations:

i. **Extending** in which the border of the image is extended as far as required. The edge pixels are replicated in lines, while corner pixels are replicated in wedges to fill the gap between edges.
ii. **Tiling** where the image is considered to be*tiled*, or repeating, and therefore pixels from the other side of the image are used.
iii. **Mirroring** where the image is considered mirrored on the edge, and therefore for every pixel index outside the image bounds, the pixel with the same distance from the edge *inside* the bounds is considered instead.
iv. **Image Cropping** where all pixels that require pixels outside the image bounds are excluded from the output.
v. **Kernel Cropping** where for pixels that require values out of bounds, only the pixels within the bounds are considered and the kernel is cropped and normalized accordingly to fit the case.
vi. **Constant Bounds** where all pixels outside of the image bounds are considered to have a constant value, usually black, white, or some shade of grey.

In this article, we implement the mirroring method.

## IV. Blurring

Blurring is a technique that attempts to smoothen the look of an image by smoothening the transition of colour between pixels. There are different types of natural and digital blurs, but we will chiefly be reviewing two of these: *Box Blur* and *Gaussian Blur*.

## A. Box Blur

In a Box Blur, the blur radius *r* corresponds to a convolution kernel $K_{n\times n}$ where $□ = 2□ + 1$. The box blur is the simplest of convolution operations, with the kernel K being defined as

$$□_{□,□} = \frac{1}{□^2} = \frac{1}{4□^2 + 4□ + 1}$$

This operation corresponds to setting the value of each pixel to the *average* of the values of all its neighbouring pixels within a radius *r*.

*Figure 2*
*Left: Original image.Right: Image after a box blur with r = 5*

One problem with the box blur is that it gives equal weight to all pixels within the radius, when, to preserve accuracy, the current pixel should be given the highest weightage, followed by pixels in ascending order of the distance from the current pixel. However, this also makes the box blur faster to compute than other blurring methods due to the relatively simple computations required.

Repeated applications of box blur approximate a different kind of blur in which the current pixel does indeed have greater weightage.[5] This is because of the Central Limit Theorem.[6] If we consider each pixel $I_{x,y}$ of the image a discrete random variable, with a probability distribution $I$, it can be shown that each pixel $H_{x,y}$ of the output image $H$ is a sample mean of the original probability distribution $I$. Applying this same convolution operation multiple times, the distribution of the sample means must therefore approximate the normal distribution according to the Central Limit Theorem. In this case, repeated application of box blur approximates the *Gaussian* blur.

## B. Gaussian Blur

In the Gaussian blur, every pixel being blurred is given higher weightage than the pixels surrounding it, which are given lower weightage the farther they are from the current pixel. This is due to the nature of the Gaussian kernel and the Gaussian distribution.

The Gaussian function G with a mean $\mu = 0$ is defined as

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

Where σ is the standard deviation.

The two-dimensional Gaussian kernel G can therefore be defined as[7]:

$$G_{x,y} = G(x)G(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{y^2}{2\sigma^2}\right)$$

$$= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

The standard deviation $\sigma$ determines the size of the kernel. The relationship between the radius and the standard deviation must be defined according to requirement. The most common relation, used by nVidia and OpenCV[8], is to set the blur radius $r = \text{round}(3\sigma)$ where $\text{round}(x)$ is the nearest integer function, which sets the kernel size $n$ to be $6\sigma + 1$.

*Figure 3*
*Left: Original image. Right: Gaussian blur with standard deviation σ = 2 (kernel size 13).*

However, this approach of convolving the 2D Gaussian matrix is very computationally expensive. A better approach would be to convolve the image first with the horizontal Gaussian vector $G_x$ and then with the vector □□, which dramatically decreases the computational complexity while resulting in the same values, since the 2D Gaussian Kernel is equivalent to the cross product of the two Gaussian vectors $G_x$ and $G_y$.

## V. Image Sharpening

Image sharpening can be achieved by using a kernel that removes the influence of the surrounding pixels on the current pixel, thereby undoing the effects of convolutions such as Gaussian or box blurs. The sum of the sharpening kernel must still be unity as to keep the pixel values within range.The simplest sharpening kernel of size 3 would be:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

***Figure 4: A 3x3 sharpening kernel***

This kernel gives *negative* influence to the surrounding pixels of an image, thereby contrasting the current pixel more than before. The result of convolving this kernel can be seen below.

*Figure 5*
*Left: Original image. Right: Image convolved with the sharpening kernel in Figure 4.*

Image sharpening kernels can also be chosen if information about the blur is known. For example, for a directional blur, or a motion blur, with direction left to right, giving the pixels on the left side negative weight while the right side is given positive weight may lead to a better deblur than a kernel designed to undo Gaussian blur.

## VI. Edge Detection

Edge detection is aimed at detecting *edges*, which are defined as curves in an image at which there is a discontinuity in image brightness.[9] These discontinuities often correspond to discontinuities in depth, surface orientation, material properties, or illumination.

The most used method for this is applying the Sobel operator to the image. This operator uses 2 3x3 kernels $G_x$ and $G_y$which are convolved with the image. The result is the approximation of the horizontal and vertical derivative at each pixel. The Euclidean distance between the results of both kernels can be used to compute the final image, similar to the Gaussian kernel.

For a given image *A*, the Sobel method is to compute the following images:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

To generate the final image G, we take the Euclidean distance between the values of $G_x$ and $G_y$:

$$\mathbf{G}_{x,y} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

For noisy images, sometimes the Gaussian blur is applied before the Sobel operator[10] to reduce the number of edges detected, ensuring only major edges are detected. This is called the Laplacian of Gaussian.

## VII. Conclusion

We have looked at the basics of image processing in this article. There are many other kinds of each of these operations: Stack blur, Circular blur, Motion blur, etc. for blurs, Canny, Kovalevsky, Differential, etc. approaches for edge detection, et cetera. These are only the most basic of techniques that are still in use today. All of these different techniques have different applications.

With the increase in AI applications, there may be models in the future trained specifically to deblur or detect edges, that do not explicitly use any of these known approaches. However, these approaches are still computationally less expensive and faster to calculate than any AI approaches for now.

## VIII. References


***[1].*** ***Murray, James D; VanRyper, William; Encyclopedia of Graphics File Formats, ISBN 1565921615***

***[2].*** ***"DIBs and Their Uses", Microsoft Help and Support***

***[3].*** ***Murray, James D; VanRyper, William; Encyclopedia of Graphics File Formats, ISBN 1565921615***

***[4].*** ***BITMAPINFOHEADER structure, Microsoft Developer Network, 21 June 2023***

***[5].*** ***W3C SVG1.1 specification, 15.17 Filter primitive 'feGaussianBlur'***

***[6].*** ***Pascal Getreuer, A Survey of Gaussian Convolution Algorithms, Image Processing On Line, 3 (2013), pp. 286–310.***

***[7].*** ***R.A. Haddad and A.N. Akansu, A Class of Fast Gaussian Binomial Filters for Speech and Image Processing, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 39, pp 723-727, March 1991.***

***[8].*** ***Smooth.cpp, OpenCV/modules/imgproc/src/smooth.cpp, line 3782, The OpenCV repository on GitHub.***

***[9].*** ***H.G. Barrow and J.M. Tenenbaum (1981),Interpreting line drawings as three-dimensional surfaces, Artificial Intelligence, vol 17, issues 1–3, pages 75–116.***

***[10].*** ***Fisher, Perkins, Walker & Wolfart (2003). Spatial Filters - Laplacian of Gaussian.***