Image Fundamentals

December 10, 2025

Contents

1 Introduction

To work with any vision-based system, the first step is to understand how images are represented inside a computer and how we can manipulate them. Instead of thinking of images as photographs, we treat them as grids of numerical values—matrices or tensors that describe brightness or channel intensities at each pixel. Once we view images in this numerical form, we can apply simple operations like grayscaling, resizing, normalization, thresholding, smoothing, and basic filtering to clean and structure the data. These foundational tools allow us to transform raw images into forms that are easier to analyze, segment, and interpret, forming the basis for building an OCR pipeline and more advanced machine-learning models later

2 Colour and Colour Spaces

2.1 Colour

Colour itself is not just a property of light, but the result of how light interacts with objects and how our visual system interprets it. Digital images capture this interaction in a simplified form—by storing colour or brightness information as arrays of numbers.

2.2 Colour Spaces

Colour space or colour models is an abstract mathematical model which describes the range of colours as tuples of numbers, typically 3 or 4 numbers or colour components.
Most color models in use today are oriented either toward hardware such as for color monitors and printers (RGB,CMY,CMYK) or toward applications where color manipulation is a goal such as in the creation of color graphics for animation (HSI)

2.2.1 RGB Space

In the RGB color model, every color is represented as a combination of three primary components: Red (R), Green (G), and Blue (B). We treat these three values as coordinates in a 3-dimensional Cartesian space, where each axis corresponds to one primary color. When we normalize the values of R, G, and B to lie within the range [0, 1], the entire set of possible RGB colors forms a unit cube.

- Black lies at the origin (0, 0, 0).
- White lies at the opposite corner (1, 1, 1).
- Primary colors appear at the axis corners: (1, 0, 0), (0, 1, 0), (0, 0, 1).
- Secondary colors (cyan, magenta, yellow) appear at the mid-edge positions formed by mixing two primaries.
- Any valid RGB color is simply a point inside or on the surface of this cube.
- The grayscale line is the diagonal from black to white, where R = G = B.
- Moving along this diagonal increases brightness without changing hue.

This geometric view is useful because it lets us interpret colors as vectors extending from the origin to any point in the cube.

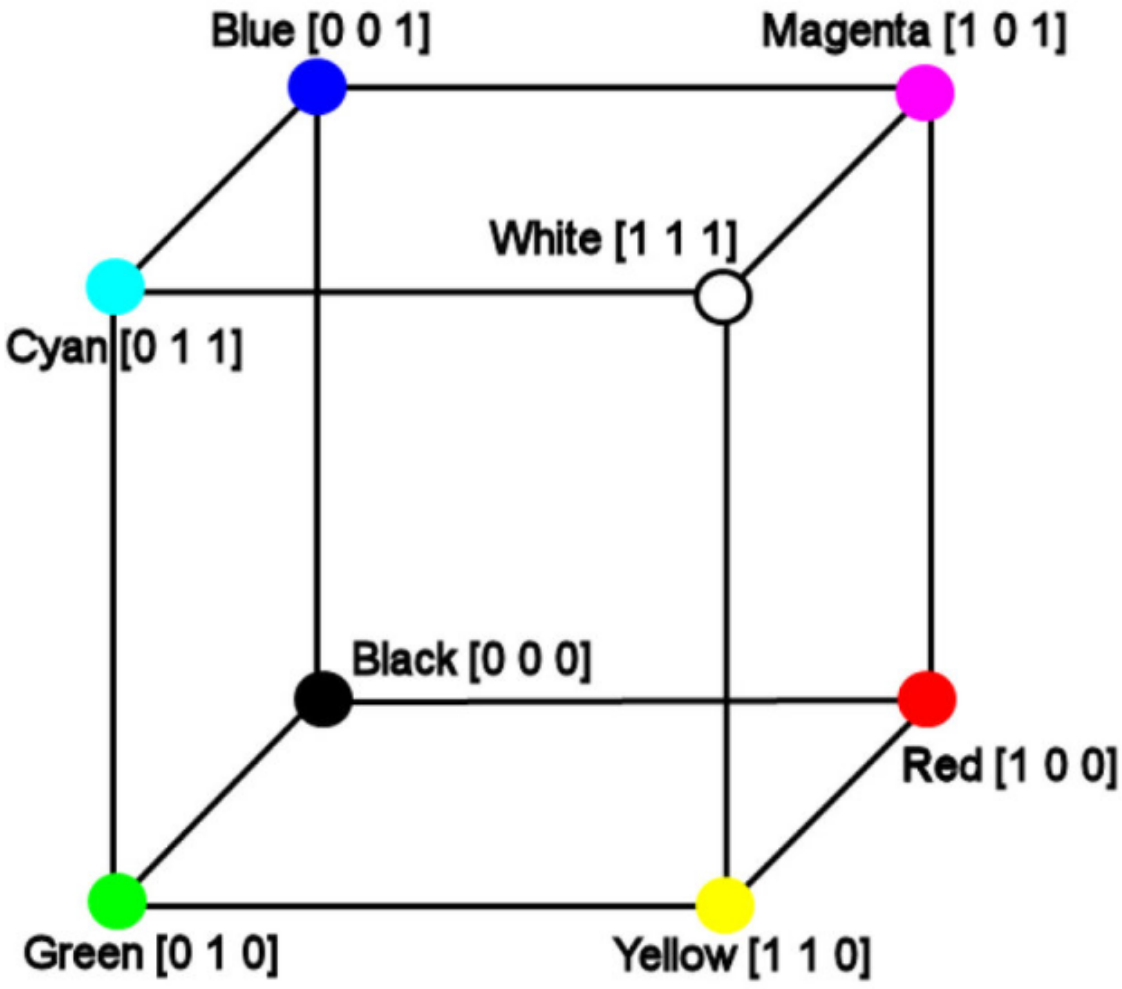

Figure 1: RGB Colourspace

2.2.2 HSI colour space

The HSI (Hue, Saturation, Intensity) model represents color in a form that aligns closely with human perception. Instead of describing color through primary components, HSI separates visual information into three intuitive attributes:

- **Hue (H)**: Represents the dominant wavelength or pure color (e.g., red, yellow, green, cyan). Hue is measured as an angle around a color wheel or hexagon.
- **Saturation (S)**: Measures the amount of white light mixed with a color. Highly saturated colors contain little or no white; low-saturation colors move toward pastel or grayish tones. Geometrically, saturation is the radial distance from the intensity axis.
- **Intensity (I)**: Represents the achromatic intensity or brightness of the color—how light or dark it appears, independent of hue. It is measured along a vertical axis and is similar to grayscale level

Together, hue + saturation form the chromatic (color-carrying) information, while intensity carries the achromatic (brightness-only) information. This separation makes HSI extremely useful in image processing tasks where color interpretation or illumination adjustment must be handled independently.

3 Digital images

Most images we use in computer vision are digital images. This means they are stored not as continuous pictures, but as grids of numbers. Every number in this grid represents the brightness (or color intensity) of one small point in the image, called a pixel.
A real-world scene is continuous, but a computer cannot store infinite detail.So we convert the scene into a digital form using two steps:

- Sampling – choosing a regular grid of points in the scene. Example: Taking a picture at 1000×1000 resolution means sampling the scene at 1,000 rows and 1,000 columns.
- Quantization – assigning a numerical intensity value to each sampled point. Example: In an 8-bit grayscale image, each pixel is an integer from 0 to 255, where
  0 = black
  255 = white
  middle values = shades of gray

After these steps, the image becomes a 2-D function f(x, y), where x and y are now integer coordinates on the grid (X=0..M-1, Y=0..N-1),
In computer vision, we treat every digital image as a matrix (or tensor) of numbers. These numbers describe the brightness or color of each pixel. For OCR, this matrix view is critical, because all preprocessing, segmentation, and feature extraction operate directly on these numeric values.

3.1 Binary Image

A binary image contains only two possible values:
0 → background (black)
1 or 255 → foreground (white, usually the text)

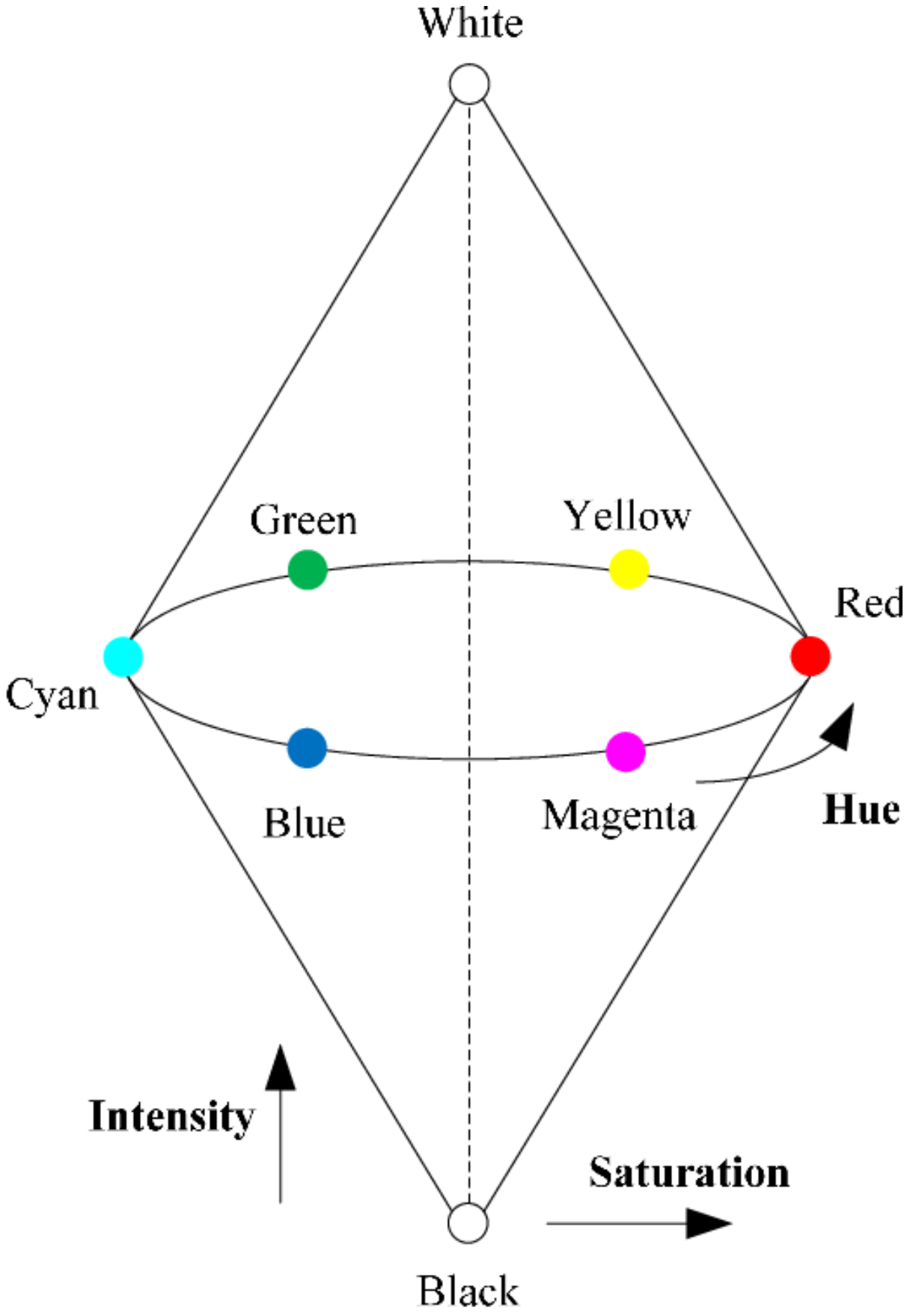

Figure 2: HSI Colourspace

Representation:
A 2D matrix of size M × N
Each entry is either 0 or 1 (or 255 depending on convention)

$$\text{Binary Image example} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

3.2 Grayscale Image

A grayscale image stores only brightness (no color).
Representation:

- A 2D matrix of size M × N
- Each entry is an intensity value, usually:
  - 0 = black
  - 255 = white
- values in between = shades of gray.

$$\text{Greyscale example} = \begin{bmatrix} 11 & 50 & 70 \\ 43 & 180 & 140 \\ 76 & 20 & 255 \end{bmatrix}$$

3.3 RGB Image

An RGB image stores color using three separate grayscale matrices (channels):

- R(x, y) → red intensity
- G(x, y) → green intensity
- B(x, y) → blue intensity

Representation: A 3D matrix (tensor) of size M × N × 3

3.4 HSI Image

HSI separates color into perceptual components:

- H(x, y) → hue (which color)
- S(x, y) → saturation (how pure / how much white is mixed)
- I(x, y) → intensity (brightness, achromatic component)

Representation: A 3D matrix of size M × N × 3, similar to RGB.But channels have different meanings.

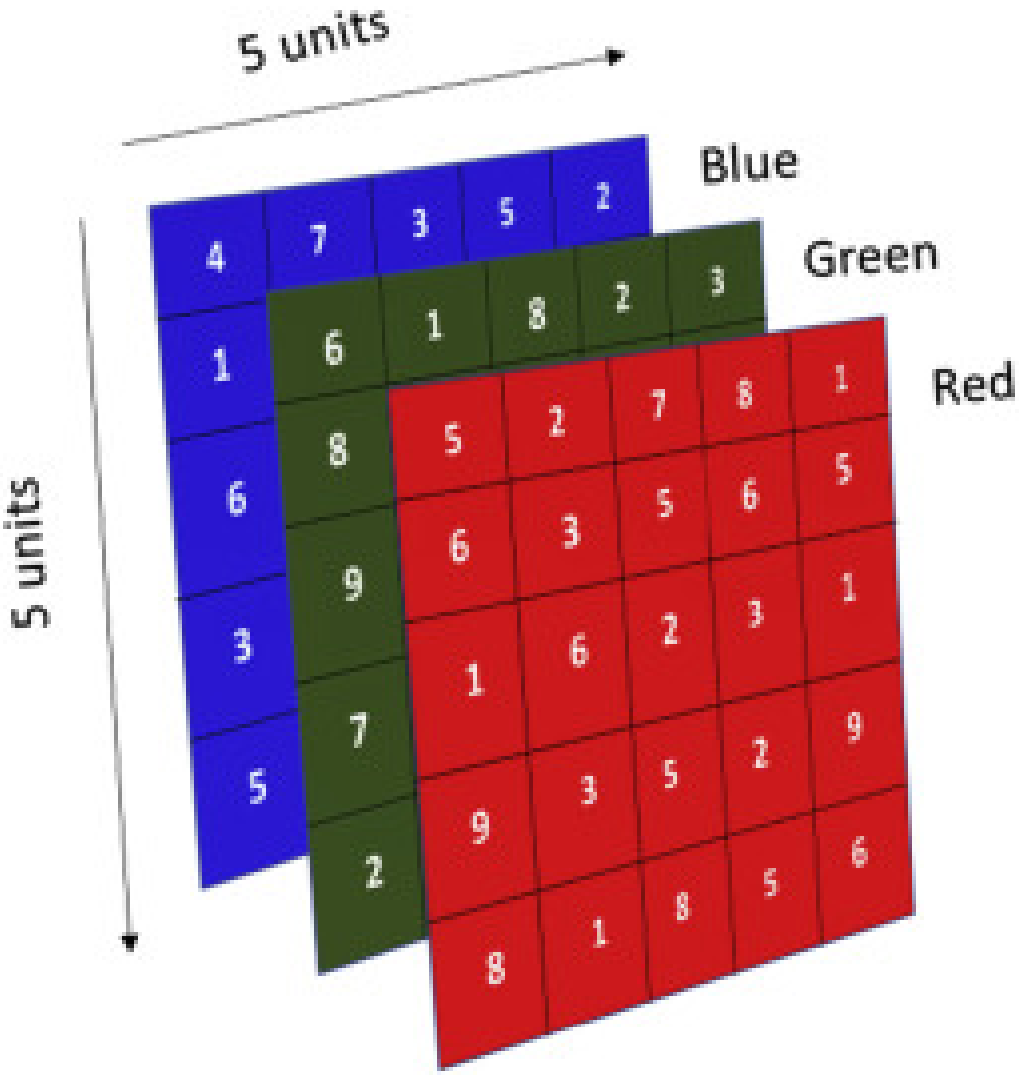

Figure 3: RGB image

4 Pixel Level Operations

Pixel-level operations are image processing techniques that work by directly modifying the numerical values of individual pixels in an image matrix. These operations act on the image before any higher-level analysis and typically change brightness, color, or local structure by manipulating pixel intensities. They are the essential first steps in preparing images for tasks like thresholding, segmentation, and OCR.

4.1 Greyscaling

Grayscaling is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white.
Why needed in OCR?

- OCR works on shape(shape of characters are used in detection/classification), not color
- Simplifies the data (reduces 3 channels $\rightarrow$ 1 channel)

4.1.1 Methods to change an image from rgb to greyscale

Using built in cv2 function

- Using the cv2.cvtColor() function
- Using the cv2.imread() function with flag=zero-In this method, we can directly load an image in grayscale mode by passing the flag 0 to cv2.imread(). This saves us from having to convert the image separately after loading.

Weighted Method
This method uses standard luminance weights ($0.2989R + 0.5870G + 0.1140B$) to account for human visual sensitivity—more to green, less to red, least to blue. It produces a more realistic and visually accurate grayscale image

4.2 Normalization

Normalization is the process of adjusting pixel values so that they fall within a common and consistent range, usually [0, 1] instead of [0, 255]. It is done by dividing every pixel intensity by the maximum possible value (often 255 for 8-bit images):
$I_{norm} = I/255$ (considering 8 bit image, maximum intensity is 255)
This does not change the structure of the image—only the scale of the numbers.
Why Normalization Is Useful

- Ensures all images have comparable intensity ranges
- Makes mathematical operations (distance, filtering, gradients) more stable
- Helps machine learning models converge faster
- Prevents large pixel values from dominating computations

4.3 Thresholding

Image thresholding works on grayscale images, where each pixel has an intensity value between 0 (black) and 255 (white). The thresholding process involves converting this grayscale image into a binary image, where pixels are classified as either foreground (object of interest) or background based on their intensity values and a predetermined threshold. Pixels with intensities above the threshold are assigned to the foreground, while those below are assigned to the background.

4.3.1 Simple Thresholding

Simple thresholding uses a single threshold value to classify pixel intensities. If a pixel's intensity is greater than the threshold, it is set to 255 (white); otherwise, it is set to 0 (black)

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > T \\ 0, & \text{if } f(x,y) \leq T \end{cases}$$

4.4 Adaptive Thresholding

Adaptive thresholding addresses one of the main limitations of simple (global) thresholding its inability to handle images with varying lighting conditions. Instead of using a single global threshold value for the whole image, adaptive thresholding calculates the threshold for small regions around each pixel. This approach provides better results for images where illumination changes across different parts.

$$T(x,y) = \mu_{N(x,y)} - C$$

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > T(x,y) \\ 0, & \text{otherwise} \end{cases}$$

- $f(x,y)$: input pixel intensity.
- $N(x,y)$: local neighbourhood window.
- $\mu_{N(x,y)}$: mean intensity of the neighbourhood.
- C: small constant to adjust the threshold.C adjusts how aggressive the thresholding is (higher C $\rightarrow$ darker output)
- $T(x,y)$: local threshold for that pixel.
- $g(x,y)$: output binary pixel (1 = white, 0 = black).

Note Thresholding is taught here only to illustrate how traditional OCR separated text from background. Although many thresholding variants exist, a simple global threshold is enough to understand the basic classical pipeline. Modern neural-network OCR does not use thresholding, since CNNs learn directly from grayscale or color images. Grayscale conversion and normalization, however, are still used because they simplify inputs and stabilize training. Also note that many more pixel-level operations exist, but we focus only on those relevant for building a basic OCR pipeline.

5 Filtering

Filtering is the process of modifying an image by replacing each pixel with a weighted combination of its neighbors. This is done to achieve useful effects such as smoothing noise, sharpening edges, or highlighting character strokes for OCR.
Filtering is performed mathematically using correlation or convolution, where a small matrix called a kernel slides over the image, and at each position we compute a weighted sum of nearby pixels.

- In correlation, the kernel is used as-is.
- In convolution, the kernel is flipped before sliding.

For most symmetric kernels, the two operations give identical results, so filtering in practice is just correlation with different kernels. Filtering is important for OCR because it makes characters clearer, reduces noise, and enhances edges, creating cleaner inputs for thresholding, segmentation, or basic feature extraction.

5.1 convolution

During convolution, using the kernel, we slide through the image matrix. However, there is an important step before this sliding. The kernel must be flipped horizontally and vertically before convolving with the image.

$$K_{\text{flipped}} = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$

$$(I * K)(1,1) = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \odot \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} = 1(4) + 2(3) + 4(2) + 5(1) = 4 + 6 + 8 + 5 = 23$$

5.2 Correlation

Correlation is very much similar to convolution. Yet, in correlation, no flipping of kernel is required. Using the original kernel, without flipping it, we use it directly and slide it through the image matrix. The operation is then same as convolution.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \qquad K = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$(I \star K)(1,1) = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 1(1) + 2(2) + 4(3) + 5(4) = 1 + 4 + 12 + 20 = 37$$

Note on OpenCV: In practice, OpenCV's filtering functions (e.g., `cv2.filter2D`) perform correlation, not true mathematical convolution, even though people casually refer to it as convolution. Because many commonly used kernels (like Gaussian blur, Laplacian, and Sobel magnitude) are symmetric, convolution and correlation produce identical results, which is why the difference rarely matters in applied image processing.

5.3 Some common filters and its function

Filter	Kernel Example	Purpose
Mean Blur	$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$	Smooths the image, reduces random noise before thresholding/OCR.
Gaussian Blur	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$	Removes noise while preserving edges better than mean blur.
Sharpening	$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$	Enhances edges and fine details in characters.
Sobel-X (Edge)	$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$	Detects vertical edges; useful for stroke structure in OCR.
Sobel-Y (Edge)	$\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$	Detects horizontal edges in characters.
Laplacian	$\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix}$	Second-order edge detector; highlights rapid intensity changes.
Emboss	$\begin{bmatrix}-2 & -1 & 0\\ -1 & 1 & 1\\ 0 & 1 & 2\end{bmatrix}$	Creates a 3D relief effect; shows gradient directions clearly.

Notes:

- In this context, *smoothing* and *blurring* mean the same thing: we replace each pixel with an average of its neighbours to reduce noise.
- Mean blur is the simplest smoothing filter; it treats all neighbours equally, but can smear edges.
- Gaussian blur uses larger weights near the centre and smaller weights farther away, so it smooths noise while preserving edges slightly better. It is commonly used before thresholding in classical OCR.
- The size of the blurring filter (e.g., 3×3, 5×5) determines the degree of smoothing: a larger kernel causes more blurring and greater loss of detail.
- Sharpening filters emphasise edges and fine details by boosting differences between a pixel and its neighbours. This can make strokes appear crisper but may also amplify noise.
- Sobel-X and Sobel-Y are first-derivative edge detectors. Sobel-X responds strongly to vertical edges, and Sobel-Y to horizontal edges. Combined, they highlight character stroke structure and are a classic starting point for edge-based feature extraction.
- The Laplacian is a second-derivative edge detector that responds to rapid intensity changes in all directions. It is usually applied after some smoothing because it is very sensitive to noise.
- Emboss filters are mainly for visualization: they create a shaded, "3D" look that encodes gradient direction. They are rarely used directly in OCR pipelines but help illustrate the effect of directional filters.

You can watch the first 3 videos of this playlist to get an idea how laplacian and other gradient operators can be used to detect edges