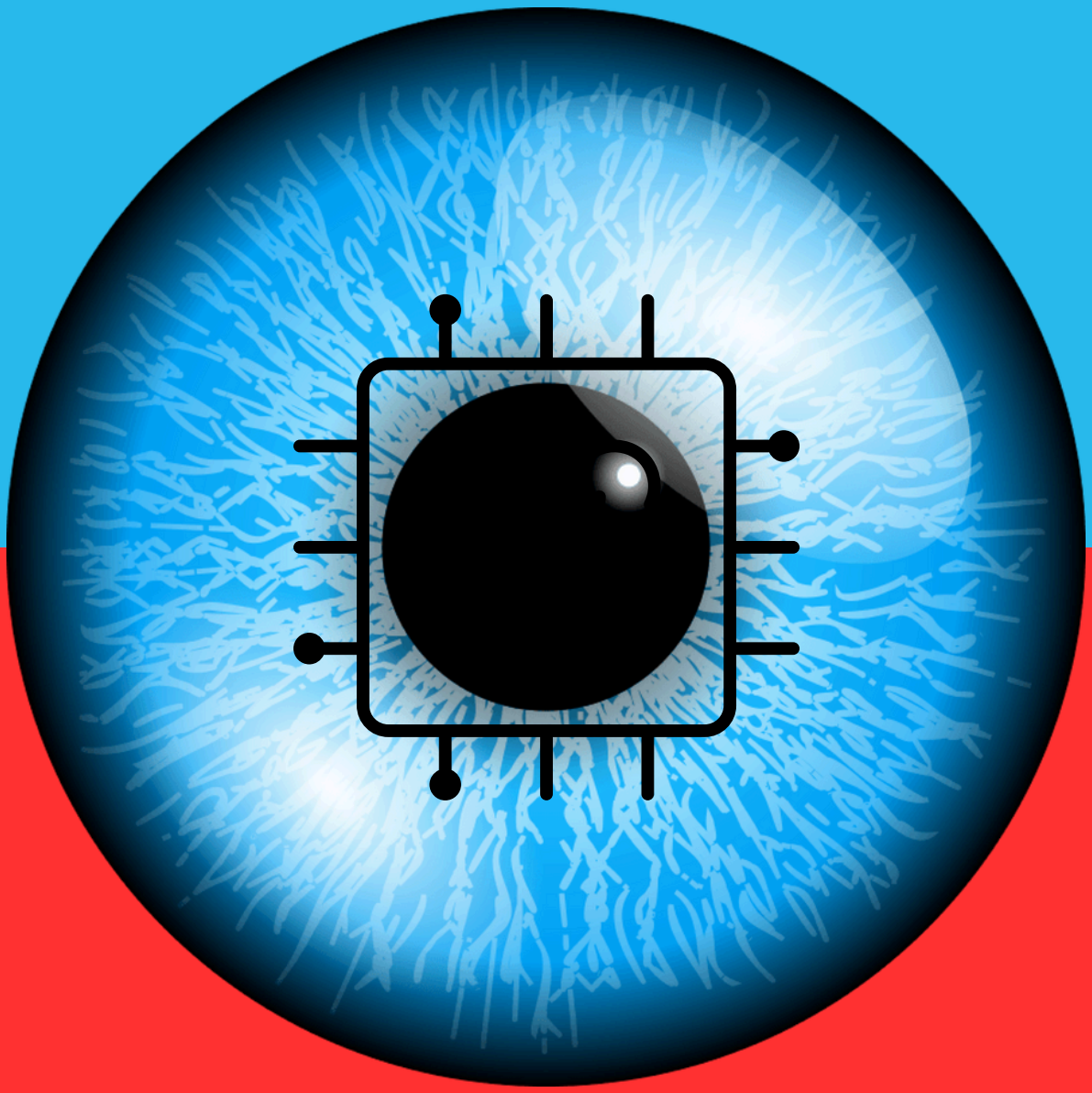
Computer Vision

for Highschoolers



Nishant Pandey

Author's Message

Hi Readers,

I am Nishant. You made it here! The purpose of this book is to provide an intuitive understanding of what computer vision is and its far-reaching capabilities. The resources are based on the documentation of OpenCV and my own understanding. This book is a free, public resource—a project born out of my eagerness to help students get started with computer vision. Getting start with computer vision is overwhelming due to which I have structured the content to guide you through each level, ensuring you have a solid grasp of the material.

If you have any suggestions, please feel free to contact me at **ml.nishantpy@gmail.com**

"An early start allows the unimaginable to become imaginable."

Before getting started

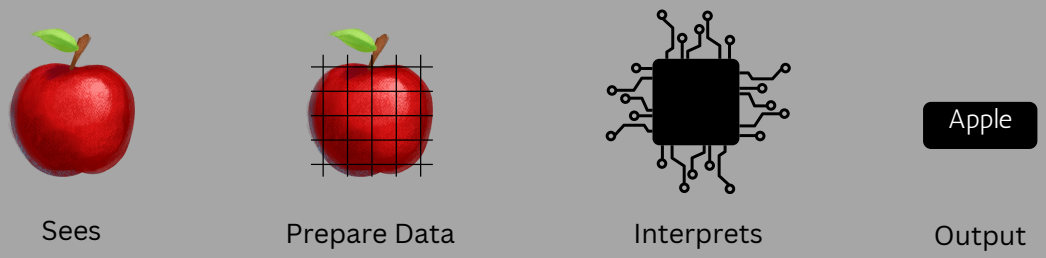
Hi Readers, I am Nishant. You made it here! The purpose of this book is to provide an intuitive understanding of what computer vision is and its far-reaching capabilities. The resources are based on the documentation of OpenCV and my own understanding. This book is a free, public resource—a project born out of my eagerness to help students get started with computer vision. Getting start with computer vision is overwhelming due to which I have structured the content to guide you through each level, ensuring you have a solid grasp of the material. If you have any suggestions, please feel free to contact me at ml.nishantpy@gmail.com. "An early start allows the unimaginable to become imaginable."

Module 1

Intuition : Computer Vision

Computer vision and the human eye both strive to decode the visual world, yet their methods are strikingly different. The eye captures light, and the brain unravels its mysteries, much like a webcam records and a processor deciphers its image. Introducing Computer Vision—a technological marvel that transcends mere sight, harnessing the full might of computational power to transform raw visual data into profound insights.

Computer vision is a multidisciplinary field that enables machines to interpret and understand visual information from the world, mimicking the human ability to see.



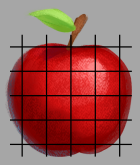
The diagram above represents the fundamental structure of computer vision. Each stage of the process involves various components that contribute to the overall system.

Computer vision, a branch of AI, aims to replicate or even surpass human visual capabilities. Consider the "Find the Difference" puzzles: while it can take a human a significant amount of time to spot discrepancies between two images, computer vision can identify these differences in mere milliseconds. Although writing the code for such technology might be time-consuming, the efficiency and speed it brings to the future are undeniable. With computer vision, tasks that once required hours of painstaking effort can now be accomplished almost instantly.

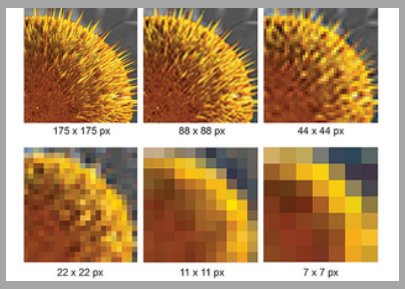
What makes this technological marvel possible? A multitude of critical components work together to enable computer vision. In the following chapter, we will explore an overview of these foundational elements, shedding light on how each contributes to the sophisticated capabilities of computer vision.

Terms and Terminologies I

- **Pixel:** The smallest unit of an image, representing a single point in a grid of colors and intensities. Each square is a pixel which carry numerical value (as we know computer understand numbers)
- **Resolution**: The amount of detail an image holds, defined by the number of pixels along its width and height. The higher the number of pixel, the higher the image quality

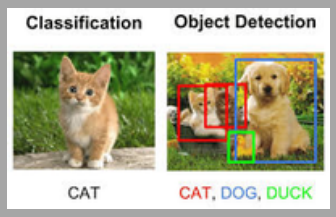


a. Representation of Pixel



b. Resolution of an image

- **Image Classification**: The task of assigning a label to an entire image based on its content.
- **Object Detection**: Identifying and locating objects within an image, typically using bounding boxes.

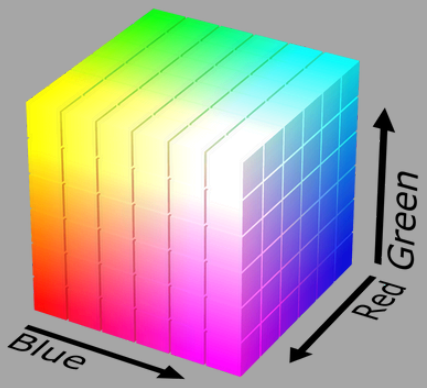


a. Differences in Classification and Detection

- **Image Segmentation:** Image segmentation is a fundamental process in computer vision that involves dividing an image into multiple segments or regions, each representing different parts of the image.

a. Illustration of Image Segmentation

Pixel and Color Representation



At the heart of image processing are pixels, the tiny building blocks of an image. Think of them as the dots on a grid that come together to form a picture. Each pixel holds information about color and brightness. In color images, this is typically represented using the RGB model, where each pixel has three values: red, green, and blue. By combining these colors in various intensities, you get the full spectrum of colors seen in the image.. This is called **RGB model**

Each pixel in an image is typically represented by three values corresponding to the **intensity** of red, green, and blue light, respectively.

- Red (R): Varies from 0 to 255
- Green (G): Varies from 0 to 255
- Blue (B): Varies from 0 to 255

When the values of these colors are combined, they can produce any color in the visible spectrum. For example:

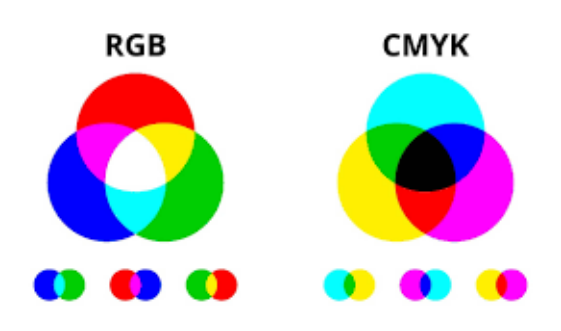
- Red: (255, 0, 0) - Full intensity of red || No green or blue
- Green: (0, 255, 0) - Full intensity of green || No red or blue
- Blue: (0, 0, 255) - Full intensity of blue || No red or green
- Black: (0, 0, 0) - No light
- White: (255, 255, 255) - Full intensity of all three colors

So, What is the mystery behind the range 0 -255 ?

It comes from the use of 8-bit binary numbers, where each bit can be either 0 or 1. An 8-bit number can represent 2 power of 8, i.e 256, different values, ranging from 0 to 255.

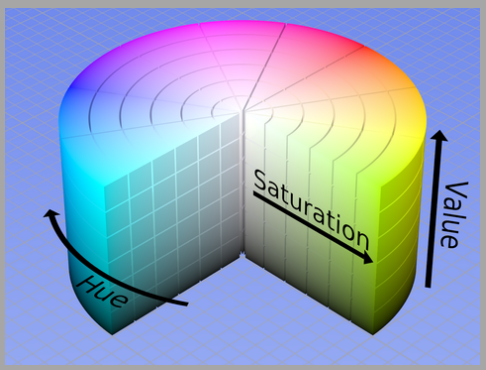
Other Color Models

- **CMYK Models**: CMYK, Cyan Magenta Yellow and Key(black) color model, is based on subtractive color mixing, which means that colors are created by subtracting light. When you mix these inks, they absorb (subtract) specific wavelengths of light and reflect others.



BRAINSTROM => RGB VS CMYK is equality to ability to intensify color vs ability to absorb. Think and see the diagram!

- **HSV**: The HSV model describes colors more intuitively by separating color information (hue) from intensity (value) and purity (saturation). The HSV (Hue, Saturation, Value) color model is necessary because it provides a more intuitive way for humans to understand and manipulate colors compared to the RGB (Red, Green, Blue) model.



Hue: Represents the type of color (e.g., red, blue, green) and corresponds to the angle on a color wheel (0°-360°). This matches how humans typically think about color.
Saturation: Describes the intensity or purity of the color. Fully saturated colors are vivid, while less saturated colors appear more washed out or grayish.
Value: Indicates the brightness or lightness of the color, from black (0%) to full brightness (100%).

Some code represention in HSV color model :

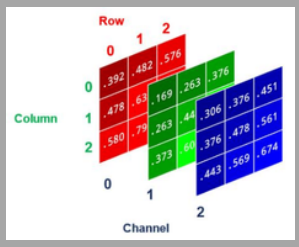
- Red: (0°, 100%, 100%)
- Green: (120°, 100%, 100%)
- Blue: (240°, 100%, 100%)
-

Conversion of RGB to HSV is one of the major learnings that yet to come in the following chapters.

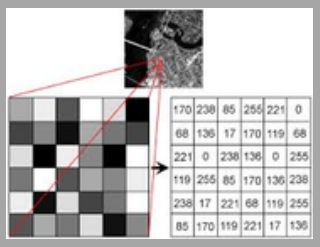
Image Channels

Imagine you have a coloring book, but instead of coloring each picture with one color, you color different parts of the picture with red, green, and blue transparencies. When you overlay these colored transparencies, you see the complete, colored image.

RGB Channels: When you combine the red, green, and blue channels, each pixel's color is created by mixing the corresponding intensities of these three colors. That's how the RGB channel works.



GrayScale Channels: A grayscale channel is a **single-channel** image where each pixel represents a shade of gray. Each pixel of the channel has intensity value ranging from 0 to 255 , where 0 gives the complete black and 255 gives complete white

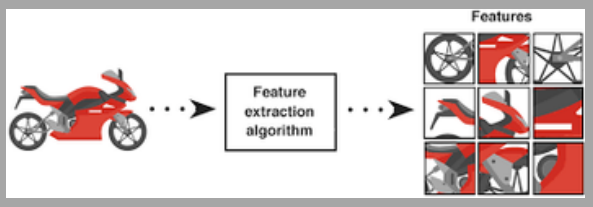


These two are the most commonly used channels. However, there are many other image channels, just like different color models.

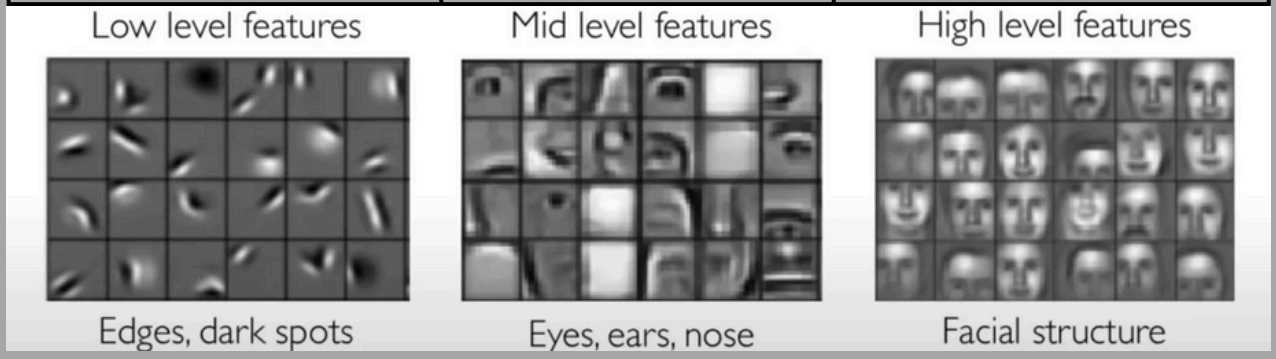
Features in Computer Vision

In computer vision, a feature refers to a piece of information extracted from an image that is important for a particular task, such as object recognition or image classification. Features can include edges, textures, colors, shapes, or more complex patterns. They serve as the building blocks for understanding and analyzing visual data, allowing algorithms to identify and differentiate between various elements within an image.

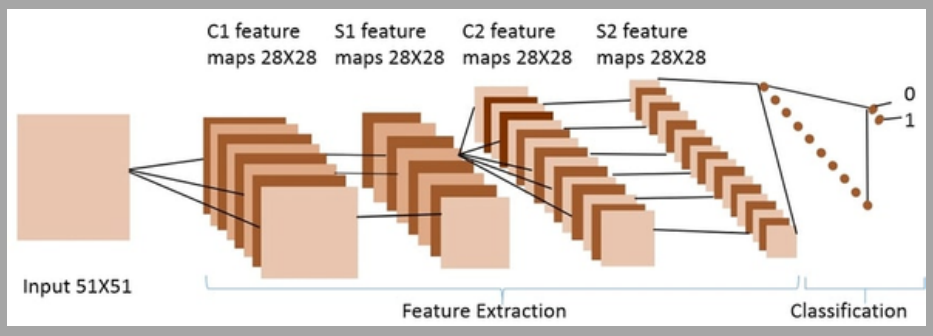
In general there are three level of features : low level, mid level and high level . ,



Low Level Features	Mid Level Features	High Level Features
These are basic visual elements directly extracted from the image, such as edges, corners, and textures	These features represent more complex structures and patterns, often formed by combining low-level features.	These features are abstract and semantically rich, representing the meaningful content of the image.



Pixels serve as the raw data for computer vision models, representing the smallest units of an image that collectively form a visual scene. Features, on the other hand, are the insights that these models extract from the pixels. These features need to be extracted and processed to do tasks such as classification and detection. The features could be extracted using different filter/kernel that lies over the image channels.

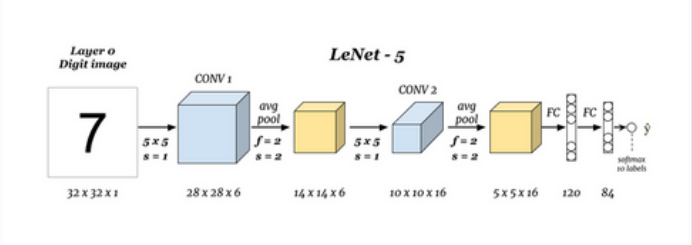


Computer Vision Models

computer vision model is a computational system or algorithm designed to interpret and understand visual data from images or videos. These models use machine learning, especially deep learning techniques, to identify patterns, recognize objects, and extract meaningful information from visual inputs. Use of which model is answered on your desired task to do. The following gives some examples of CV models and for what purposes they are mostly used.

1.Convolutional Neural Networks (CNNs)

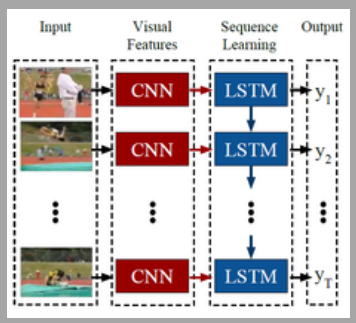
CNNs are a type of deep learning model specifically designed for processing **structured grid data** like images. They use convolutional layers to automatically detect and learn hierarchical features from raw pixel data, making them highly effective for tasks such as image classification, object detection, and segmentation.



2. Recurrent Neural Networks (RNNs)

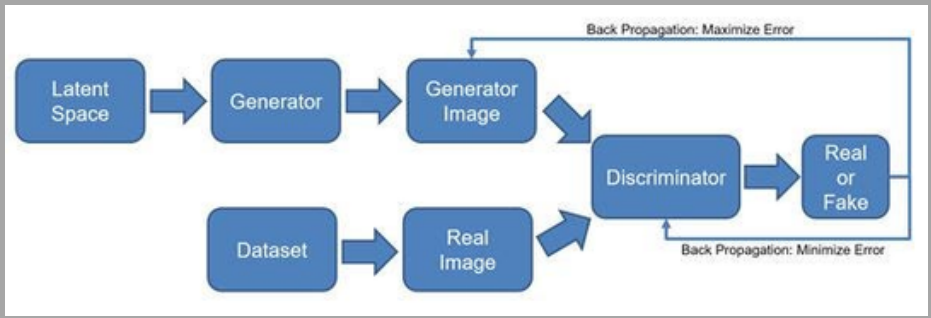
RNNs are designed for **sequential data processing** and are often used in tasks where temporal dynamics are important, such as video analysis and activity recognition. They maintain a memory of previous inputs, allowing them to recognize patterns over time.

Here, LSTM is a RNN based model, which will b.e taking about it later



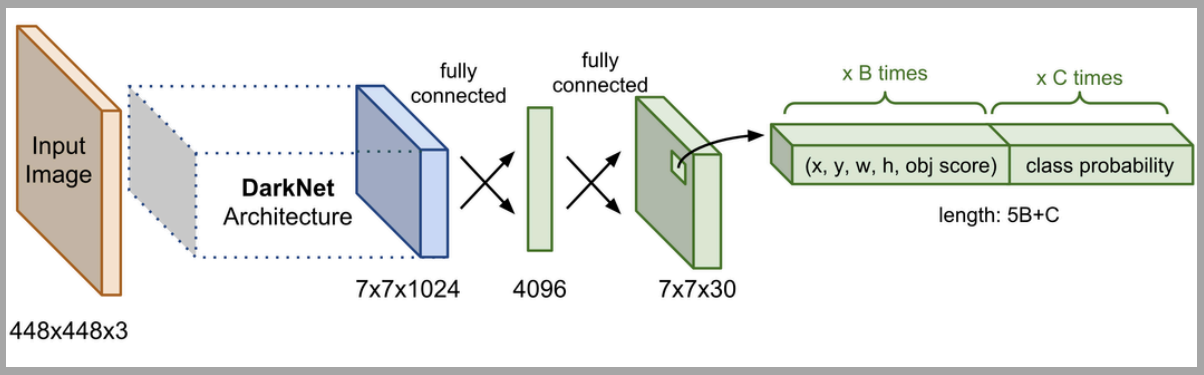
3. Generative Adversarial Networks (GANs)

GANs consist of two networks—a generator and a discriminator—that compete against each other. The generator creates synthetic images, while the discriminator tries to distinguish between real and fake images. GANs are used for **image generation**, image-to-image translation, and creating high-quality synthetic data.



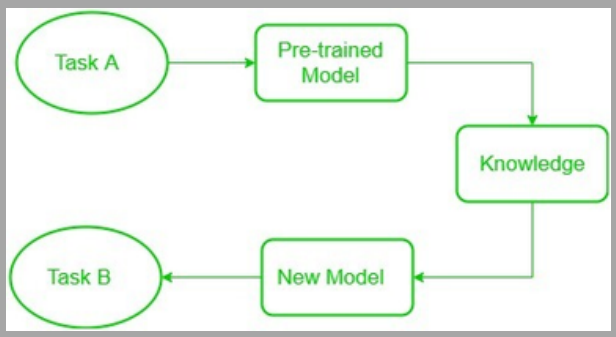
3. You Only Look Once (YOLO)

YOLO is **object detection model** known for its speed and accuracy. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell in a single pass through the network, enabling real-time processing.



10. Transfer Learning Models

Transfer learning involves taking a **pre-trained model** and fine-tuning it for a specific task. This approach leverages the knowledge gained from training on large datasets and applies it to new, similar problems, often leading to improved performance and faster training times.



These are the most popular and trending models and architectures used in computer vision. DONOT get overwhelmed by it. This chapter just gets you familiar with purposes of different models for different works. We , in coming chapter, go in-depth about these models and their working principle

Image Processing Techniques

Image Acquisition

Image acquisition is the first step in any image processing system. This process involves capturing an image using a device such as a camera or scanner. Once the image is captured, it can be converted into a digital form that can be processed by a computer. Think of it as taking a picture with your smartphone; the moment you press the button, the camera captures light and converts it into a digital image that you can see on your screen.

Image Enhancement: Brightness, Contrast, and Histogram Equalization

1. Brightness

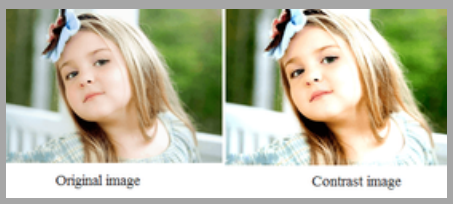
This refers to the overall lightness or darkness of an image. When you increase the brightness, you add a certain value to the pixel's intensity. . If the original pixel has values (R,G,B), the new values (R′,G′,B′)are calculated as:

$R' = R + b, \quad G' = G + b, \quad B' = B + b$

where b is the brightness level

2. Contrast

Adjusting contrast involves changing the range of intensity values in the image to be more spread out or closer together.I f a pixel has an intensity value I, adjusting contrast can be done by multiplying each pixel value by a constant factor C (contrast factor), followed by adding or subtracting a constant to shift the midpoint. The new intensity is $I' = C \cdot (I - 128) + 128$ where 128 is the midpoint for an 8-bit image.



3. Histogram Equalization

This technique redistributes the pixel intensity values so that they cover the full range of the histogram evenly. The intensity of each pixel is mapped to a new value based on the cumulative distribution function (CDF) of the histogram.

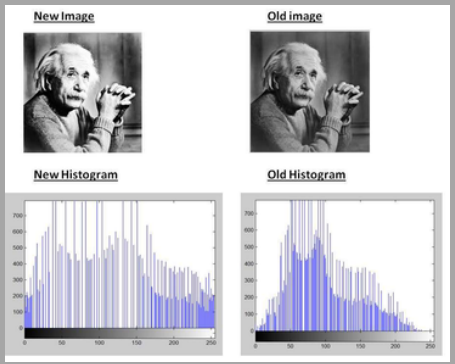
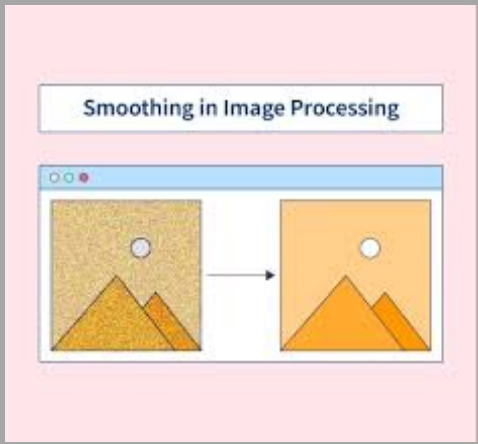


Image Filtering: Smoothing, Sharpening, and Edge Detection

1. Smoothing

Smoothing (blurring) reduces noise and fine details by averaging the intensity values of neighboring pixels. Smoothing involves replacing each pixel's value with an average (or some other function) of the values of its neighboring pixels. This process reduces the intensity variations between adjacent pixels, thereby "blurring" the image.



2. Sharpening:

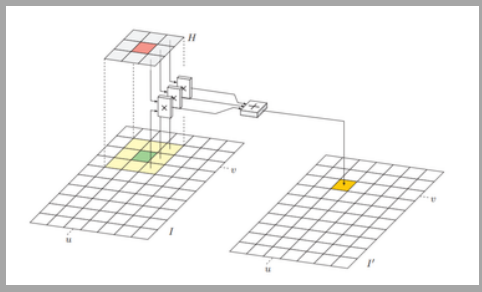
harpening is an image processing technique used to enhance the edges and fine details in an image, making objects appear crisper and more defined. Sharpening typically involves the following steps:

- Blurring the Image: To create a contrast reference.
- Subtracting the Blurred Image from the Original: To detect edges.
- Enhancing the Edges: By adding a scaled version of the edge details back to the original image.

3. Edge Detection:

Edge detection is about finding the boundaries within an image. It identifies points where the image brightness changes sharply, which usually corresponds to the edges of objects. Detecting edges is crucial for recognizing shapes and features in the image

Image Processing Techiques mainly aims to to remove noise, and prepare images for further analysis, making it easier to extract meaningful information from them. it is actually adding a layer depending on which technique is used



Libraries and Tools for Computer Vision

Various libraries and tools have been developed to facilitate tasks such as image processing, object detection, and pattern recognition. OpenCV and torch vision are the two popular libraries for computer vision. It is advised to learn OpenCV as beginner as it gives an step wise intuition on what is happening inside the hood.

OpenCV

Using opencv requires a package to be installed. Make sure you have python and pip install in your code editor. Use `pip install package_name` in your terminal to dowload any packages as per the need.

```python
import cv2
from matplotlib import pyplot as plt #requires to plot(show) the images

# Read the image using OpenCV
image_path = 'path_to_your_image.jpg'
image = cv2.imread(image_path)

# Display the image using Matplotlib
plt.imshow(image_rgb)
plt.title("Displayed Image")

plt.show()
```

Prerequisites :

1. Run pip install opecv-python matplotlib (in your terminal)
2. Add any image make sure to add its image path

Torchvision

Torchvision is a sub component of pytorch which is specifically use to processing data from the images.. Using pytorch is mostly preferred in professional works as I can facilitate in both application of AI i.e Natural Language Processing (NLP) and Computer Vision (CV)

```python
import torch
from torchvision import transforms
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load the image
file_path = 'path_to_your_image.jpg'  # Replace with your image file path
image = mpimg.imread(file_path)

# Convert the image to a tensor
transform = transforms.ToTensor()
image_tensor = transform(image)

# Display the image
plt.imshow(image_tensor.permute(1, 2, 0))
plt.show()
```

Prerequisites :

1. Run pip install torch torchvision matplotlib (in your terminal)
2. Add any image make sure to add its image path

Coding Session I

So far, we have come a long way, from what introducing what an pixel is to introduction of library for computer vision. Now we will have some hands-on on what we have covered so far and let's play with images.

***import codes these basic libraries may not be shown everytime: numpy, pandas, matplotlib, seaborn, cv2*

Let's begin with uploading image and doing some basic operation.

The images used in his session is found here Also follow the comments in code too , this will definitely help you. I am using Google Colab, I assume you have skills on using google colab and mounting to drive.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img_path = 'data/bird.jpg'
image = cv2.imread(img_path)

# # Display the image
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

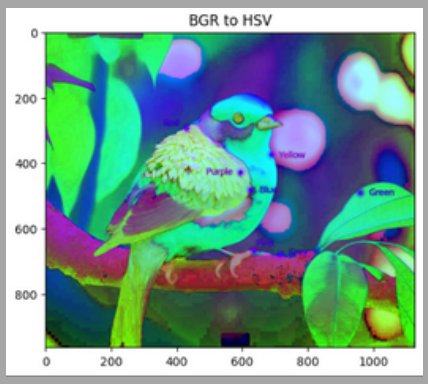
Be sure, you image path is correct.
its output comes in new window, which is annoying later when we have to run code multiple times or we use matplotlib to show images instead of cv2.imshow() and cv2.destroyAllWindows()

> cvtColor means convertColor
Grayscale Image , remember ?
cmap means colormap.

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Display the grayscale image
cv2.imshow('Grayscale Image', gray_image)
plt.imshow(gray_image, cmap='gray')
plt.title("Original")
```

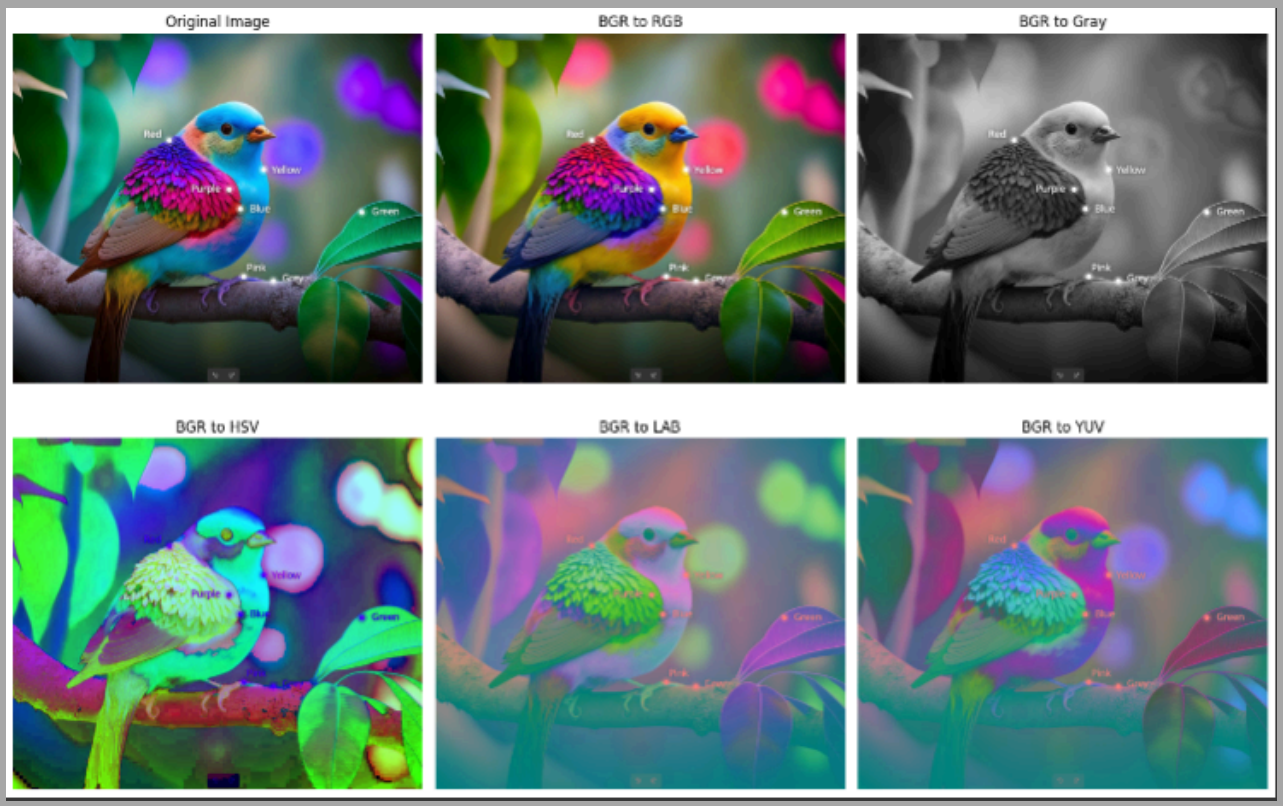
The image is by default in BGR. There is difference in BGR and RBG. It is because the positioning order of three channels are different and order does matter.

```
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
plt.subplot(111)
plt.imshow(hsv_image)
plt.title("BGR to HSV")
```



This is all about converting images in different different color models. Explore and Experiment , and that's how you learn.

There are diferent cobination to try : RGB,, BGR, GRAY, HSV, LAB and YUV



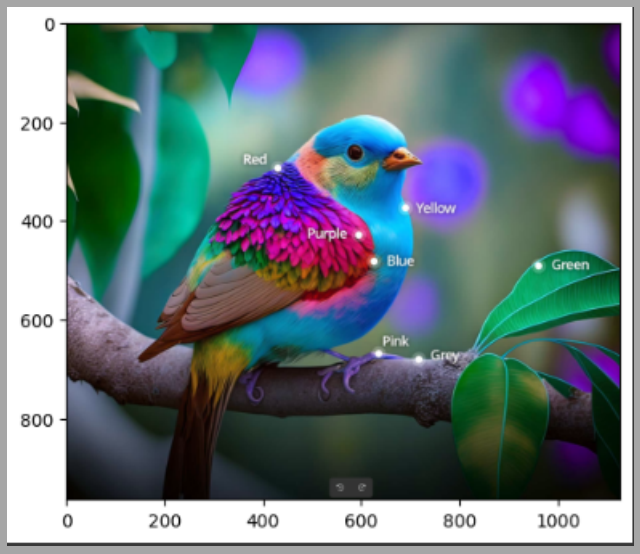
Your desired output is supposed to match to any one of these..

Geometric Operations in Image

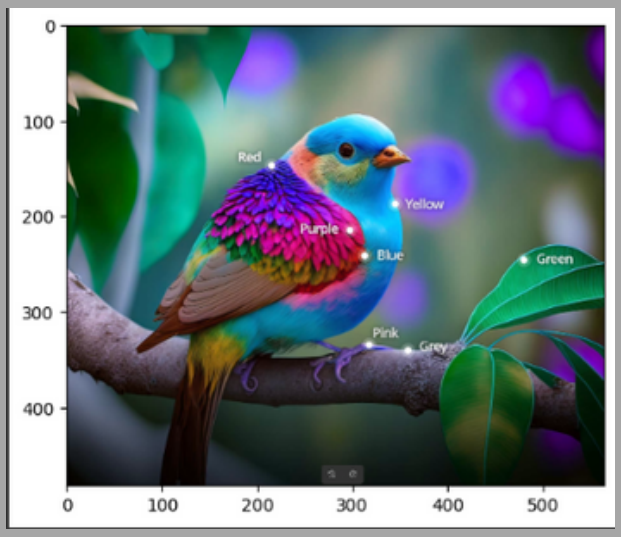
1.Resize and Rotation

res refers to resize.
fx = 0,5 means to resize by 0.5 times in x axis while fx = 0.5 means resize by 0.5 in y axis

```
# scaling
img_path = 'data/bird.jpg'
image = cv2.imread(img_path)

# interpolation: cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_AREA
res = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation = cv2.INTER_NEAREST)
```



Original

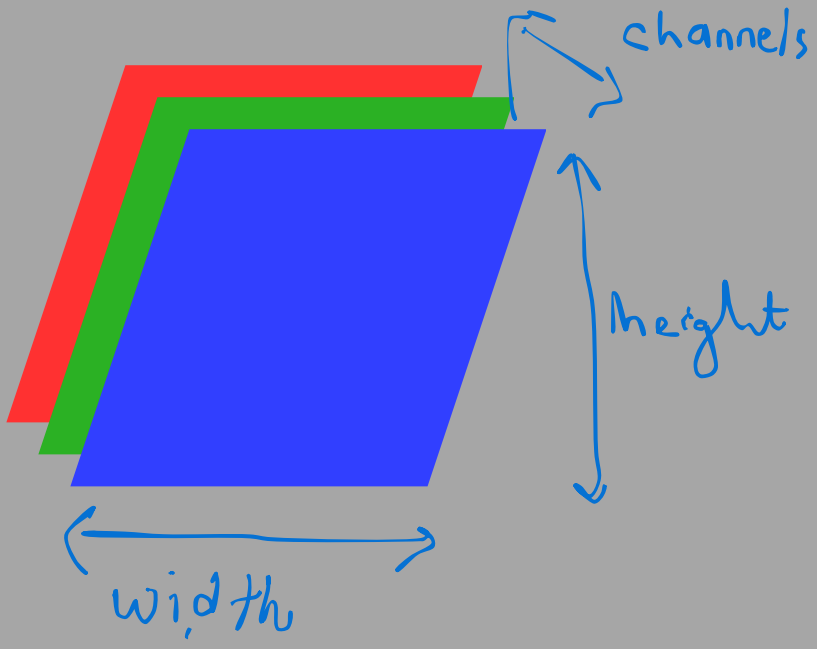


Resized

Didinot find the difference ? Have a look at the coordinates.

```
image.shape
# (height, width, channels)
```

```
(964, 1127, 3)
```



Now after resizing, we are about the rotate resized image (we can also rotate the original image too)

```
# rotation
# Get the image dimensions (height, width)
(h, w) = image.shape[:2] # (:2 means get first two)

# Calculate the center of the image
center = (w // 2, h // 2)

# Define the rotation matrix
angle = 45  # Rotate 45 degrees
scale = 1.0  # No scaling
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

# Apply the rotation
rotated_image = cv2.warpAffine(image, rotation_matrix, (h,w))

# Convert to RGB for displaying with Matplotlib
rotated_image_rgb = cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB)

# Display the result
plt.imshow(rotated_image_rgb)
plt.axis('off')
plt.show()
```

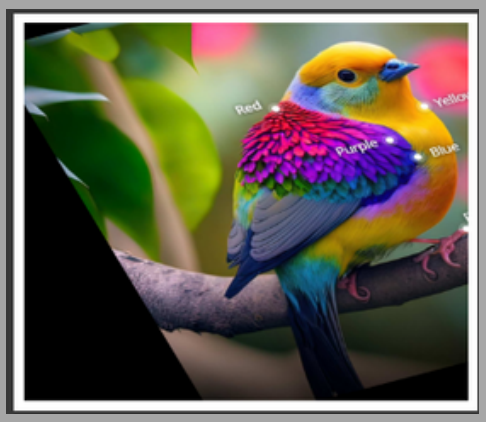
Note :

1. For rotating we donot need channels. So we fetch on heiht and weight in the first line
2. In second line of code, we are assuming that we are about to rotate from the center of the image. Divide by 2 ? This is exactly what rotating from center mean. **Pause and Ponder.**
3. Rotate by what degree || Is scale required?
4. Rotation, by a computer, is done by multiply the matrix called "Rotation Matrix" to that image,
5. Conversion of Color. Not required to do. Still I do it, RGB looks good :)

2. Affline Transformation Matrix

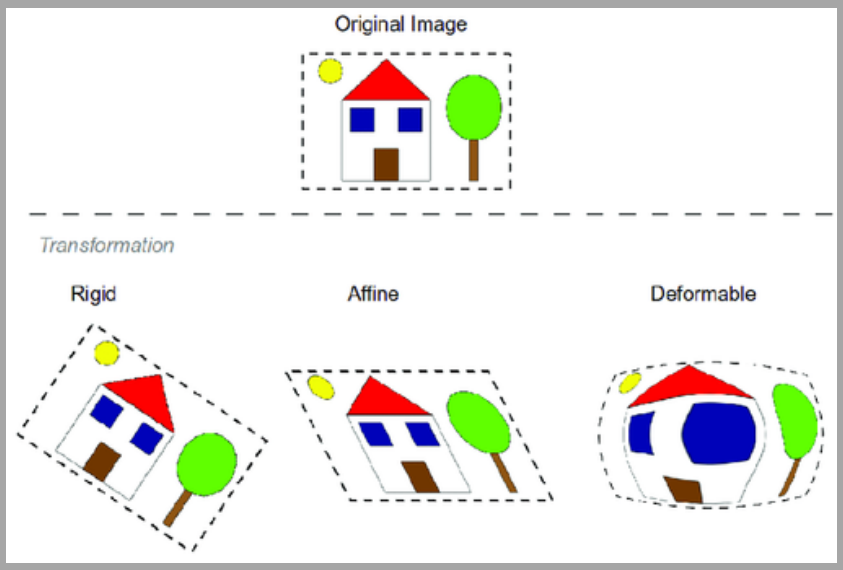
An affine transformation is a linear mapping method that preserves points, straight lines, and planes. It includes operations like translation, scaling, rotation, and shearing. In essence, it transforms shapes while **maintaining their parallelism and ratios.**

```python
# Affine transformtio
# Define the points for affine transformation
srcPoints = np.float32([[50, 50], [200, 50], [50, 200]])
dstPoints = np.float32([[10, 100], [200, 50], [100, 250]])

# Get the affine transformation matrix
affine_matrix = cv2.getAffineTransform(srcPoints, dstPoints)

# Apply the affine transformation
affine_transformed_image = cv2.warpAffine(image, affine_matrix, (image.shape[1], image.shape[0]))

# Convert to RGB for displaying with Matplotlib
affine_transformed_image_rgb = cv2.cvtColor(affine_transformed_image, cv2.COLOR_BGR2RGB)

# Display the result
plt.imshow(affine_transformed_image_rgb)
plt.axis('off')
```



Notes

- Source Points: Original coordinates (srcPoints).
- Destination Points: Target coordinates (dstPoints).
- Matrix Calculation: Compute affine matrix with cv2.getAffineTransform.
- Apply Transformation: Use cv2.warpAffine to apply the matrix.
- Display: Convert to RGB and display with Matplotlib.



3. Deformable Transformation Matrix

A deformable transformation is a type of transformation that allows an image to be warped or deformed in a flexible manner, beyond simple linear changes. It has huge cases which will be covered in later modules.

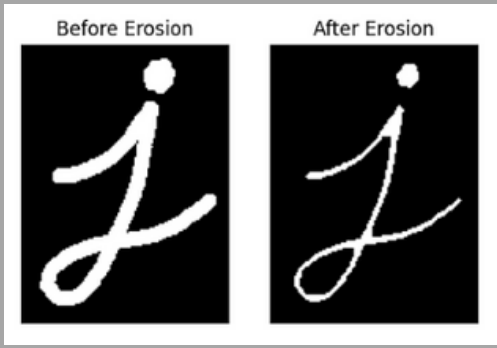
Morphological Transformation of Image

Note : We are using new image here. It is found at the same folder I previously provided

1. **Erosion**

- it erodes away the boundaries of foreground object (Always try to keep foreground in white).. The kernel(a new upper layer) slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).
- All the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image.

```
# Erosion
img_path = 'data/j.png'
image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

# Define a kernel (structuring element)
kernel = np.ones((5,5), np.uint8)

# Apply erosion
erosion = cv2.erode(image, kernel, iterations=1)

plt.figure(figsize=(8,7))
plt.subplot(131),plt.imshow(image, cmap='gray'),plt.title('Before Erosion')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(erosion, cmap='gray'),plt.title('After Erosion')
plt.xticks([]), plt.yticks([])
# plt.subplot(133),plt.imshow(dilation, cmap='gray'),plt.title('Dilation')
# plt.xticks([]), plt.yticks([])
plt.show()
```



It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

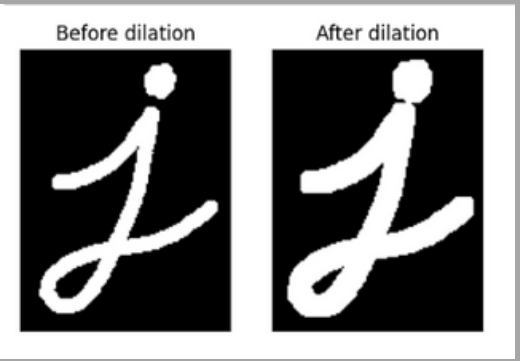
Notes :

1. np.ones(5,5) create a matrix of size 5*5 with value 1 and uinit8 is a data types which stands for 8-bit unsigned integers
2. This size of kernel (layer) goes through all part of image and makes changes accordingly

2. Dilation

- It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases.
- Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases.

```
img_path = 'data/j.png'
image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

# Define a kernel (structuring element)
kernel = np.ones((5,5), np.uint8)

# Apply dilation
dilation = cv2.dilate(image, kernel, iterations=1)

plt.figure(figsize=(8,7))
plt.subplot(131),plt.imshow(image, cmap='gray'),plt.title('Before dilation')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(dilation, cmap='gray'),plt.title('After dilation')
plt.xticks([]), plt.yticks([])
# plt.subplot(133),plt.imshow(dilation, cmap='gray'),plt.title('Dilation')
# plt.xticks([]), plt.yticks([])
plt.show()
```



Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

The process is same here as just in the Erosion.

Nishant Pandey