Finding **Region of interest (ROIs)** in pictures and videos is a fundamental component of computer vision. It is necessary to analyze and process certain regions of an image or video, known as ROIs, because they contain significant information. This blog includes the definition of ROIs and their significance in computer vision applications and methods, such as **feature-based, segmentation-based**, and **deep learning-based** ones, for finding ROIs in pictures and videos. We will look at how ROIs may be used to improve the precision of object detection and improvise tracking systems. We will also discuss some of the challenges associated with ROIs in computer vision, such as **scalability** and **computational efficiency.** When working with computer vision tasks, the OpenCV library provides various techniques for locating the region of interest using OpenCV in an image or video.

What is the Region of Interest in Computer Vision?

In "Computer Vision," the term "region of interest" (ROI) designates a particular area or region in an image or video that includes "crucial information" that has to be examined and processed. A portion of the overall picture with "special interest" is considered as the "ROI."

Computer vision applications can benefit from the usage of ROIs. Computer vision algorithms extract the "pertinent information" required to carry out these tasks precisely by recognizing the ROIs.

Definition and Explanation of ROI

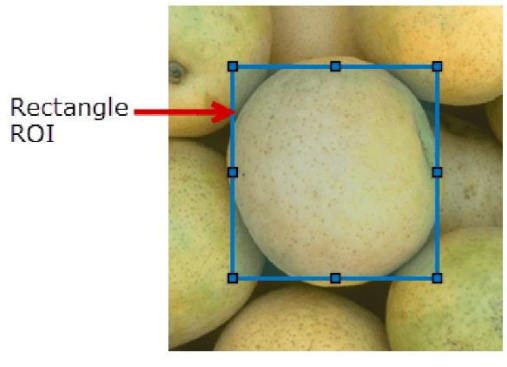
In computer vision, the term "region of interest" (ROI) designates a particular area or region in an image or video that is of special interest and importance to the job at hand. The ROI is frequently identified by its position, size, form, or other visual characteristics, but it can also be a subset of the full picture or video.

ROIs are used in various computer vision applications that include object recognition, tracking, and segmentation. In tracking, an ROI may be described as the region where the item is anticipated to travel, but in object recognition, an ROI can be defined as the area that includes an object of interest.

Types of ROIs (rectangular, circular, polygonal, etc.)

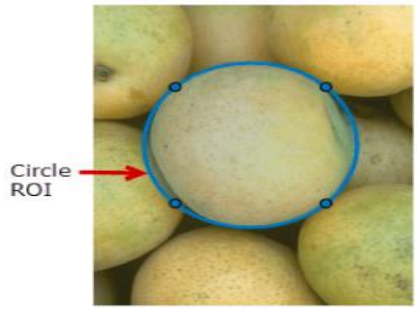
Depending on the particular needs of the application, ROIs can be created in a variety of forms, including different sizes, shapes, and kinds. The following are examples of the most typical ROIs used in computer vision:

1. Rectangular ROIs:



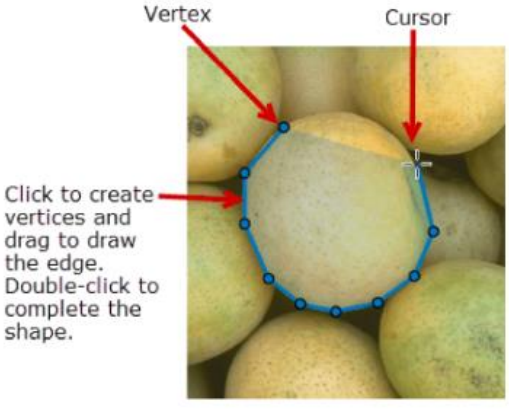
The width and height of a rectangular ROI determine its shape, and its location is frequently determined by its centre or top-left corner. Since rectangular ROIs are simple to define and implement, they are frequently used in computer vision applications.

2. Circular ROIs:



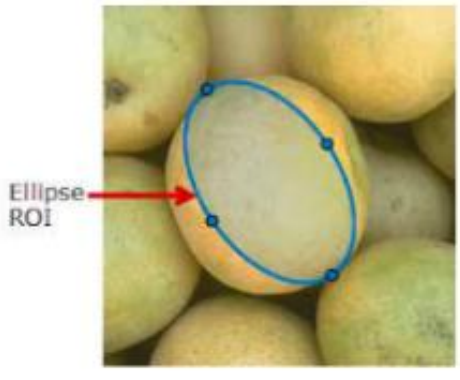
Circular ROIs have a radius and a centre, and they are frequently employed in situations where the ROI's form is important. For instance, certain applications of medical imaging employ circular ROIs to assess the size and form of tumors.

3. Polygonal ROIs:



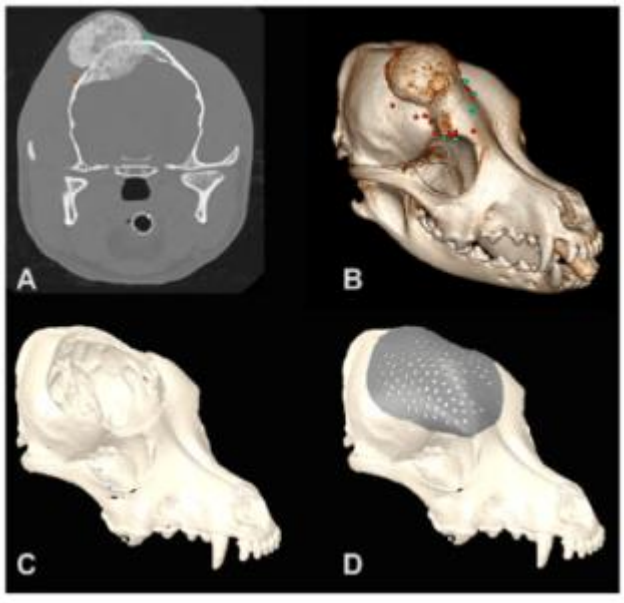
These ROIs may be shaped to suit the contours of the item of interest. Polygonal ROIs are defined by a group of interconnected vertices. When an object's shape is erratic or non-uniform, like in facial recognition or gesture identification, polygonal ROIs are frequently utilized.

4. Elliptical ROIs:



Elliptical ROIs are employed in situations when the ROI's form is important, but a circular shape is inappropriate. Elliptical ROIs are characterized by their major and minor axes as well as their centre.

5. Freeform ROIs:



Freeform ROIs can be used to extract ROIs that are particular to the user's requirements since they are specified by a user-defined shape.

Why ROIs are Used in Computer Vision?

By defining an ROI, computer vision algorithms can focus their analysis on a subset of the visual data rather than processing the entire image or video.

- The usage of ROIs in computer vision is particularly significant in applications such as object detection, tracking, and segmentation.
- Overall, the use of ROIs in computer vision allows algorithms to focus their analysis on the most relevant and informative parts of an image or video, reducing processing time and improving accuracy.

Importance of selecting the right ROI

The ROI specifies which region of the image or video the algorithm will examine, and picking the incorrect ROI may produce results that are incorrect or irrelevant.

- The size of the area should be one of the main considerations when choosing an ROI.
- If the ROI is too high, the algorithm might take longer to process the data and might be more likely to include unrelated data in the analysis.
- On the other hand, if the ROI is too small, crucial features might be missed, leading to unreliable results.

Methods for Selecting Region of Interest

In computer vision, there are various ways to choose a region of interest (ROI), each having benefits and drawbacks. Here are a few of the most popular techniques:

User-defined ROI: With this technique, the ROI is chosen manually by the user by tracing a polygon or bounding box around the subject of interest. The user may pick any region in the image or video using this simple, versatile way. It may not be appropriate for situations when the item of interest is moving swiftly or is challenging to monitor due to its time-consuming nature.

Manual Selection of ROIs

Object identification: Using object detection methods, the ROI may be automatically chosen based on the object of interest in the picture or video. This technique is quick and effective, making it appropriate for

real-time applications like surveillance or self-driving cars. However, it needs a trained model and might not be appropriate for scenes that are cluttered or complex.

Background subtraction: Background subtraction algorithms are able to distinguish between the foreground and background areas in an image or video and choose the ROI depending on the foreground area. The accuracy of this technique can be increased by combining it with object detection algorithms to track moving objects in videos. However, it might not be appropriate in situations where the target object is static or partially obscured.

Automatic Selection of ROIs Using Techniques Such as Edge Detection, Color Segmentation, and Object Detection

In computer vision applications, automatic ROI selection employing methods like edge detection, colour segmentation, and object detection is becoming more and more common. These methods can automate the selection of ROIs, reducing time spent and improving accuracy.

When selecting the ROI, edge detection techniques can automatically identify the edges of objects in an image or video. The boundaries of the vehicle may be utilized to define the ROI in applications like vehicle detection, where this strategy is very helpful.

Advantages and Disadvantages of Each Method

In computer vision applications, there are various ways to choose regions of interest (ROIs), each with pros and cons. Here are some advantages and disadvantages of each technique:

Manually choosing ROIs:

Advantages:

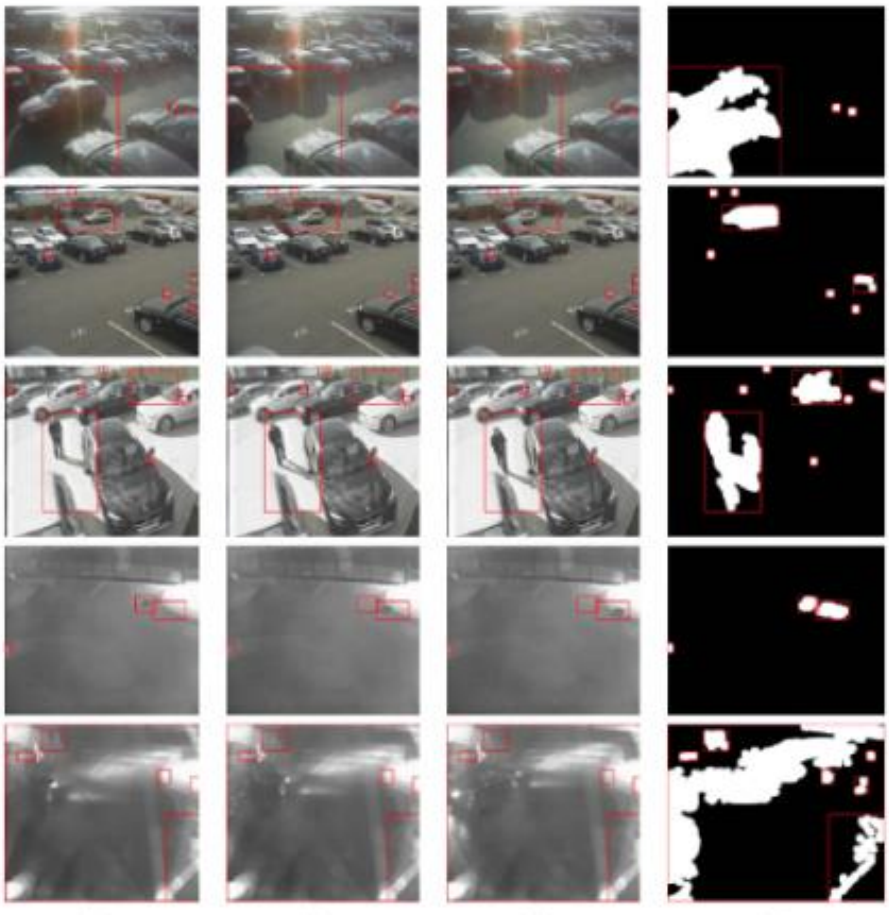
1. The ROI can be customized to meet the particular requirements of the application.
2. can be used in circumstances where other methods might not work.
3. If the user is familiar with the subject of interest, it may be more accurate.

Disadvantages:

1. Time-consuming, particularly if the user wants to choose several ROIs if the item of interest is moving fast.
2. Can be subjective, causing variations in ROI selection among users.
3. For real-time applications, ineffective.

Automatic selection of ROIs using edge detection:

An approach used in computer vision and image processing to identify and extract regions of interest in a picture is the automatic selection of ROIs (Regions of Interest) using edge detection. Edge detection is a process that analyses changes in pixel intensities to identify the boundaries or edges of objects in an image. This data may be used to identify sections of a picture containing items or features of interest.



Advantages:

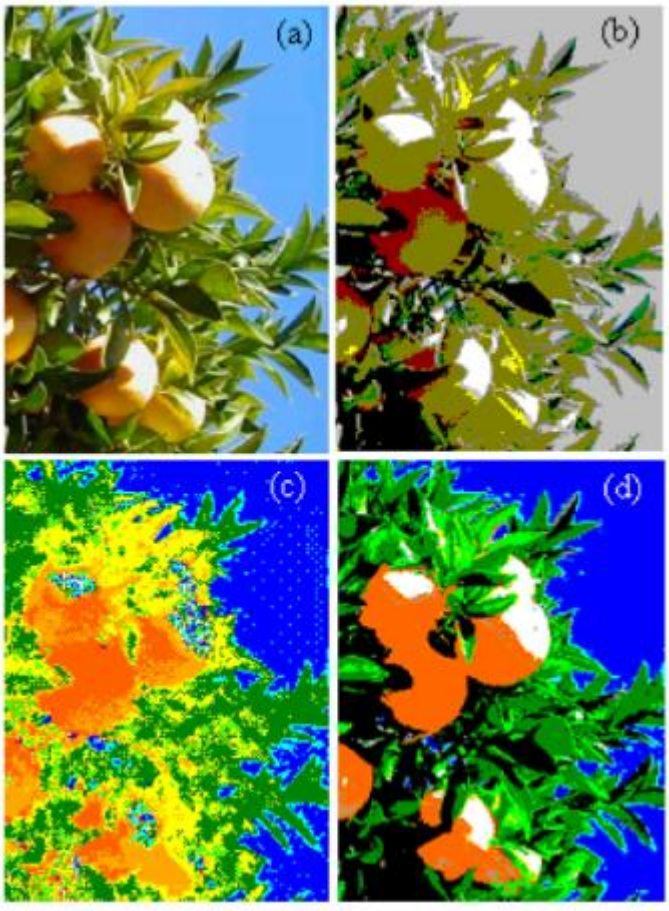
1. It is effective and quick, making real-time applications possible.
2. It Can occasionally be more precise than manual selection, especially if the object of interest has clearly defined edges.
3. Can be used to simultaneously detect multiple objects.

Disadvantages:

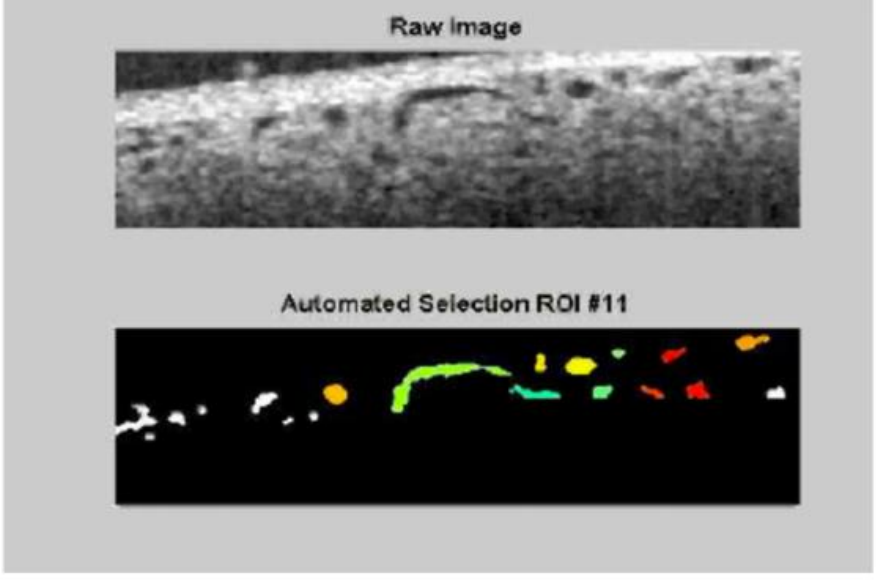
1. When the item of interest lacks clearly defined borders, it cannot be effective.
2. Situations with intricate backdrops or occlusions might not perform effectively.

Automatic selection of ROIs using color segmentation:

Colour segmentation is a method used in computer vision and image processing to detect and extract regions of interest in an image based on their colour.



Colour segmentation is the technique of identifying sections in a picture with similar colour properties. This data may be used to identify sections of a picture containing items or features of interest.

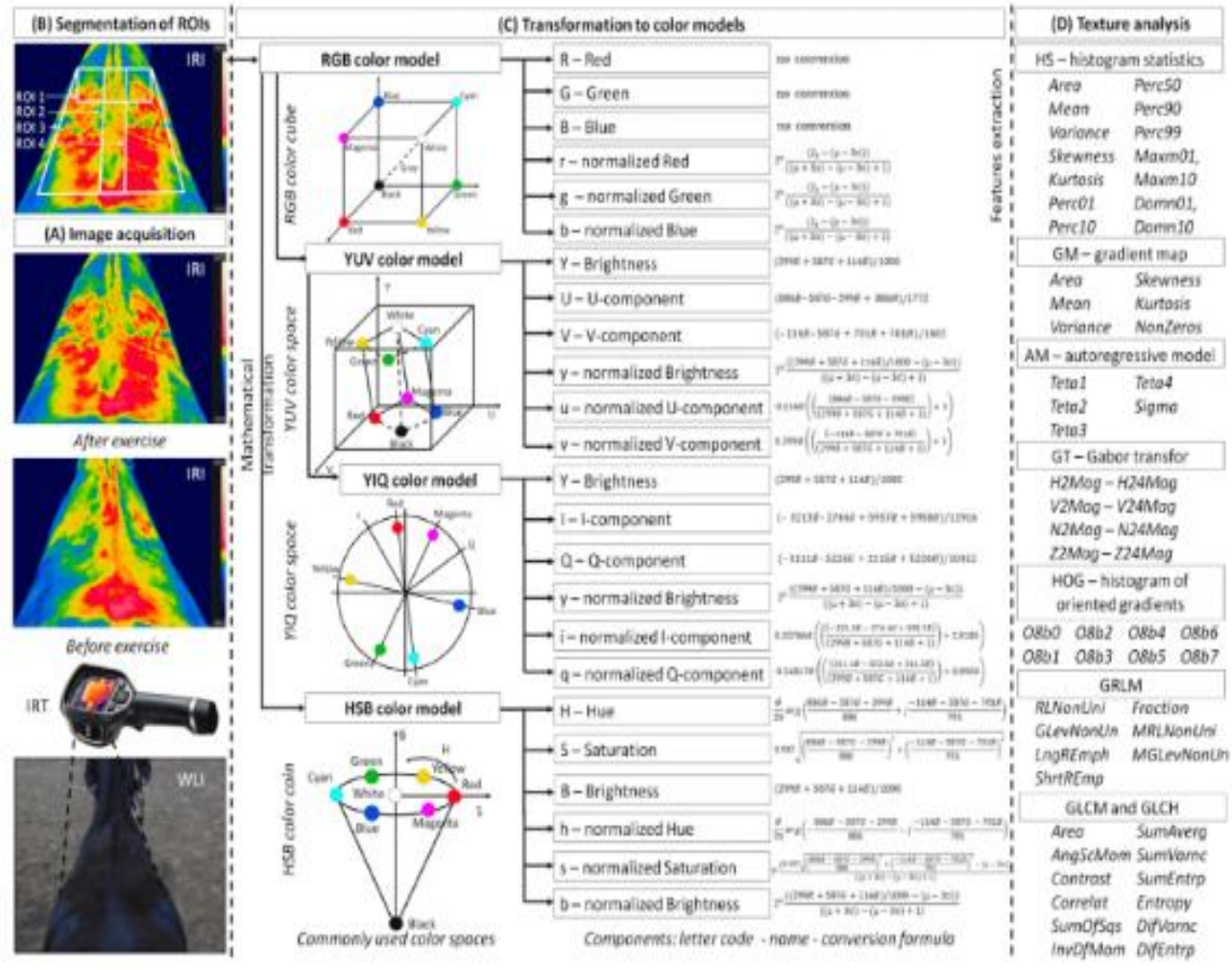


Advantages:

1. Can be useful when the object of interest has a recognizable colour.
2. Can be applied to real-time ROI selection.

Disadvantages:

1. When the object of interest lacks a recognizable colour, it can not operate effectively.



2. Alterations in lighting conditions could have an impact.

Automatic selection of ROIs using object detection:

Automatic selection of ROIs (Regions of Interest) using object detection is a computer vision and image processing approach for identifying and extracting regions of interest in an image based on the presence of specified objects.

The technique of identifying the position and boundaries of items in a picture is known as object detection. This data may be used to identify sections of a picture containing items or features of interest.

(a) (b)

Advantages:

1. It is effective and quick, making real-time applications possible.
2. In some circumstances, especially when the item of interest is complicated or challenging to identify, it may be more accurate than manual selection.
3. Can be used to simultaneously detect multiple objects.

Disadvantages:

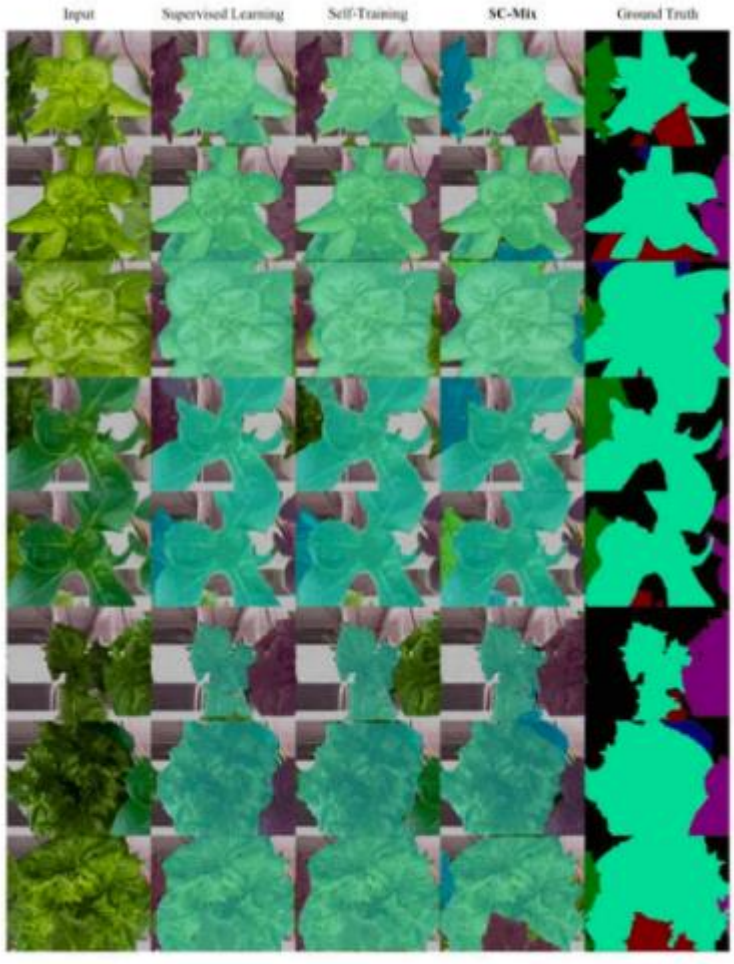
1. It is dependent on training data, and it could not perform effectively for items that aren't represented in the training data.
2. Costly computationally, particularly for deep learning-based methods.
3. In circumstances where the item of attention is partially obscured or has a complicated background, it might not be successful.

Applications of ROI in Computer Vision

Region of interest (ROI) selection is a core component of computer vision and has several uses in a variety of fields. In computer vision, the following are some of the most typical ROI applications:

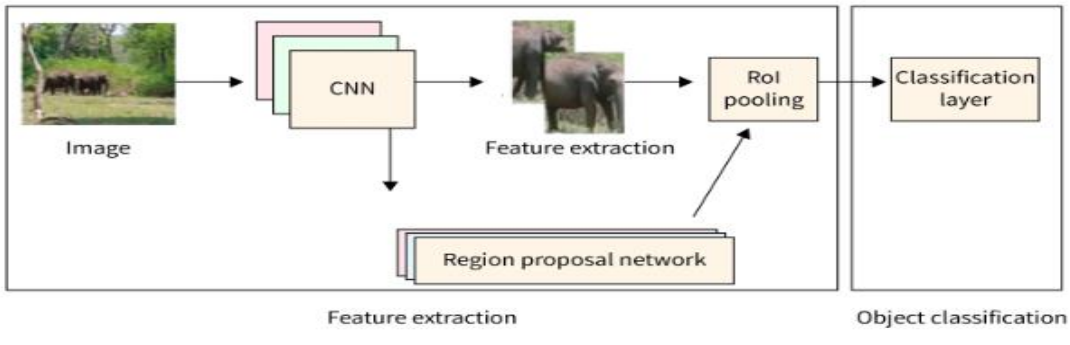
1. **Object tracking and surveillance:**
   Real-time video feeds may be followed to find items of interest using ROI selection, which is also used for surveillance. When a user has to keep an eye on a certain region or item, such as in security and surveillance applications, this is helpful.
2. **Medical imaging:**
   The ROI selection technique is used in medical imaging to separate and examine particular bodily sections or organs. Consequently, medical professionals can accurately diagnose and track patient conditions.
3. **Autonomous cars:**
   Self-driving cars employ ROI selection to find and follow things, including other cars, people walking along the road, and traffic signals. This is essential to ensure the security of travelers and other drivers.
4. **Robotics:**
   Robots can recognize and move things in their surroundings by using ROI selection. This is helpful in situations like manufacturing, where robots must carefully choose and put products.
5. **Agriculture:**
   In the field of agriculture, ROI selection may be used to detect and categorize crops, keep track of

soil moisture, and spot pest infestations. Farmers may use this to optimize their yields and make data-driven decisions.



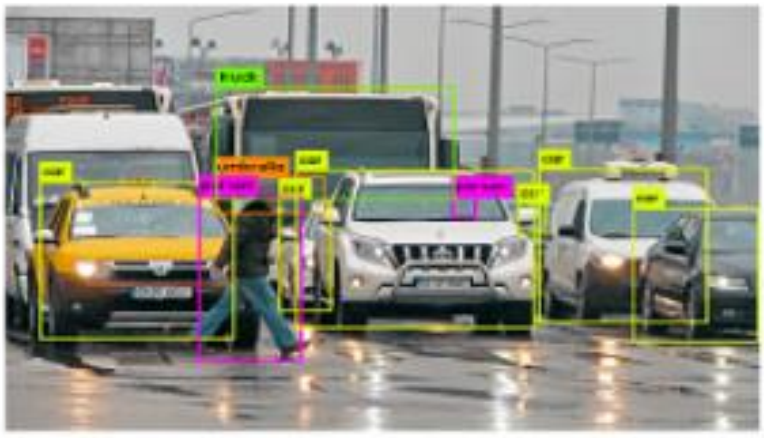
Object Tracking and Detection

One of the most common uses of Region of Interest (ROI) in computer vision is object tracking and detection. While object detection refers to the process of detecting the presence of an object of interest in an image or video frame, object tracking involves finding a specific object of interest in a sequence of images or video frames.



Various methods, such as feature-based methods, appearance-based methods, and deep learning-based methods, can be used to track and detect objects.

A neural network is trained using deep learning techniques to find and follow the item of interest. Due to its great accuracy and resilience, this method has gained popularity in recent years.



Face Detection and Recognition

Another common use of Region of Interest (ROI) in computer vision is face identification and recognition. Face recognition identifies and verifies an individual's identification based on their facial traits, whereas face detection identifies the existence and placement of faces in an image or video frame.

Convolutional Neural Networks (CNNs), in particular, have demonstrated outstanding effectiveness in both face identification and recognition using deep learning-based techniques. CNNs may be trained to recognize certain facial patterns and traits, which makes it possible for them to recognize and reliably detect faces in a variety of contexts.

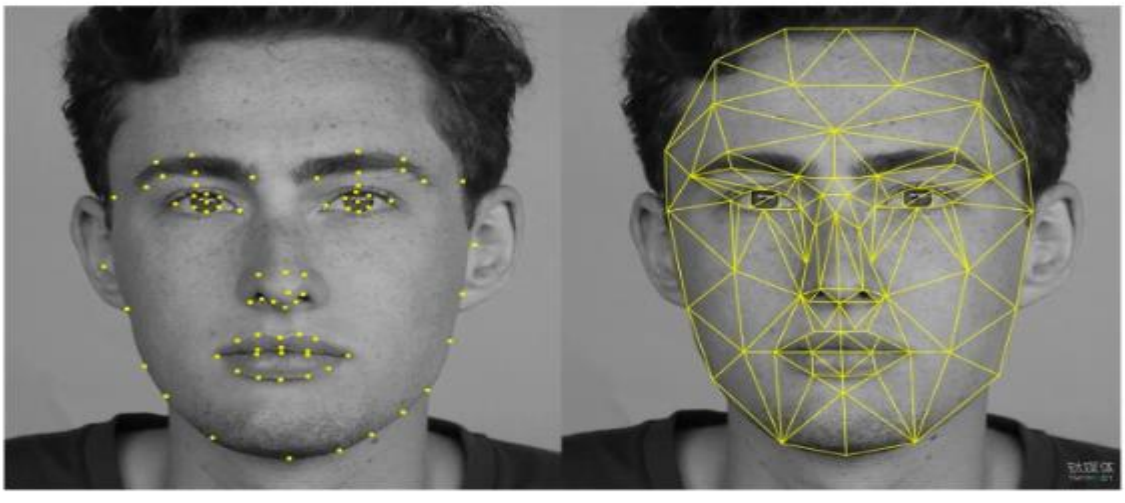
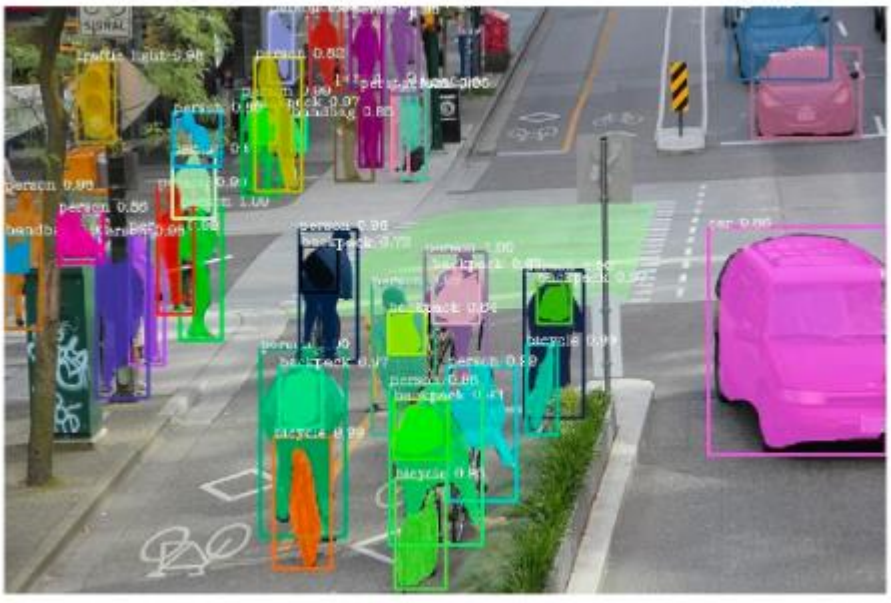


Image Segmentation

Another crucial use of Region of Interest (ROI) in computer vision is image segmentation. In order to segment a picture, it must be divided into a number of regions or segments, each of which represents a different item or portion of the image. The region or area of interest for image segmentation may be defined using ROI.

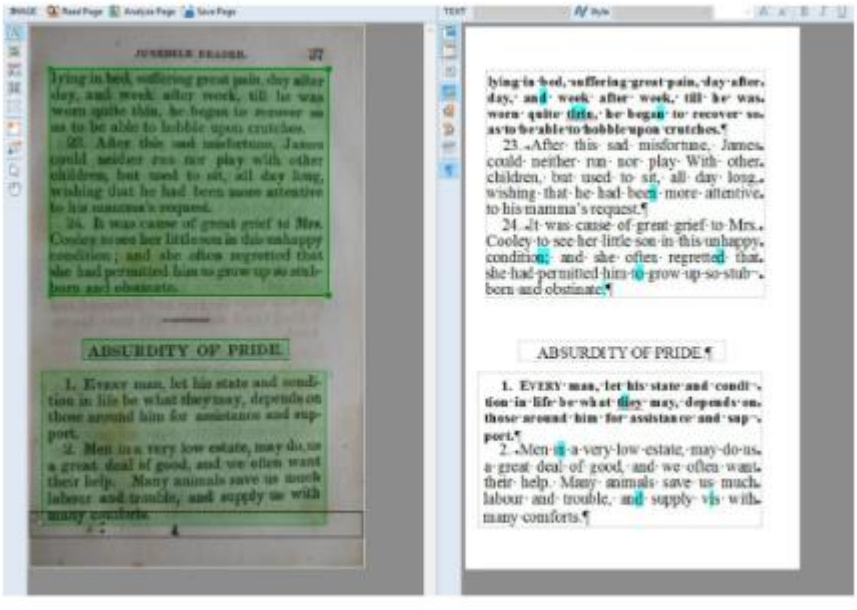
Medical imaging, object identification, and autonomous driving are just a few of the computer vision applications that image segmentation may be used for. Image segmentation can be used in medical imaging to recognize and separate certain organs or tissues for examination or diagnosis.



Optical Character Recognition (OCR)

Region of Interest (ROI) is another tool used in computer vision for optical character recognition (OCR). OCR is the process of identifying and extracting text from a document or an image. The region or area of interest containing the text in the picture or document may be identified using ROI.

OCR has several uses in a variety of fields, including banking, healthcare, and education. OCR can be used in the financial industry to automatically extract data from invoices, receipts, and other financial documents, saving time and labor by eliminating the need for manual data entry. OCR may be used in the healthcare industry to extract and analyze patient data from medical records, allowing for more effective and precise diagnosis and treatment.



Medical Imaging

Another use of Region of Interest (ROI) in computer vision is in medical imaging. Many of the details in medical pictures, like X-rays, CT scans, and MRI scans, are unrelated to the diagnosis or treatment of a specific medical problem. ROI can help you locate and extract the region or area of interest that contains important data.

ROI has a wide range of applications in medical imaging, including tumor detection and segmentation, organ identification, and image registration. For instance, ROI may be used to locate the area of the picture containing the tumor while detecting and segmenting tumors.

Autonomous Vehicles and Robotics

Region of Interest (ROI) in computer vision has several vital applications in robotics and autonomous vehicles. ROI is used in various fields to locate and follow real-time items of interest.

ROI can be used, for instance, in autonomous cars to recognize and track pedestrians, other vehicles, and traffic signs. The computer vision system may concentrate on analyzing just the pertinent data by defining the zone of interest around these objects, improving accuracy and efficiency.

In robotics, ROI can be used to track the movement of objects or parts within a system. For example, in a manufacturing plant, ROI can be used to track the movement of components along a production line. By identifying the region of interest around the components, the computer vision system can track their movement and ensure that they are assembled correctly.

Other Use Cases

Region of Interest (ROI) in computer vision has numerous additional use cases outside the ones just listed. Here are a few instances:

1. **Sports analytics:**

ROI may be used in sports analytics to track and examine player mobility in activities like football, basketball, and tennis. Computer vision systems can follow the player's movement, spot trends, and offer insights into their performance by specifying the region of interest surrounding them.

2. **Security and surveillance:**

   ROI may be applied to security and surveillance applications to identify and monitor people and things. ROI can be used, for instance, in airports to follow the movement of passengers and spot those who could be carrying contraband.

3. **Agriculture:**
   Crop health and growth trends may be examined using ROI. Computer vision systems can analyze crop development, spot disease or pest infestations, and offer insights into agricultural production by specifying the zone of interest around the crops.

4. **Retail analytics:**

   ROI is a tool that may be used to study consumer behavior in retail settings. Computer vision systems may track consumer movement and behavior, identify shopping trends, and offer insights into client preferences by specifying the zone of interest surrounding certain parts of a business.

5. **Robotics in construction:**

   ROI may be used to analyze and monitor the development of building projects. Robotics in construction. Computer vision systems can monitor progress and follow the movement of tools and supplies on a construction site by defining the zone of interest around certain areas in real time.

Feature Extraction: HOG (Histogram of Oriented Gradients)

Histogram of Oriented Gradients, also known as HOG, is a feature descriptor like the Canny Edge Detector, SIFT (Scale Invariant and Feature Transform) . It is used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in the localized portion of an image. This method is quite similar to Edge Orientation Histograms and Scale Invariant aFeature Transformation (SIFT). The HOG descriptor focuses on the structure or the shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features. For the regions of the image it generates histograms using the magnitude and orientations of the gradient.

Steps to calculate HOG Features

1. Take the input image you want to calculate HOG features of. Resize the image into an image of 128x64 pixels (128 pixels height and 64 width). This dimension was used in the paper and was suggested by the authors as their primary aim with this type of detection was to obtain better results on the task of pedestrian detection. As the authors of this paper were obtaining exceptionally perfect results on the MIT pedestrian database, they decided to produce a new and significantly more challenging dataset called the 'INRIA' dataset (http://pascal.inrialpes.fr/data/human/), containing 1805 (128x64) images of humans cropped from a varied set of personal photos.

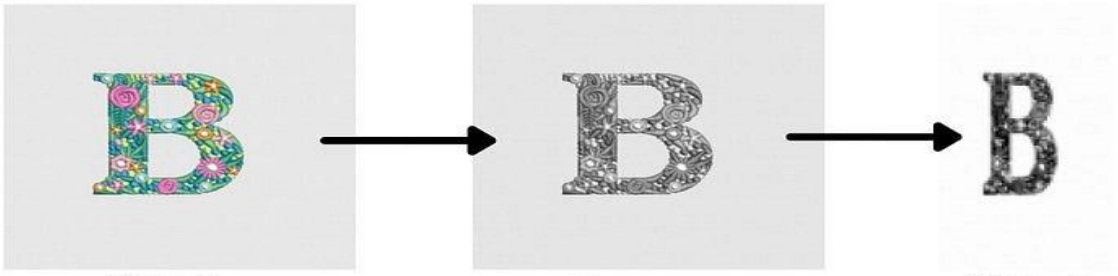


Figure 1 Figure 2 Figure 3

Figure 1 : The image imported to get HOG features of. Figure 2 : The imported image grayscale for the process. Figure 3 : Resized and grayscale image of the imported image. (Image by author)

2. The gradient of the image is calculated. The gradient is obtained by combining magnitude and angle from the image. Considering a block of 3x3 pixels, first Gx and Gy is calculated for each pixel. First Gx and Gy is calculated using the formulae below for each pixel value .

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \quad G_y(r,c) = I(r-1,c) - I(r+1,c)$$

where r, c refer to rows and columns respectively. (Image by author)

After calculating Gx and , magnitude and angle of each pixel is calculated using the formulae mentioned below.

$$Magnitude(\mu) = \sqrt{G_x^2 + G_y^2} \quad Angle(\theta) = |\tan^{-1}(G_y/G_x)$$

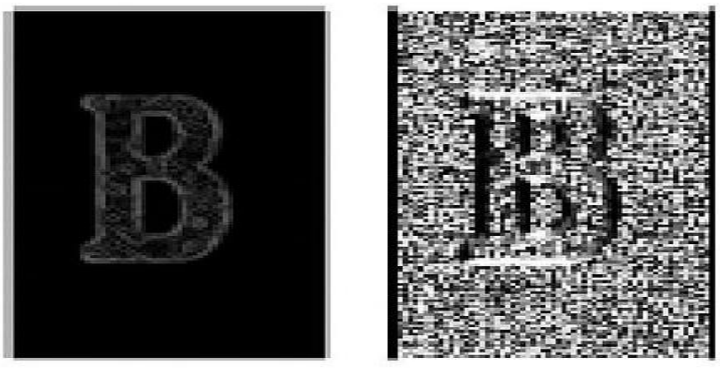


Figure 4 Figure 5

Figure 4 : Visualization of magnitude of the image. Figure 5 : Visualization of angle of the image. (Image by author)

3. After obtaining the gradient of each pixel, the gradient matrices (magnitude and angle matrix) are divided into 8x8 cells to form a block. For each block, a 9-point histogram is calculated. A 9-point histogram develops a histogram with 9 bins and each bin has an angle range of 20 degrees. Figure 8 represents a 9 bin histogram in which the values are allocated after calculations. Each of these 9-point histograms can be plotted as histograms with bins outputting the intensity of the gradient in that bin. As a block contains 64 different values, for all 64 values of magnitude and gradient the following calculation is performed. As we are using 9 point histograms, hence :

$$Number\ of\ bins = 9(ranging\ from\ 0^\circ\ to\ 180^\circ)$$
$$Step\ size(\Delta\theta) = 180^\circ\ /\ Number\ of\ bins = 20^\circ$$

(Image by author)

Each Jth bin, bin will have boundaries from :

$[\Delta\theta \cdot j, \Delta\theta \cdot (j+1)]$

(Image by author)

Value of the centre of each bin will be :

$$C_j = \Delta\theta(j + 0.5)$$

(Image by author)

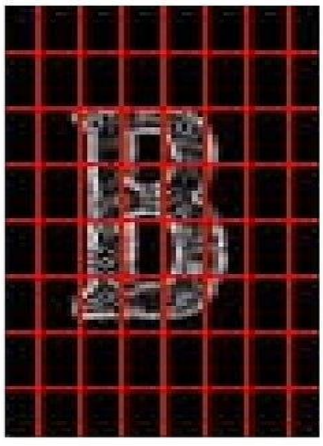


Figure 6

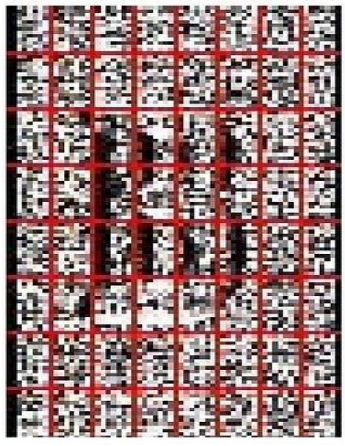


Figure 7

Figure 6 : 8x8 blocks on the magnitude image. Figure 7 : 8x8 blocks on an angle image. (Image by author)

Value									
Bins	0	20	40	60	80	100	120	140	160

Figure 8 : Representation of a 9 bin histogram. This one single histogram will be unique for one 8x8 block made up of 64 cells. All 64 cells will add their Vj and Vj+1 value to the jth and (j+1)th index of the array respectively. (Image by author)

4. For each cell in a block, we will first calculate the jth bin and then the value that will be provided to the jth and (j+1)th bin respectively. The value is given by the following formulae :

$$j = \lfloor \left(\frac{\theta}{\Delta\theta} - \frac{1}{2} \right) \rfloor$$

$$V_j = \mu \cdot \left[\frac{\theta}{\Delta\theta} - \frac{1}{2} \right]$$

$$V_{j+1} = \mu \cdot \left[\frac{\theta - C_j}{\Delta\theta} \right]$$

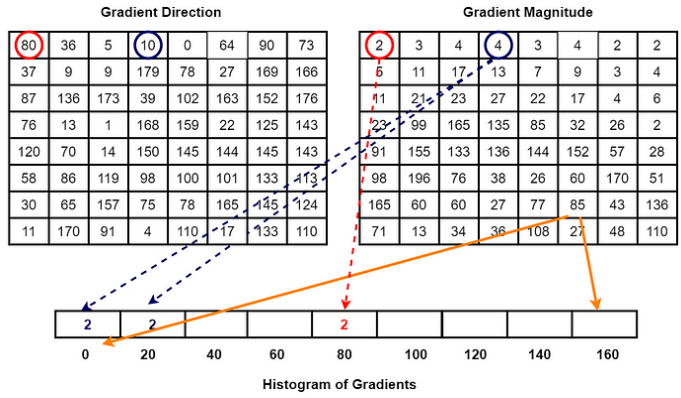
(Image by author)

5. An array is taken as a bin for a block and values of Vj and Vj+1 is appended in the array at the index of jth and (j+1)th bin calculated for each pixel.

6. The resultant matrix after the above calculations will have the shape of 16x8x9.

7. Once histogram computation is over for all blocks, 4 blocks from the 9 point histogram matrix are clubbed together to form a new block (2x2). This clubbing is done in an overlapping manner

with a stride of 8 pixels. For all 4 cells in a block, we concatenate all the 9 point histograms for each constituent cell to form a 36 feature vector.



$$f_{bi} = [b_1, b_2, b_3, ..., b_{36}]$$

Traversing of 2x2 grid box around the image in order to make a combined fbi from 4 blocks. (Image by author)

8. Values of fb for each block is normalized by the L2 norm :

$$f_{bi} \leftarrow \frac{f_{bi}}{\sqrt{\|f_{bi}\|^2 + \varepsilon}}$$

Where ε is a small value added to the square of fb in order to avoid zero division error. In code value taken of is 1e-05. (Image by author)

9. To normalize, the value of k is first calculated by the following formulae :

$$k = \sqrt{b_1^2 + b_2^2 + b_3^2 + ... + b_{36}^2}$$

$$f_{bi} = \left[\left(\frac{b_1}{k}\right), \left(\frac{b_2}{k}\right), \left(\frac{b_3}{k}\right), ..., \left(\frac{b_{36}}{k}\right)\right]$$

(Image by author)

10. This normalization is done to reduce the effect of changes in contrast between images of the same object. From each block. A 36 point feature vector is collected. In the horizontal direction there are 7 blocks and in the vertical direction there are 15 blocks. So the total length of HOG features will be : 7 x 15 x 36 = 3780. HOG features of the selected image are obtained.

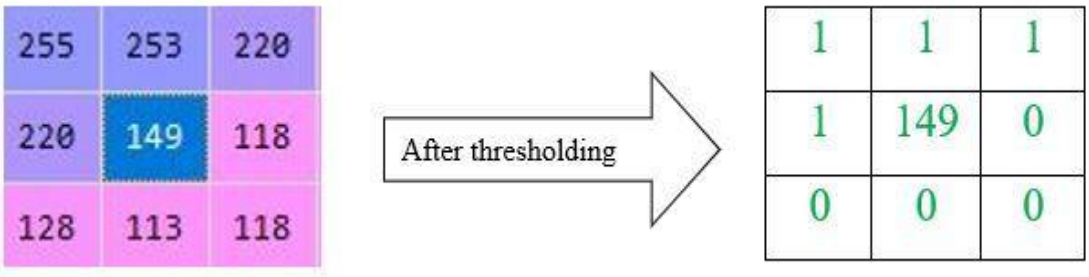
Visualization of HOG features on the same image using skimage library. (Image by author)

Feature Extraction: Local Binary Pattern

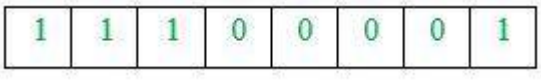
There are lots of different types of texture descriptors are used to extract features of an image. Local Binary Pattern, also known as LBP, is a simple and grayscale invariant texture descriptor measure for classification. In LBP, a binary code is generated at each pixel by thresholding it's neighbourhood pixels to either 0 or 1 based on the value of the centre pixel. The rule for finding LBP of an image is as follows:

1. Set a pixel value as center pixel.
2. Collect its neighbourhood pixels (Here I am taking a 3 x 3 matrix so; total number of neighbourhood pixel is 8)
3. Threshold it's neighbourhood pixel value to 1 if its value is greater than or equal to centre pixel value otherwise threshold it to 0.
4. After thresholding, collect all threshold values from neighbourhood either clockwise or anti-clockwise. The collection will give you an 8-digit binary code. Convert the binary code into decimal.
5. Replace the center pixel value with resulted decimal and do the same process for all pixel values present in image.

Let's take an example to understand it properly. Let's take a pixel value from the above output to find its binary pattern from its local neighbourhood. So, I am taking a value '149' (present at 15th row and 19nd column) and its 8 neighbourhood pixels to form a 3 x 3 matrix.



Collect the thresholding values either clockwise or anti-clockwise. Here, I am collecting them clockwise from top-left. So, after collecting, the binary value will be as follows:

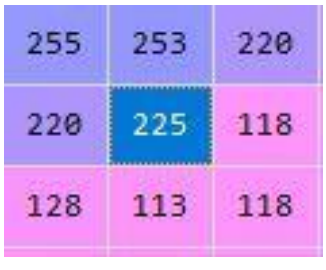


Then, convert the binary code into decimal and place it at center of matrix.

1 x 27 + 1 x 26 + 1 x 25 + 0 x 24 + 0 x 23 + 0 x 22 + 0 x 21 +1 x 20

= 128 + 64 + 32 + 0 + 0 + 0 + 0 + 1

= 225



Now, the resulted matrix will look like,Now, let's do it using python

Feature Extraction of Images using GLCM (Gray Level Cooccurrence Matrix)

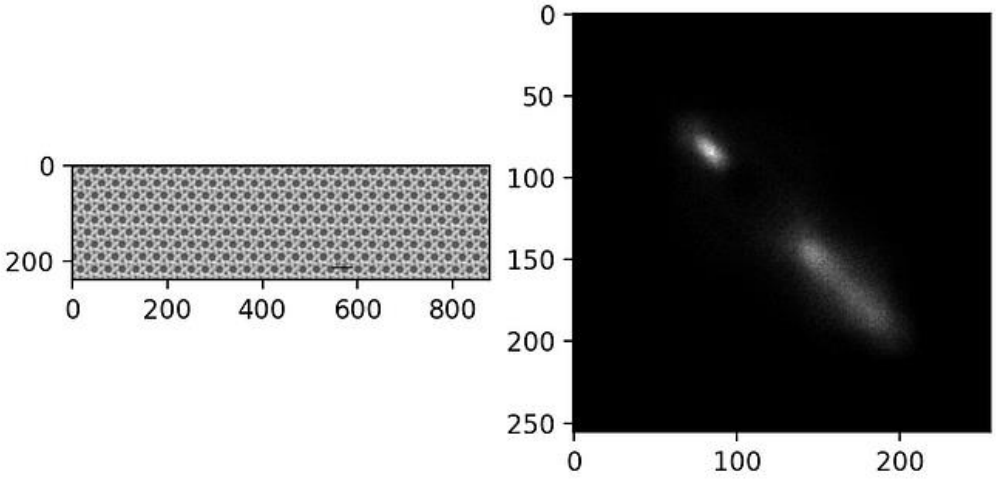


Image and its GLCM with 255 levels, 50 pixel offset along Vertical direction (90 degree)

Feature extraction plays a pivotal role in image processing and computer vision tasks. Images contain vast amounts of data, and extracting meaningful information from them is essential for various applications.

Some common feature extraction methods are Local Binary Pattern (LBP), Histogram of Gradients (HOG) and Gray Level Co-occurrence Matrix (GLCM). LBP captures texture by comparing local pixel patterns, HOG describes object shape through gradient orientation histograms, while GLCM characterizes texture via spatial pixel intensity relationships. In this blog I will explain about GLCM. You can also read about HOG in my other blog.

Unlike simple pixel intensity-based features, GLCM considers the relationships between neighboring pixels, providing rich texture information. This makes GLCM particularly well-suited for tasks that require analyzing fine-grained texture patterns, such as medical image analysis.

Understanding Gray Level Co-occurrence Matrix (GLCM)

“Gray Level” in GLCM

Pixel intensities as “grays”: Each pixel in a grayscale image holds an intensity value, typically ranging from 0 (black) to 255 (white) for 8-bit images. These intensity values are often referred to as “grays” in the context of GLCM.

“Co-occurrence Matrix” in GLCM

Capturing Spatial Relationships Neighborly analysis: The key concept of GLCM lies in analyzing how often gray levels (intensities) occur together within an image, specifically considering neighboring pixels.

Pixel offsets and directions: By counting co-occurrences in a specific pixel offset (often 1 or 2) and specific directions (e.g., horizontal, vertical, diagonal), GLCM captures the spatial arrangement of textures.

So these are 3 parameters used in calculating GLCM:

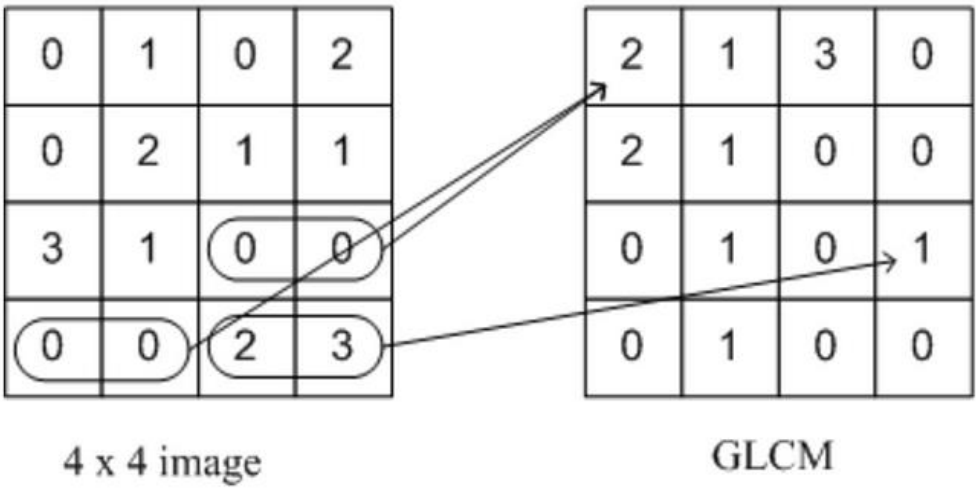
Distance (d): The displacement between two pixels.

Angle (θ): The direction in which pixel pairs are considered, typically in 0°, 45°, 90°, and 135°.

Number of Gray Levels (G): The number of discrete intensity levels in the image.

Constructing the Co-occurrence Matrix

Imagine a table with rows and columns labeled with all possible grayscale values (e.g., 0–255). Each cell represents the frequency with which a specific pair of gray levels co-occur at a given offset and direction. For example, if cell (50, 60) has a value of 20, it means that the gray level 50 occurs 20 times next to a pixel with intensity 60 at the specified offset and direction.



GLCM Calculation with GL intensity 4, Offset 1 Pixel and Angle 0 degree (Horizontal)

Features from GLCM

Once the GLCM is computed, various statistical measures can be derived from it to characterize the texture and structure of the image. Some common features are

Contrast: Measures the local variations in the image. High contrast values indicate large differences between neighboring pixel intensities.

Dissimilarity: Measures the average difference in intensity between neighboring pixels. High dissimilarity values indicate greater heterogeneity in texture.

Homogeneity: Reflects the closeness of the distribution of elements in the GLCM to the GLCM diagonal. High homogeneity values indicate that elements are concentrated along the diagonal, suggesting a more uniform texture.

Energy (or Angular Second Moment): Represents the orderliness or homogeneity of the image. High energy values indicate more uniform texture.

Correlation: Measures the linear dependency between pixel pairs. High correlation values indicate a more predictable texture.

What is Image Compression?

In the field of Image processing, the compression of images is an important step before we start the processing of larger images or videos. The compression of images is carried out by an encoder and output a compressed form of an image. In the processes of compression, the mathematical transforms play a vital role. A flow chart of the process of the compression of the image can be represented as:

In this article, we try to explain the overview of the concepts involved in the image compression techniques. The general representation of the image in a computer is like a vector of pixels. Each pixel is represented by a fixed number of bits. These bits determine the intensity of the color (on grayscale if a black and white image and has three channels of RGB if colored images.)

Why Do We Need Image Compression?

Consider a black and white image that has a resolution of 1000*1000 and each pixel uses 8 bits to represent the intensity. So the total no of bits required = 1000*1000*8 = 80,00,000 bits per image. And **consider if it is a video with 30 frames per second of the above-mentioned type images then the total bits for a video of 3 secs is: 3*(30*(8, 000, 000))=720, 000, 000 bits**

As we see just to store a 3-sec video we need so many bits which is very huge. So, we need a way to have proper representation as well to store the information about the image in a minimum

no of bits without losing the character of the image. Thus, image compression plays an important role.

Basic steps in image compression:

- Applying the image transform
- Quantization of the levels
- Encoding the sequences.

Transforming the Image

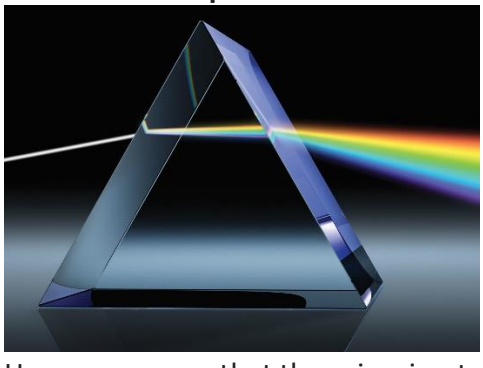
What is a transformation (Mathematically):

It is a function that maps from one domain (vector space) to another domain(other vector space). Assume, **T is a transform, f(t):X->X' is a function then, T(f(t)) is called the transform of the function.**

In a simple sense, we can say that T changes the shape (representation) of the function as it is a mapping from one vector space to another (without changing basic function f(t) i.e. the relationship between the domain and co-domain).

We generally carry out the transformation of the function from one vector space to the other because when we do that in the newly projected vector space we infer more information about the function.

A real life example of a transform:



Here we can say that the prism is a transformation function in which it splits the white light (f(t)) into its components i.e the representation of the white light. And we observe that we can infer more information about the light in its component representation than the white light one. This is how transforms help in understanding the functions in an efficient manner.

Transforms in Image Processing

The image is also a function of the location of the pixels. i.e I(x, y) where (x, y) are the coordinates of the pixel in the image. So we generally transform an image from the spatial domain to the frequency domain.

Why Transformation of the Image is Important:

- It becomes easy to know what all the principal components that make up the image and help in the compressed representation.
- It makes the computations easy.
    - Example: finding convolution in the time domain before the transformation:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau.$$

- Finding convolution in the frequency domain after the transformation:

$$(f*g)(t) = F(s)G(s)$$

- So we can see that the computation cost has reduced as we switched to the frequency domain. We can also see that in the time domain the convolution was equivalent to an integration operator but in the frequency domain, it becomes equal to the simple product of terms. So, this way the cost of computation reduces.

So this way when we transform the image from domain to the other carrying out the spatial filtering operations becomes easier.

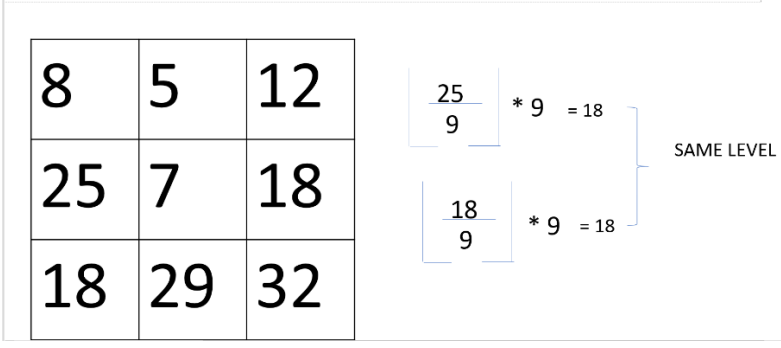
Quantization

The process quantization is a vital step in which the various levels of intensity are grouped into a particular level based on the mathematical function defined on the pixels. Generally, the newer level is determined by taking a fixed filter size of "m" and dividing each of the "m" terms of the filter and rounding it its closest integer and again multiplying with "m".

Basic quantization Function: [pixelvalue/m] * m

So, the closest of the pixel values approximate to a single level hence as the no of distinct levels involved in the image becomes less. Hence we reduce the redundancy in the level of the intensity. So thus quantization helps in reducing the distinct levels.

Eg: (m=9)



Thus we see in the above example both the intensity values round up to 18 thus we reduce the number of distinct levels (characters involved) in the image specification.

Symbol Encoding

The symbol stage involves where the distinct characters involved in the image are encoded in a way that the no. of bits required to represent a character is optimal based on the frequency of the character's occurrence. In simple terms, In this stage codewords are generated for the different characters present. By doing so we aim to reduce the no. of bits required to represent the intensity levels and represent them in an optimum number of bits.

There are many encoding algorithms. Some of the popular ones are:

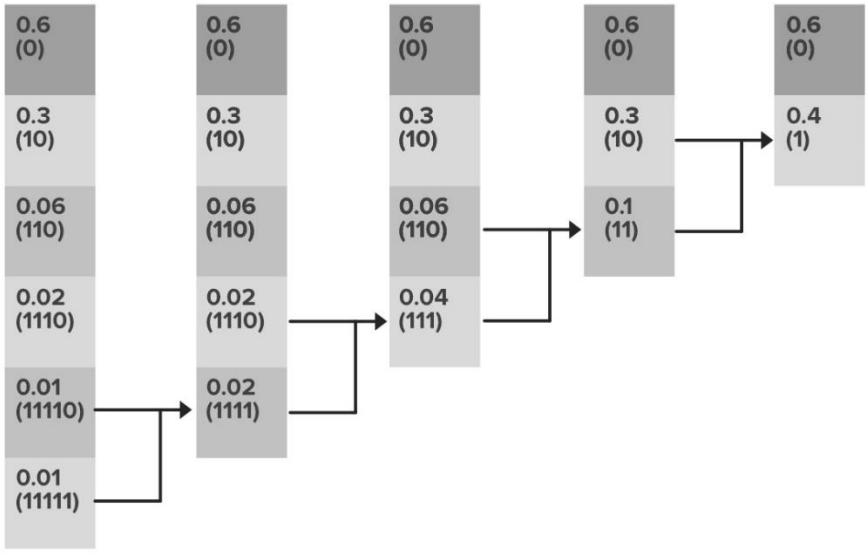
- Huffman variable-length encoding.
- Run-length encoding.

In the Huffman coding scheme, we try to find the codes in such a way that none of the codes are the prefixes to the other. And based on the probability of the occurrence of the character the length of the code is determined. In order to have an optimum solution the most probable character has the smallest length code.

Example:

SYMBOLS (with instensity value gray scale)	Probability (arranged in decreasing order)	Binary Code	Huffman code	Length of Huffman code
a1 - 18	0.6	00010010	0	1
a2 - 25	0.3	00011001	10	2
a3 - 255	0.06	11111111	110	3
A4 - 128	0.02	10000000	1110	4
a5 - 200	0.01	11001000	11110	5
a6 - 140	0.01	10001100	11111	5

We see the actual 8-bit representation as well as the new smaller length codes. The mechanism of generation of codes is:



So we see how the storage requirement for the no of bits is decreased as:
Initial representation–average code length: 8 bits per intensity level.
After encoding–average code length:
*(0.6*1)+(0.3*2)+(0.06*3)+(0.02*4)+(0.01*5)+(0.01*5)=1.56 bits per intensity level*
Thus the no of bits required to represent the pixel intensity is drastically reduced.

Thus in this way, the mechanism of quantization helps in compression. When the images are once compressed its easy for them to be stored on a device or to transfer them. And based on the type of transforms used, type of quantization, and the encoding scheme the decoders are designed based on the reversed logic of the compression so that the original image can be re-built based on the data obtained out of the compressed images.

Image compression models.

Fig. 3.1 shows, a compression system consists of two distinct structural blocks: an encoder and a decoder. An input image f(x, y) is fed into the encoder, which creates a set of symbols from the input data. After transmission over the channel, the encoded representation is fed to the decoder, where a reconstructed output image f^(x, y) is generated. In general, f^(x, y) may or may not be an exact replica of f(x, y). If it is, the system is error free or information preserving; if not, some level of distortion is present in the reconstructed image. Both the encoder and decoder shown in Fig. 3.1 consist of two relatively independent functions or subblocks. The encoder is made up of a source encoder, which removes input redundancies, and a channel encoder, which increases the noise immunity of the source encoder's output. As would be expected, the decoder includes a channel decoder followed by a source decoder. If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively.

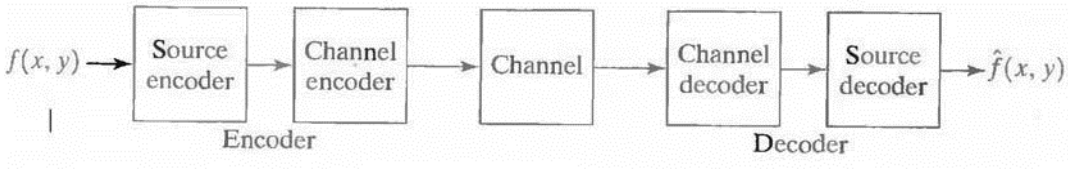
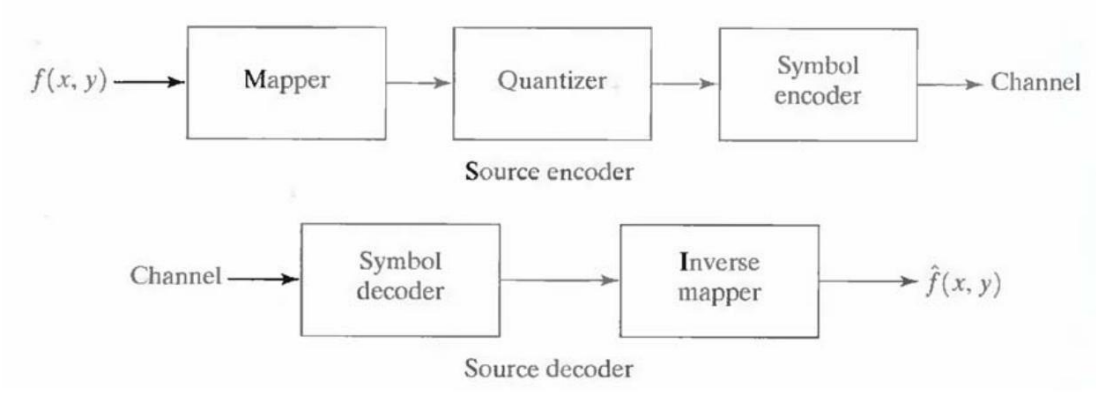


Fig.3.1 A general compression system model The Source Encoder and Decoder:

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisual redundancies in the input image. The specific application and associated fidelity requirements dictate the best encoding approach to use in any given situation. Normally, the approach can be modeled by a series of three independent operations. As Fig. 3.2 (a) shows, each operation is designed to reduce one of the three redundancies. Figure 3.2 (b) depicts the corresponding source decoder. In the first stage of the source encoding process, the mapper transforms the input data into a (usually nonvisual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.



a
b

Fig.3.2 (a) Source encoder and (b) source decoder model

Run-length coding is an example of a mapping that directly results in data compression in this initial stage of the overall source encoding process. The representation of an image by a set of transform coefficients is an example of the opposite case. Here, the mapper transforms the image into an array of coefficients, making its interpixel redundancies more accessible for compression in later stages of the encoding process.

The second stage, or quantizer block in Fig. 3.2 (a), reduces the accuracy of the mapper's output in accordance with some preestablished fidelity criterion. This stage reduces the psychovisual redundancies of the input image. This operation is irreversible. Thus it must be omitted when error-free compression is desired.

In the third and final stage of the source encoding process, the symbol coder creates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term symbol coder distinguishes this coding operation from the overall source encoding process. In most cases, a variable-length code is used to represent the mapped and quantized data set. It assigns the shortest code words to the most frequently occurring output values and thus reduces coding redundancy. The operation, of course, is reversible. Upon completion of the symbol coding step, the input image has been processed to remove each of the three redundancies.

Figure 3.2(a) shows the source encoding process as three successive operations, but all three operations are not necessarily included in every compression system. Recall, for example, that the quantizer must be omitted when error-free compression is desired. In addition, some compression techniques normally are modeled by merging blocks that are physically separate in

Fig. 3.2(a). In the predictive compression systems, for instance, the mapper and quantizer are often represented by a single block, which simultaneously performs both operations.

The source decoder shown in Fig. 3.2(b) contains only two components: a symbol decoder and an inverse mapper. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in Fig. 3.2(b)

The Channel Encoder and Decoder:

The channel encoder and decoder play an important role in the overall encoding-decoding process when the channel of Fig. 3.1 is noisy or prone to error. They are designed to reduce the impact of channel noise by inserting a controlled form of redundancy into the source encoded data. As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of this "controlled redundancy." One of the most useful channel encoding techniques was devised by R. W. Hamming (Hamming [1950]). It is based on appending enough bits to the data being encoded to ensure that some minimum number of bits must change between valid code words. Hamming showed, for example, that if 3 bits of redundancy are added to a 4-bit word, so that the distance between any two valid code words is 3, all single-bit errors can be detected and corrected. (By appending additional bits of redundancy,

multiple-bit errors can be detected and corrected.) The 7-bit Hamming (7, 4) code word h1, h2, h3…., h6, h7 associated with a 4-bit binary number b3b2b1b0 is

$$
\begin{aligned}
h_1 &= b_3 \oplus b_2 \oplus b_0 \qquad & h_3 &= b_3 \\
h_2 &= b_3 \oplus b_1 \oplus b_0 \qquad & h_5 &= b_2 \\
h_4 &= b_2 \oplus b_1 \oplus b_0 \qquad & h_6 &= b_1 \\
& & h_7 &= b_0
\end{aligned}
$$

where denotes the exclusive OR operation. Note that bits h1, h2, and h4 are even- parity bits for

the bit fields b3 b2 b0, b3b1b0, and b2b1b0, respectively. (Recall that a string of binary bits has even parity if the number of bits with a value of 1 is even.) To decode a Hamming encoded result, the channel decoder must check the encoded value for odd parity over the bit fields in which even

parity was previously established. A single-bit error is indicated by a nonzero parity word c4c2c1, where

$$
\begin{aligned}
c_1 &= h_1 \oplus h_3 \oplus h_5 \oplus h_7 \\
c_2 &= h_2 \oplus h_3 \oplus h_6 \oplus h_7 \\
c_4 &= h_4 \oplus h_5 \oplus h_6 \oplus h_7.
\end{aligned}
$$

If a nonzero value is found, the decoder simply complements the code word bit position indicated by the parity word. The decoded binary value is then extracted from the corrected code word as h3h5h6h7.

Variable length codes.

Variable-Length Coding:

The simplest approach to error-free image compression is to reduce only coding redundancy. Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. It can be eliminated by coding the gray levels. To do so requires construction of a variable- length code that assigns the shortest possible code words to the most probable gray levels. Here, we examine several optimal and near optimal techniques for constructing such a code. These techniques are formulated in the language of information theory. In practice, the source symbols may be either the gray levels of an image or the output of a gray-level mapping operation (pixel differences, run lengths, and so on).

Huffman coding:

The most popular technique for removing coding redundancy is due to Huffman (Huffman [1952]). When coding the symbols of an information source individually, Huffman coding yields the smallest possible number of code symbols per source symbol. In terms of the noiseless coding theorem, the resulting code is optimal for a fixed value of n, subject to the constraint that the source symbols be coded one at a time.

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Figure 4.1 illustrates this process for binary coding (K-ary Huffman codes can also be constructed). At the far left, a hypothetical set of source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reduction, the bottom two probabilities,

0.06 and 0.04, are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source are also ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols (at the far right) is reached.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As Fig. 4.2 shows, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and 1 would work just as well). As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrarily appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source is reached. The final code appears at the far left in Fig.4.2. The average length of this code is and the entropy of the source is 2.14 bits/symbol. The resulting Huffman code efficiency is 0.973.

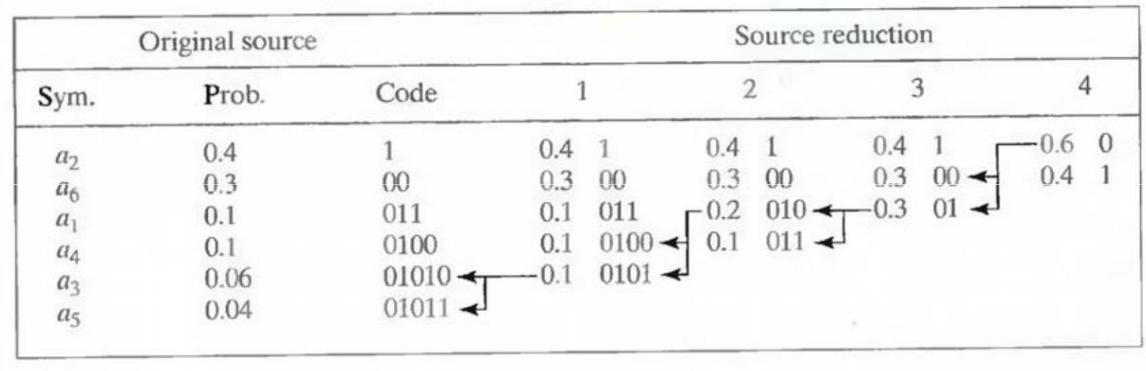


Fig.4.1 Huffman source reductions.

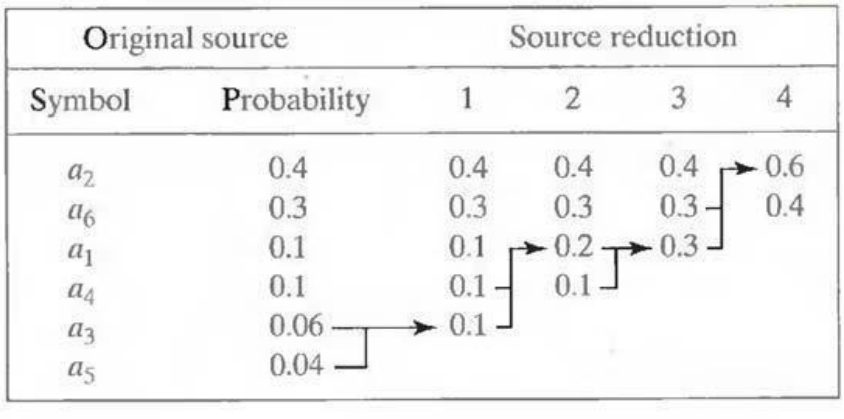


Fig.4.2 Huffman code assignment procedure.

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ bits/symbol}$$

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple lookup table manner. The code itself is an instantaneous uniquely decodable block code. It is called a block code because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous, because each code word in a string of code symbols can be decoded without referencing succeeding symbols. It is uniquely decodable, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code of Fig. 4.2, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a3.

The next valid code is 011, which corresponds to symbol a1. Continuing in this manner reveals the completely decoded message to be a3a1a2a2a 6.

Arithmetic encoding process

Arithmetic coding:

Unlike the variable-length codes described previously, arithmetic coding generates nonblock codes. In arithmetic coding, which can be traced to the work of Elias, a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence. Because the technique does not require, as does Huffman's approach, that each source symbol translate into an integral number of code symbols (that is, that the symbols be coded one at a time), it achieves (but only in theory) the bound established by the noiseless coding theorem.

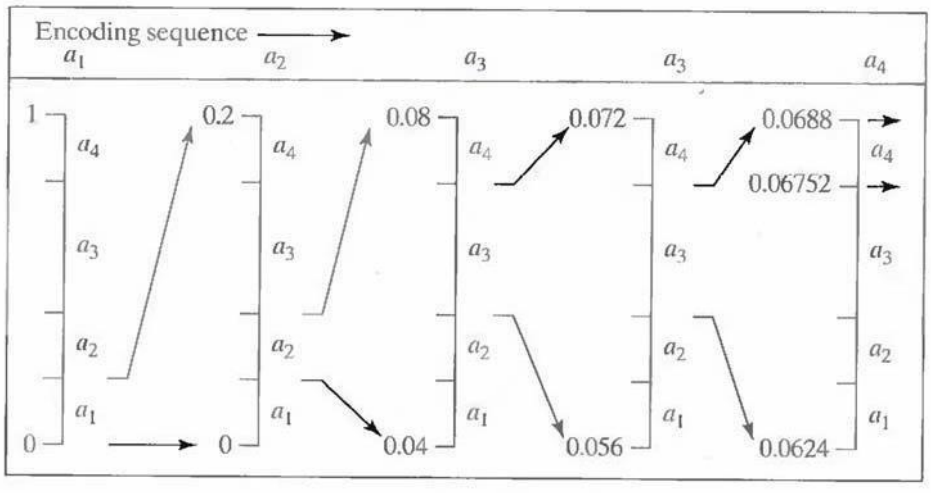


Fig.5.1 Arithmetic coding procedure

Figure 5.1 illustrates the basic arithmetic coding process. Here, a five-symbol sequence or message, a1a2a3a3a4, from a four-symbol source is coded. At the start of the coding process, the message is assumed to occupy the entire half-open interval [0, 1). As Table 5.2 shows, this interval is initially subdivided into four regions based on the probabilities of each source symbol. Symbol ax, for example, is associated with subinterval [0, 0.2). Because it is the first symbol of the message being coded, the message interval is initially narrowed to [0, 0.2). Thus in Fig. 5.1 [0, 0.2) is expanded to the full height of the figure and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the original source symbol probabilities and the process continues with the next message symbol.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

Table 5.1 Arithmetic coding example

In this manner, symbol a2 narrows the subinterval to [0.04, 0.08), a3 further narrows it to [0.056, 0.072), and so on. The final message symbol, which must be reserved as a special end-of-Message indicator, narrows the range to [0.06752, 0.0688). Of course, any number within this subinterval—for example, 0.068—can be used to represent the message.

In the arithmetically coded message of Fig. 5.1, three decimal digits are used to represent the five-symbol message. This translates into 3/5 or 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which is 0.58 decimal digits or 10- ary units/symbol. As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by the noiseless coding theorem.

In practice, two factors cause coding performance to fall short of the bound: (1) the addition of the end-of-message indicator that is needed to separate one message from an-other; and (2) the use of finite precision arithmetic. Practical implementations of arithmetic coding address the latter problem by introducing a scaling strategy and a rounding strategy (Langdon and Rissanen [1981]). The scaling strategy renormalizes each subinterval to the [0, 1) range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being represented accurately.

Bit plane coding method.

Bit-Plane Coding:

An effective technique for reducing an image's interpixel redundancies is to process the image's bit planes individually. The technique, called bit-plane coding, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images and compressing each binary image via one of several well-known binary compression methods.

Bit-plane decomposition:

The gray levels of an m-bit gray-scale image can be represented in the form of the base 2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_1 2^1 + a_0 2^0.$$

Based on this property, a simple method of decomposing the image into a collection of binary images is to separate the m coefficients of the polynomial into m 1-bit bit planes. The zeroth-order

bit plane is generated by collecting the a0 bits of each pixel, while the (m - 1) st-order bit plane

contains the am-1, bits or coefficients. In general, each bit plane is numbered from 0 to m-1 and is constructed by setting its pixels equal to the values of the appropriate bits or polynomial coefficients from each pixel in the original image. The inherent disadvantage of this approach is that small changes in gray level can have a significant impact on the complexity of the bit planes. If a pixel of intensity 127 (01111111) is adjacent to a pixel of intensity 128 (10000000), for instance, every bit plane will contain a corresponding 0 to 1 (or 1 to 0) transition. For example, as the most significant bits of the two binary codes for 127 and 128 are different, bit plane 7 will contain a zero-valued pixel next to a pixel of value 1, creating a 0 to 1 (or 1 to 0) transition at that point.

An alternative decomposition approach (which reduces the effect of small gray-level variations) is to first represent the image by an m-bit Gray code. The m-bit Gray code gm-1... g2g1g0 that corresponds to the polynomial in Eq. above can be computed from

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$
$$g_{m-1} = a_{m-1}.$$

Here, denotes the exclusive OR operation. This code has the unique property that successive code words differ in only one bit position. Thus, small changes in gray level are less likely to affect all m bit planes. For instance, when gray levels 127 and 128 are adjacent, only the 7th bit plane will contain a 0 to 1 transition, because the Gray codes that correspond to 127 and 128 are 11000000 and 01000000, respectively.

Lossless predictive coding.

The error-free compression approach does not require decomposition of an image into a collection of bit planes. The approach, commonly referred to as lossless predictive coding, is based on eliminating the interpixel redundancies of closely spaced pixels by extracting and coding only the new information in each pixel. The new information of a pixel is defined as the difference between the actual and predicted value of that pixel.

Figure 8.1 shows the basic components of a lossless predictive coding system. The system consists of an encoder and a decoder, each containing an identical predictor. As each successive pixel of the input image, denoted fn, is introduced to the encoder, the predictor generates the anticipated value of that pixel based on some number of past inputs. The output of the predictor is then rounded to the nearest integer, denoted f^n and used to form the difference or prediction error which is coded using a variable-length code (by the symbol encoder) to generate the next element of the compressed data stream.

$$e_n = f_n - \hat{f}_n,$$

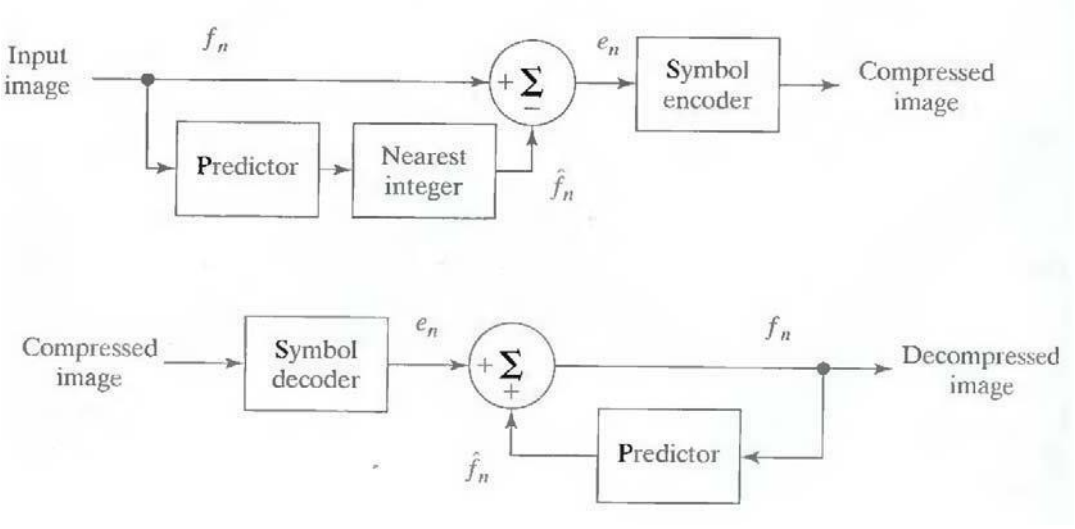


Fig.8.1 A lossless predictive coding model: (a) encoder; (b) decoder

The decoder of Fig. 8.1 (b) reconstructs en from the received variable-length code words and performs the inverse operation

$$f_n = e_n + \hat{f}_n.$$

Various local, global, and adaptive methods can be used to generate f^n. In most cases, however, the prediction is formed by a linear combination of m previous pixels. That is,

$$\hat{f}_n = \text{round}\left[\sum_{i=1}^{m} \alpha_i f_{n-i}\right]$$

where m is the order of the linear predictor, round is a function used to denote the rounding or

nearest integer operation, and the αi, for i = 1,2,..., m are prediction coefficients. In raster scan applications, the subscript n indexes the predictor outputs in accordance with their time of

occurrence. That is, fn, f^n and en in Eqns. above could be replaced with the more explicit notation f (t), f^(t), and e (t), where t represents time. In other cases, n is used as an index on the spatial coordinates and/or frame number (in a time sequence of images) of an image.

In 1-D linear predictive coding, for example, Eq. above can be written as

where each subscripted variable is now expressed explicitly as a function of spatial coordinates x and y. The Eq. indicates that the 1-D linear prediction f(x, y) is a function of the previous pixels on the current line alone. In 2-D predictive coding, the prediction is a function of the previous pixels in a left-to-right, top-to-bottom scan of an image. In the 3-D case, it is based on these pixels and the previous pixels of preceding frames. Equation above cannot be evaluated for the first m pixels of each line, so these pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process. A similar comment applies to the higher-dimensional cases.

Lossy predictive coding.

In this type of coding, we add a quantizer to the lossless predictive model and examine the resulting trade-off between reconstruction accuracy and compression performance. As Fig.9 shows, the quantizer, which absorbs the nearest integer function of the error-free encoder, is inserted between the symbol encoder and the point at which the prediction error is formed. It maps

the prediction error into a limited range of outputs, denoted e^n which establish the amount of compression and distortion associated with lossy predictive coding.

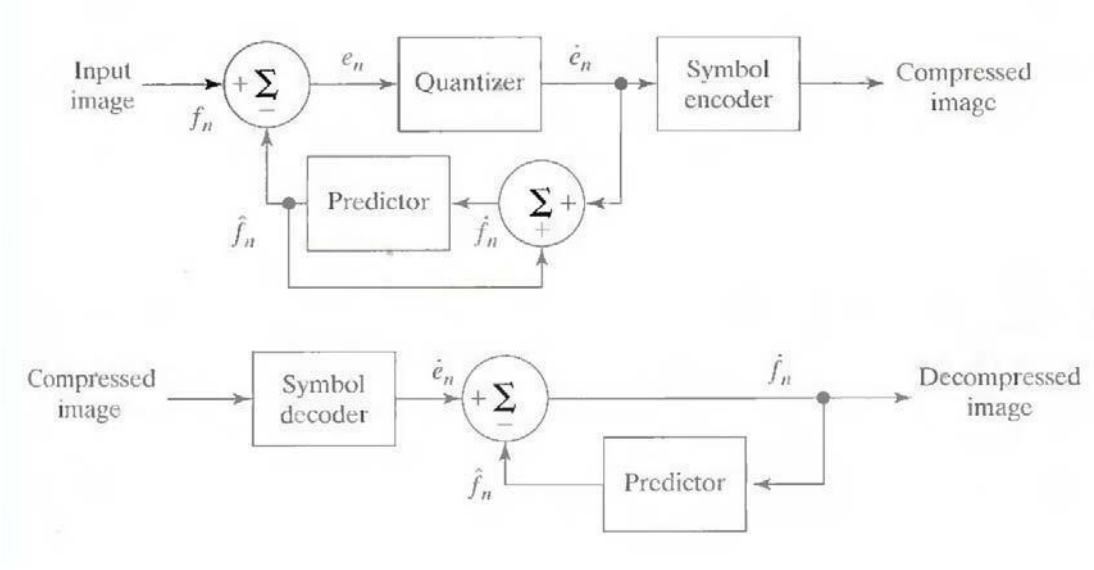


Fig. 9 A lossy predictive coding model: (a) encoder and (b) decoder.

In order to accommodate the insertion of the quantization step, the error-free encoder of figure must be altered so that the predictions generated by the encoder and decoder are equivalent. As Fig.9 (a) shows, this is accomplished by placing the lossy encoder's predictor within a feedback

loop, where its input, denoted f˙n, is generated as a function of past predictions and the corresponding quantized errors. That is,

$$\dot{f}_n = \dot{e}_n + \hat{f}_n$$

This closed loop configuration prevents error buildup at the decoder's output. Note from Fig. 9 (b) that the output of the decoder also is given by the above Eqn.

Optimal predictors:

The optimal predictor used in most predictive coding applications minimizes the encoder's mean- square prediction error

$$E\{e_n^2\} = E\{[f_n - \hat{f}_n]^2\}$$

subject to the constraint that

$$\dot{f}_n = \dot{e}_n + \hat{f}_n \approx e_n + \hat{f}_n = f_n$$

and

$$\hat{f}_n = \sum_{i=1}^{m} \alpha_i f_{n-i}.$$

That is, the optimization criterion is chosen to minimize the mean-square prediction error, the quantization error is assumed to be negligible (e˙n ≈ en), and the prediction is constrained to a linear combination of m previous pixels.1 These restrictions are not essential, but they simplify the analysis considerably and, at the same time, decrease the computational complexity of the predictor. The resulting predictive coding approach is referred to as differential pulse code modulation (DPCM).

Transform Coding

All the predictive coding techniques operate directly on the pixels of an image and thus are spatial domain methods. In this coding, we consider compression techniques that are based on modifying the transform of an image. In transform coding, a reversible, linear transform (such as the Fourier transform) is used to map the image into a set of transform coefficients, which are then quantized and coded. For most natural images, a significant number of the coefficients have small magnitudes and can be coarsely quantized (or discarded entirely) with little image distortion. A variety of transformations, including the discrete Fourier transform (DFT), can be used to transform the image data.

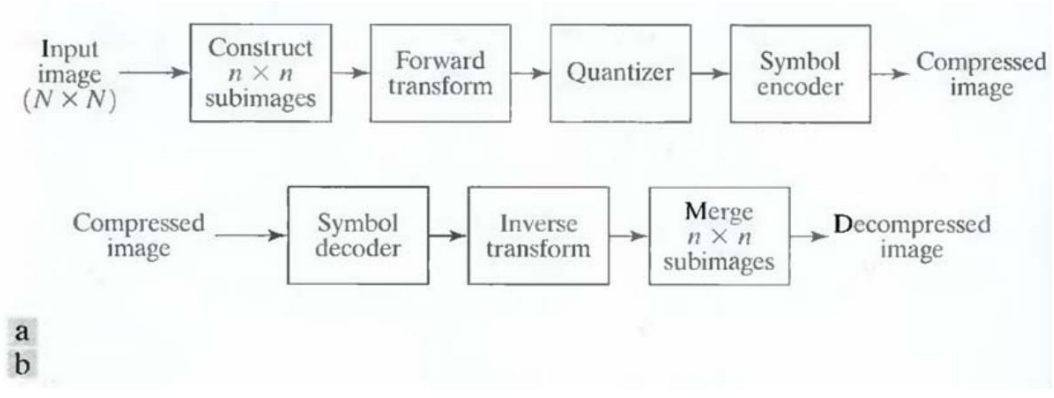


Fig. 10 A transform coding system: (a) encoder; (b) decoder.

Figure 10 shows a typical transform coding system. The decoder implements the inverse sequence of steps (with the exception of the quantization function) of the encoder, which performs four relatively straightforward operations: subimage decomposition, transformation, quantization, and coding. An N X N input image first is subdivided into subimages of size n X n, which are then transformed to generate (N/n) 2 subimage transform arrays, each of size n X n. The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients. The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least information. These coefficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by coding (normally using a variable-length code) the quantized

coefficients. Any or all of the transform encoding steps can be adapted tolocal image content, called adaptive transform coding, or fixed for all subimages, called nonadaptive transform coding.

Wavelet Coding:

The wavelet coding is based on the idea that the coefficients of a transform that decorrelates the pixels of an image can be coded more efficiently than the original pixels themselves. If the transform's basis functions—in this case wavelets—pack most of the important visual information into a small number of coefficients, the remaining coefficients can be quantized coarsely or truncated to zero with little image distortion.

Figure 11 shows a typical wavelet coding system. To encode a 2J X 2J image, an analyzing wavelet, Ψ, and minimum decomposition level, J - P, are selected and used to compute the image's discrete wavelet transform. If the wavelet has a complimentary scaling function ϕ, the fast wavelet transform can be used. In either case, the computed transform converts a large portion of the

original image to horizontal, vertical, and diagonal decomposition coefficients with zero mean and Laplacian-like distributions.

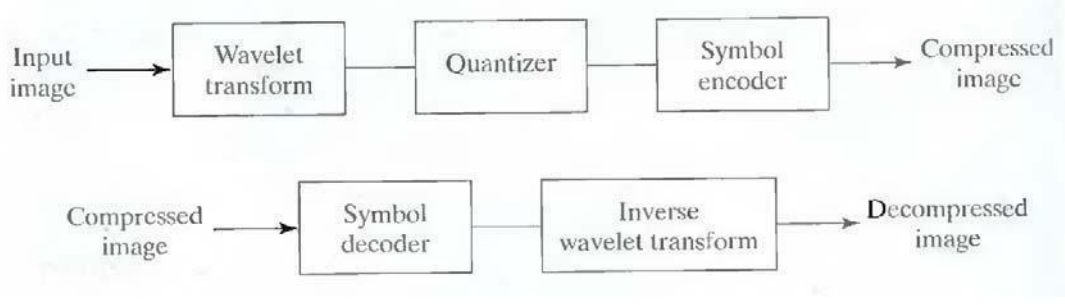


Fig.11 A wavelet coding system: (a) encoder; (b) decoder.

Since many of the computed coefficients carry little visual information, they can be quantized and coded to minimize intercoefficient and coding redundancy. Moreover, the quantization can be adapted to exploit any positional correlation across the P decomposition levels. One or more of the lossless coding methods, including run-length, Huffman, arithmetic, and bit-plane coding, can be incorporated into the final symbol coding step. Decoding is accomplished by inverting the encoding operations—with the exception of quantization, which cannot be reversed exactly.

The principal difference between the wavelet-based system and the transform coding system is the omission of the transform coder's subimage processing stages.

Because wavelet transforms are both computationally efficient and inherently local (i.e., their basis functions are limited in duration), subdivision of the original image is unnecessary

Data Compression is a technique in which the size of data is reduced without loss of information. **Lossy compression** and **Lossless compression** are the categories of data compression method.

The main difference between the two compression techniques (lossy compression and Lossless compression) is that, The lossy compression technique does not restored the data in its original form, after decompression on the other hand lossless compression restores and rebuilt the data in its original form, after decompression.

Difference between Lossy Compression and Lossless Compression:

S.NO	Lossy Compression	Lossless Compression
1.	Lossy compression is the method which eliminate the data which is not noticeable.	While Lossless Compression does not eliminate the data which is not noticeable.
2.	In Lossy compression, A file does not restore or rebuilt in its original form.	While in Lossless Compression, A file can be restored in its original form.
3.	In Lossy compression, Data's quality is compromised.	But Lossless Compression does not compromise the data's quality.
4.	Lossy compression reduces the size of data.	But Lossless Compression does not reduce the size of data.
5.	Algorithms used in **Lossy** compression are: Transform coding, Discrete Cosine Transform, Discrete Wavelet Transform, fractal compression etc.	Algorithms used in **Lossless** compression are: Run Length Encoding, Lempel-Ziv-Welch, Huffman Coding, Arithmetic encoding etc.
6.	Lossy compression is used in Images, audio, video.	Lossless Compression is used in Text, images, sound.
7.	Lossy compression has more data-holding capacity.	Lossless Compression has less data-holding capacity than Lossy compression technique.
8.	Lossy compression is also termed as irreversible compression.	Lossless Compression is also termed as reversible compression.