Image Compression – Image compression is used to make image file size smaller so that they take up less space on computer and can be shared faster over the internet. The goal is to reduce the file size without changing how the image looks. In computers images are made up of tiny blocks called **pixels** and each pixel stores information intensity value (colour). When an image is very large it contains millions of these numbers which means the file size becomes big.

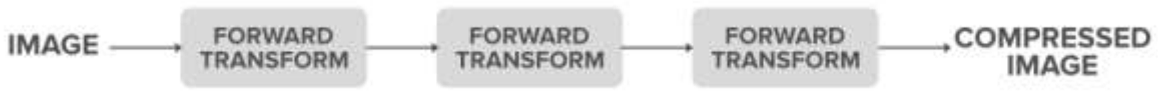
Steps in Image Compression

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**

So to store a 3-sec video we need so many bits which can be problematic as it takes up so much storage. Thus image compression plays an important role in this.

Step 1: Transforming The Image

- This is the first step where we transform the image data to a new form that is easier to compress. It rearrange the image information so that the important parts are more noticeable and the unnecessary parts can be reduced.
- So we use **transform** like the Discrete Cosine Transform (DCT) 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 and T(f(t)) is called the transform of the function.**



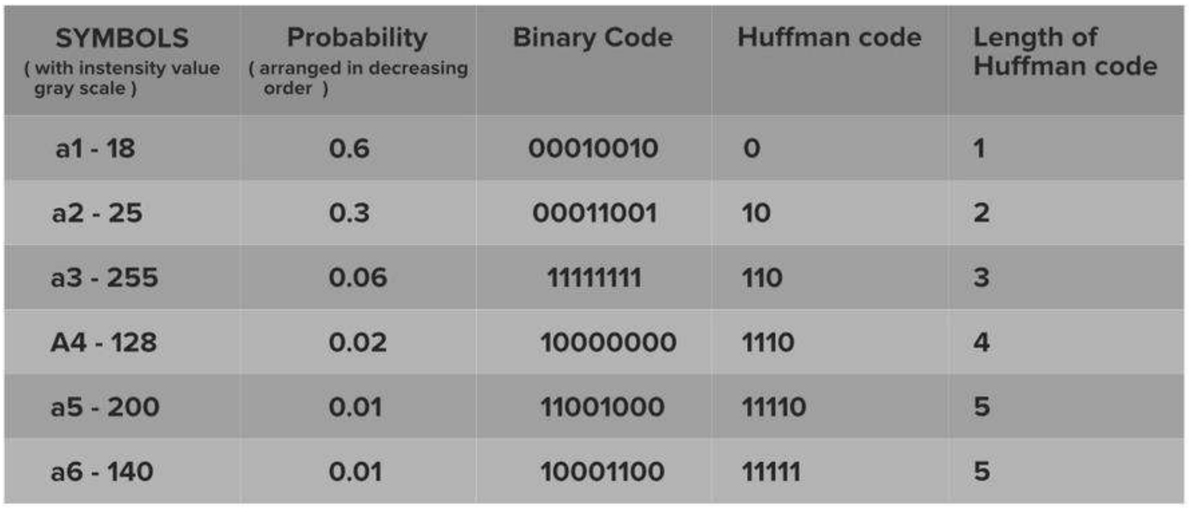
As shown in the diagram, the original image goes through a series of **Forward Transforms** which help to reduce the amount of data while keeping the core image content. At the end of this process we get a **Compressed Image** that looks almost the same but takes up much less space.

Step 2: Quantization

- Once we have transformed the image we get many numbers. Some of these numbers are very small and don't contribute much to how the image looks. In quantization we **reduce the number of different values** by rounding them off to the nearest level.
- This helps to reduce the amount of data even if it means we lose a tiny bit of image detail this is called **lossy compression**. For example if two nearby pixel values are 17 and 19 we can round both of them to 18. This saves space because we treat similar values as the same.

Step 3: Symbol Encoding

After quantization we are left with few values and some values like 0s might repeat many times. In this step we use **smart coding methods** to save more space. One common method is **Huffman coding**. It gives **shorter codes** to values that occur more often and **longer codes** to less common values. This way we use few bits for frequently used data.



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

Huffman Coding Scheme

Another method is **Run-Length Encoding** which is useful when the same value repeats a lot like a row of 0s. Instead of storing each zero separately we store it like this: (0, 12) meaning "zero repeats 12 times."
By using these three steps we can reduce **the size of an image** without losing too much visual quality. This makes it easier to store, share or stream images and videos efficiently.

Coding redundancy - Redundancy refers to **"storing extra information to represent a quantity of information".** So that is the redundancy of data now apply this concept on digital images we know that computer store the images in pixel values so sometimes image has duplicate pixel values or maybe if we remove some of the pixel values they don't affect the information of an actual image. Data Redundancy is one of the fundamental component of **Data Compression**.

Types of Redundancy in the context of neighboring pixels-: Broadly we have three types of redundancy in images pixels

1. **Spatial Redundancy-:** In spatial redundancy there is a correlation between the neighboring pixel values.
2. **Spectral Redundancy-:** In spectral redundancy there is a correlation between different color planes or spectral bands.
3. **Temporal Redundancy**-: In temporal redundancy there is a correlation between adjacent frames in the sequence of image.

- Codes designed based on histogram do not exploit the correlation between pixels.
• These correlations result from the structural or geometrical relationships between the objects in the image.
• The value of any pixel is highly correlated to its neighbors and the information carried by individual pixels is small.

Interpixel redundancy - is a fundamental concept in digital image processing and plays a significant role in the field of image compression. It refers to the redundancy that exists due to the strong correlation between neighboring pixels in most natural and man-made images. In typical images, especially photographs or scanned documents, the intensity or color of one pixel is often very similar to the intensity or color of its adjacent pixels. This occurs because real-world scenes do not usually change dramatically from one point to the next; instead, they exhibit gradual transitions in brightness and color. As a result, much of the data in an image is redundant in the sense that once you know the value of a pixel, you can reasonably predict the value of nearby pixels with a high degree of accuracy.

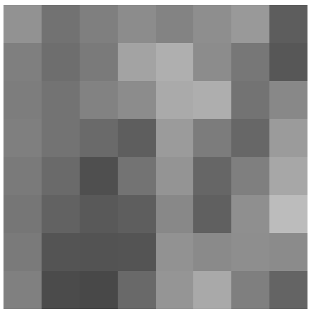
This spatial redundancy between pixels means that an image contains more data than is strictly necessary to convey its visual content. For instance, in a clear blue sky or a smooth wall, hundreds or thousands of neighboring pixels may have nearly identical values. Storing each of these pixel values individually results in the consumption of excessive memory and storage space. Interpixel redundancy can be exploited by image compression algorithms to reduce the size of the image data without significantly affecting its perceived quality. Techniques such as **predictive coding** and **transform coding** are designed specifically to take advantage of this type of redundancy. Predictive coding works by estimating the value of a pixel based on its neighbors and then encoding the difference between the predicted and actual value, which typically requires fewer bits. Transform coding methods, such as the **Discrete Cosine Transform (DCT)** used in JPEG compression, convert spatial domain information into frequency components, many of which can be discarded or quantized without noticeably degrading the image.

The removal or reduction of interpixel redundancy is crucial for efficient storage, transmission, and processing of images, especially in applications such as web browsing, video streaming, medical imaging, and satellite imaging. By minimizing the amount of data needed to represent an image, we can achieve faster data transfer rates, lower bandwidth usage, and more efficient storage management. Importantly, the goal is to retain as much of the visually significant information as possible while discarding or compressing redundant pixel data. Interpixel redundancy is just one of several types of redundancies in image processing, alongside others like **coding redundancy**, **psychovisual redundancy**, and **temporal redundancy** (in video processing), but it remains one of the most directly exploitable in spatial image compression.

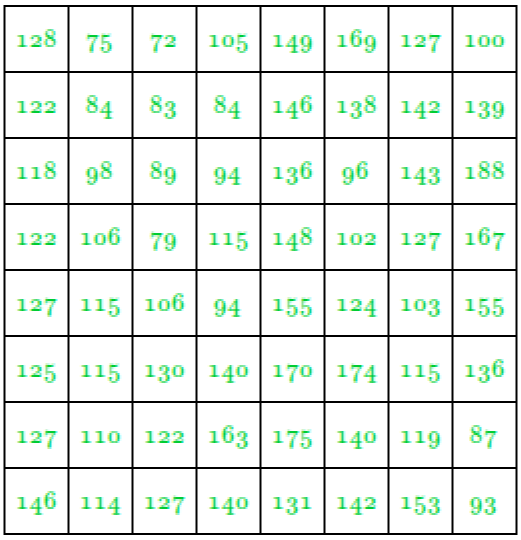
Huffman coding - is one of the basic compression methods, that have proven useful in image and video compression standards. When applying Huffman encoding technique on an Image, the source symbols can be either pixel intensities of the Image, or the output of an intensity mapping function.
The first step of Huffman coding technique is to reduce the input image to a ordered histogram, where the probability of occurrence of a certain pixel intensity value is as

```
prob_pixel = numpix/totalnum
```

where
numpix
is the number of occurrence of a pixel with a certain intensity value and
totalnum
is the total number of pixels in the input Image. Let us take a 8 X 8 Image



The pixel intensity values are :



128	75	72	105	149	169	127	100
122	84	83	84	146	138	142	139
118	98	89	94	136	96	143	188
122	106	79	115	148	102	127	167
127	115	106	94	155	124	103	155
125	115	130	140	170	174	115	136
127	110	122	163	175	140	119	87
146	114	127	140	131	142	153	93

This image contains 46 distinct pixel intensity values, hence we will have 46 unique Huffman code words. It is evident that, not all pixel intensity values may be present in the image and hence will not have non-zero probability of occurrence. From here on, the pixel intensity values in the input Image will be addressed as leaf nodes. Now, there are 2 essential steps to build a Huffman Tree :

1. **Build a Huffman Tree :**
2.
    1. Combine the two lowest probability leaf nodes into a new node.
    2. Replace the two leaf nodes by the new node and sort the nodes according to the new probability values.
    3. Continue the steps (a) and (b) until we get a single node with probability value 1.0. We will call this node as **root**
2. **Backtrack from the root, assigning ‘0’ or ‘1’ to each intermediate node, till we reach the leaf nodes**

Arithmetic coding -

Arithmetic coding is a type of entropy encoding utilized in lossless data compression. Ordinarily, a string of characters, for example, the words “hey” is represented for utilizing a fixed number of bits per character.
In the most straightforward case, the probability of every symbol occurring is equivalent. For instance, think about a set of three symbols, A, B, and C, each similarly prone to happen. Straightforward block encoding would require 2 bits for every symbol, which is inefficient as one of the bit varieties is rarely utilized. In other words, A = 00, B = 01, and C = 10, however, 11 is unused. An increasingly productive arrangement is to speak to a succession of these three symbols as a rational number in base 3 where every digit speaks to a symbol. For instance, the arrangement “ABBCAB” could become 0.011201. In arithmetic coding as an incentive in the stretch [0, 1). The subsequent stage is to encode this ternary number utilizing a fixed-guide paired number of adequate exactness toward recuperating it, for example, 0.00101100102 — this is just 10 bits. This is achievable for long arrangements in light of the fact that there are productive, set up calculations for changing over the base of subjectively exact numbers.

When all is said and done, arithmetic coders can deliver close ideal output for some random arrangement of symbols and probabilities (the ideal value is – log2P bits for every symbol of likelihood P). Compression algorithms that utilize arithmetic coding go by deciding a structure of the data – fundamentally an expectation of what examples will be found in the symbols of the message. The more precise this prediction is, the closer to ideal the output will be.

When all is said and done, each progression of the encoding procedure, aside from the absolute last, is the equivalent; the encoder has fundamentally only three bits of information to consider: The following symbol that should be encoded. The current span (at the very beginning of the encoding procedure, the stretch is set to [0,1], yet that will change) The probabilities the model allows to every one of the different symbols that are conceivable at this stage (as referenced prior, higher-request or versatile models imply that these probabilities are not really the equivalent in each progression.) The encoder isolates the current span into sub-spans, each speaking to a small amount of the current span relative to the likelihood of that symbol in the current setting. Whichever stretch relates to the real symbol that is close to being encoded turns into the span utilized in the subsequent stage. At the point when the sum total of what symbols have been encoded, the subsequent span unambiguously recognizes the arrangement of symbols that delivered it. Any individual who has a similar last span and model that is being utilized can remake the symbol succession that more likely than not entered the encoder to bring about that last stretch. It isn’t important to transmit the last stretch, in any case; it is just important to transmit one division that exists in that span. Specifically, it is just important to transmit enough digits (in whatever base) of the part so all divisions that start with those digits fall into the last stretch; this will ensure that the subsequent code is a prefix code.

Lossy Compression-
Lossy Compression reduces file size by permanently removing some of the original data. It’s commonly used when a file can afford to lose some data or if storage space needs to be significantly freed up.

Advantages of Lossy Compression

- **Smaller File Sizes**: Lossy compression significantly reduces file sizes, making it ideal for web use and faster loading times.

- **Widely Supported**: Many tools and software support lossy formats (e.g., JPEG for images, MP3 for audio).
- **Efficient for Multimedia**: Effective for compressing multimedia files without noticeable quality loss.

Disadvantages of Lossy Compression

- **Quality Degradation**: Due to data removal, lossy files may exhibit reduced quality.
- **Not Suitable for Critical Data**: Inappropriate for situations where data integrity is crucial.

Lossless compression reduces file size by removing unnecessary metadata without any discernible loss in picture quality. The original data can be perfectly reconstructed after decompression.

Advantages of Lossless Compression

- **No Quality Loss**: Lossless compression maintains original quality during compression and decompression.
- **Suitable for Text and Archives**: Ideal for text-based files, software installations, and backups.
- **Minor File Size Reduction**: Reduces file size without compromising quality significantly.

Disadvantages of Lossless Compression

- **Larger Compressed Files**: Compared to lossy formats they compressed larger files.
- **Less Efficient for Multimedia**: Not as effective for multimedia files

Difference between Lossy Compression and Lossless Compression

Lossy Compression	Lossless Compression
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.
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.
In Lossy compression, Data’s quality is compromised.	But Lossless Compression does not compromise the data’s quality.
Lossy compression reduces the size of data.	But Lossless Compression does not reduce the size of data.

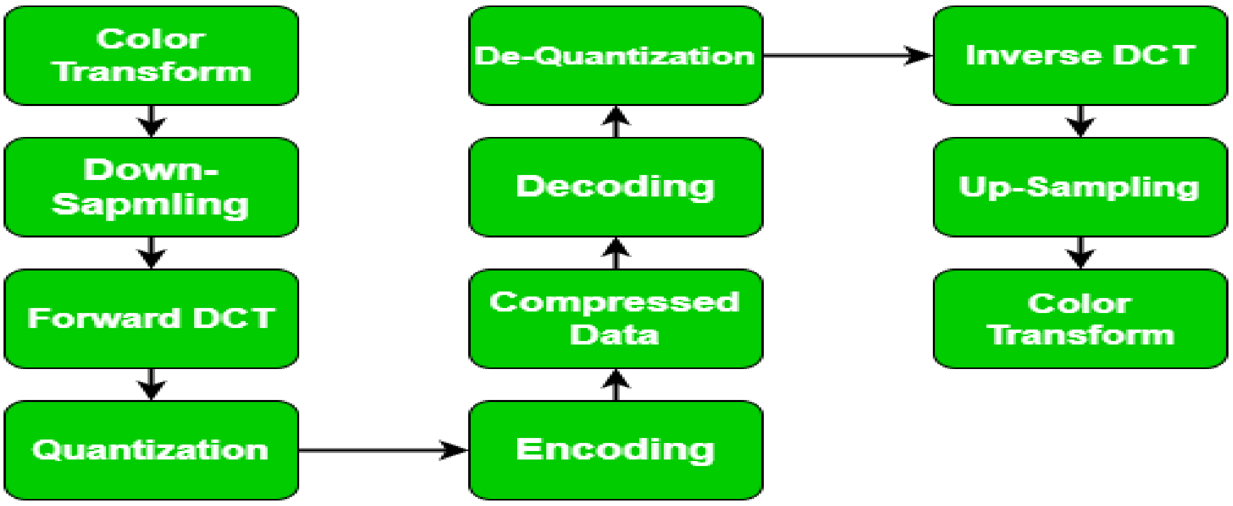
Lossy Compression	Lossless Compression
Lossy compression has more data-holding capacity.	Lossless Compression has less data-holding capacity than Lossy compression technique.
Lossy compression is also termed as irreversible compression.	Lossless Compression is also termed as reversible compression.

JPEG compression-

JPEG stands for Joint Photographic Experts Group. We perform such type of compression to reduce the size of the file without damaging its quality. By reducing the size we can store it in a huge amount which was not possible earlier. Reducing the size of images will also improve the efficiency of the system as it will give less load on it.

Process Of JPEG Compression :

Firstly, we convert the R, G, B color format to Y, Cb, Cr format. Some colors are more sensitive to human eyes and thus are high-frequency colors. Some colors of chromium compounds like Cb and Cr are less sensitive to human eyes thus can be ignored. Then we reduce the size of pixels in downsampling. We divide our image into 8*8 pixels and perform forward DCT(Direct Cosine Transformation). Then we perform quantization using quantum tables and we compress our data using various encoding methods like run-length encoding and Huffman encoding.
In the second stage, we decompress our data, It involves decoding where we decode our data, and we again de-quantize our data by referring to the quantization table. Then we perform Inverse DCT and upsampling to convert it into original pixels and finally, color transformation takes place to convert the image into its original color format.



Algorithm of JPEG Data Compression :

1. **Splitting –**
   We split our image into the blocks of 8*8 blocks. It forms 64 blocks in which each block is referred to as 1 pixel.

2. **Color Space Transform –**
   In this phase, we convert R, G, B to Y, Cb, Cr model. Here Y is for brightness, Cb is color blueness and Cr stands for Color redness. We transform it into chromium colors as these are less sensitive to human eyes thus can be removed.

3. **Apply DCT –**
   We apply Direct cosine transform on each block. The discrete cosine transform (DCT) represents an image as a sum of sinusoids of varying magnitudes and frequencies.

4. **Quantization –**
   In the Quantization process, we quantize our data using the quantization table.

5. **Serialization –**
   In serialization, we perform the zig-zag scanning pattern to exploit redundancy.

6. **Vectoring –**
   We apply DPCM (differential pulse code modeling) on DC elements. DC elements are used to define the strength of colors.

7. **Encoding –**
   In the last stage, we apply to encode either run-length encoding or Huffman encoding. The main aim is to convert the image into text and by applying any encoding we convert it into binary form (0, 1) to compress the data.