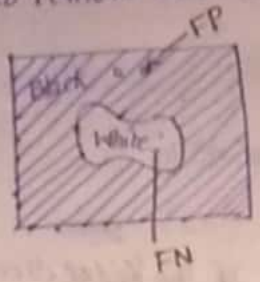
## Morphological Image Processing :-

How to remove black dots and white dots?




This is one kind of classification algorithm.

Classification of processes :-

1) True positive : Correctly Identified
2) True Negative : Correctly Rejected
3) False Positive : Incorrectly Identified
4) False Negative : Incorrectly Rejected

Fire Alarm Example

There is a Fire and Alarm rangs — True positive

There is no Fire and alarm does not rang — True Negative

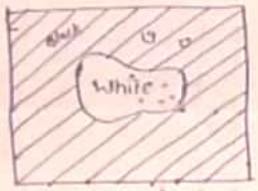
There is no Fire and Alarm rangs — False Positive

There is Fire and Alarm does not rang — False Negative

False +ve & False -ve are not in classification problem

Operates on — threshold image (BIW)

Ideal ideal the combination of True Positives and True Negatives must be 100 then only our system working correctly




Here we have to convert black dots into white and white into black dots.

How to do ?...

→ Set of i/p pixels produces set of o/p pixels.

→ Key element — structuring element { Simple binary array  (or)  }
Operation { Erosion, opening, closing, dilation }

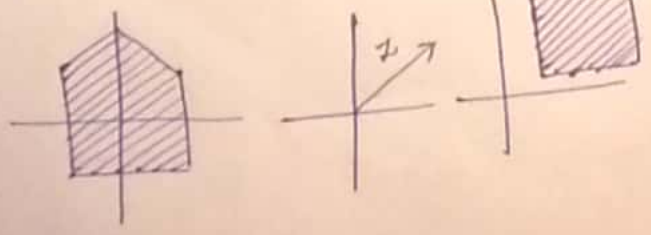
→ A small pixel template that helps to produce the new image from the old one

→ Simple operations of set B

$B_z$ = Translation of B by a vector z

$\{c/c = b+z; b \in B\}$

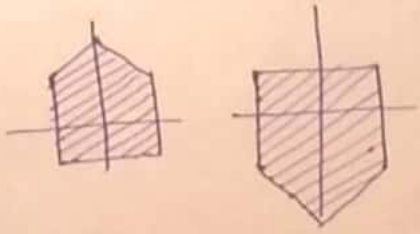
$\{(x+z_1, y+z_1)/(x,y) \in B\}$




$\hat{B}$ : Reflection of B

$\{c/c = -b, b \in B\}$

$\{(-x,-y)/(x,y) \in B\}$



**Erosion:-**

Some operation on set B such that $B_z$ is subset of A

$$A \ominus B = \{z \mid B_z \subseteq A\}$$

(structuring element)

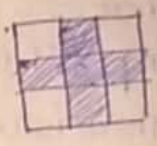
set of structure element points fits fully inside A is called "Erosion."

Erosion element and set of object.

→ If we do erosion operation on an object then the size of the object decreases.

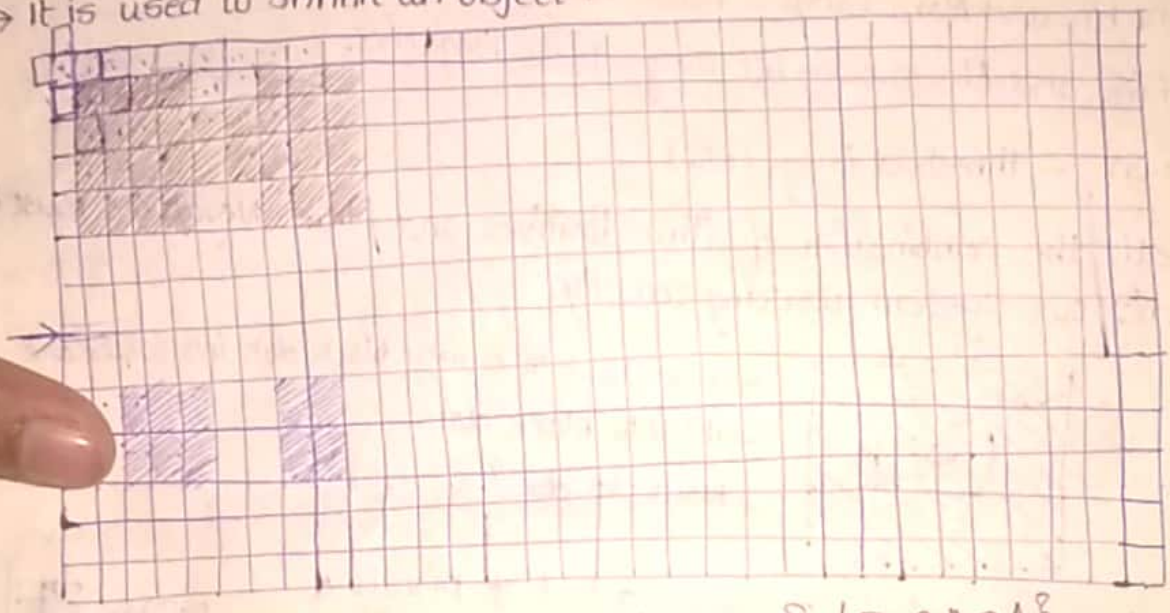
→ Erosion removes thin lines and isolated dots.

$$A \ominus B \subseteq A$$



→ This operation directly depends type of structuring element.

→ It is used to shrink an object.



**Dilation:-** It is reverse of Erosion. $A \oplus B = \{z \mid B_z \cap A \subseteq A\}$

If atleast one element in A

Dilation operation is used to expand an object.

If it has any overlap with

Find pixels such that shifted structuring element has any overlap with the original set.

It increases size of the object. It connects two isolated objects.

This operation is also always depends on type of structuring element.

Thin lines becomes thicker.

The operation that does not change the size and bridge the gaps.

They are:-

(i). Morphological opening

It is defined as to perform erosion and then dilation.

$$A \cdot B = (A \ominus B) \oplus B$$

Erode, then Dilate

- It breaks the narrow bridges and eliminates the structures.

(ii). Morphological closing

Here we first dilate the given object and then erode the object with the same structuring element.

$$A \cdot B = (A \oplus B) \ominus B$$

- Fuses the narrow break points (bridges) and eliminates small holes.

- In Morphological opening, thin lines are removed whereas in morphological closing thin lines are not removed.

```
cv.getStructuringElement(cv.MORPH_RECT, (5,5))
                        (cv.MORPH_CROSS_LINES, (3,3))
                        (cv.MORPH_BLACKHAT, (5,5))
                        (cv.MORPH_ELLIPSE, (5,5))
```

RECT

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

CROSS LINES

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
cv.imshow('Original', img)
kernel = np.ones((5,5), np.uint8)
erosion = cv.erode(img, kernel, iteration=1)
dilation = cv.dilate(img, kernel, iteration=1)
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

# Color Image Processing

Colour is not just digital but it involves physics and the human vision system.
- Cons are faster in bright and is responsible for identifying colors in morning.
- Rods are responsible for identifying in night

Colour is a spectral power distribution.

There are 3 types of cones. They are:

1) S (Small)
2) M (Medium)
3) L (Large)

Red light sensitive cones – 65%
Green light sensitive cones – 33%
Blue light sensitive cones – 2%

- Each type cone is responsible for a particular color.
- There are 6 – 7 million cones.

Primary and Secondary colors:

- Newspapers technology uses secondary color

CMYK (Cyan, magenta, yellow, black)

Red + Green = Yellow
Red + Blue = Magenta
Green + Blue = Cyan

Trichromatic Theory:-

R for L
G for M
B for S

$$C_i = \int_{-\infty}^{\infty} R_i(\lambda) P$$

How many colors can we see?

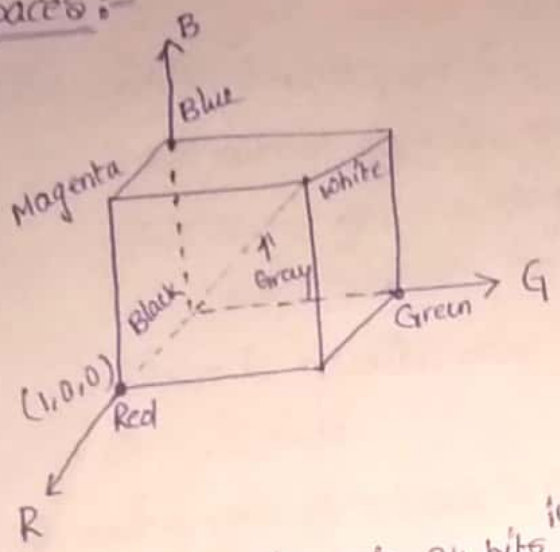
Half toning
Dithering

Some linear combinations of points given new colors

Saturation means the amount of a particular color

Image Inpainting

## Color Spaces :-




The total no. of colours in 24 bits image $= (2^8)^3 = 2^{24} = 16,777,216$ (> 16 million)

OpenCV stores the image in BGR format not in RGB format.

## Advantages :-

The problem with RGB colorspace is -

- We cannot quantize the colors
- We cannot get the range of a particular color completely (we cannot quantize the colors)
- There is no difference between chromatic and Achromatic colors in RGB.

CMYK- We can't get complete black in CMY

- It is extensively used in printers, printer mathines.

~~Converting~~ :

3 quantities are going to calculate are:

1) Hue
2) Saturation
3) Value

→ Hue & Saturation describe colour. These two are responsible for color

1). Hue:-

It is the dominant wavelength or angle made with the 0 present at particular one. It is dominant.

| | |
|---|---|
| Red - 0° | Cyan - 180° |
| Yellow - 60° | Blue - 240° |
| Green - 120° | Megenta = 300° |

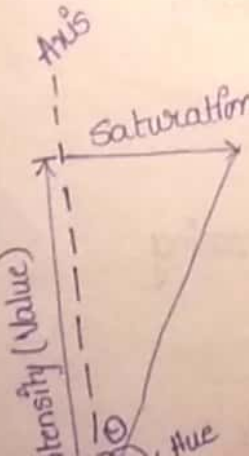
2). Saturation:-

* the purity of the color. It is distance blw centre to Red

3). Value(V):-

Intensity axis

Perpendicular distance of particular color from the axis.



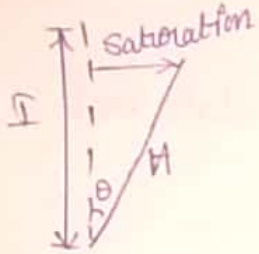
-These saturated colors have lightness 0.5 in HSL, while in HSV -they have value 1.

-The complete saturated color has 0.5 lightness in HSL and 1 in HSV.

→ Hue & Saturation fixed -then intensity decreases then darkness of the image increases.

→ Fixed Hue & Intensity -then if the Saturation decreases then -the purity of the image decreases.

⇒ Fixed Intensity & Saturation -then if the Hue increases then -the color changes.




Given RGB, how to convert RGB into HSI

Converting RGB to HSV:-

$$V = \frac{1}{3}(R+G+B)$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]$$

$$H = \begin{cases} \theta & B \le G \\ 360-\theta & B > G \end{cases}$$

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}$$

Color Image Processing

1) Full color Image Processing

(i)

(ii)

2) Pseudo color image processing

→ Individually apply on each plane i.e., R, G & B.

→ Vector

HSV Range

Since we have only 8 bits to store values

Hue - 0 - 360 ← 0 - 180

Saturation - 0 - 255

Value - 0 - 255

# Pseudo Coloring

Hardly we can identify only 2 dozens of grayscale levels.

$f(x,y)$

$g_R(x,y) = 0.66 \times f(x,y)$

$g_G(x,y) = 0.22 \times f(x,y)$

$g_B(x,y) = \boxed{\phantom{0}} f(x,y)$

Harris

Eastman Colors ??

In old movies

There are 24 frames in 1 second

Coming Topic.... Uniform Color Space

— Correct name $L^* a^* b^*$

+a for Red

−a for

+b for Yellow

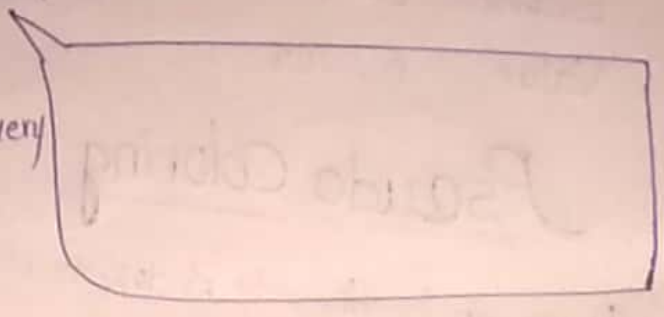
−b for Blue

# Image Compression

There are 2 compression techniques. They are:

1) Loss less compression
2) Lossy compression

Why do we need compression? ....



Zip file – loss less compression

+Lossy compression is used and very good whenever the human vision unable to see the change

→ Jpg

PPM, PGM, BMP } No compression

GIF, TIFF } loss less & uses LZ coding

PGM — 8 bits for grayscale picture

8+8+8=24 bits for color image

400×600

24DKY24

JPEG – Joint Photographic Expert Group – Lossy compression & uses DCT

Compression Ratio:-

Compression Ratio (CR) = $\frac{\text{No. of bits per pixel before}}{\text{No. of bits per pixel after}} > 1$

for real world images, if we go with loss less compression then

$$2 < CR < 10$$

It depends on histogram of the image.

Probability Density function: (PDF)

$$P_i = P(I(x,y)=i) = \frac{\text{No. of pixels having } I(x,y)=i}{\text{Total No. of pixels}}$$

Average no. of bits per pixel = $L_{avg} = \sum_{i=1}^{L} P_i * b_i$

In Uniform coding technique, No. of bits for pixel is always 8 bits

∴ So, Average # of bits, $L_{AVG}$ = 8 bits

We use less no. of bits for high probability pixels value.

For less probability pixel value, it uses more no. of bits.

Ques 1:-

| Level | Probability ($P_i$) | Uniform Coding | Variable length code |
|---|---|---|---|
| 0 | 0.19 | 000 | 00 |
| 1 | 0.25 | 001 | 11 |
| 2 | 0.21 | 010 | 01 |
| 3 | 0.16 | 011 | 101 |
| 4 | 0.08 | 100 | 1001 |
| 5 | 0.06 | 101 | 10001 |
| 6 | 0.03 | 110 | 100001 |
| 7 | 0.02 | 111 | 100000 |

0.19×2
0.25×2
0.
0.38
0.5
0.42
0.48
0.32
0.30
0.18
0.3(2
2.70

$$L_{avg} = \sum_{i=0}^{7} P_i * b_i$$

$$= 0.19*2 + 0.25*2 + 0.21*2 + 0.16*3 + 0.08*4 + 0.06*5 + 0.03*6 + 0.02*6$$

$$= 2.7$$

Compression Ration (CR) $= \frac{3}{2.7} = 1.1$

Entropy :-

$$\text{Entropy} = -\sum_{i=1}^{N} P_i \log P_i$$

If we use uniform distribution then entropy will be

$$\text{Entropy} = -\sum_{i=1}^{N} \frac{1}{N} \log\left(\frac{1}{N}\right) = -N\left[\frac{1}{N}\log\frac{1}{N}\right] = -1 \log N^{-1}$$

$\boxed{\text{Entropy} = \log N}$ [worst case]

Best case: only 1 gray level (delta) $\Rightarrow$ Entropy $= -[1 \log 1] = 0$ $\boxed{\text{Entropy} = 0}$ [Best case]

/* Entropy would be avg. no. of bits */

Entropy is the lowest no. of average bits per symbol that can be used to code the distribution.

The technique which results average bits per pixel which is similar to entropy is "Huffman Coding".

# Huffman Coding :- (Minimum Lav)

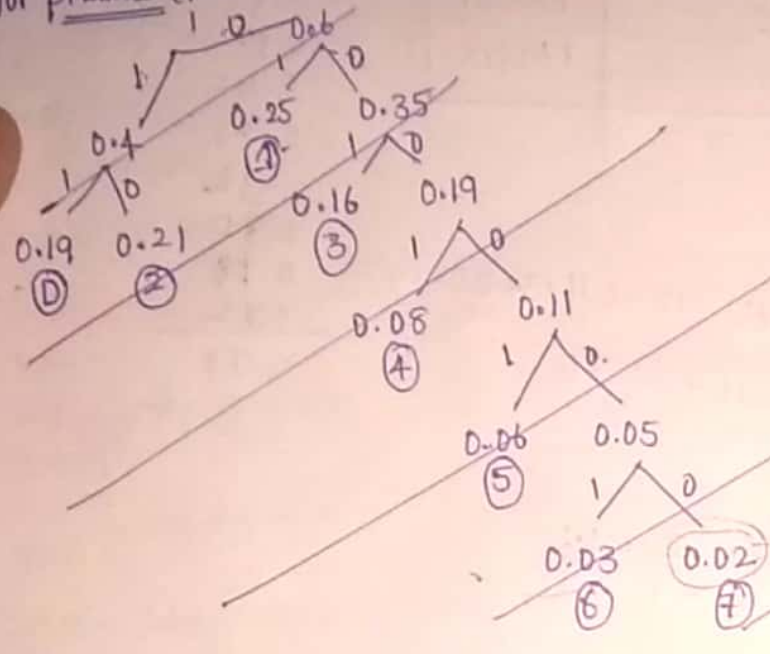
i). Arrange symbols in decreasing $P_i$

ii). Merge the 2 nodes with lowest $P_i$

iii). Assign 0/1 to top/Bottom branch

iv). Read code from Root to leaf

for prblm1 (previous example problem)




level

0 — 11

1 — 01

2 — 10

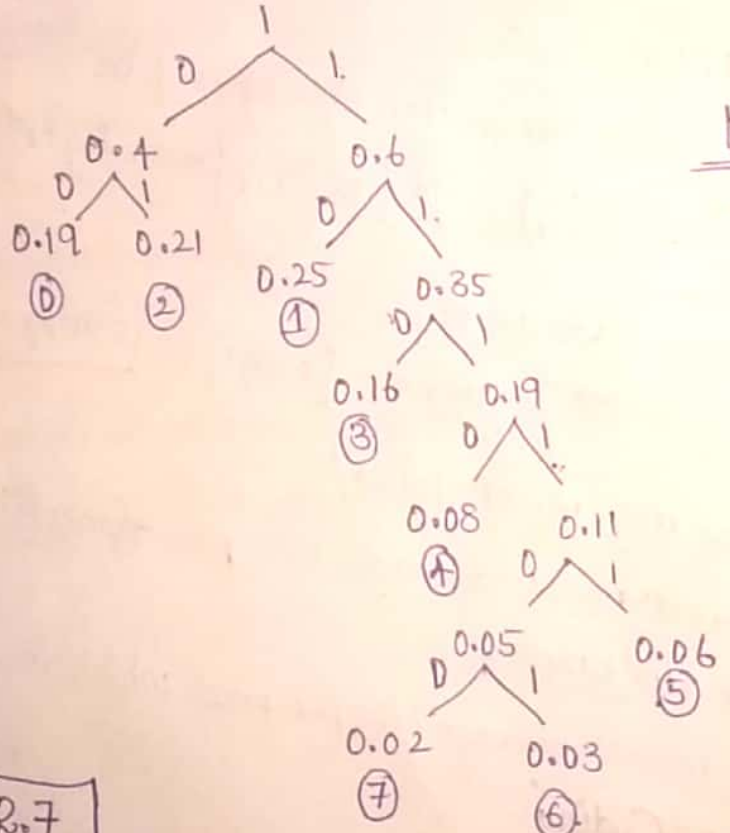
3 — 001

4 — 0001

5 — 00001

6 — 000001

7 — 000000




level

0 — 00

1 — 10

2 — 01

3 — 110

4 — 1110

5 — 11111

6 — 111101

7 — 111100

Lav = 2.7

Huffman code is prefix code

0.38 +
0.5

| Code A | Code B |
|---|---|
| 1 - 10 | 1 - 0 |
| 2 - 00 | 2 - 10 |
| 3 - 11 | 3 - 110 |
| 4 - 110 | 4 - 111 |

Given two coding techniques where code B is prefix code and code A is not.

Encode : 1 2 3 4 3 2 1

Code A : 10 00 11 110 11 00 10

Code B : 0 10 110 111 110 100

Using prefix code, we can easily decode the data without ambiguity.

Huffman coding is prefix code in which no prefixes are not matched and using this we can easily decode.

## Truncated Huffman Coding :-

Without probabilities, we cannot do Huffman coding (lossless technique).

If we have stream of data i.e., the probabilities are unknown, then how can calculate the average no. of bits per pixel ? ....

Coming topic ... [LZ Coding & LZ Variance]

## Lempel-Ziv Compression Techniques :-

LZ77 & LZ78 encoding algorithms.

Lossless compression techniques

1) static 2) Adaptive 3)

# LZ78 algorithm

R141113

## LZ78 Compression algorithm

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| A | B | BC | BCA | BA | BCA | BCAAB |

| Output | Index | |
|---|---|---|
| (0, A) | 1 | A |
| (0, B) | 2 | B |
| (2, C) | 3 | BC |
| (3, A) | 4 | BCA |
| (2, A) | 5 | BA |
| (4, A) | 6 | BCAA |
| (6, B) | 7 | BCAAB |

```
prefix ← empty; Dictionary ← empty; DictionaryIndex ← 1
while (characterstream is not empty)
{
    char ← next character from characterstream
    if (prefix + char is in Dictionary)
        prefix ← prefix + char
    else
    { if (prefix is empty)
        codewordforprefix ← 0.
      else
      { codewordforprefix ← Dictionary for prefix;
        output: (codewordforprefix, char)
        insertinDictionary (DictionaryIndex, prefix+char)
        DictionaryIndex ++;
        prefix ← empty
      }
    }
    if (prefix is not empty)
    {
        codewordforprefix ← Dictionary Index for prefix;
        output: (codewordforprefix);
    }
```

B|A|BA|AB|R|RR|A

| | | |
|---|---|---|
| (0, B) | 1 | B |
| (0, A) | 2 | A |
| (1, A) | 3 | BA |
| (2, B) | 4 | AB |
| (0, R) | 5 | R |
| (5, R) | 6 | RR |

Output :-

(0, B) (0, A) (1, A) (2, B) (0, R) (5, R) (2, )

# LZW Encoding Algorithm

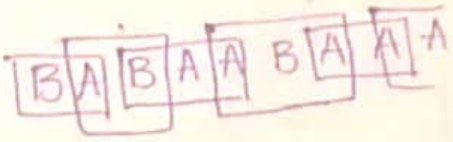
```
prefix ← first input character
while (not end of character stream)
{     char ← next input character;
      if (prefix + char exists in Dictionary)
          prefix ← prefix + char;
      else
      {   output: the code for prefix;
          insertInDictionary((codeWord, prefix + char));
          CodeWord++;
          prefix ← char;
      }
}
output: the code for prefix
```

BABAABAAA

prefix ← ~~B~~ ~~A~~ ~~B~~A ~~A~~ ~~B~~ ~~A~~ ~~A~~ A

CodeWord ← ~~2~~56 ~~2~~57 ~~2~~58 ~~2~~59 ~~2~~60 261

char ← ~~A~~ ~~B~~ ~~A~~ ~~A~~ ~~B~~ ~~A~~ ~~A~~ A




| Output | Index | Dictionary String |
|---|---|---|
| <66> | 256 | BA |
| <65> | 257 | AB |
| <256> | ~~258~~ | BAA |
| <257> | 259 | ABA |
| <65> | 260 | AA |
| <260> | | |