Faculty of Electrical Engineering
University of Montenegro, Podgorica

MULTIMEDIA SYSTEMS AND SIGNALS

DIGITAL IMAGE

- Image: two-dimensional analog function ***f(x,y)***
- Digital image – after digitalisation; represented by a two-dimensional set of samples – **pixels**
- Depending on the number of bits used for pixels representation, we have

-**Binary image** - one pixel one bit,
-**Computer graphics** - four bits per pixel
-**Grayscale image** - eight bits per pixel
-**Color image** – each pixel is represented with 24 or 32 bits

Increasing the number of bits reduces the quantization error, i.e., increases the signal-to-noise ratio by 6 dB per bit

Memory requirements:
Grayscale image - $8xN_1xN_2$ bits
Color image - $3x8xN_1xN_2$ bits

Image analysis:
- **spatial distribution of pixels** – gives the information about the positions of grayscaled values

- **distribution of pixels in different image regions** - described by the joint density distribution

DIGITAL IMAGE

- Beside the spatial distribution of pixels which provides the information about the positions of grayscaled values , distribution of pixels in different image regions can bring important information
- Such a distribution can be described by the **joint density distribution**

$$p(x_i)=\sum_{k=1}^{N}\pi_k p_k(x_i), \quad i=1,2,...,M$$

x_i - gray level of the *i*-th pixel
$p_k(x_i)$ - probability density function (**pdf**) for a region *k*
π_k - weighting factor

- **pdf** for a region *k* can be described by the generalized Gaussian function:

$$p_k(x_i)=\frac{\alpha\beta_k}{2\Gamma(1/\alpha)}e^{[-|\beta_k(x_i-\mu_k)|^{\alpha}]}, \quad \alpha>0, \quad \beta_k=\frac{1}{\sigma_k}\left[\frac{\Gamma(3/\alpha)}{\Gamma(1/\alpha)}\right]^{\frac{1}{2}}$$

Γ - gamma function
μ_k - mean
σ_k - variance

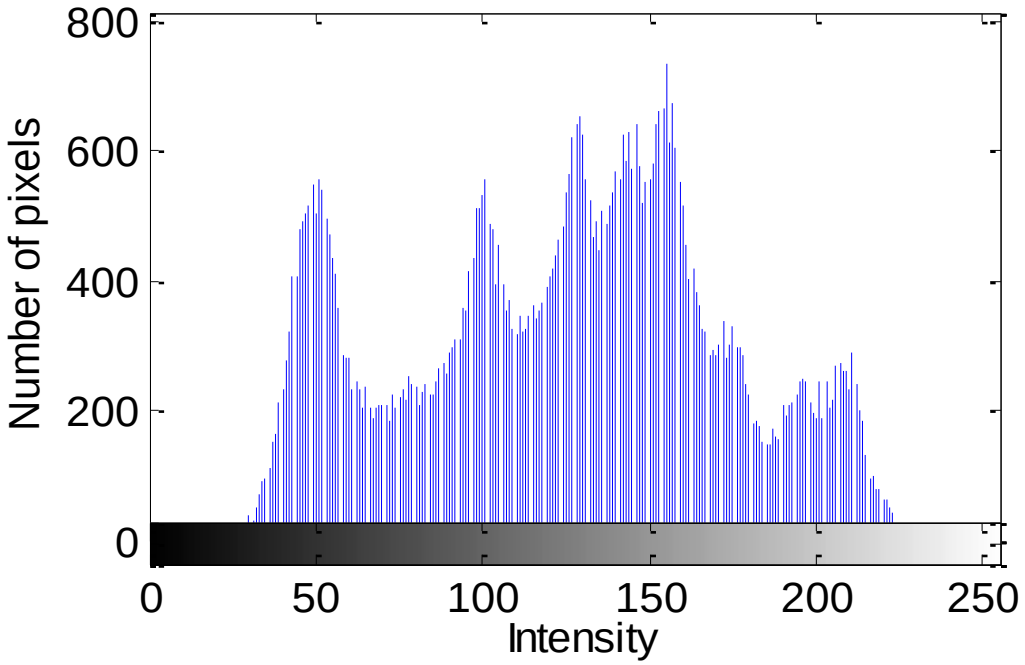
DIGITAL IMAGE

$\alpha = 2$ - the Gaussian distribution is obtained

$\alpha = 1$ - the Laplace distribution is obtained

- We can use generalized form of pdf to describe the image histogram
- Histogram provides information about the occurrence of certain pixel values

Histogram of a grayscale image "Lena"



Elementary algebraic operations with images

Consider two images:

- ***a*(*i,j*)** – pixel on the (i,j) position of the first image
- ***b*(*i, j*)** - pixel on the (i,j) position of the second image

Addition or **subtraction** of two images is done by adding or subtracting the corresponding pixels of an image

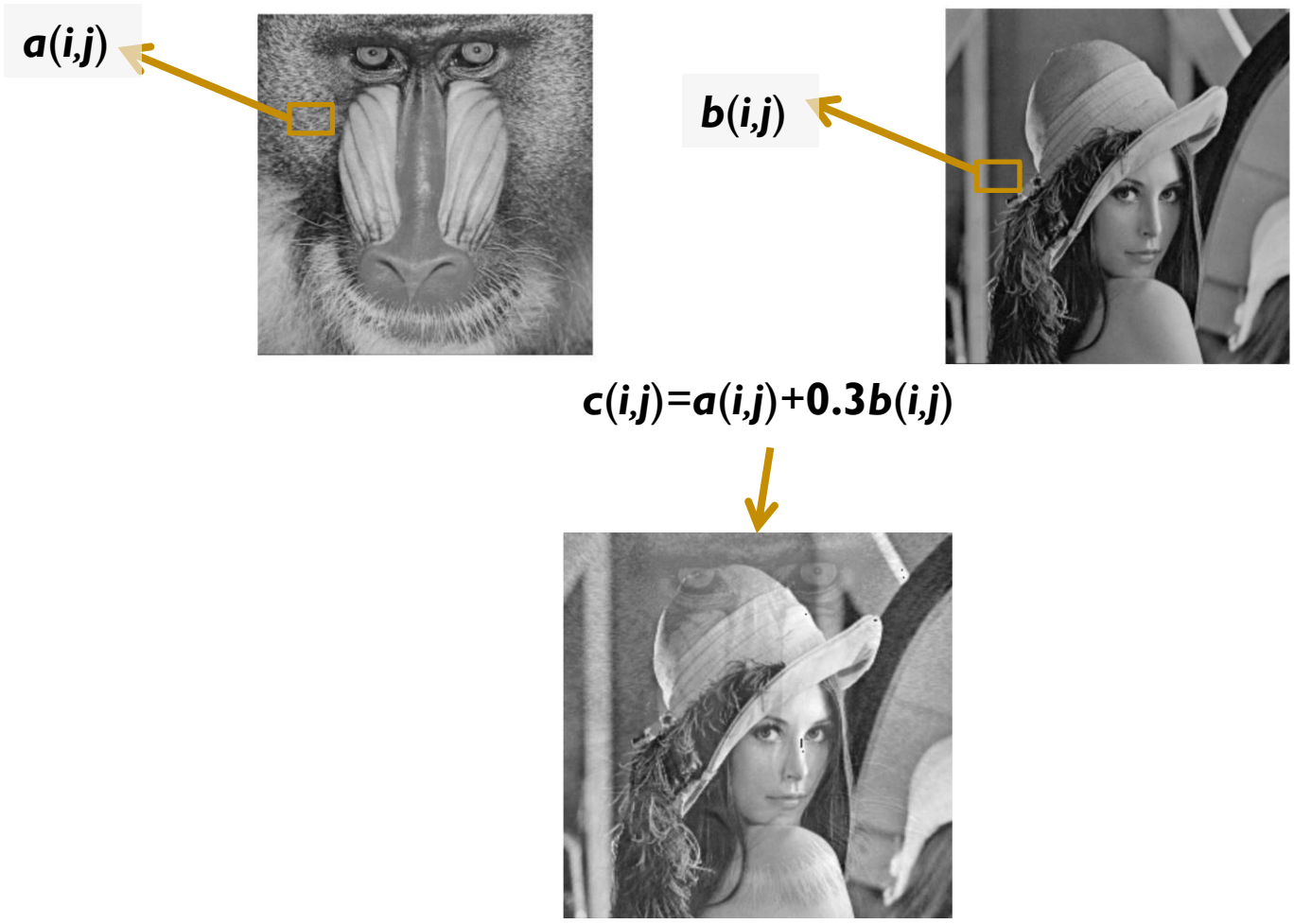
$$c(i,j)=a(i,j)\pm b(i,j) \quad - \quad \textit{resulting pixel}$$

Multiplying the image by a constant term *k* we get:

$$c(i,j)=ka(i,j)$$

- **Quantisation** (rounding to integer values) is performed in order to represent the result of these and other operations as a new image, we must perform quantization
- result is limited in the range of 0 to 255 (grayscale image is assumed)

Elementary algebraic operations with images



Resulting image obtained by adding 30% of "Baboon" to “Lena”

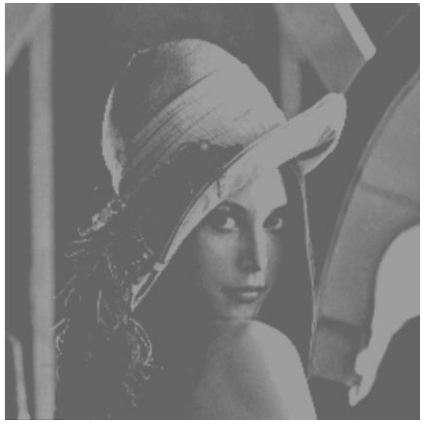
Elementary algebraic operations with images

- Relationship used to obtain a negative of a grayscale image:

$$n(i,j) = 255 - a(i,j)$$

- **Clipping** - cutting the pixels values over a certain value c_{max} and below a certain value c_{min}):

$$b(i,j) = \begin{cases} c_{\max} & a(i,j) > c_{\max} \\ a(i,j) & c_{\max} \geq a(i,j) \geq c_{\min} \\ c_{\min} & a(i,j) < c_{\min} \end{cases}$$



The result of clipping for

$c_{max} = 156,\ c_{min} = 100$

Basic geometric operations

- Translation of an image ***a*(*i*,*j*)** - moving the pixels in one or both directions for a certain number of positions

Translated image by embedding 31 rows and 31 columns of black color (zero value)

- Coordinates of the imege can be written by the vector $\begin{bmatrix} x \\ y \end{bmatrix}$

- Coordinates after rotation can be obtained as:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

"Lena"
rotated by 45°

- After image rotation, we to transform points from the polar coordinate system to the rectangular coordinate system
- This transform is performed with certain approximations

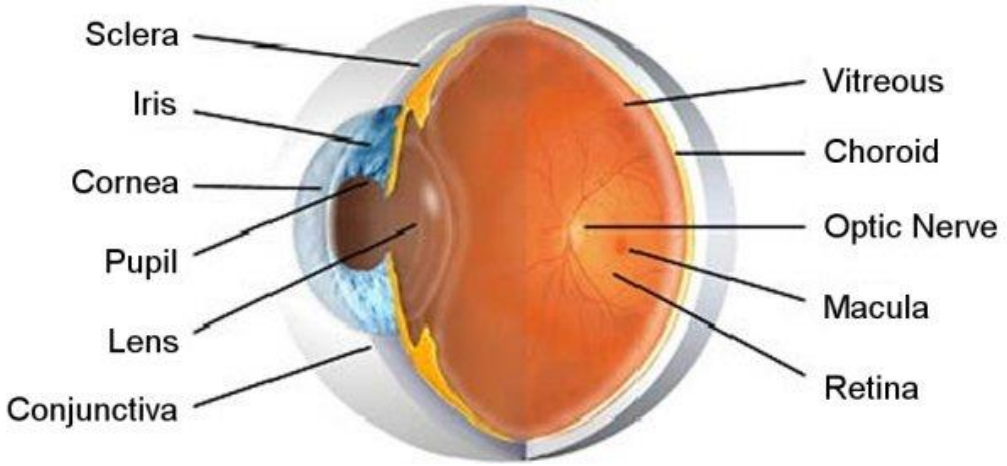
The characteristics of the human eye

- Different image processing algorithms can be defined that will meet some important perceptual criteria
- Image is projected in the eye with light rays coming into the eye
- An important feature is the eye sensitivity to the change of light intensity
- The eye does not perceive linearly the changes in the light intensity, but **logarithmically**
- In cases of very high or very low-light intensity, eye has ability to saturate

- There are two types of cells in the eye:

elongated (rod cells or rods) **-** 125 million
cone-like (cone cells or cones) - 5.5 million

- Rods respond to light and cones respond to colors



The characteristics of the human eye

- An eye is not equally sensitive to three primary colors: red, green and blue
- Relative ratio of these sensitivities is: *Red : Green : Blue* = 30% : 59% : 11%

- The eye is able to identify approximately between 40 and 80 shades of gray
- For color images it can recognize between 15 and 80 million of colors
- Cons detect light that enter the eye
- The image in the brain is actually obtained as the sum of images in primary colors (Televisions, monitors, and other displays follow the human three-color model)

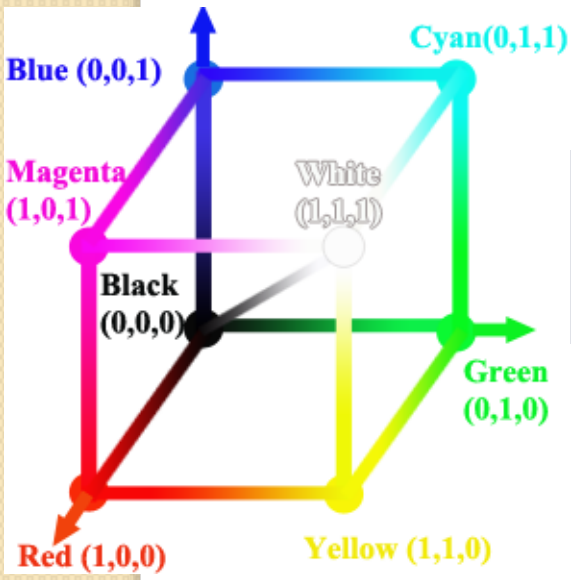
- Image quality can be represented by three dimensions: **fidelity, usefulness** and **naturalness**

-The usefulness is a major metric for medical imaging;
-The fidelity is the major metric for paintings
-The naturalness is the most important feature in virtual reality applications

Color models

- Color - one of the most important image characteristic
- Color is invariant to translation, rotation and scaling
- Various color systems are used to model the color image

1. RGB is one of the most commonly used color systemsThis system can be represented by the color cube



The gray level is defined by the line R = G = B

- RGB model is based on the human perception of colors, and thus has been used for displaying images (monitors, TV, etc.)

- RGB model uses fact that the image can be viewed as a vector function of three coordinates for each position within the image

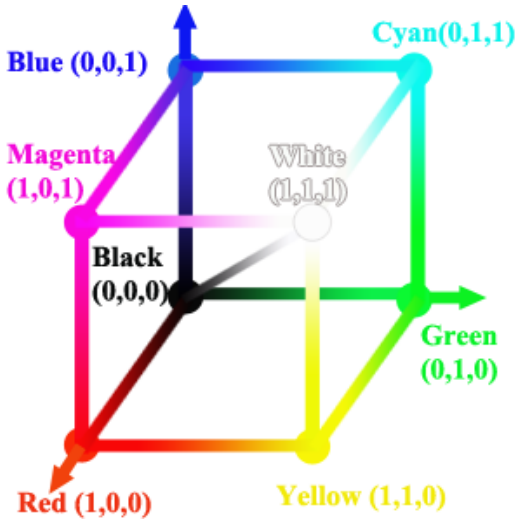
- RGB model is sometimes called the **additive model** - the image is obtained by adding the components in primary colors
- Each point in the image can be represented by the sum of values of the three primary colors (R, G, B)

Color models

- A size of an RGB digital image depends on how many bits we use for quantization
  - *n* = 8 bits - values range from 0 to 255
- The value 0 (coordinate=0) means the absence of a color
- The value 255 (coordinate=1) denotes the color data with maximum intensity
  (0,0,0) represents black
  (1,1,1) represents white
- The **luminescence** for grayscale images is calculated as the mean pixel values of the RGB components of color images

- Colors used in **CMY color model** are obtained by combining two of the three primary colors (R, G, B)

G+B=C (cyan);
R+B=M (magenta);
R+G=Y (yellow);
R+G+B=W (white)



Color models

- CMY color model is basically the most commonly used in printers, because the white is obtained by the absence of all three colors
- Black could be obtained by combining all three colors together – however, the printers usually have a separate cartridge for the black color
- **CMYK color** model - CMY model including the black color
- The connection between CMY and RGB (from the color cube):

$$C=1-R,\quad M=1-G,\quad Y=1-B$$

while the CMYK model can be obtained as:

$$K=\min(C,M,Y),\quad C=C-K,\quad M=M-K,\quad Y=Y-K$$

- **YUV color model**:
- color is represented by three components: **luminescent** (Y) and two **chrominance** (UV) components
- YUV to RGB is obtained from the following equations:

$$Y = 0.299R + 0.587G + 0.114B$$
$$U = 0.564(B - Y)$$
$$V = 0.713(R - Y)$$

Some commonly present noise probability distributions

- Image noise may occur during image transmission over communication channel
- The most common types of noise are;
- **Impulse noise (**manifested as a set of black and white pulses in the picture)
- **Gaussian noise** (occurs as a result of atmospheric discharges, or due to electromagnetic fields generated by various appliances)
- **Two-sided impulse noise model** - impulse noise takes two fixed values *a* (negative impulse) and *b* (positive impulse) with equal probabilities *p*/2:

$$f_I(i,j)=\begin{cases} a & \text{with a probability } p/2 \\ b & \text{with a probability } p/2 \\ f(i) & \text{with a probability } (1-p) \end{cases}$$

"Lena" affected by an impulse noise with density 0.05

Some commonly present noise probability distributions

- Thermal noise - modeled as white Gaussian noise
- Its distribution is given by:

μ - mean

σ - standard deviation of noise

$$P_g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- **Uniformly distributed noise :**

-The gray level values of the noise are evenly distributed across a specific range.

-The quantization noise can be approximated by using uniform distribution. The corresponding pdf is defined by:

$$P_u(x) = \begin{cases} \frac{1}{b-a}, & for\ a \le x \le b \\ 0, & otherwise. \end{cases}$$

$\mu = (a+b)/2$ - mean

$\sigma^2 = (b-a)^2/12$ - standard deviation of noise

"Lena" affected by zero-mean white Gaussian noise whose variance is equal to 0.02

$$P_R(x) = \begin{cases} \frac{2}{\beta}(x-\alpha)e^{-(x-\alpha)^2/\beta}, & for\ x \ge \alpha \\ 0, & otherwise \end{cases}$$

$$\mu = \alpha + \sqrt{\pi\beta/4}, \quad \sigma^2 = \beta(4-\pi)/4$$

Radar images may contain noise that is characterized by the **Rayleigh distribution**

Filtering in the spatial domain

- Goal of filtering of noisy images - noise reduction and highlighting image details
- The commonly used filters in the spatial domain are **mean** and **median** filters
- The use of these filters depends on the nature of the noise that is present within the image
- In case of **aditive noise**, spatial domain filters are suitable

Mean filtar

- Used to filter the images affected by **Gaussian white noise**,
- Calculates the average pixel intensity within an image part captured by a specified window
- Wide window can cause blurring of image details, but results in better noise reducton (the noise might converge to zero).

Filtering in the spatial domain

- **Mean filtar:**

- consider a window of size $(2N_1+1)\times(2N_2+1)$

- signal affected by noise is given by: $z(i,j)=f(i,j)+n(i,j)$

$$z_f(i,j)=\frac{1}{(2N_1+1)(2N_2+1)}\sum_{n=i-N_1}^{i+N_1}\sum_{m=j-N_2}^{j+N_2}z(n,m)$$

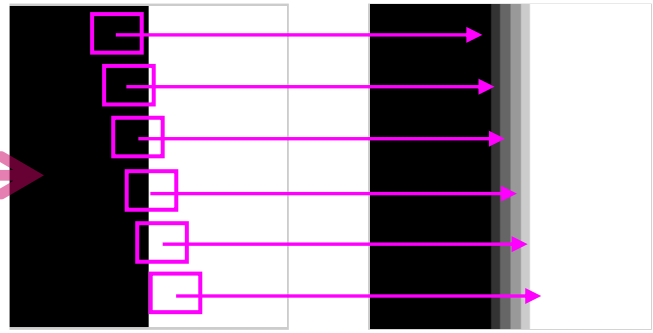
The output of the arithmetic mean filter

- output of this filter is mean value of pixels captured by the window

$z(n,m)$ - pixel value within the window

$h(i,j)=1/(2N_1+1)(2N_1+1)$ - impulse response of the filter

blurring after applying a mean filter on the image edge (the mask size used is 5x5)



-window size of 3x - 9 points

- window size 5x5 - 25 points

Second window will be more effective for noise reduction, but will introduce more smoothed edges and blurred image

Filtering in the spatial domain

"Lena" affected by Gaussian noise is zero mean and variance 0.02

Filtered image obtained by using 3x3 moving filter window

Filtered image obtained by using the 5x5 mean moving filter window

The geometric mean filter can be used instead arithmetic mean filter
Output of the filter is:

$$z_f(i,j) = \left(\prod_{n=i-N_1}^{i+N_1} \prod_{m=j-N_2}^{j+N_2} z(n,m) \right)^{\frac{1}{(2N_1+1)(2N_2+1)}}$$

Filtering in the spatial domain

- **Geometric mean filter** - less blurring
  - preserves more image details

original image

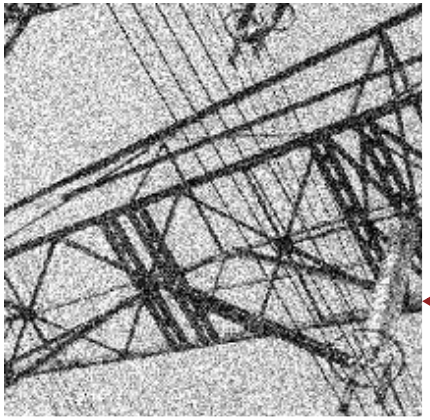
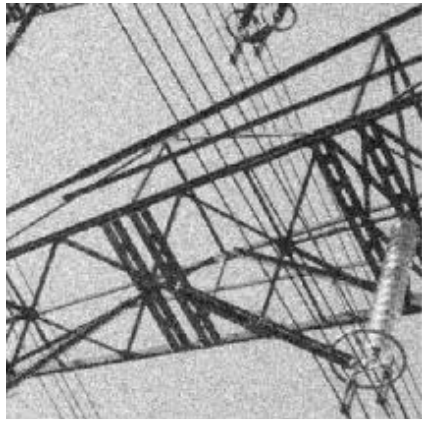


image affected by Gaussian noise with - 0.05 mean and variance 0.025

image filtered by using arithmetic mean



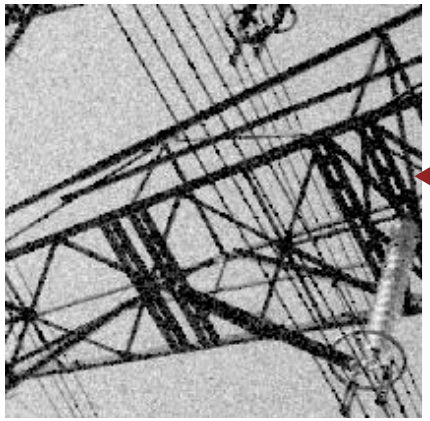


image filtered by using geometric mean

Filtering in the spatial domain

- **Median filtar**

- filter out the impulse noise
- If a sequence has an odd number of elements - sort the elements in an array. The median value is the middle element of the sorted array
- If a sequence has an even even number of elements, the median is the mean of the two central elements of sorted array

Sequence I: *3 14 7 1 5*

1 3 ***5*** *7 14*

median

Sequence II: *1 12 7 4 9 2*

1 2 ***4 7*** *9 12*

median → *5.5*

Filtering in the spatial domain

1. Median of image can be determined by rewriting the matrix as a vector **z**:{$z(k), k \in [1,N]$} and then finding the median value z_m for the vector:

$$z_m = med\left(z(1),\ldots,z(k),\ldots,z(N)\right) = \begin{cases} z_s(\lfloor N/2 \rfloor + 1), & N \text{ is odd} \\ \dfrac{z_s(N/2) + z_s(N/2+1)}{2}, & N \text{ is even} \end{cases}$$

$\mathbf{z_s}$ is sorted version of **z**

2. Another way to calculate the median of a matrix is to calculate the median value for columns and then for rows (or vice versa)

- These two approaches usually do not produce exactly the same results

Filtering in the spatial domain

- Suppose $(2N_1+1)(2N_2+1)$ window width
- $z(i,j)$ - the central pixel in the filter window
- From all the pixels within the window we form a matrix:

$$\begin{bmatrix} z(i-N_1, j-N_2) & \dots & z(i, j-N_2) & \dots & z(i+N_1, j-N_2) \\ \vdots & & \vdots & & \vdots \\ z(i-N_1, j) & \dots & z(i, j) & \dots & z(i+N_1, j) \\ \vdots & & \vdots & & \vdots \\ z(i-N_1, j+N_2) & \dots & z(i, j+N_2) & \dots & z(i+N_1, j+N_2) \end{bmatrix}$$

1. Compare and sort the matrix elements in the appropriate sequence and determine the median of each of these sequences
2. Repeat the previous procedure, except that instead of pixels from the filter window, we use the median values already obtained

$$q_n = med\{z(i, j-N_2), z(i, j-N_2+1), \dots, z(i, j+N_2)\}$$

output of separable median filter →

$$q(i,j) = med\{q_n;\ n \in (i-N_1, i+N_1)\}$$

Filtering in the spatial domain

"Lena" affected by impulse noise with density 0.05

image filtered by a median filter with 3x3 window

image filtered by a median filter with 5x5 window

- **α-trimmed mean filter** - compromise between the median and arithmetic mean filter
- After sorting the windowed pixels, a few lowest and highest samples are discarded by using parameter α, while the remaining pixels are averaged

Filtering in the spatial domain

α-trimmed mean filter →

$$z_\alpha(i,j)=\frac{1}{(N-2[\alpha N])}\sum_{n=[\alpha N]+1}^{N-[\alpha N]} z_s(n)$$

- $z_s(n)$ is the vector of sorted pixels from the window $N_1 x N_2$, $N=N_1 N_2$
- [.] denotes rounding to the greatest integer
- Parameter α takes the values from the range $0 \le \alpha < 0.5$

$\alpha=0.5$ corresponds to median, for odd N

$\alpha=0$ it performs as the moving average filter

- the same operation can be performed separately on the rows and columns:

$$z_\alpha(i,j)=\frac{1}{(N_1-2[\alpha N_1])(N_2-2[\alpha N_2])}\sum_{n=[\alpha N_1]+1}^{N_1-[\alpha N]}\sum_{m=[\alpha N_2]+1}^{N_2-[\alpha N]} z(m,n)$$

Filtering in the frequency domain

- Filters in the frequency domain are designed on the basis of a priori knowledge about signal frequency characteristics
- The most significant frequency content of images is usually concentrated at low frequencies ⟶ in many applications, the images are usually filtered with low-pass filters

$$H(\omega_1,\omega_2)=\begin{cases}1 & |\omega_1|\le W_1 \text{ and } |\omega_2|\le W_2\\ 0 & \text{otherwise}\end{cases}$$

low-pass filter

$$H(\omega_1,\omega_2)=\begin{cases}1, & W_{11}\le|\omega_1|\le W_{12},\ W_{21}\le|\omega_2|\le W_{22}\\ 0, & \text{otherwise}\end{cases}$$

bandpass filter

$$H(\omega_1,\omega_2)=\begin{cases}1, (|\omega_1|>W_1 \text{ and } |\omega_2|>W_2) \text{ or } (\omega_1^2+\omega_2^2>W)\\ 0, \quad \text{otherwise}\end{cases}$$

high-pass filter

$$H(\omega_1,\omega_2)=\begin{cases}1 & \omega_1^2+\omega_2^2\le W\\ 0 & \text{otherwise}\end{cases}$$

circular low-pass filter

Edge detection

- Edges of the image should be obtained by simple differentiation
- The image is always more or less affected by noise

direct application of differentiation is not effective

- One of the most commonly used algorithm for edge detection is based on the **Sobel matrix**
- The image is analyzed pixel by pixel, using the Sobel matrix as a mask
- The matrix elements are the weights which multiply the pixels within the mask
- The sum is calculated by adding all the obtained values and resulting value is compared with a threshold
  - The central pixel belongs to the edge if the sum is greater than the threshold, the and vice versa

$$S_v = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Sobel matrix for vertical edges

$$S_h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel matrix for horizontal edges

Edge detection

- The edges are obtained by → $L(i,j) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} a(i+m, j+n) S(m+2, n+2)$

$S(m,n)$ - **filtering function** (e.g., the Sobel matrix S_h and S_v)

- After calculating L_h and L_v (using the horizontal and vertical matrix), the overall L is: $L = \sqrt{L_h^{\,2} + L_v^{\,2}}$
- Obtained value is compared with a threshold and the results are represented in a binary form
- Local threshold values are frequently used

They are calculated based on the mean response of edge detector around the current pixel

- A threshold value can be calculated as:

$$T(i,j) = \bar{L}(i,j)(1+p) = \frac{1+p}{2N+1} \sum_{k=i-N}^{i+N} \sum_{l=j-N}^{j+N} L(k,l)$$

- p has a value between 0 and 1

Edge detection

Illustration of edge detection

original image

L_v

L_h

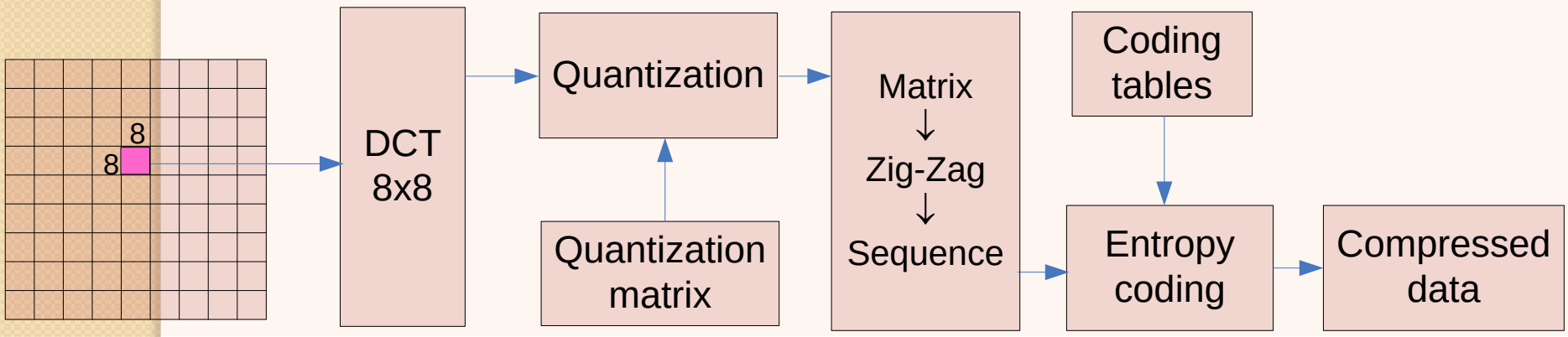
L

Data compression

- Multimedia information is very demanding concerning the memory space
- Needs much processing power
- May require higher bit rates compared to the available bit rates of the communication channels
- These aspects lead to the use of compression algorithms
- Data compression - **lossless compression** and **lossy compression**

JPEG image compression algorithm

- **JPEG algorithm** - significant compression ratio, high image quality JPEG algorithm is analyzed across several blocks used for image compression - **block performing discrete cosine transform** (DCT) on the 8x8 image blocks,
  - **- quantization block,**
  - **- zig-zag matrix scanning block,**
  - **- entropy coding block.**



JPEG image compression algorithm

DCT of an 8x8 image block

$$DCT(k_1,k_2)=\frac{C(k_1)}{2}\frac{C(k_2)}{2}\sum_{i=0}^{7}\sum_{j=0}^{7}a(i,j)\cos(\frac{(2i+1)k_1\pi}{16})\cos(\frac{(2j+1)k_2\pi}{16})$$

$$C(k_1)=\begin{cases}\frac{1}{\sqrt{2}} & for \quad k_1=0\\ 1 & for \quad k_1>0\end{cases}$$

$$C(k_2)=\begin{cases}\frac{1}{\sqrt{2}} & for \quad k_2=0\\ 1 & for \quad k_2>0\end{cases}$$

- The DCT coefficient (0,0) is **DC component**
- The DC component carries an information about the mean value of 64 coefficients
- The remaining 63 coefficients are **AC coefficients**

- The samples of the grayscaled image whose values are in the range $[0, 2^n - 1]$ are shifted to the range $[-2^{n-1}, 2^{n-1}-1]$

- where *n* is number of bits used to represent samples

- Then the DCT is applied
- For 8-bit samples case, the shifted range is $[-128, 127]$
- The corresponding DCT coefficients will be in the range:

$[-1024, 1023]$ and they require additional 3 bits

JPEG image compression algorithm

Original image "Lena"

The image based on the first 128x128 elements

The image based on the first 64x64 elements

The image based on the first 25x25 elements

JPEG image compression algorithm

- From previous slide follows that the most important transform coefficients of images are concentrated at low frequencies

1. Apply DCT and low-pass filtering
2. Take the first 128x128 coefficients, then the first 64x64 coefficients, and finally the first 25x25 elements
3. Apply the inverse DCT

- Reconstructed images are shown on previous slide
- Although the number of coefficients is significantly decreased, the image retains much of the information

- 32x32 and 16x16 blocks slightly improve coding gain compared to the 8x8 blocks, the JPEG compression still uses 8x8 blocks due to the quite easier calculation

JPEG image compression algorithm

- Image is first decomposed into 8x8 blocks. Next, the DCT is calculated for each 8x8 block
- DCT coefficients are divided by weighting coefficients, representing the elements of quantization matrix:

$$DCT_q(k_1,k_2)=round\{\frac{DCT(k_1,k_2)}{Q(k_1,k_2)}\}$$

- Q - quantization matrix
- DCT_q - quantized coefficients.

- calculating coefficients of the quantization matrix

```
for  i = 0 : N
     for  j = 0 : N
Q(i+1, j+1) = 1+[(1+i+ j)*quality];
     end
end
```

- ***quality*** parameter ranges from 1 to 25
- Higher values of ***quality*** - better compression but worse image quality

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

compression matrix for *quality*=2

JPEG image compression algorithm

- In practical applications, the quantization matrices are derived from the **experimental quantization matrix**
- The experimental quantization matrix is defined for 50% compression ratio (quality factor - QF = 50)

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Coefficients of the quantization matrix Q_{50}

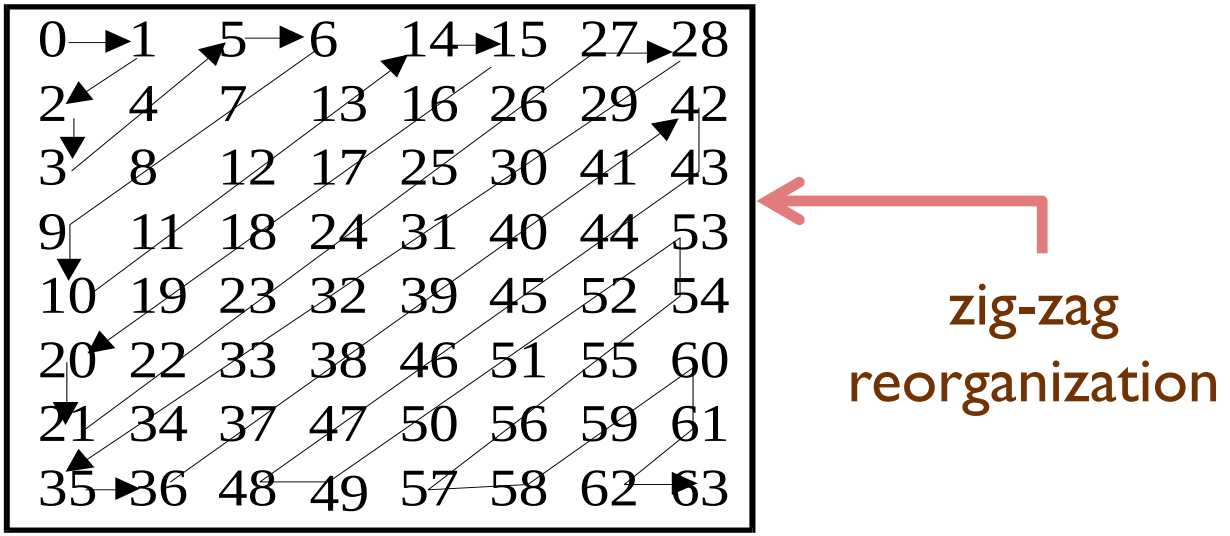
- Matrices for other compression degrees $\Rightarrow Q_{QF} = round(Q_{50} \cdot q)$

$$q = \begin{cases} 2 - 0.02QF, & for \quad QF \geq 50 \\ \dfrac{50}{QF}, & for \quad QF < 50 \end{cases}$$

JPEG image compression algorithm

- DCT coefficients of 8x8 blocks are divided by the corresponding coefficients of quantization matrices and rounded to the nearest integer values

- After quantization, the **zig-zag** reordering is applied to the 8x8 matrix
- Vector of 64 elements is formed
- The values are sorted from low-frequency coefficients to the high-frequency coefficients
- The entropy coding is next applied based on the Huffman coding



JPEG image compression algorithm

- Each AC coefficient is encoded with two symbols: ***(a,b)=(runlength,size)***
- The *runlength* provides the information about the number of consecutive zero coefficients preceding the non-zero AC coefficient.
- The *runlength* is encoded with 4 bits, so it can be used to represent 15 consecutive zero coefficients
- Symbol (15,0) represents 16 consecutive zero AC coefficients and it can be up to three (15,0) extensions
- This symbol also contains information on the number of bits required to represent the coefficient's value (*size*)
- The second symbol is **the amplitude of the coefficient** (which is in the range [-1023,1024])
- Second symbol can be represented with up to 10 bits

JPEG image compression algorithm

- Example:

- Sequence 0,0,0,0,0,0,0,239, is coded as (7,8) 239
- (0,0) symbol denotes the end of the block

- The differences between DC coefficients is coded instead of their values (there is a strong correlation between the DC coefficients from adjacent blocks)

$[-2048, 2047]$ → - Range of the DC coefficients

- DC coefficient is coded by two symbols

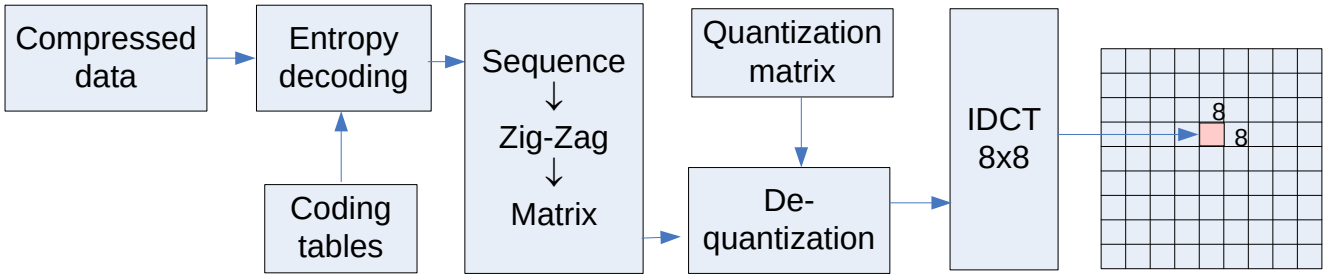
- ***first symbol*** is the number of bits (*size*) used to represent the amplitude
- ***second symbol*** is the amplitude itself

- Amplitude for both DC and AC coefficients are encoded by using the variable-length integer code

Amplitude range	Size
-1,1	1
-3,-2,2,3	2
-7,-6,-5,-4,4,5,6,7	3
-15,…,-8,8,…,15	4
-31,…,-16,16,…,31	5
-63,…,-32,32,…,63	6
-127,…,-64,64,…,127	7
-255,…,-128,128,…,255	8
-511,…,-256,256,…,511	9
-1023,…,-512,512,…,1023	10

JPEG image compression algorithm

- DECODING:



- The samples are returned into to the matrix form
- De-quantization followed by the inverse DCT is performed

$$DCT_{dq} = DCT_q(k_1, k_2) \cdot Q(k_1, k_2)$$

- Inverse DCT transformation

$$a(i,j) = \sum_{k_1=0}^{7} \sum_{k_2=0}^{7} \frac{C(k_1)}{2} \frac{C(k_2)}{2} DCT(k_1, k_2) \cos\left(\frac{(2i+1)k_1\pi}{16}\right) \cos\left(\frac{(2j+1)k_2\pi}{16}\right)$$

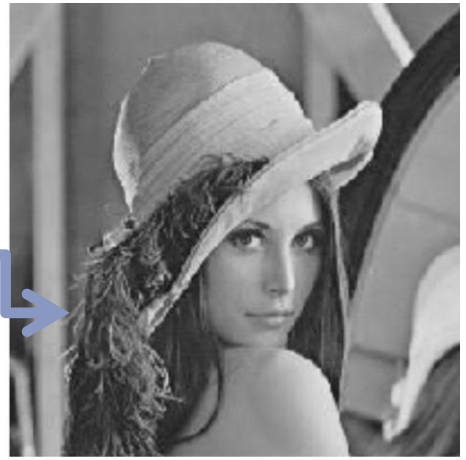
- Duantization/de-quantization and rounding procedures introduce an error
- The error is proportional to the quantization step

JPEG image compression algorithm

- Examples of compressed images with different qualities

original "Lena" image

"Lena" after applying JPEG compression quality of 70%



"Lena" after JPEG compression quality of 25%

"Lena" after JPEG compression quality 5%

Examples

1. Calculate the memory requirements for an image of size 256x256 pixels, in the case of:

a) Binary image,

b) Grayscale image,

c) Color image.

Solution:

Binary image: $256 \cdot 256 \cdot 1 = 65536$ bits

Grayscale image: $256 \cdot 256 \cdot 8 = 524288$ bits

Color image usually contains 3 different matrices for ach color channel and required three time higher memory space than the grayscale image:

$256 \cdot 256 \cdot 8 \cdot 3 = 1572864$ bits

Examples

2. If the values of R, G, and B components in the RGB systems are known and for a certain pixel they are given by R=0.5, G=0.2, B=0.8, determine the corresponding values of the components from the YUV color model.

$$Y = 0.299R + 0.587G + 0.114B$$
$$U = 0.564(B - Y)$$
$$V = 0.713(R - Y)$$

Y=0.299·0.5+0.587·0.2+0.114·0.8= 0.358
U=0.564·(0.8-0.358)=0.25
V=0.713·(0.5-0.358)=0.1

Examples

3. Write a Matlab code which will load color image (e.g., lena.jpg), determine the image size, and then converts the color image into grayscaled version by using the Matlab built-in function rgb2gray, as well as by using the following formula:

$$Grayscale = \frac{R_{value} + G_{value} + B_{value}}{3}$$

```
I=imread('lena.jpg');  % load image
size(I)                % image size
ans =
   512   512     3
I1=rgb2gray(I);
imshow(I1)         % show image
I2=double(I);
I2=(I(:,:,1)+I(:,:,2)+I(:,:,3))/3;
imshow(uint8(I2));
```

Note: The color channels are obtained as: I(:,:,1), I(:,:,2), I(:,:,3)

Examples

4. Write a code in Matlab that will create a negative of image *"cameraman.tif"*.

```
I=imread('cameraman.tif');
I=double(I);

N=255-I;
imshow(uint8(N))
imshow(uint8(I))
```

Original

Negative

Examples

5. Write a code in Matlab that will provide a simple image darkening and brightening procedure by decreasing/increasing original pixels values for 40%.

Solution:

```
I=imread('cameraman.tif');
I=double(I);
B=I+0.4*I;          % brightening
figure(1), imshow(uint8(B))
D=I-0.4*I;          % darkening
figure(2), imshow(uint8(D))
```

Examples

6. Starting from the grayscale image "*cameraman.tif*", make a version of binary image by setting the threshold on value 128.

A binary image will have values 255 at the positions (*i,j*) where the original image has values above the threshold. On the remaining positions the pixels in the binary image will be 0.

$$B(i,j) = \begin{cases} 255, & I(i,j) > threshold \\ 0, & I(i,j), \quad otherwise \end{cases}$$

```
I=imread('cameraman.tif');
for i=1:m
        for j=1:n
                if I(i,j)>128
                I(i,j)=255;
                else
                I(i,j)=0;
                end
         end
       end
imshow(I)
```

Examples

7. Consider a color image *lena.jpg*. Transform the image into grayscaled one and add a white Gaussian noise with variance 0.02.

Solution:

```
I=imread('lena.jpg');
I=rgb2gray(I);
I=double(I);
I1=imnoise(I,'gaussian',0,0.02);
imshow(uint8(I1))
```

Examples

8. Calculate he mean and median values for vectors:

a) v_1=[12 22 16 41 -3]; b) v_2=[12 9 22 16 41 -3];

a) v_1=[12 22 16 41 -3];
mean= 17.6
sorted_v1=[-3 12 16 22 41];
median=16.

b) v_2=[12 9 22 16 41 -3]
mean=16.16

sorted_ v_2=[-3 9 12 16 22 41]
median=(12+16)/2=14.

Examples

9. By using the Matlab function *imnoise*, add the impulse noise ('salt & pepper' with a density 0.1) to the image "lena.jpg". Then perform the image filtering by using median filter realized by Matlab function medfilt2 for two-dimensional median filter form.

```
I=imread('lena.jpg');
I=rgb2gray(I);
imshow(I)
% Noisy image 'salt & pepper' (density 0.1)
In=imnoise(I,'salt & pepper',0.1);
imshow(In)
% Image filtering
If=medfilt2(In);
imshow(If)
```

Original image

Noisy image

Filtered image

Examples

10. Write your own code for median filtering in Matlab: the filtering should be applied to image *cameraman.tif* which is corrupted by the impulse noise with density 0.1. Use the window of size 3x3.

```
I=imread('cameraman.tif');
In=imnoise(I,'salt & pepper',0.1);
M= In;
[m,n]=size(In);
a=double(In);
for i=1:m
for j=1:n
b=a(max(i,i-1):min(m,i+1),max(j,j-1):min(n,j+1));
c=b(:);
M(i,j)=med(c);
end
end
figure(1),imshow(In)
figure(2),imshow(uint8(In))
```

Examples

11. Write a Matlab code that filter an image corrupted by Gaussian noise with zero mean and variance equal to 0.01, where we use the window of size 5x5. It is necessary to include the image boundaries as well.

```
I=imread('cameraman.tif');
I_n =imnoise(I,'gaussian',0,0.1);
M= I_n;
[m,n]=size(I_n);
a=double(I_n);
for i=1:m
for j=1:n
b=a(max(i,i-2):min(m,i+2),max(j,j-2):min(n,j+2));
c=b(:);
M(i,j)=mean(c);
end
end
figure(1),imshow(I_n)
figure(2),imshow(uint8(I_n))
```

Examples

13. For a given block of 8x8 DCT coefficients and the given JPEG quantization matrix *Q*, perform the quantization, zig-zag scanning and determine the intermediate symbol sequence.

$$D = \begin{bmatrix} 80 & 50 & 26 & 10 & 33 & 11 & 0 & 0 \\ 22 & -28 & 34 & 10 & 0 & 0 & 0 & 0 \\ 14 & 10 & 17 & 11 & 5 & 0 & 5 & 0 \\ 56 & 17 & 20 & 12 & 0 & 12 & 8 & 0 \\ 10 & 12 & 8 & 3 & 2 & 0 & 7 & 0 \\ 10 & 13 & 17 & 3 & 0 & 2 & 2 & 0 \\ 6 & 0 & 5 & 10 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad Q = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{bmatrix}$$

Solution:

DCT coefficients from the 8x8 block are divided by the quantization matrix and rounded to the integer values, as follows:

$$D_q = round(D/Q) = \begin{bmatrix} 27 & 10 & 4 & 1 & 3 & 1 & 0 & 0 \\ 4 & -4 & 4 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 6 & 2 & 2 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Examples

After performing the zig-zag scanning of the matrix D_q, the sequence is obtained in the form:

27, 10, 4, 2, -4, 4, 1, 4, 1, 6, 1, 2, 2, 1, 3, 1, 0, 1, 2, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ...

Final the intermediate symbol sequence is given by:

(5)(27), (0,4)(10), (0,3)(4), (0,2),(2), (0,3)(-4), (0,3)(4), (0,1)(1), (0,3)(4), (0,1)(1), (0,3)(6), (0,1)(1), (0,2)(2), (0,2)(2), (0,1)(1), (0,2)(3), (0,1)(1), (1,1)(1), (0,2)(2), (0,1)(1), (0,1)(1), (1,1)(1), (0,1)(1), (0,1)(1), (8,1)(1), (6,1)(1), (9,1)(1), (0,0).

The code words for the symbols (*a*,*b*) are given in the table:

	Code word
(0,1)	**00**
(0,2)	**01**
(0,3)	**100**
(0,4)	**1011**
(1,1)	**1100**
(6,1)	**1111011**
(8,1)	**111111000**
(9,1)	**111111001**
(0,0) EOB	**1010**