Image Processing and Computer Vision

Presented by: AIADI Oussama

Image compression using JPEG

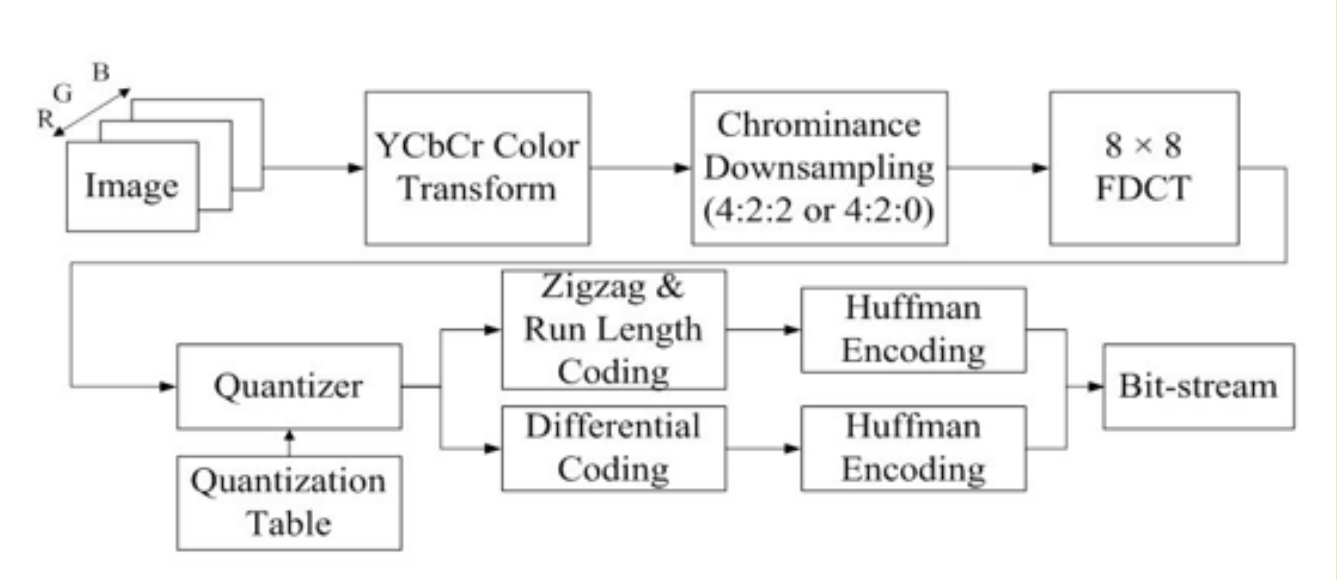
=

BMP image = 891 KB

JPEG image = 67 KB

Image compression using JPEG

- The main aim behind compression process is to make the file *smaller*, especially for the case of videos (sequence of images).
- File size is proportional to the storage requirement in hard disk, cloud, memory supports…etc. i.e., increasing file size will require more memory space.
- For large files (e.g., video), the transmission bandwidth is limited, so, it is important to upload/load small size files.



The general flowchart of JPEG compression algorithm

Image compression using JPEG

Image

Y

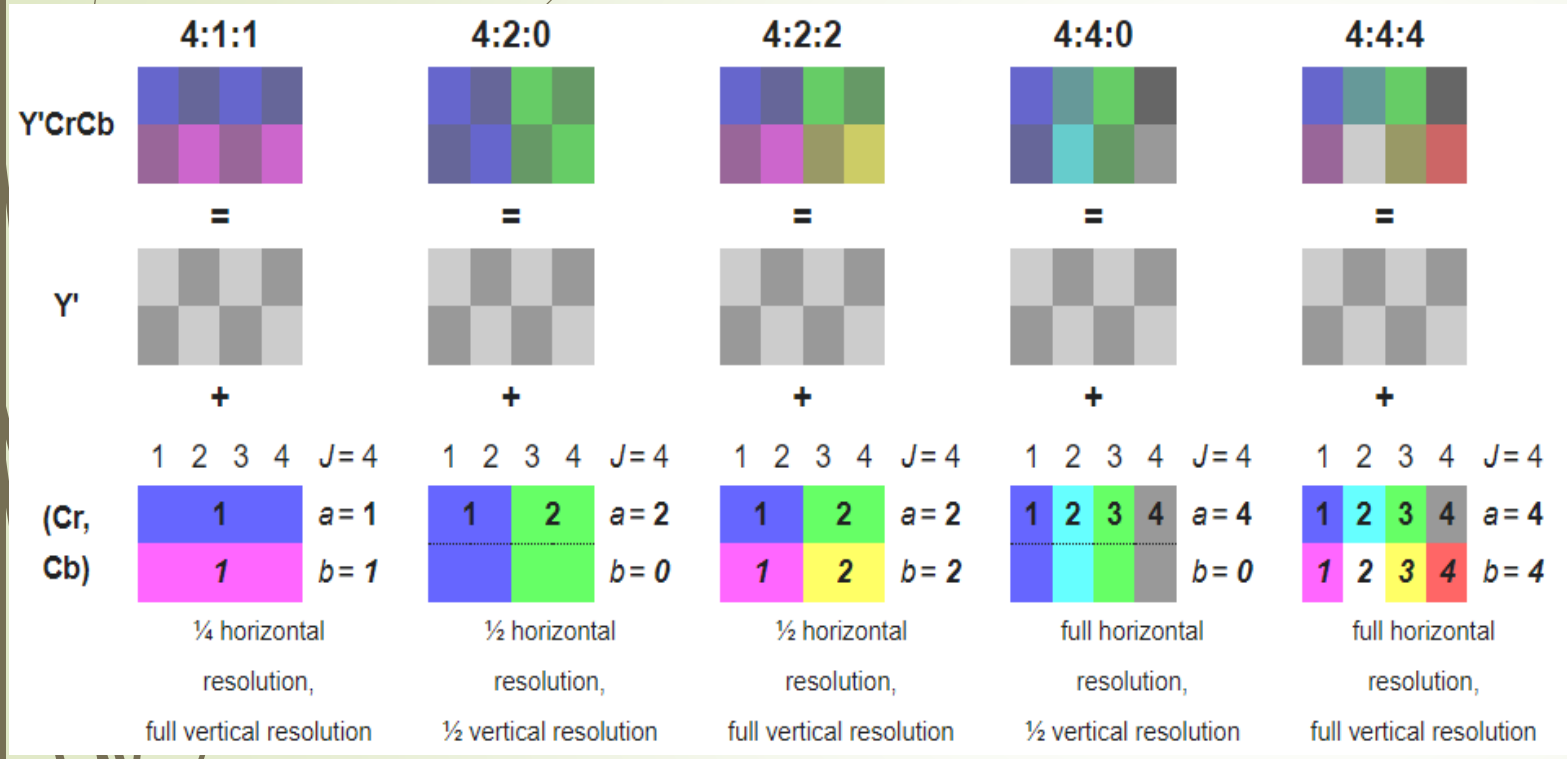
Cb

Cr

Typically, in JPEG images are converted from RGB to YCbCr color space before proceeding to the subsequent steps. This is because in this color space achromatic (which carry out more information) and chromatic channels are separated.

Researchers have shown that humans are more sensitive to different levels of brightness than it is to differences in color. Since the achromatic brightness component (Y) is separated from the color (chromatic) components, these components are subsampled e.g., instead of considering 255 level (i.e., 8 bits), we can consider 16 level (i.e., 4 bits). This is called chroma. subsampling and it allows reducing the amount of bit allocated to the image.

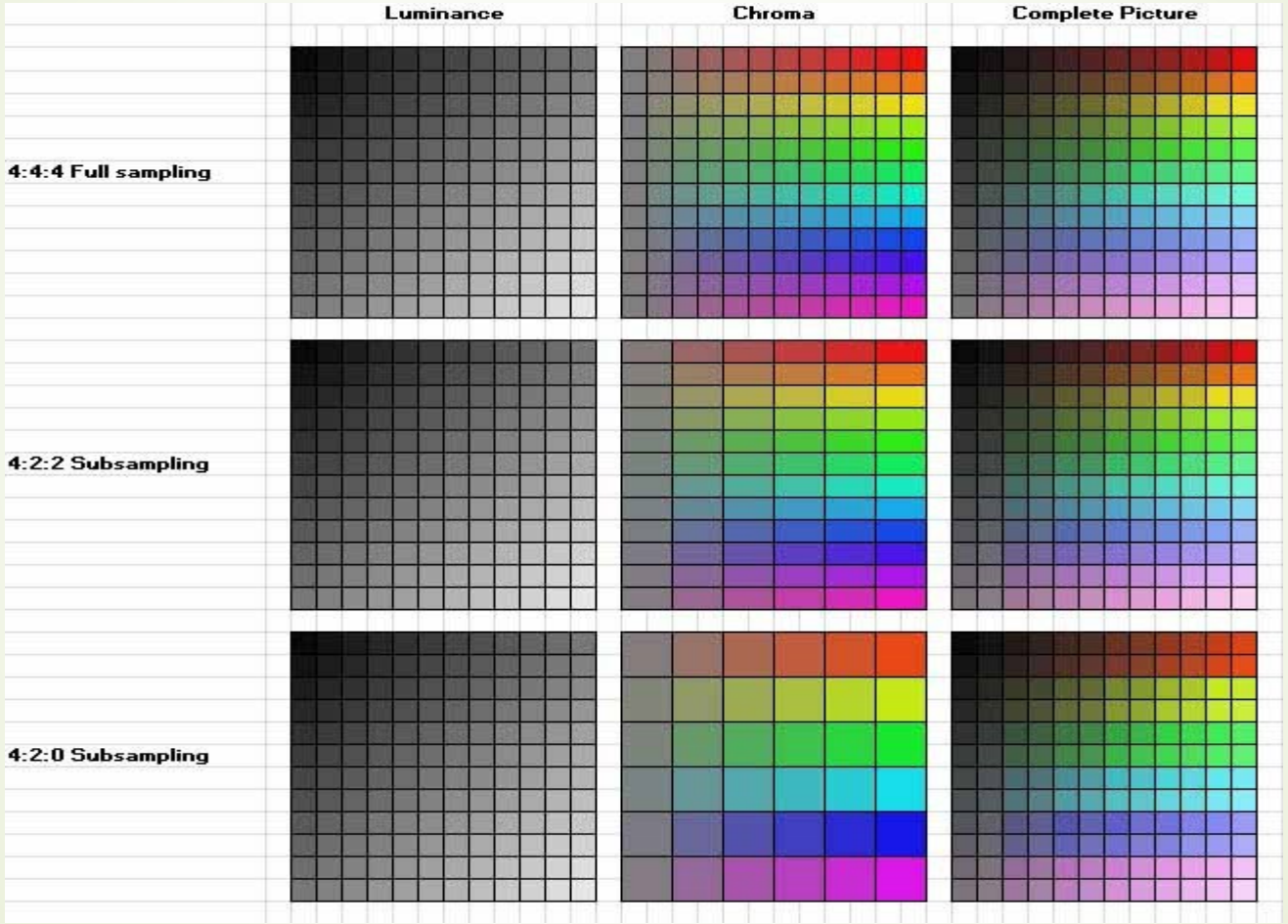
Image compression using JPEG



- ***J*** stands for the width of the conceptual region (index), in this example is equal to 4
- ***a*** is the number of chrominance (color) samples in the first row of ***J*** pixels.
- ***b*** represents the number of chrominance in the second row compared to the first row i.e., number of changes in color vertically.

The chroma (color) channels subsampling schemes. We can notice that achromatic channel (Y) is not samlped. In addition, we can see that quality of image kept roughly unchanged with the increasing samling ratio (the image contrast does not change a lot) e.g., for the scheme 4:2:0, where both horizantal and vertical resolution are subsampled by half, the image seems to have roughly the same quality (compared to 4:4:4 scheme i.e., no subsamlping is done). To confirm that, we see how the subsampling looks like when dealing with larger images.

Image compression using JPEG



The effect of chroma (color) channels subsampling on a large image.

Image compression using JPEG

JPEG bases mainly on the use of dicrete cosine transform (DCT). The first step in DCT is to split the image into 8x8 blocks.

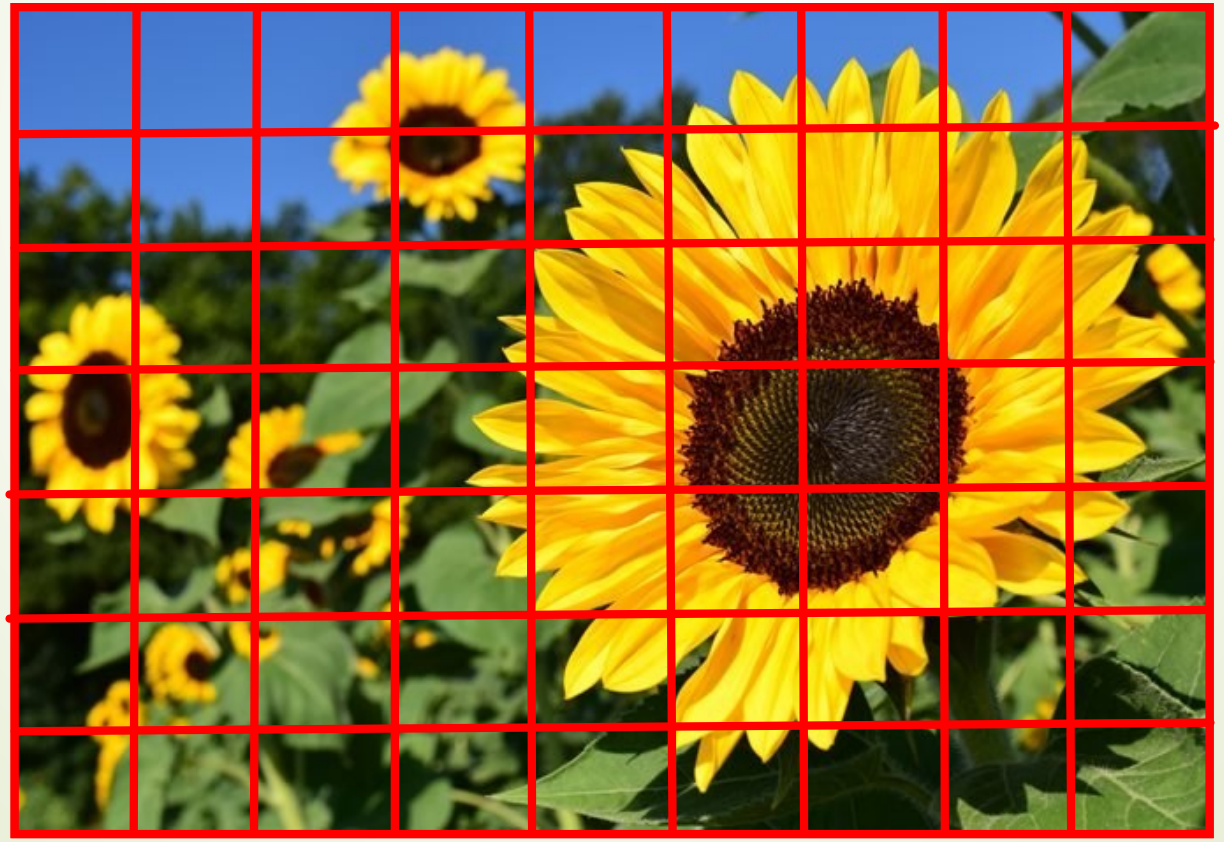


Image compression using JPEG

Any given signal can be break down to some set of elementary signals

+

+

+

=

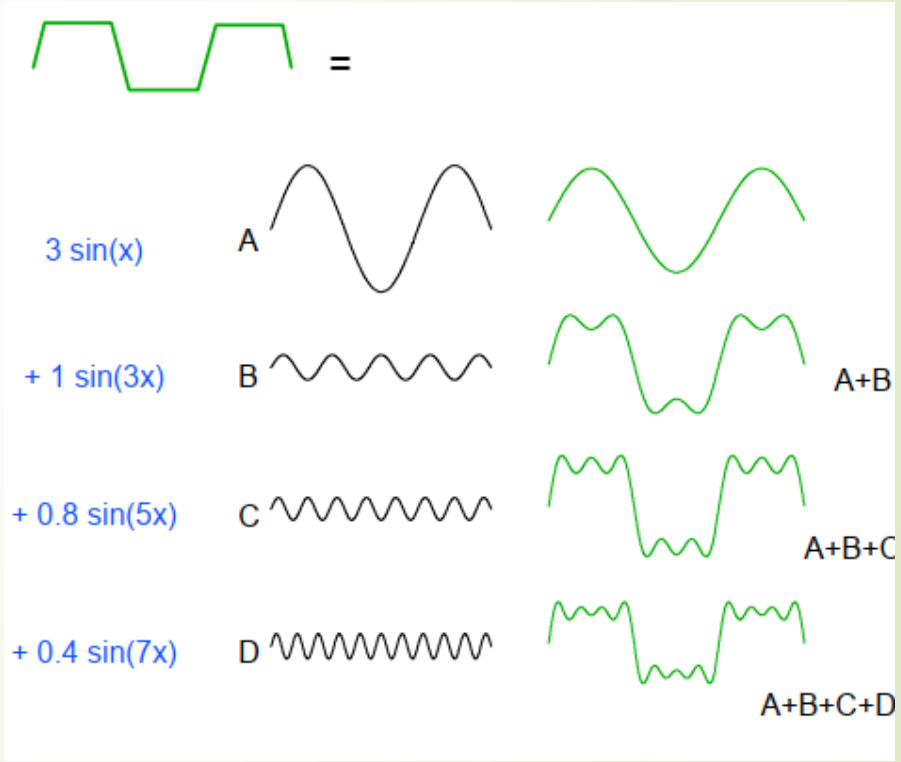
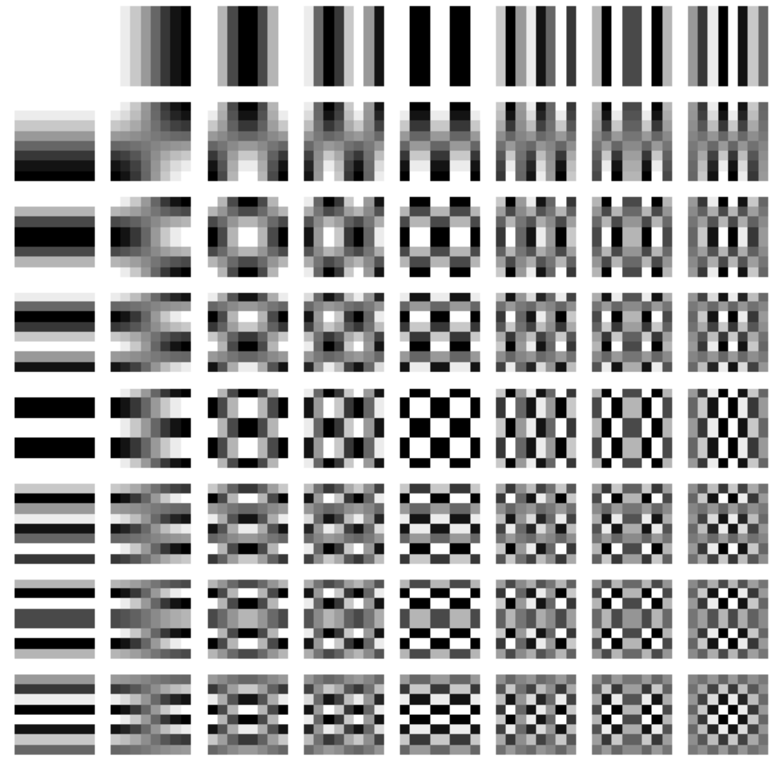


Image compression using JPEG

Similarly for the case of 2D images, any image can be expressed as a linear combination of bases functions



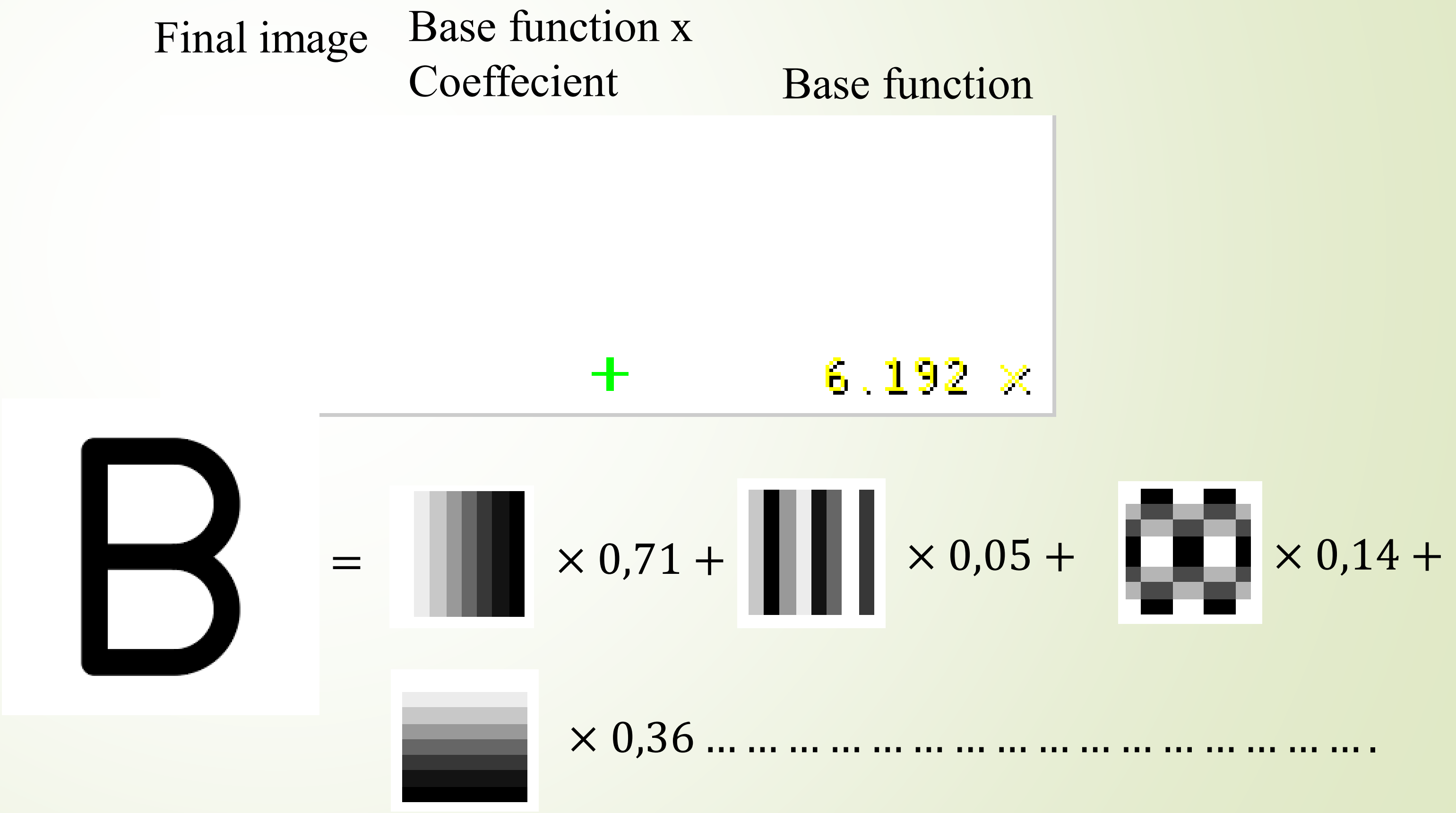


Image compression using JPEG

Another illustration

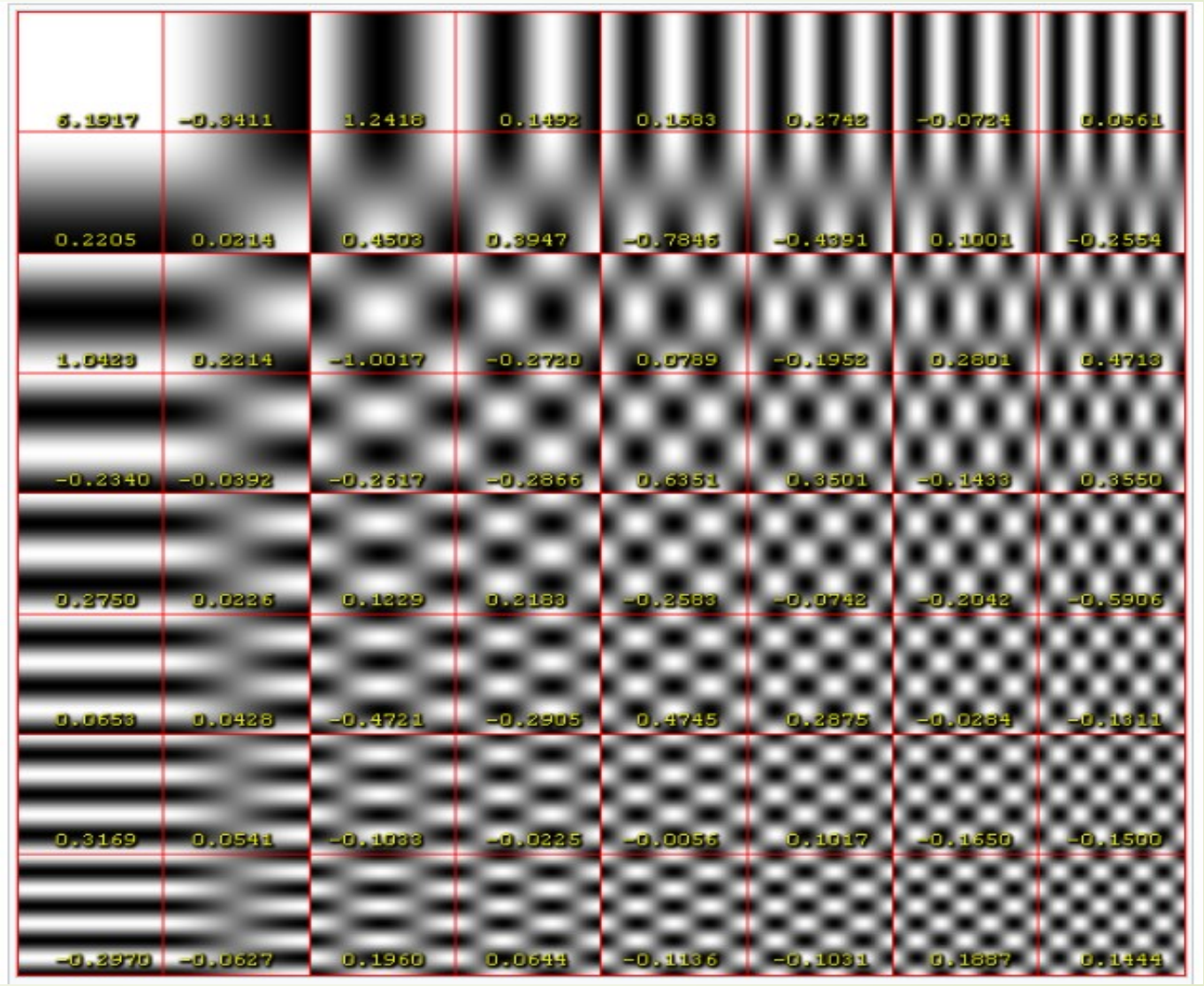
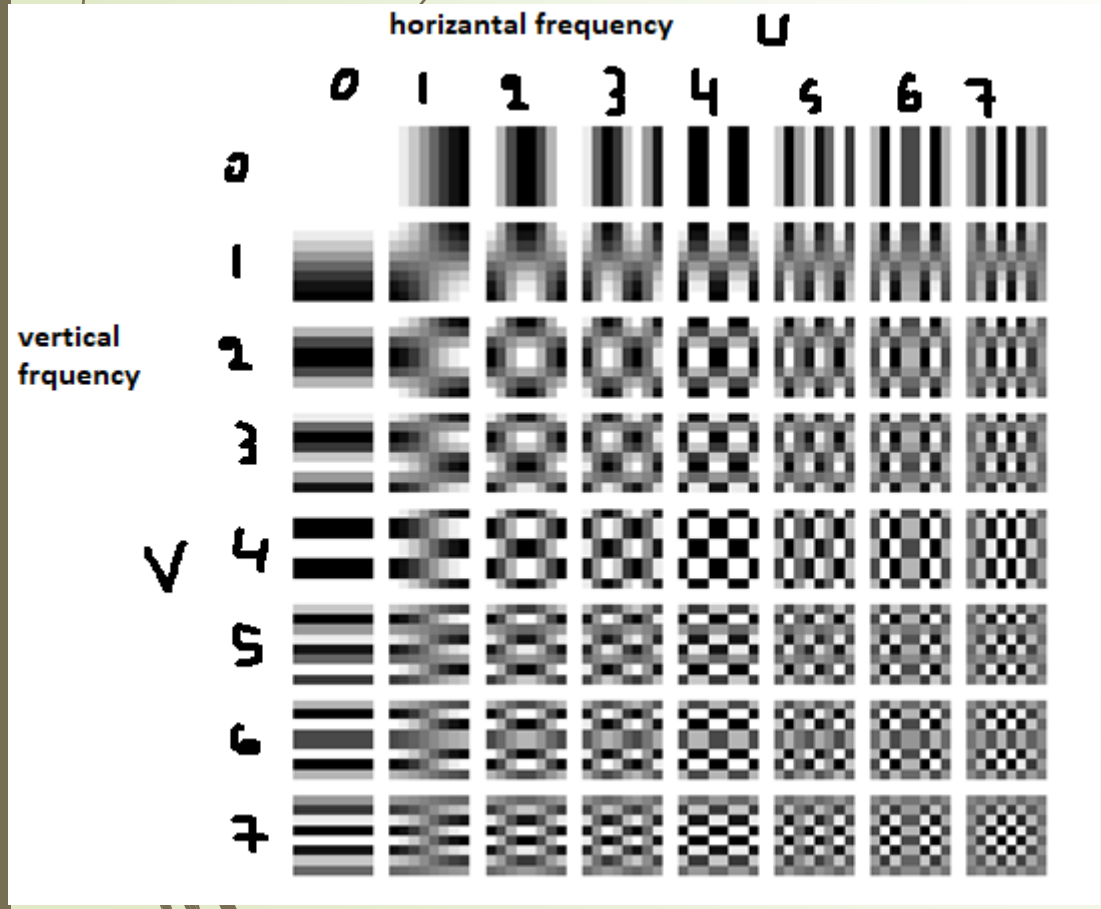
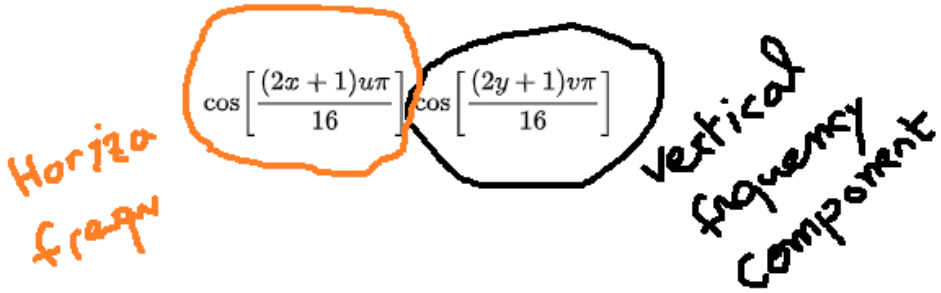


Image compression using JPEG

How do these bases generated ?



$$\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

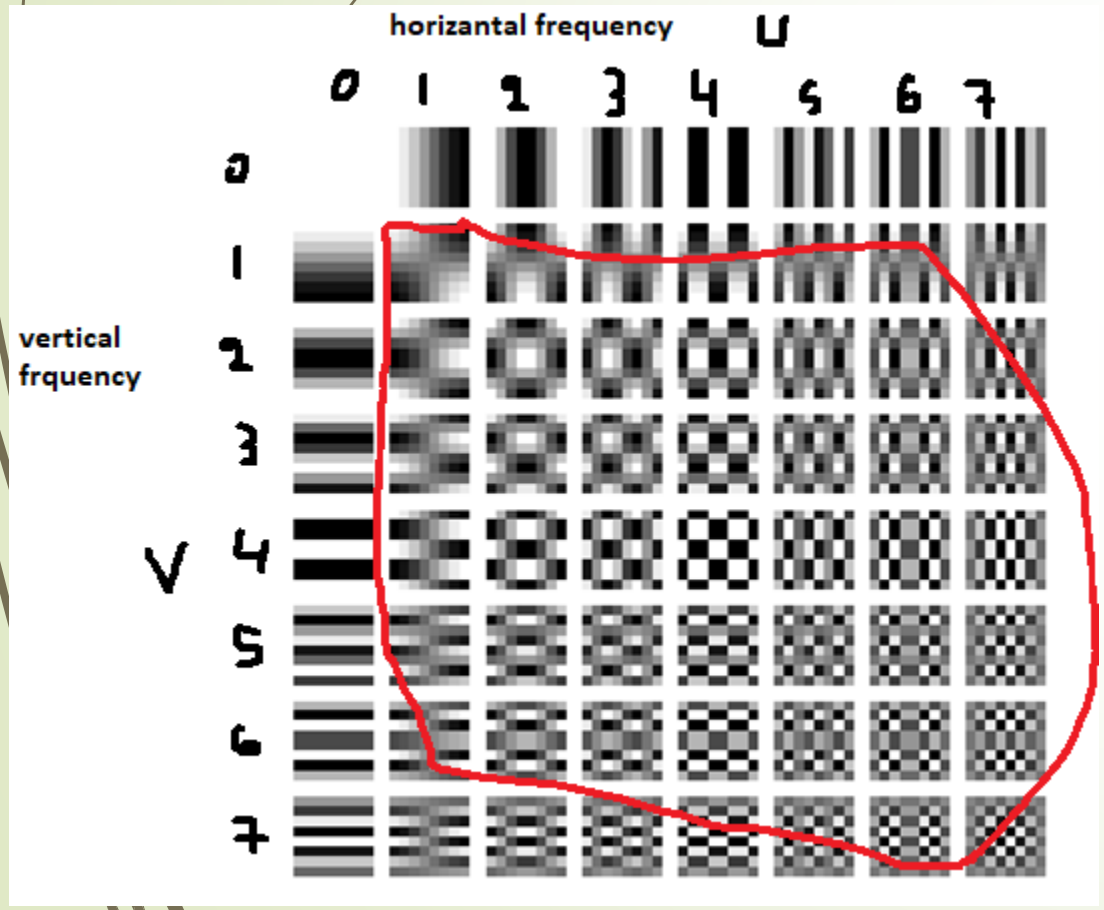


$$0 \leq u < 8$$

$$0 \leq v < 8$$

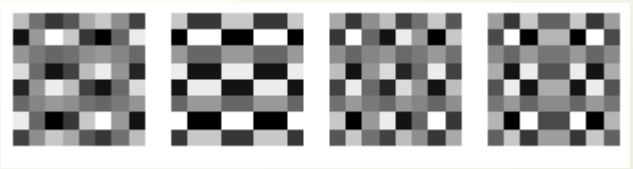
Image compression using JPEG

Each entry is the product of vertical and horizantal frequnecies



By going towards the bottom right, we get more components with a high frequency (i.e., rapid change in intensity)

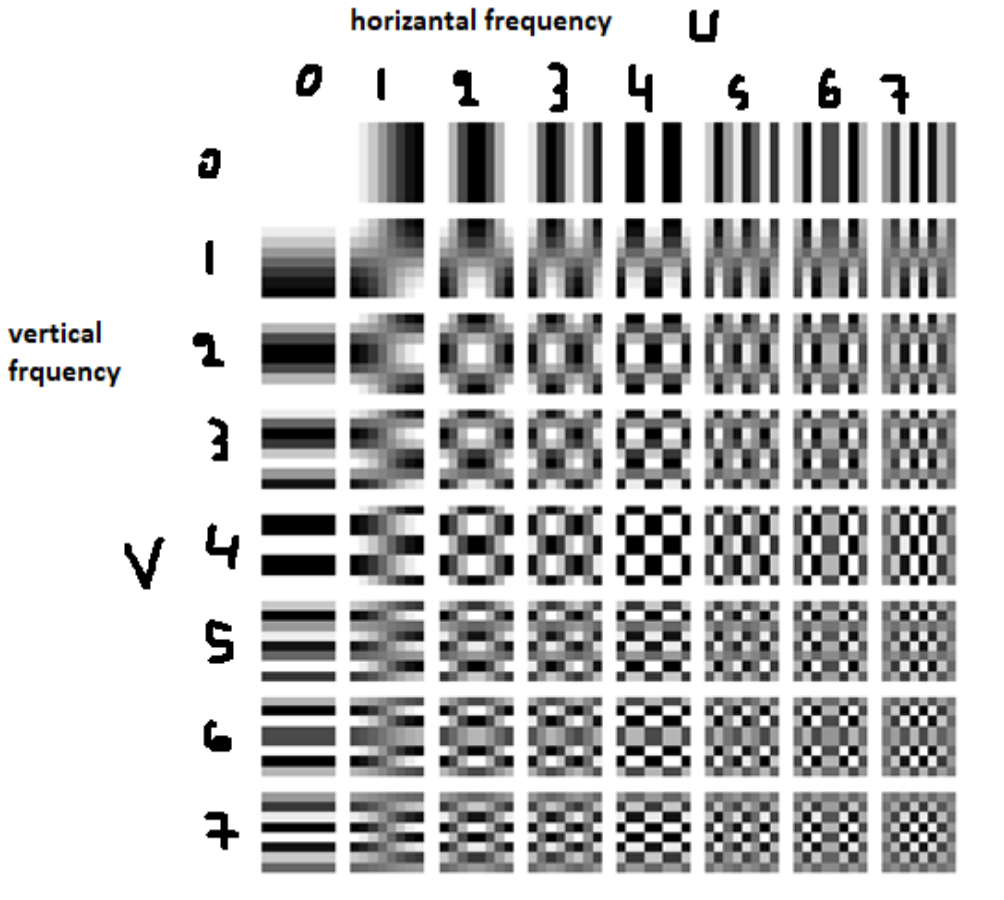
Low frequency components



High frequency components

Image compression using JPEG

Sample code for generating DCT bases




```
function bank = Generate_DCT_filter_bank(blockSize)

for u=0:(blockSize-1)
    for v=0:(blockSize-1)
        for x=0:(blockSize-1)
            for y=0:(blockSize-1)
                filter(x+1,y+1)= cos(((2*x+1)*u*pi)/16)   * cos(((2*y+1)*v*pi)/16);
            end
        end
        bank{u+1,v+1} = filter;
    end
end

end
```

Image compression using JPEG

How can we generate the DCT coeffecients

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} - 128$$

$$g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

($x \rightarrow$, $y \downarrow$)

$$G_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

($u \rightarrow$, $v \downarrow$)

- u is the horizontal spatial frequency, for the integers $0 \leq u < 8$.
- v is the vertical spatial frequency, for the integers $0 \leq v < 8$.
- $\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } u = 0 \\ 1, & \text{otherwise} \end{cases}$ is a normalizing scale factor to make
- $g_{x,y}$ is the pixel value at coordinates (x, y)
- $G_{u,v}$ is the DCT coefficient at coordinates (u, v).

Image compression using JPEG

Quantization

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

$$B_{j,k} = \text{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0, 1, 2, \ldots, 7; k = 0, 1, 2, \ldots, 7$$

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Image compression using JPEG

Quantization

As has already been mentioned, the human eye is sensitive to the difference in brightness, but within a relatively large region, because typically the eye is not able to detect the small variation in brightness over a small region. This is exactly the **cornerstone** of the JPEG algorithm, as this will allow us to significantly reduce the amount of information by eliminating the high frequency components. Through many subjective experiments by considering the human visual system, the JPEG has adopted this quantization matrix.

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

It evidently appears that the top left corner has lower values compared to the other sides of the matrix, as we target to keep the low frequency components (located in the top left) and omit the high frequency components.

Image compression using JPEG

Quantization

The quantization matrix used in the last step is at level equal to 50, which is a good compromise between compression ratio and image quality. Now, if we want less compression and higher quality, we can opt for a quality **level > 50**. In the contrary, if need a higher compression and less image quality, we can consider a quality **level < 50**.

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

$$Q_{10} = \begin{bmatrix} 80 & 60 & 50 & 80 & 120 & 200 & 255 & 255 \\ 55 & 60 & 70 & 95 & 130 & 255 & 255 & 255 \\ 70 & 65 & 80 & 120 & 200 & 255 & 255 & 255 \\ 70 & 85 & 110 & 145 & 255 & 255 & 255 & 255 \\ 90 & 110 & 185 & 255 & 255 & 255 & 255 & 255 \\ 120 & 175 & 255 & 255 & 255 & 255 & 255 & 255 \\ 245 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

$$Q_{90} = \begin{bmatrix} 3 & 2 & 2 & 3 & 5 & 8 & 10 & 12 \\ 2 & 2 & 3 & 4 & 5 & 12 & 12 & 11 \\ 3 & 3 & 3 & 5 & 8 & 11 & 14 & 11 \\ 3 & 3 & 4 & 6 & 10 & 17 & 16 & 12 \\ 4 & 4 & 7 & 11 & 14 & 22 & 21 & 15 \\ 5 & 7 & 11 & 13 & 16 & 12 & 23 & 18 \\ 10 & 13 & 16 & 17 & 21 & 24 & 24 & 21 \\ 14 & 18 & 19 & 20 & 22 & 20 & 20 & 20 \end{bmatrix}$$

$$Q_{new} = \frac{50}{Q_L} \times Q_{50}$$

$$Q_{10}(1,1) = \frac{50}{10} \times 16 = 80$$

$$Q_{new} = (100 - Q_L)/50 \times Q_{50}$$

$$Q_{90}(1,1) = \frac{100 - 90}{50} \times Q_{50}(1,1) = 3.2$$

Image compression using JPEG

Zig-Zag ordering

The zigzag scan is used to map the 8x8 matrix to a 1x64 vector. Zigzag scanning is used to group low-frequency coefficients to the top level of the vector and the high coefficient to the bottom. To remove the large number of zero in the quantized matrix, the zigzag matrix is used.

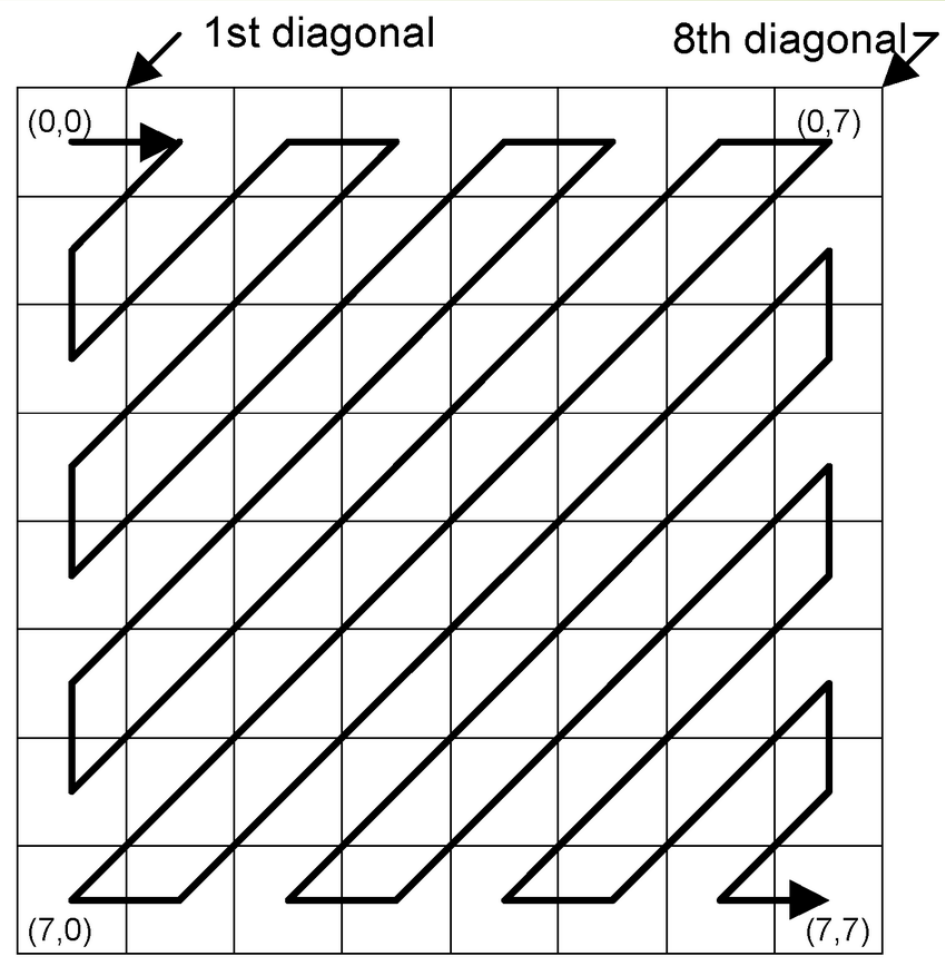


Image compression using JPEG

DCT coefficients encoding

- In this step, different DCT coefficients are encoded differently. As for the top left DC (direct current) coefficient (which typically large), it is firstly encoded using the different pulse code modulation (DPCM). The principle is simple **smaller numbers = fewer bits.**

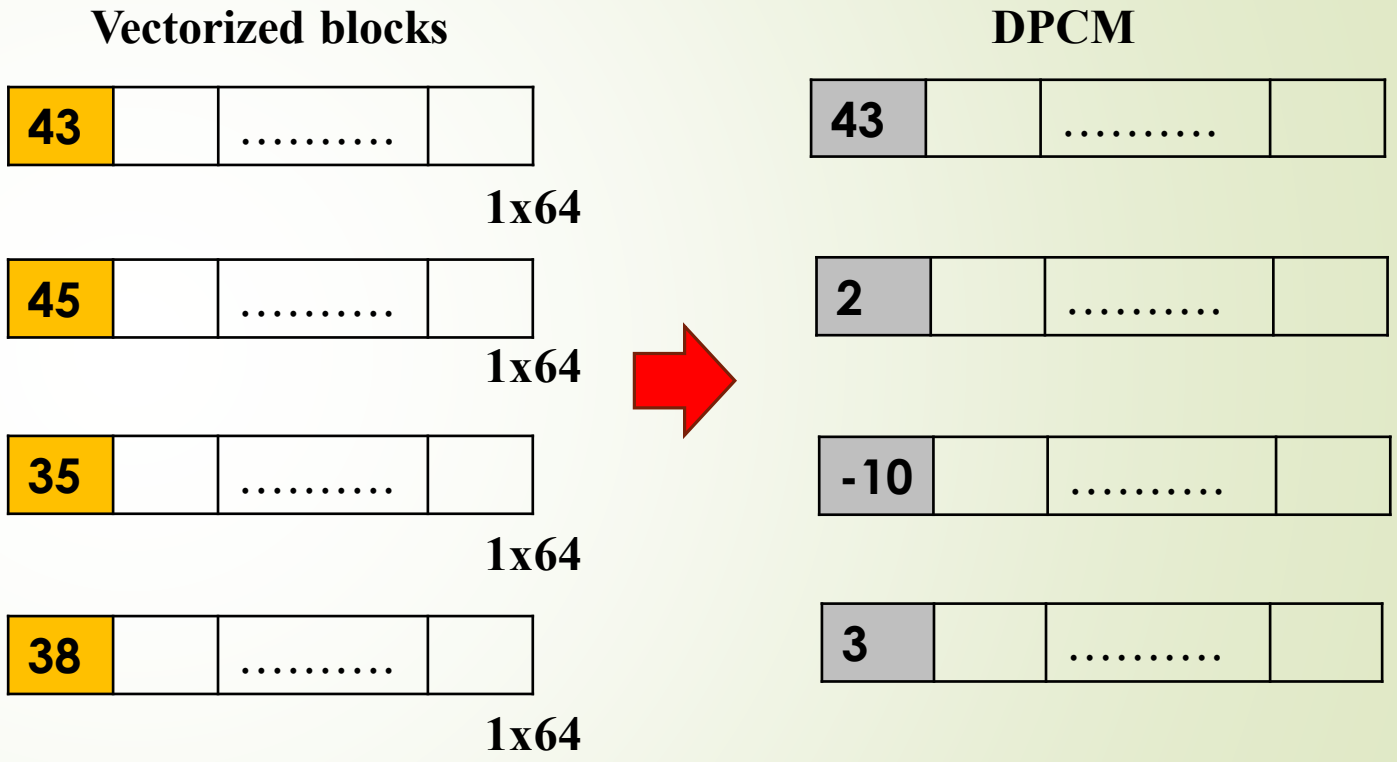


Image compression using JPEG

DCT coefficients encoding

- In fact, the normalization (subtract 128) is usually applied if we are not interested by the DPCM process. This is because if we do not apply the DPCM, the DC coefficients will be so large (need more bits), thus, applying the normalization (-128) will reduce the DC coefficients.
- Applying the both techniques (-128 and DPCM) is beneficial because the first brings the mean of values to zero (roughly), and the second will produce more zeros because DC coefficients are usually close to each other.

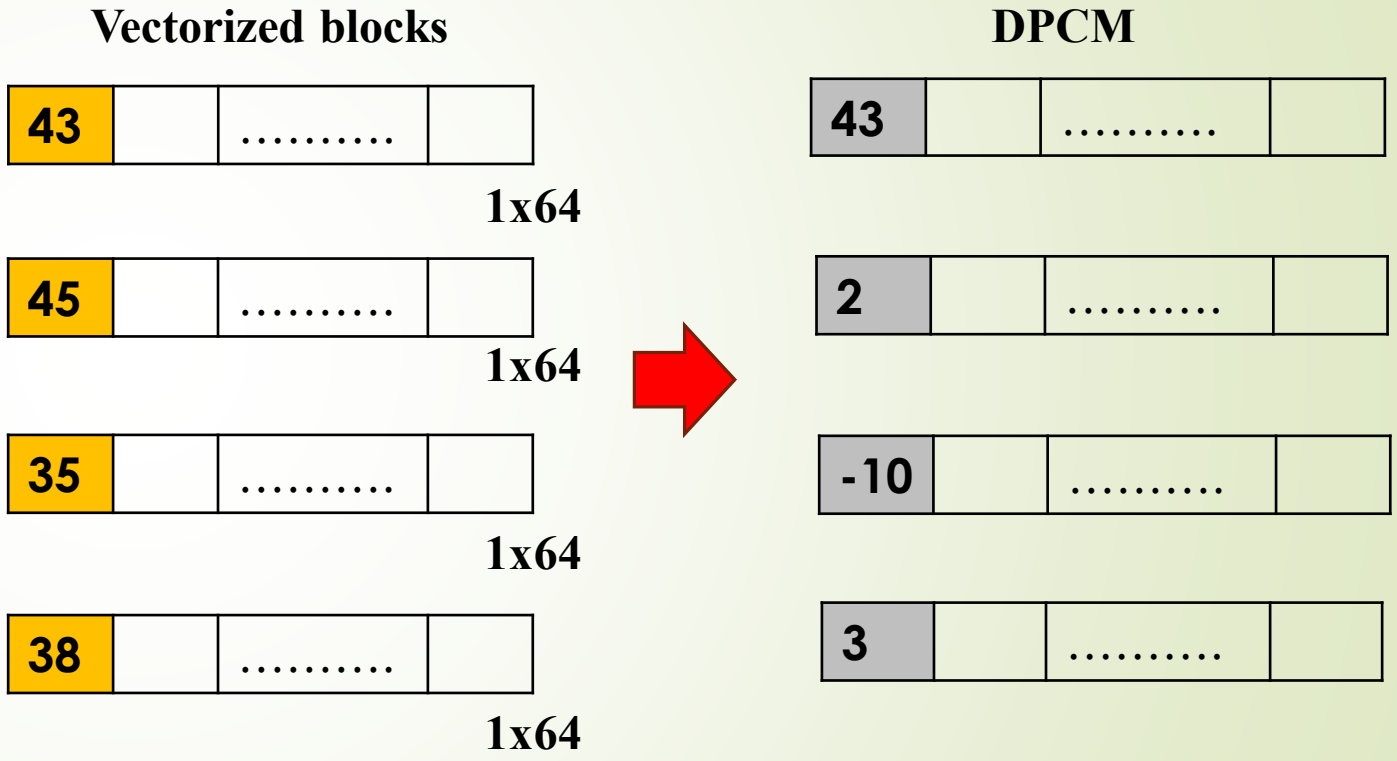


Image compression using JPEG

DCT coefficients encoding

- The AC (alternating current) coefficients are encoded using the runlength encoding scheme.

[(run-length, size), value]

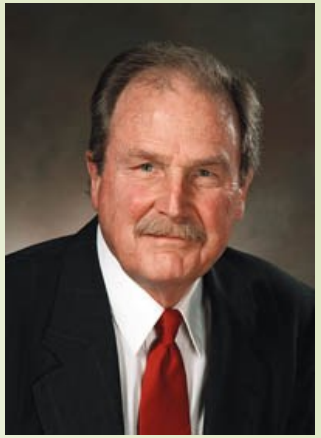
- **run-length:** number of zeros before the concerned number
- **Size:** number of bits needed to encode the number
- **Value:** the actual value of the number

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[(0, 2), -3]
[(1, 2), -3]
[(0, 2), -2]
[(0, 3), -6]
……
[(0, 0)]

[(0, 0)] = EOB (end of block)

Image compression using JPEG

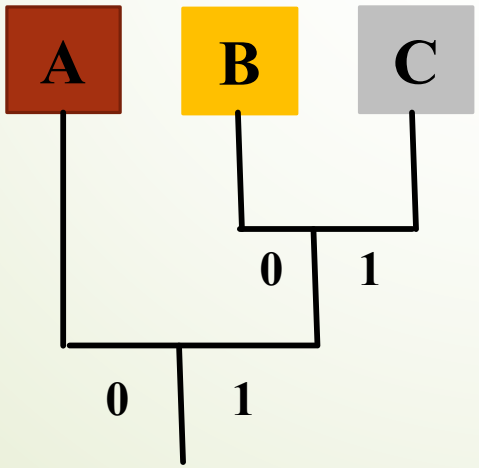


Huffman encoding

- The next step is to apply Huffman encoding on both AC and DC coefficients.
- Huffman coding is based on representing the most frequent patterns with a low number of bits, whereas, representing the less frequent patterns with a higher number of bits.
- Given the following text: **'ABAABBABCACAC'**, this text is made up of three elements 'A', 'B' and 'C'. Now, to represent this text, we need at least **2 bits** (**A: 00, B: 01, C:10**), if we use the ASCII code we need **7 bits** for each element. To encode this text we need: 2 x 13 (element) = **26 bits.**

- **Huffman coding**

Element	Frequency
A	6
B	4
C	3



Element	Representation
A	0
B	01
C	11

To encode this text using Huffman: **6** x 1 + **4** x 2 + **3** x 2 = **20 bits**

Image compression using JPEG

Huffman encoding

- Indeed, we can encode both DC and AC coefficients using Huffman codes that are generated from the data. In this case, we need two passes on our data, the first pass is to count the frequency and the second pass to encode the data. To reduce the compression time, JPEG provide us with standard Huffman tables we can use for data encoding.

SIZE	Code Length	Code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Huffman Table for DC component SIZE field

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,..., -4, 4,..., 7	000,..., 011, 100,...111
4	-15,..., -8, 8,..., 15	0000,..., 0111, 1000,..., 1111
.		.
.		.
11	-2047,..., -1024, 1024,... 2047	...

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
... 15/A	More	Such rows

Partial Huffman Table for AC Run/Size Pairs

Image compression using JPEG

Huffman encoding for DC coefficients **(size, value)**

- Suppose we are given a DC coefficient that is equal to **-7**
- From the table of size / value, we can see that (-7) can be represented by **000**
- The size of this value from the same table is **3**
- From the table of size / code length we can see that size **3** correponds to **100,** thus, the final representation will be **100 000**

SIZE	Code Length	Code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Huffman Table for DC component SIZE field

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,..., -4, 4,..., 7	000,..., 011, 100,...111
4	-15,..., -8, 8,..., 15	0000,..., 0111, 1000,..., 1111
.		.
.		.
11	-2047,..., -1024, 1024,... 2047	...

Image compression using JPEG

Huffman encoding for AC coefficients ((Run-length, size), value)

- Suppose we are given an AC coefficient that is equal to **[(0,3),-8]**
- From the table of run-length / size, we can see that (**0,3**) can be represented by **100**
- From the table of size / value, we can see that **-8** is represented by **0111**
- Thus, the final representation will be **100 0111**
- The final bit stream of the block will look like **100 000 100 0111 ……..1010** (we can compute the number of bits before and after).

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,…, -4, 4,…, 7	000,…, 011, 100,…111
4	-15,…, -8, 8,…, 15	0000,…, 0111, 1000,…, 1111
.		.
.		.
11	-2047,…, -1024, 1024,… 2047	…

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
… 15/A	More	Such rows

Partial Huffman Table for AC Run/Size Pairs

Image compression using JPEG

Huffman encoding for AC coefficients ((Run-length, size), value)

- Suppose we are given an AC coefficient that is equal to **[(0,3),-8]**
- From the table of run-length / size, we can see that (**0,3**) can be represented by **100**
- From the table of size / value, we can see that **-8** is represented by **0111**
- Thus, the final representation will be **100 0111**
- The final bit stream of the block will look like **100 000 100 0111 ……..1010** (we can compute the number of bits before and after).

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,…, -4, 4,…, 7	000,…, 011, 100,…111
4	-15,…, -8, 8,…, 15	0000,…, 0111, 1000,…, 1111
.		.
.		.
11	-2047,…, -1024, 1024,… 2047	…

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
… 15/A	More	Such rows

Partial Huffman Table for AC Run/Size Pairs

Image compression using JPEG

Huffman encoding

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,..., -4, 4,..., 7	000,..., 011, 100,...111
4	-15,..., -8, 8,..., 15	0000,..., 0111, 1000,..., 1111
.		.
.		.
11	-2047,..., -1024, 1024,... 2047	...

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
... 15/A	More	Such rows

Partial Huffman Table for AC Run/Size Pairs

- We can see that no code was assigned to the value 0, because DC compnents are typically large values, and in AC components zeros are skipped by run-length encoding.

- Note also that each of ***run*** and ***size*** are represented using 4 bits, which means that they can go until 15! But what to do if we have more than 15 zeros (e.g., 17) before a no-zero AC component? JPEG has defined a special Huffman code for this case: ***run*** /***size*** is encoded as **(15,0)(0)**

Image compression using JPEG

Decoding: return back to the original subimage

The quantized subimage

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Entry-wise product by quantization matrix

Which is resemble the DCT matrix coefficients

$$\begin{bmatrix} -416 & -33 & -60 & 32 & 48 & -40 & 0 & 0 \\ 0 & -24 & -56 & 19 & 26 & 0 & 0 & 0 \\ -42 & 13 & 80 & -24 & -40 & 0 & 0 & 0 \\ -42 & 17 & 44 & -29 & 0 & 0 & 0 & 0 \\ 18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Using the inverse DCT

Input= coeff. Matrix, output= image values

$$f_{x,y} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} \alpha(u)\alpha(v) F_{u,v} \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]$$

- x is the pixel row, for the integers $0 \leq x < 8$.
- y is the pixel column, for the integers $0 \leq y < 8$.
- $\alpha(u)$ is defined as above, for the integers $0 \leq u < 8$.
- $F_{u,v}$ is the reconstructed approximate coefficient at coordinates (u, v).
- $f_{x,y}$ is the reconstructed pixel value at coordinates (x, y)

Rounding the values

$$\begin{bmatrix} -66 & -63 & -71 & -68 & -56 & -65 & -68 & -46 \\ -71 & -73 & -72 & -46 & -20 & -41 & -66 & -57 \\ -70 & -78 & -68 & -17 & 20 & -14 & -61 & -63 \\ -63 & -73 & -62 & -8 & 27 & -14 & -60 & -58 \\ -58 & -65 & -61 & -27 & -6 & -40 & -68 & -50 \\ -57 & -57 & -64 & -58 & -48 & -66 & -72 & -47 \\ -53 & -46 & -61 & -74 & -65 & -63 & -62 & -45 \\ -47 & -34 & -53 & -74 & -60 & -47 & -47 & -41 \end{bmatrix}$$

Adding 128

$$\begin{bmatrix} 62 & 65 & 57 & 60 & 72 & 63 & 60 & 82 \\ 57 & 55 & 56 & 82 & 108 & 87 & 62 & 71 \\ 58 & 50 & 60 & 111 & 148 & 114 & 67 & 65 \\ 65 & 55 & 66 & 120 & 155 & 114 & 68 & 70 \\ 70 & 63 & 67 & 101 & 122 & 88 & 60 & 78 \\ 71 & 71 & 64 & 70 & 80 & 62 & 56 & 81 \\ 75 & 82 & 67 & 54 & 63 & 65 & 66 & 83 \\ 81 & 94 & 75 & 54 & 68 & 81 & 81 & 87 \end{bmatrix}$$

Reconstructed image which approximates The original subimage (with an error margin)

Image compression using JPEG

Decoding: return back to the original image

Original image

Reconstructed image

$$\text{Compression ratio} = \frac{\text{size of original image}}{\text{size of compressed image}}$$

For an image of 128x256, we need **(128x256x8)/8 = 32768** bytes

- ***If*** the size of the image after compression is 15632 bytes = 2.09 ≈ 2 (written also as **2:1** i.e., for each 2 bytes in the original image we have 1 byte in the compressed image).

- ***If*** the size of the image after compression is 9042 bytes = 3.62 ≈ 4 (written also as **4:1** i.e., for each 4 bytes in the original image we have 1 byte in the compressed image).

In color images, the compression procedure is applied on each channel separately

Image compression using JPEG

Effect of artifacts

Artifacts appear after the **lossy** compression process are noticeable distortions of image. In the figure below, artifacts are increased from right to left. Artifacts are proportional to the compression ratio and inversely proportional to the image quality.

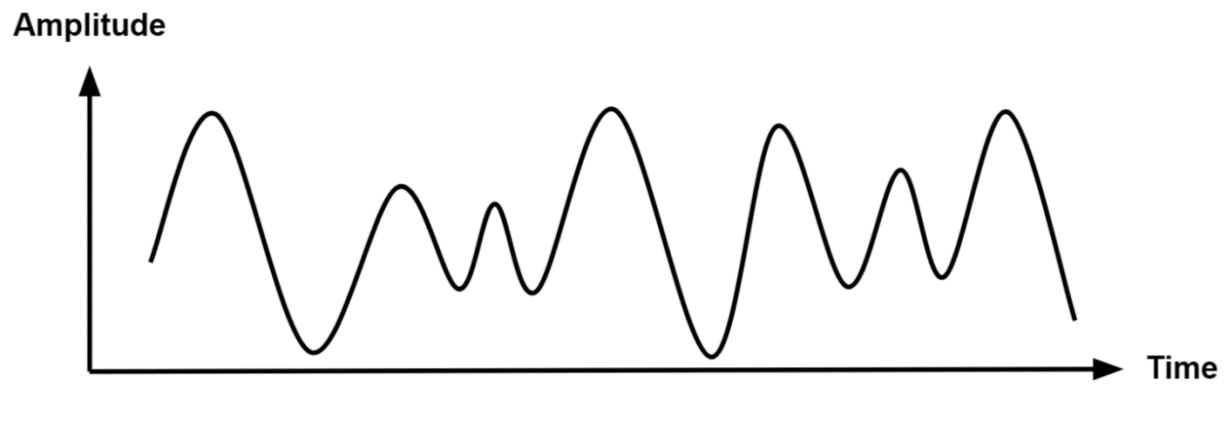
Compression

Quality

Fourier transform

What is Fourier transform

As it has already mentioned, transformation like DCT and Fourier transform the signal (e.g., image) from time domain (in our case image is in spatial domain) to frequency domain i.e., express the signal (e.g., image) as a combination of frequencies. Speech of a person can recorded as a **continuous** signal over time.

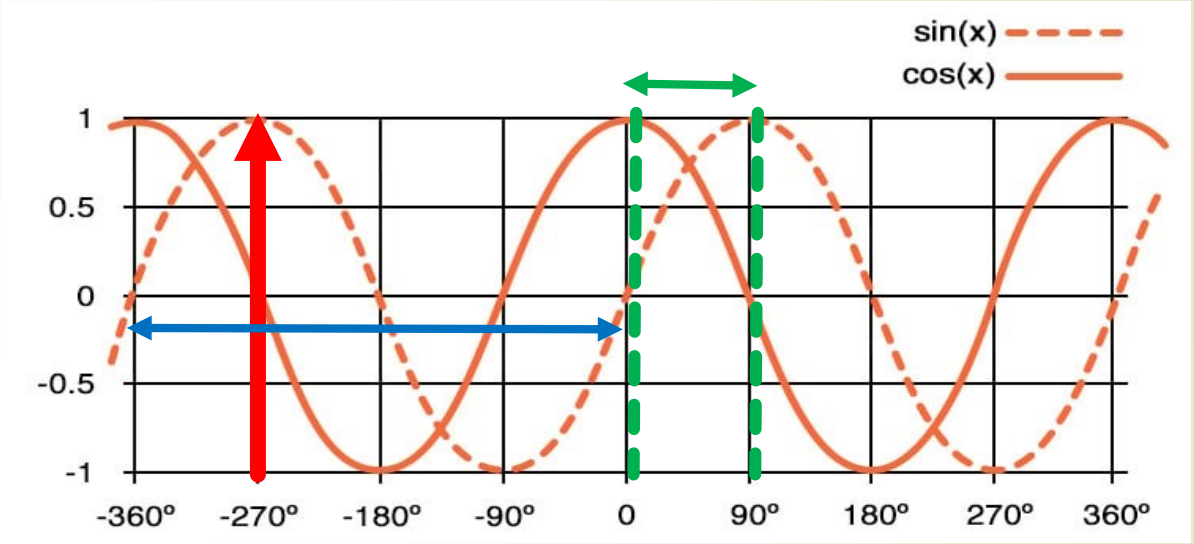


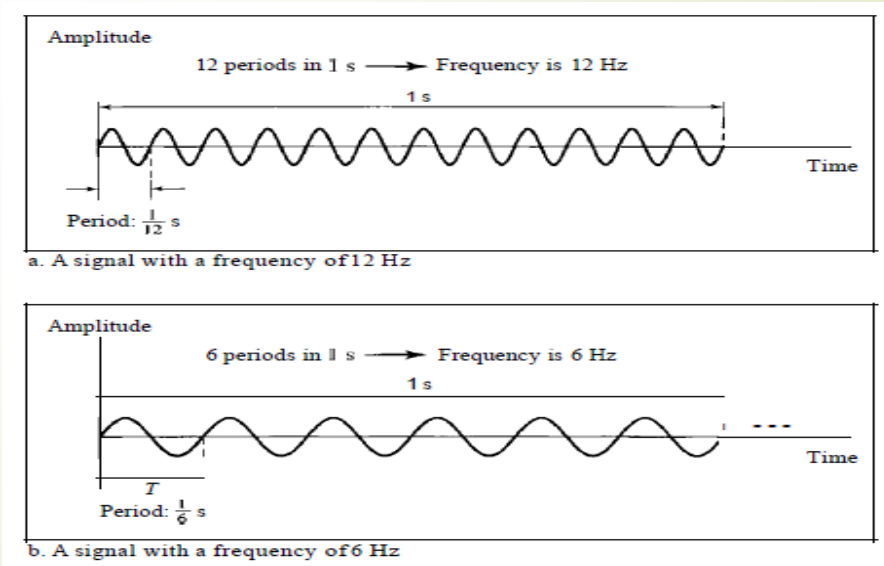
Fourier transform

Signal properties

A signal has several properties, among which

- Amplitude: represents the signal strength
- Phase: the amount of signal shifting
- Period: when the signal complete one cycle
- Frequency: it is equal to (1 / time of one period), e.g., a signal complete one cycle in 0.2 seconds and another one complete the same cycle in 1 second.

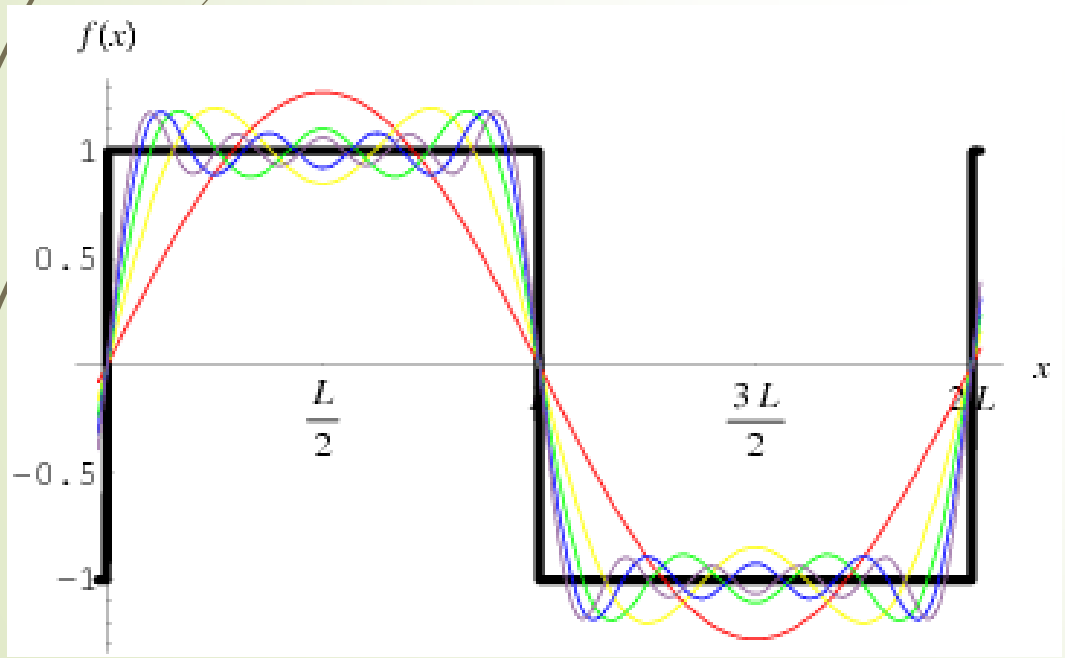


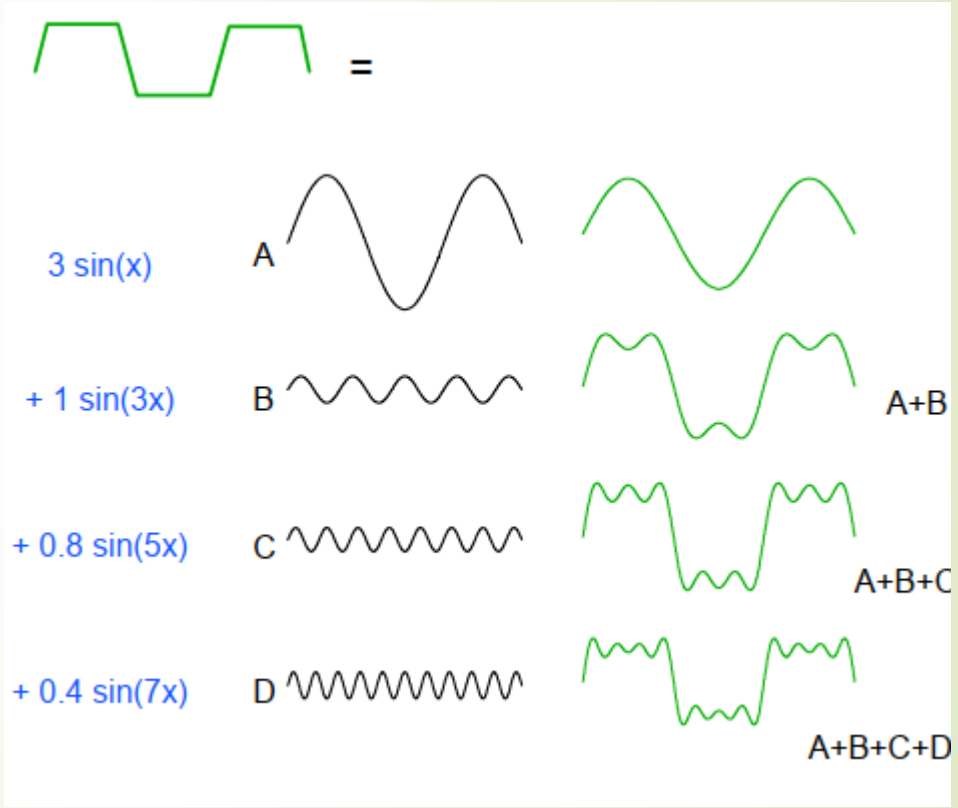


Fourier transform

Fourier transform

Similar to the principle of DCT, Fourier transform expresses a specific signal as a weighted sum of sinusoids **(cos / sin)** i.e., linear combination of sinusoids.





Fourier transform

1D Fourier transform

The 1D Fourier transform is given by

$$X_f = \int_{-\infty}^{+\infty} x_t e^{-j2\pi ft}$$

$[-\infty \ +\infty]$ because the signal is continuous
j stands for the complex representation to facilitate the integral calculation
f represents the frequency, it is equal to 1/T (T is the period)
t is the time factor
X_f is a complex number

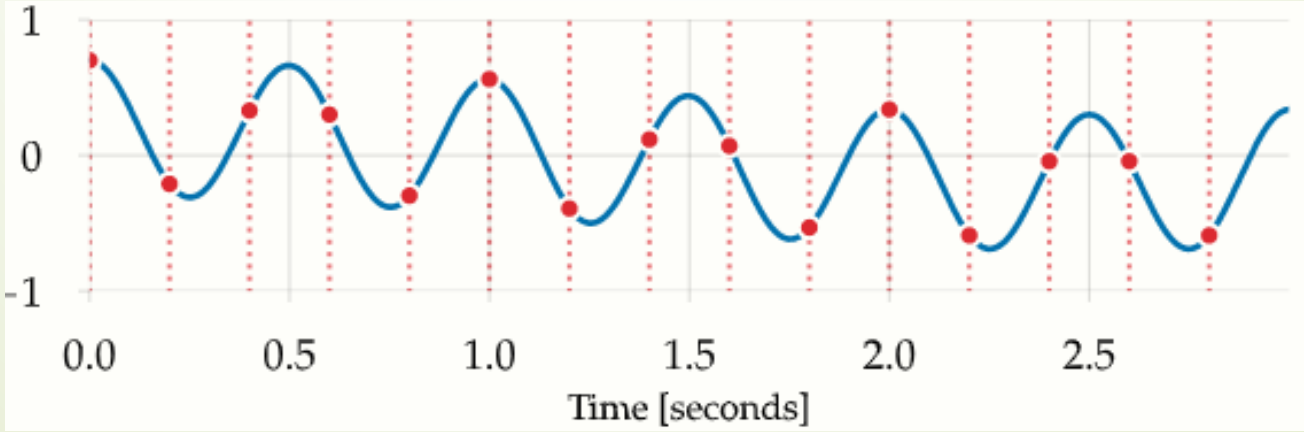
Fourier transform

1D Fourier transform

The discrete version of Fourier transform is given by

$$X_k = \sum_{i=0}^{N-1} x_i e^{-j2\pi ki/N}$$

In the discrete case, we sample N samples from the signal being transformed. N should be a trade-off between number of points required to faithfully represent the signal and the method efficiency (i.e., high number = high calculations + high storage requirements).



Fourier transform

1D Fourier transform

The discrete version of Fourier transform is given by

$$X_k = \sum_{i=0}^{N-1} x_i e^{-j2\pi ki/N}$$

N is the number of samples
x_i is the sample value (sinusoids value) at position i
i: the current sample
k: index for the frequency being generated $k \epsilon [0 \;\; N-1]$
X_k is a complex number, it is called Fourier coefficient. Because it is based on x_i and sinusoids, this coefficient encodes information about the amplitude and phase.
$\boldsymbol{2\pi}$ because sinusoids need $\boldsymbol{2\pi}$ to complete a full cycle.

Fourier transform

1D Fourier transform

But, where is sinusoids in the Fourier expression $e^{-j2\pi ki/N}$

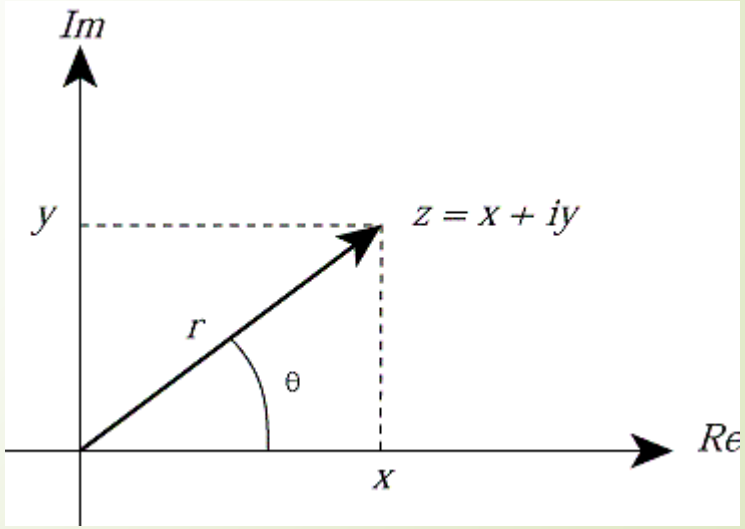
The above expression is written as Euler formula $e^{-j.x} = cos(x) + j\ sin(x)$

$$e^{-j2\pi ki/N} = cos(2\pi ki/N) + j\ sin(2\pi ki/N)$$

The relation between complex and polar coordinates

$$z = x + i\,y = r\,cos(\theta) + i\,r\ sin(\theta) = r\,[cos(\theta) + i\ sin(\theta)]$$

$z = re^{-\theta}$ (using Euler)



Fourier transform

1D Fourier transform (example)

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2-i \\ -i \\ -1+2i \end{pmatrix}.$$

Calculating the DFT of $\mathbf{x}$ using Eq.1

$$X_0 = e^{-i2\pi 0 \cdot 0/4} \cdot 1 + e^{-i2\pi 0 \cdot 1/4} \cdot (2-i) + e^{-i2\pi 0 \cdot 2/4} \cdot (-i) + e^{-i2\pi 0 \cdot 3/4} \cdot (-1+2i) = 2$$
$$X_1 = e^{-i2\pi 1 \cdot 0/4} \cdot 1 + e^{-i2\pi 1 \cdot 1/4} \cdot (2-i) + e^{-i2\pi 1 \cdot 2/4} \cdot (-i) + e^{-i2\pi 1 \cdot 3/4} \cdot (-1+2i) = -2-2i$$
$$X_2 = e^{-i2\pi 2 \cdot 0/4} \cdot 1 + e^{-i2\pi 2 \cdot 1/4} \cdot (2-i) + e^{-i2\pi 2 \cdot 2/4} \cdot (-i) + e^{-i2\pi 2 \cdot 3/4} \cdot (-1+2i) = -2i$$
$$X_3 = e^{-i2\pi 3 \cdot 0/4} \cdot 1 + e^{-i2\pi 3 \cdot 1/4} \cdot (2-i) + e^{-i2\pi 3 \cdot 2/4} \cdot (-i) + e^{-i2\pi 3 \cdot 3/4} \cdot (-1+2i) = 4+4i$$

results in $\mathbf{X} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -2-2i \\ -2i \\ 4+4i \end{pmatrix}.$

Fourier transform

2D discrete Fourier transform (2D DFT)

For an $M \times N$ image, the 2D DFT

$$X_d(k,L) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} x(i,j).\, e^{-j2\pi\frac{Ki}{M}+\frac{Lj}{N}}$$

Such that $x(i,j)$ is the pixel values of the image. The inverse transform is given by

$$x(i,j) = \sum_{K=0}^{M-1} \sum_{L=0}^{N-1} X_d(k,L).\, e^{+j2\pi\frac{Ki+Lj}{N.M}}$$

Fourier transform

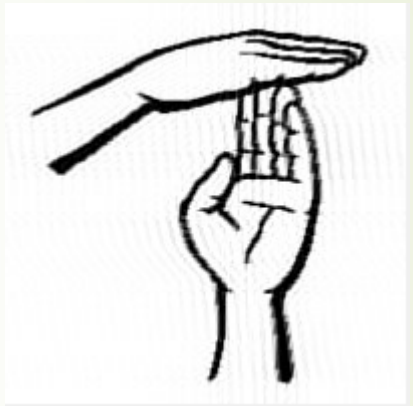
2D discrete Fourier transform (2D DFT)

Image reconstruction using 2D DFT

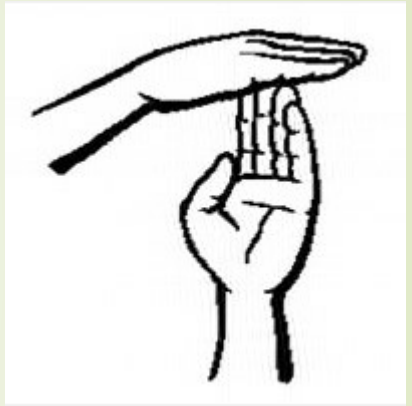
We consider the top left I(1:5,1:5), from the image of (200, 200)

We consider the top left I(1:10,1:10), from the image of (200, 200)

We consider the top left I(1:20,1:20), from the image of (200, 200)



We consider the top left I(1:35,1:35), from the image of (200, 200)



We consider the top left I(1:60,1:60), from the image of (200, 200)

we can see that the **low frequency** coefficients (i.e., top left coefficients : 0,1,…) contain information on the **general form** of the shape and the **high frequency** coefficients contain information on the **finer details** (rapid variation) of the shape. Low level frequency can reconstruct the image because they retain a high percentage of the signal energy, thus, the magnitude (amplitude) of low level frequency is typically large than high frequency amplitude.

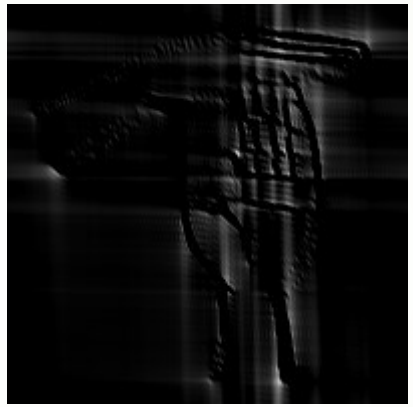
Fourier transform

2D discrete Fourier transform (2D DFT)

Image reconstruction using 2D DFT

We consider the bottom right I(195:200,195:200), from the image of (200, 200)

We consider the bottom right I(180:200,180:200), from the image of (200, 200)



We consider the bottom right I(140:200,140:200), from the image of (200, 200)

We consider the bottom right I(20:200,20:200), from the image of (200, 200)

We consider the bottom right I(5:200,5:200), from the image of (200, 200)

we can clearly notice that considering high frequency components yield minor reconstruction results. The white color of the picture is not recovered because the low frequency (0,0) which has the largest magnitude (high percentage of the signal energy) is not considered.

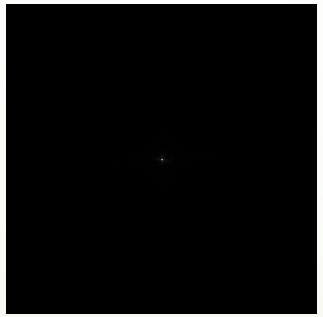
Fourier transform

2D discrete Fourier transform (2D DFT)

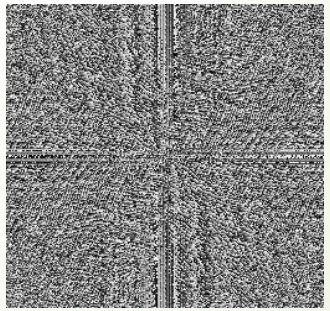
Because X_d is a complex number which encodes amplitude and phase information of a complex sinusoïdal component $e^{-j2\pi\frac{Ki}{M}+\frac{Lj}{N}}$, the amplitude and phase are calculated as follows

$$Amp = \sqrt{Re(X_d)^2+Im(X_d)^2} \qquad Ph = atan2\left(\frac{Im(X_d)}{Re(X_d)}\right)$$

The original image



Amplitude



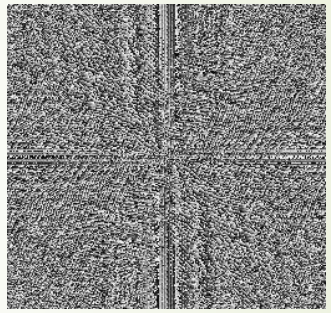
Phase

Fourier transform

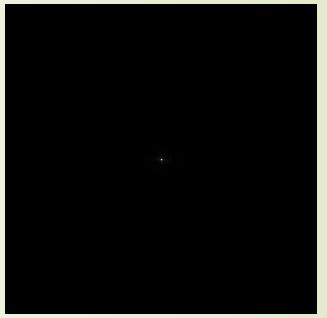
2D discrete Fourier transform (2D DFT)

- The phase image does not provide any significant information.
- We can see that the amplitude image is mostly black because most values are out of the range (the white dot in center is the DC coeffecient i.e., the coeff. with the highest value).
- Thus, values are normalized by applying the function $\boldsymbol{f(x,y) = log(1 + Amp(x,y))}$
- A common practice to improve the visualization of the amplitude image is to shift the low frequecny to the center after normalization. Swaps the first quadrant of image with the third, and the second quadrant with the fourth.

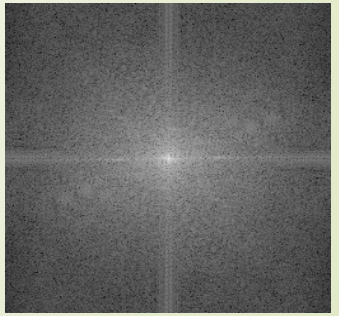
The original image



Phase



Amplitude

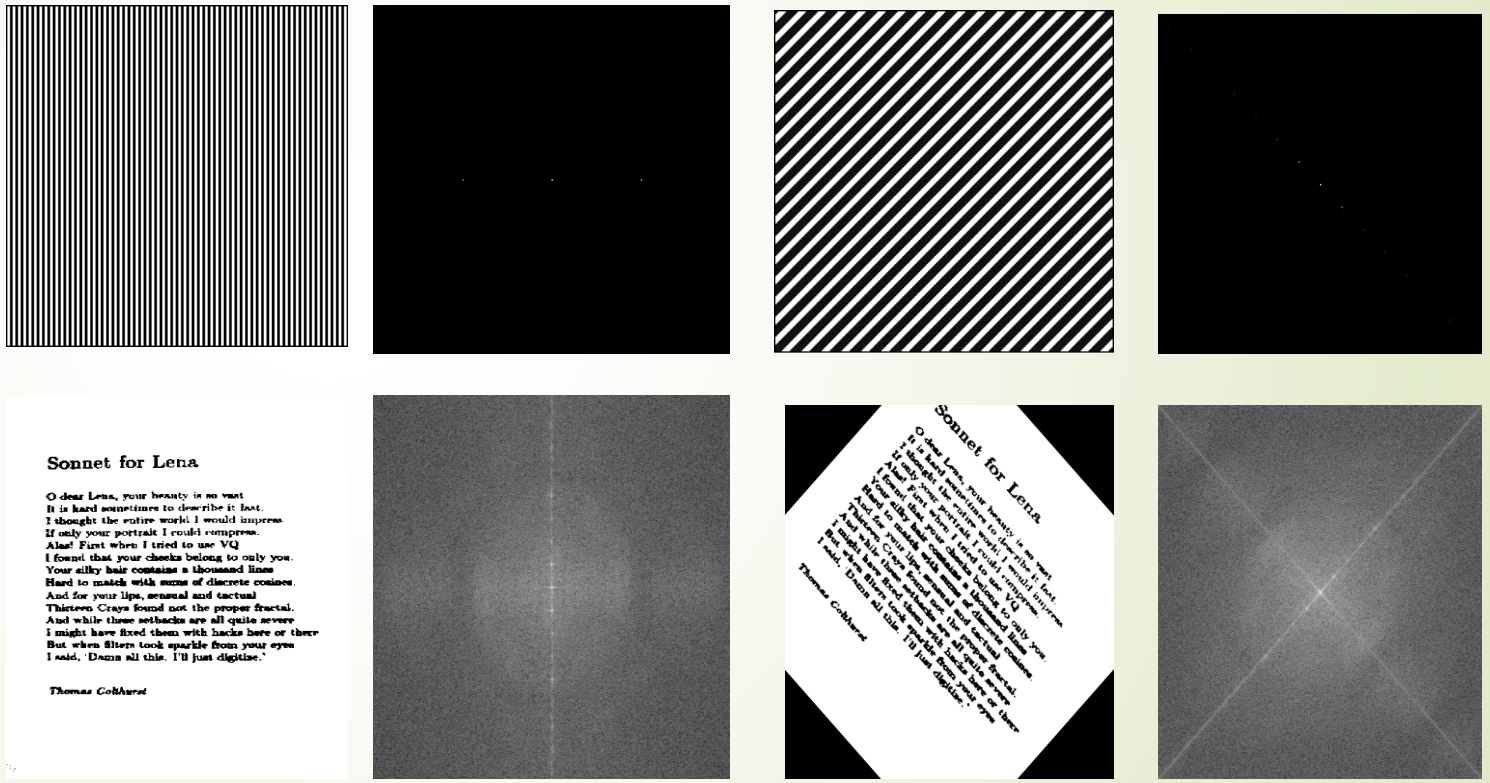


Amplitude +
LOG normalization

Fourier transform

2D discrete Fourier transform (2D DFT)

- We can see that magnitude tell us something on the structure of the image.
- JPEG is based on DCT and not DFT because the DCT have more of its energy concentrated in a small number of coefficients compared to the DFT
- There is a efficient version of DFT called fast Fourier transform (FFT) which is faster than DFT.



Fourier transform

Low and high pass filters

- Low pass filter in the frequency domain is used for smoothing the image. It attenuates the high frequency components (located on the peripheral of the image) and preserves the low frequency components (located at center). One type of such filters is the ideal low pass filter, which is applied as follows

- This filter allows to alleviate the noise because noise is a high frequecny component (rapid change).

Fourier transform

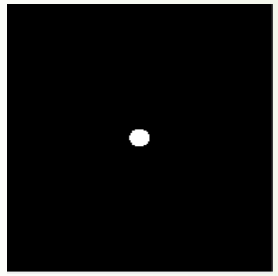
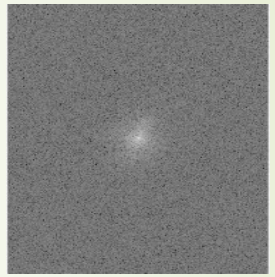
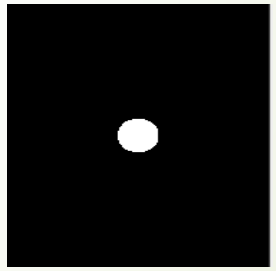
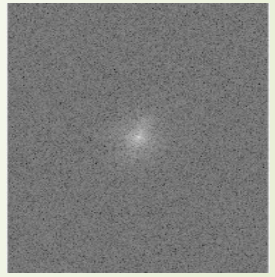
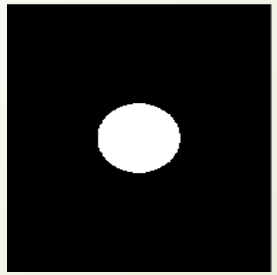
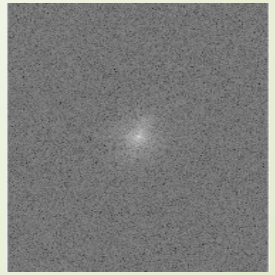
Low and high pass filters

- Apply the Fourier transform on the input image.
- Dertermine the cut-off frequency C.
- Design the ideal low pass filter
- Multiply (elemet wise) the tranformed image with the low pass filter to exclude the values greater than C.
- Apply the inverse transform and show the image.

- We can see that as the diameter of the central circle increases the results appear better. Thus, the diameter size is a compormise between the suffecient number of frequencies to reconstruct the image and the number of high frequencies to avoid.

$$H(u,v) = \begin{cases} 1 \; if \; D(u,v) \leq C \\ 0 \; if \; D(u,v) > C \end{cases} \qquad D(u,v) = \sqrt{u^2 + v^2}$$

For example, $D(0,0) = 0$ i.e., The distance between the center and itself, as the image is centered to the low frequency (0,0) by shifting.

Filter H	Amplitude	Result
		
		
		

Fourier transform

Low and high pass filters

- High pass filter in the frequency domain is used for detecting edges of the image. It attenuates the low frequency components and preserves the high frequency components (rapid changes: edges, noise,…). It is exactly the inverse of the ideal low pass filter

Fourier transform

Low and high pass filters

- Apply the Fourier transform on the input image.
- Dertermine the cut-off frequency C.
- Design the ideal low pass filter
- Multiply (elemet wise) the tranformed image with the low pass filter to exclude the values greater than C.
- Apply the inverse transform and show the image.

- We can see that as the diameter of the central circle increases the finer details become less. This is because as the diameter increases, less number of high frequencies is considered.

$$H(u,v) = \begin{cases} 1 \; if \; D(u,v) \geq C \\ 0 \; if \; D(u,v) < C \end{cases} \qquad D(u,v) = \sqrt{u^2 + v^2}$$