UNIVERSITÉ LIBRE DE BRUXELLES

Bruface Brussels Faculty of Engineering

VRIJE UNIVERSITEIT BRUSSEL

Image and Video Technology

Exercises

Boya Zhang

January 11, 2020

Engineering Science

1 Session 1

1.1 Get started

(1.3) We measure "lena.raw" quantitatively in ImageJ with "Histogram" in the "Analyze" menu.

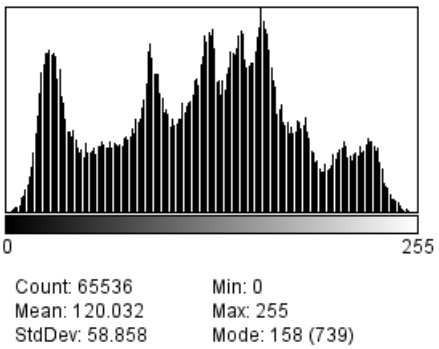


Figure 1: Histogram of lena

The meaning of all six statistics values:

1. Count: total number of the pixels
2. Mean: mean value of all the pixels
3. StdDev: the standard deviation of all the pixels
4. Min: the minimum value of pixel
5. Max: the maximum value of pixel
6. Mode: the most frequent value of pixel
   (a) 158: the most frequent value
   (b) 739: the number of the appearance of the most frequent value

1.2 Create and store a RAW 32bpp float grayscale image

(2.1 and 2.2) We create and store a RAW 32bpp float grayscale image as shown in Fig2.

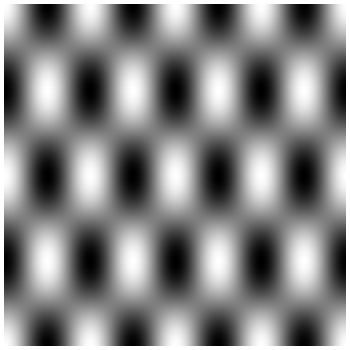


Figure 2: Gray-scale image

The total size of the image is 256 KB shown on computer. We can also get the size by calculation.

1. Number of pixels is $256 \times 256 = 65536$.
2. Size of each pixel is 32 bits.
3. 1 byte is 8 bits.

So the total size in bytes is $\frac{65536 \times 32}{8} = 262144$, which is approximately equal to 256 KB.
(2.3) We adjust the Window/Level and find that level 0.5 and window width 1 enclose the full range of pixel values as shown in Fig.3.

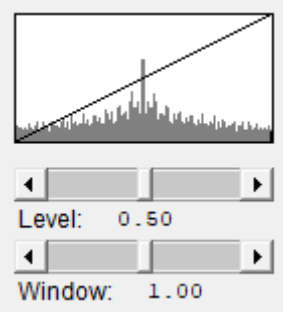


Figure 3: Histogram with Window/Level

1.3 Load and modify a RAW 32bpp float grayscale image

(3) We write a load function to read image from file and load in memory the 256×256 pixels RAW image of "Lena". Then we modify the image by multiplying pixel by pixel with the previously generated pattern.The modified image can be seen in Fig.4

Figure 4: Modified image

(3.4) Then we compute MSE and $PSNR$ of the modified image, relative to the original one with equation 1 and 2.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i.j) - K(i,j)]^2 \tag{1}$$

$$PSNR = 10 \log_{10} \frac{MAX_I^2}{MSE} \tag{2}$$

, $where MAX_I = 255$

The results are $MSE = 5326.28$, $PSNR = 10.8666$.

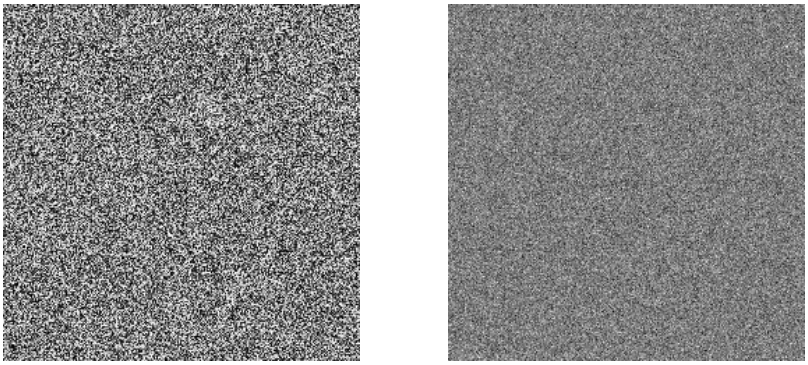
2 Session 2

2.1 Uniform and Gaussian random white noise

(4.1) First, we generate a 256×256 pixels image with uniform-distributed random numbers in $[-\frac{1}{2}, \frac{1}{2})$ and find **MSE** of the random image, compared to the expected zero mean is **0.0818778**.

(4.2) Second, we generate a 256×256 pixels image with Gaussian-distributed random numbers, when **variance** $\sigma^2 = 0.2875^2, MSE = 0.0819496$ matches the MSE of uniform random image.

(4.3) Third, we put the two uniform and Gaussian-distributed noise images side-by-side.

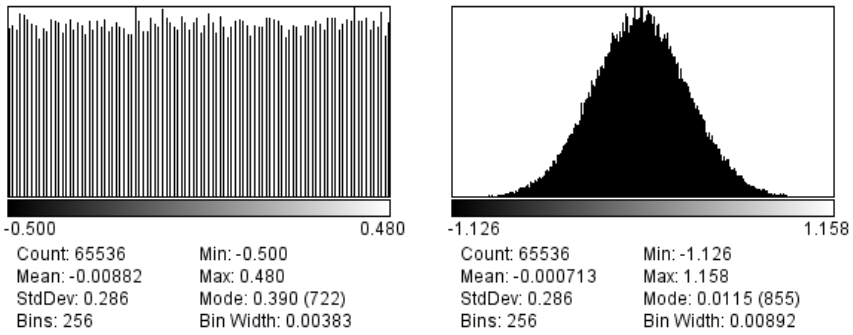


(a) Uniform random white noise (b) Gaussian random white noise

Figure 5: Comparison of noise images

(4.4) Finally, we measure statistics of the noise realizations and compare them.

Noise type	Mean	Variance	Minimum	Maximum
Uniform	-0.009	0.286^2	-0.500	0.480
Gaussian	-7.134E-4	0.286^2	-1.126	1.158



(a) Uniform random white noise (b) Gaussian random white noise

Figure 6: Histogram of noise images

2.2 Additive noise

(5) We load in memory the RAW image of "Lena" and modify the image by adding uniform or Gaussian random noise realizations. The modified image is shown below.

Figure 7: Gaussian random white noise images

We can visually see that the modified image is more blur than the original one. PSNR of noisy image with Gaussian random noise is 26.0578.

2.3 Blur with 3×3 kernel convolution

(6.1) First, we calculate values of the normalized 3×3 blur kernel, using the standard normal distribution.

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, σ is 1. We normalize G so that the summation of all the elements equal to 1.

$$G_n(x, y) = \frac{G(x, y)}{\Sigma G(x, y)} \tag{4}$$

The normalized 3×3 blur kernel is $\begin{pmatrix} 0.0751136 & 0.123841 & 0.0751136 \\ 0.123841 & 0.20418 & 0.123841 \\ 0.0751136 & 0.123841 & 0.0751136 \end{pmatrix}$

(6.2) Second, we write a blur function that applies a 3×3 kernel convolution inside a rectangular region. At this moment we don't consider the pixels at the borders.The blur image is shown in Fig.8. There is a black contour at the border.

Figure 8: Partly blurred image

(6.3) Third, we applying the 3×3 kernel convolution on the whole image. We use extend method for the border, which is providing values for the convolution by extending the nearest border pixels. Corner pixels are extended linearly and diagonally. Other edge pixels are extended in lines.[1]

Figure 9: Extend the edge

The whole image after convolution is shown below.

Figure 10: Fully blurred image

	PSNR(original)	PSNR(blurred)
Gaussian random noise	26.0578	30.4797

(6.4) We can see that "Blur is a medication for noise" because after blurring the PSNR increases, which means that the noise is reduced.

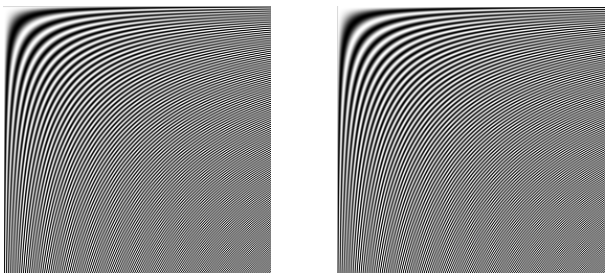
3 Session 3

3.1 Matrix of orthogonal basis functions

(7) We create a matrix with equation5 containing all DCT basis vectors for a 1D signal of length 256.

$$X_k = \sum_{n=0}^{N-1} x_n cos(\frac{\pi}{N}(n+0.5)k) \quad k = 0, 1, .., N-1 \tag{5}$$

We can see that in figure 11a from top left to bottom right the frequency is increasing as expected. In order to justify the orthogonality, we multiply the basis with its transpose and to see if the result is an identity matrix. The figure 12 shows that the result is an identity matrix as the white line represents ones and the black area represents zeros. And when we visually compare between DCT basis matrix and IDCT basis matrix(Fig.11), there is no difference.



(a) DCT basis matrix (b) IDCT basis matrix

Figure 11: Analysis and synthesis dictionaries side-by-side

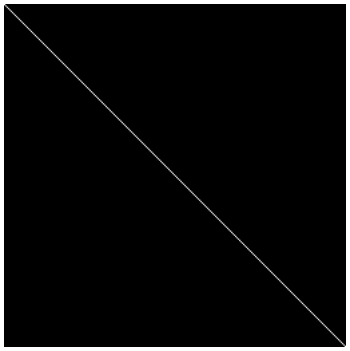


Figure 12: Identity matrix

3.2 Discrete cosine transforms (DCT)

(8.1) We write a transform function that produce DCT coefficients from the image of "Lena" as shown in Fig.13. **(8.2)** Then we write a threshold function that zero-out small (in absolute value) DCT coefficients.The threshold we choose is 1, 10 and 100. **(8.3)** The reconstructed images and their PSNR are as shown in figure14.

Figure 13: DCT coefficients from the image of "Lena"

(a) TH = 1, PSNR = 60

(b) TH = 10, PSNR = 37

(c) TH = 100, PSNR = 24

Figure 14: Reconstructed images with different thresholds

We can observe that the PSNR decrease as the threshold increases.The reason is that the higher threshold zeros out more high frequency components. Visually speaking the image quality of threshold from 1 to 10 are all good, and when the threshold goes to 100 we can clearly observe that the image quality is bad. So we can conclude that there exists one threshold value between

10 and 100 that gives us visually good quality image with the least image storage space.

4 Session 4

4.1 Lossy JPEG image approximation

(9.1) First, we create an 8×8 image containing standard JPEG quantization weights Q at 50% quality.

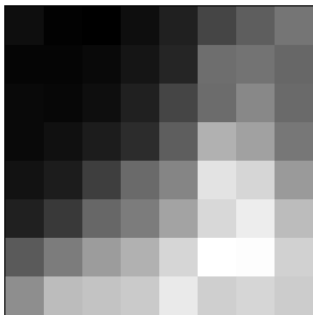
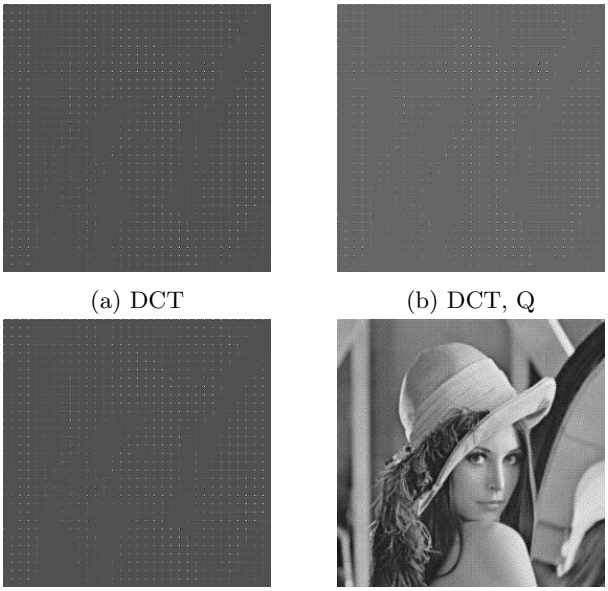


Figure 15: Q matrix

(9.2) Second, we write an approximate function that apply for each 8×8 pixels blocks: DCT, Q, IQ, IDCT.



(a) DCT (b) DCT, Q

(c) DCT, Q, IQ (d) DCT, Q, IQ, IDCT

Figure 16: Intermediate steps visualization

(9.3) Third, we write a clip function for exporting images to 8bpp integer grayscale values with a valid range [0..255]. **(9.4)**Then we create a difference image of the result with a baseline JPEG file at 50% quality.

(a) Clipped image (b) Q = 50% JPEG (c) Difference image

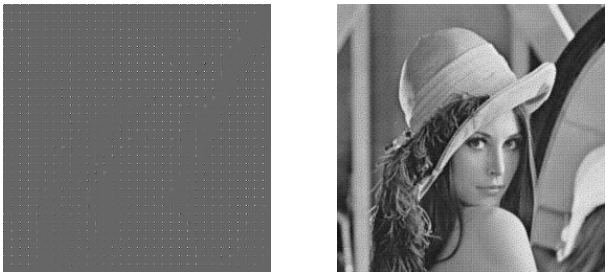
Figure 17: Comparison between two images

The difference image is almost zero everywhere. But there are still non-zero pixels. The reason can be that the standard JPEG compression operation on ImageJ contains more optimization methods.

(9.5) Last, we write two encode and decode functions for splitting the approximate method into:

1. Encoding an original image into quantized DCT coefficients;
2. Decoding quantized DCT coefficients to reconstruct the image approximation.

We can observe that the encoded image is as same as the image after DCT and Q, and the decoded image is as same as the image after DCT, Q, IQ abd IDCT.



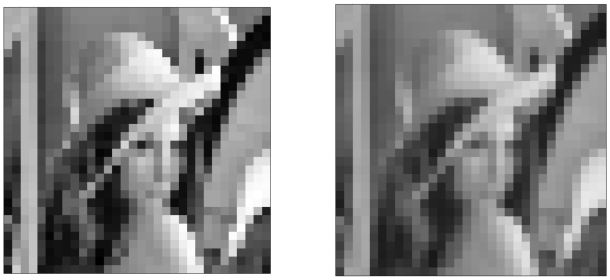
(a) Encoded image (b) Decoded image

Figure 18: Splitting the approximate method

5 Session 5

5.1 Delta encoding of DC coefficients

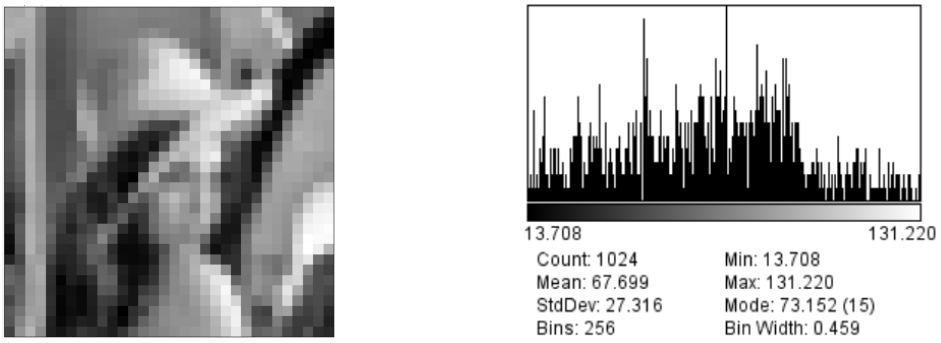
(10.1) We create a 32×32 pixels image from the quantized DC terms of each 8×8 pixels block, and then compare it with the $8\times$ downsized input image generated with imageJ.



(a) 32×32 pixels DC terms (b) $8\times$ downsized

Figure 19: Comparison between two images

When we qualitatively compare between these two images we can not see much difference. When we quantitively analyze two images, we can see that the pixel values of DC terms image are always smaller than the ones in downsized input image. We get the difference image after subtracting Fig.19a from Fig.19b. The quantitative comparison results are shown in Fig.20.



(a) Difference image (b) Histogram

Figure 20: Quantitatively comparison

(10.2 and 10.3) We create a text file containing successive differences between quantized DC terms of each block. The size of the text file is 3 KB, the size of the DC terms image is 4 KB. So it saves 25% of the memory. Then we read the delta encoded DC terms from the text file and reconstruct the low-definition image. The reconstructed image is same as Fig.19a because delta encoding is a lossless encoding method.

5.2 Run-length encoding (RLE) of AC coefficients

1. **Imagine a method for ordering and coding groups of quantized AC terms with run lengths.**
   We perform zigzag ordering method on each 8x8 block. As shown in Fig.21, there are three situations for the moving of element(i,j).
   (a) If j is an even number and the element is on the first or last row(i = 0 or i =7 in the case of 8x8 block),then we move right for one index.

(b) If i is an odd number and the element is on the first or last column(j = 0 or j =7 in the case of 8x8 block), then we move down for one index.

(c) For all the other situations, if i+j is an even number, then we move right and up for one index. Otherwise i+j is an odd number, we move left and down for one index.

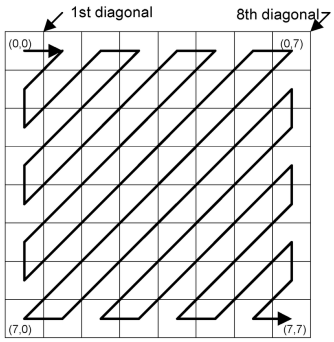


Figure 21: Zigzag-ordering

After the zigzag ordering is done correctly, we perform run-length encoding[2]. The basic idea is to make each run length represents a contiguous sequence of either zeros or arbitrary values. An simple example is as follows: the original sequence is [A A B B B C C], then the coded sequence is [2 A 3 B 2 C]. The meaning is that we have two "A"s, three "B"s, two "C"s in the sequence. And most of the time, the coding method save space.

2. **Update your encode function for printing a sequence of run lengths from coefficients.**
We first extract all the ac terms as shown in "ACterms.txt". There are 63 AC elements in each group.

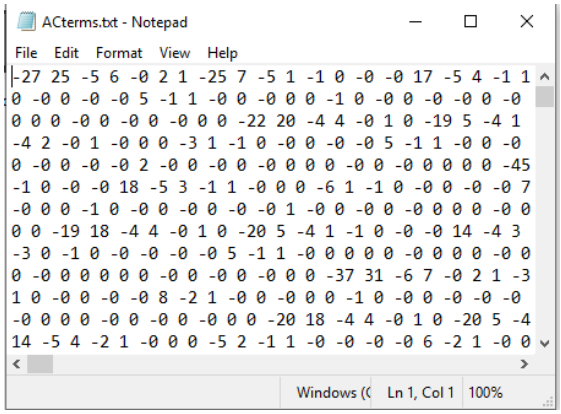


Figure 22: AC terms

We perform the zigzag ordering and run length encoding for each group and store the encoded sequence in file "encodedACterms.txt". One important thing is to use 8x8 pixels

block for zigzag ordering and then delete the first term of each group before performing run length encoding.
We take the first few elements in the first group as example(Fig.23), the encoded sequence means we have one "-27", one "-25", one "17" and so on. Concerning the text file size we can see that the AC terms takes up 158 KB while the encoded AC terms takes up 110 KB. We saved 30% of the storage space after encoding.

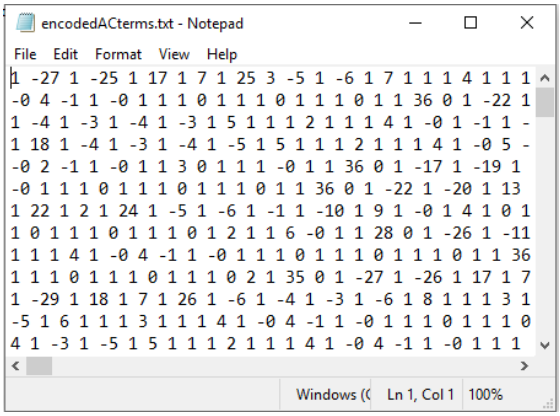


Figure 23: Encoded AC terms

3. **Update your decode function for reconstructing AC coefficients from a sequence of run lengths.**
For the decode function we first perform the run-length decoding, then we use the inverse zigzag for each group. One important thing is to insert one element in the front of each group so that the izigzag is performed on each 8x8 pixels block, and after izigzag we delete the elements inserted. The result after decoding is stored in "decodedACterms.txt". It is as same as the AC terms in "ACterms.txt" (Fig.22), so we perform the decoding successfully.

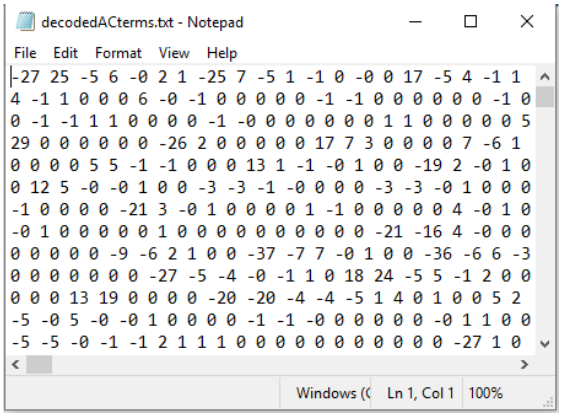


Figure 24: Decoded AC terms

5.3 Discrete probability density functions (PDF)

1. **Create an array P(i) i=0,..,N containing the number of occurrences for each RLE symbol.**
   We find array P(i) with the map container in c++.[3] The longest run length N is 107, which is the length for the number of occurrence vector computed from the RLE vector. To encode all AC coefficients, we have M = 48942, which is the number of symbols for the RLE vector.

2. **Normalize the array P such that the sum of all elements is one.**
   The elements in normalized array P are the probabilities. The map P, array P and normalized array P are stored in the text files.

3. **Write an entropy function that compute the theoretical minimum number of bits per symbol.**

$$H(X) = -\sum_{i} P(x_i) log_2 P(x_i) \tag{6}$$

   We calculate with equation 6 and get $H(X) = 3.26303$ bits/symbol, which is the theoretical minimum number of bits per symbol. The minimum possible file size for encoding AC coefficients with RLE is: $Entropy \times M = 159699$ bits.

6 References

[1] Kernel (image processing) From Wikipedia, the free encyclopedia University.
[2] Run-length encoding From Wikipedia, the free encyclopedia University.
[3] Counting-occurrences-of-integers-in-map-c From stackoverflow.
[4] Image and Video Technology Exercises (WPO) 2019-2020.