| IMC<br>———<br>ESIEE-Paris | IMC & M2 SIS & I5FI<br>Image Compression | IMC- Master SIS - I5FI |
|---|---|---|

Given by J.-F. BERCHER

# Lab on image compression

## GOAL OF THE LAB

In this lab, we will study an image compression method analog to the JPEG method, with some simplifactions with respect to the standard. The goal is still to obtain a true compression method, that begins with an image and end with a binary stream, whose decoding enable to find back an image that resemble to the original one. The programs will be implemented under Matlab. Many functions are offered. Some others will have to be implemented. Different gray level images are available (PGM format – Portable GrayMap). The function `getpgm` (or `imread`) enables to load these images as a matrix in Matlab's workspace, e.g. `A=getpgm('bird.pgm');` loads the pgm image into the matrix A. The command`imagesc` (image scaled) displays an image using all the range specified by the colormap. In our case, so as to display in gray levels, one has to choose `colormap('gray')`. Available images : bird.pgm, boat.pgm, frog.pgm, lena.pgm, math4.pgm, mandrill.pgm, peppers.pgm.

## I. LET US PLAY WITH MATLAB

**Partial image** Load an image using `getpgm`. Display your image. Create a function so as to extract a block of specified size (e.g. 8x8) at a given position in the image. Display the result. (**[Q0]**)

Matlab is a vectorial interpreter, which enables very easily to extract a submatrix. Thus, `A(k*L+1 :k*(L+1), p*L+1 :p*(L+1))` extract a submatrix of size L*L, from the row k*L+1 to the row k*(L+1) and from the column p*L+1 to p*(L+1).

**Reading of the remaining blocks** Using two for loops, display the remaining blocks in the lexicographic order (**[Q1]**)

```
NbBlocks=min(size(IMG))/block_size;
[M N]=size(IMG);
iter=0;
for m=1:block_size:M
for n=1:block_size:N
iter=iter+1;
message=sprintf('Processing block number %d',iter); disp(message);
Block=IMG(m:m+block_size-1,n:n+block_size-1);
imagesc(Block); colormap('gray')
pause
end;
end;
```

**DCT2D** (**[Q2]** )On a block A of any size, compute the 2D cosine transform (function `dct2d`). Display the result B and note in particular the importance of low frequencies. Compute the 2D inverse DCT of B, (function `idct2d`) and compare the result C to the initial image A.

**DCT2D and rounding** Round the result B to the nearest integer (function `round`). Compare the result C of the inverse DCT of B to the initial image A. Compute the quadratic distorsion according to `{mean(mean(abs(C-A).^2))}`

**DCT2D and rounding** In the previous operations, add a weightening on the result B by a matrix 1./Q (The matrix Q is read by `stdQ` – note the importance of ./) before the rounding operation. Unweight this result, noted B, by `B.*Q` (note the importance of the . in .*) before applying the inverse DCT.

**[Q3]** Compare this reconstruction to the initial image A. Compute the quadratic distortion.

## II. Let us play again with Matlab

**Zig-zag reading** The function `zz.m` reads a block in zigzag order. If Block is a matrix, and if one let Z=zz(8), then Block(Z) is a vector that contains the result of the reading in the zigzag order. In order to convince yourself test the two commands above.

– The function `RLCjpg.m` returns a matrix in the RLC format, JPEG variant, that is [Number of preceding zeros, value]. The [0 0] code marks the end of the block. Test this function on the result of the previous zigzag reading.

**Binary representation** The function dec2bin.m converts a decimal number into binary. Test the function, e.g. on 1,2, 4,5, 8,9, 27, 53

**Category Codes and binary values** The function `cat_code.m` computes the categories and binary values associated to a decimal integer.

**[Q5]** Test this function. Beware, there are two outputs. Compute the categories and binary values associated with the outputs of the RLC codes given by `RLCjpg.m`. Test this on the result of an RLC encoding. You will need to use a loop like

```
U=RLCjpg(...);
[LL MM]=size(U);
for nn=1:LL
[cat,bin]=cat_code(U(nn,:));
disp(['U=' num2str(U(nn,:)) '  Cat= ' num2str(cat) '  Bin=[' num2str(bin) ']' ]);
end;
```

**Huffman encoding** Several Huffman tables are available, for different values of the compression parameter s. These tables, called HH are contained in the files htabXX.mat, with XX $\in$ 01, 05, 1, 2, 3, 4, 5, 10, 20 (load with `load htabXX.mat;`); if [nb_zeros, cat] is a couple [Number of zeros, category], then HH(nb_zeros+1,cat+1) is the corresponding Huffman code. Actually, we have stocked in the Huffman table the indexes on the set of binary words rather than the actual (variable length) words. To each binary word is associated a decimal number, by a simple enumeration : [0 1 00 01 10 11 000 001 etc] –> [1 2 3 4 5 6 7 8 etc]. The function `bstr2idx.m` gives the translation binary words to decimal while `idx2bstr.m` realizes the inverse translation. Hence, `M=idx2bstr(HH(nb_zeros+1,cat+1))` is the codeword associated with the couple (nb_zeros, cat). If U is the RLC result, U(n,1) is the number of zeros, and the category and binary value are given by `[cat,bin]=cat_code(U(n, :));` The binary stream is then obtained by the concatenating the Huffman codeword M and the binary value bin, as [M bin].

**[Q6]** Build the binary stream corresponding to the proposed test matrix (There is only two or three lines to add after cat_code). The length of the binary stream will be obviously given using the function length. The compression ratio is given by the ration of the number of bits in the initial image (in our case number of pixels x 8 bits), to the number of bits of the encoded image (length of the binary stream). Compute the compression ratio.

## III. Final coder

Modify the script cod1.m into a new function cod2.m so as to obtain a binary sequence as the output. For each DCT block, one will have to add the following operations :

i zigzag reading,
ii RLC encoding,
iii category coding,
iv Huffman coding,
v creation of the binary stream.

Finally, you will also have to give the effective compression ratio R at the output of the encoder. The call to cod2 should be

`[BinaryStream, R, ImTr] = cod2(Img,L,Q,HH).`

The output `ImTr` will enable to decode the image using `decod1`. You will evaluate the compression ratio and distortion for several values of the parameter s and you may plot the compression-distortion curve. The implementation of the decoder from the binary sequence is optional.

*List of available functions*

For each of the following function, it is possible to obtain help by
`help name_of_function`
It is also possible to list the content by (`type`) or to edit it.

| | |
|---|---|
| cat_code.m | Compute the category codes and the associated binary value. |
| dct1d.m | Compute the 1D DCT |
| dct2d.m | Compute the 2D DCT |
| getpgm.m | Read a pgm image |
| huff.m | Huffman algorithm |
| idct1d.m | Inverse DCT 1D |
| idct2d.m | Inverse DCT 2D |
| jpgstat10.m | Cimpute statistics and Huffman tables on a base of images |
| msb.m | Most Significant Bit of a decimal number |
| prtbstr.m | Print bit stream |
| quantify.m | Quantization |
| rlcjpg.m | RLC format JPG |
| stdq.m | Weight matrix of JPG standard |
| zz.m | ZigZag reading |
| bin2dec.m | Binary to decimal |
| dec2bin.m | Decimal to binary |