Lecture "Digital Signal Processing"

Prof. Dr. Dietrich Klakow, Summer Term 2021

Assignment 1

Submission deadline: 26 April 2021, 23:59

Submission Instructions:

You have one week to solve the assignments.
The code should be well structured and commented. Do not use any Matlab-Toolbox or Python external libraries if it is not mentioned that you can use them.

- You are allowed and encouraged to hand in your solutions in a group of two students.
- There are two parts in this assignment: theoretical part and practical part.
- The practical part that is solved with Python should be submitted as an ipynb file (Jupyter Notebook or Google Colab), where every function is written in a separate block.
- The theoretical part can either be written by hand and scanned, or typed with LaTeX in the text / markdown area of ipynb file.
- Submission of both parts should be done via Microsoft Teams by one of the group members.
- Submission should be named as: Ex01_matriculationnumber1_matriculationnumber2.zip

The submission should contain the following files:

- file "README" that contains an information on all team members:
  name, matriculation number, and email address.
- code files
- file "answers.pdf" which contains answers to the questions appearing in the exercise sheet. *Note: If you use ipynb file, you don't have to submit "answers.pdf". You can embed your scanned copy or write your answers in the text / markdown area.*

1 Discrete Cosine Transform (DCT)

One-dimensional DCT operation can be expressed in matrix form of:

$$y = Cx \tag{1}$$

where $C \in \mathbb{R}^{N \times N}$ is the orthogonal DCT matrix.

1.1 (1P) DCT Matrix

Derive a 4×4 DCT matrix starting from expressing the formula of 1-D DCT.

1.2 (0.5P) DCT of a given signal

Given a 1-D signal of $(2, -1, 0, 1)^T$, obtain its DCT transformed signal.

1.3 (0.5P) Inverse DCT of a given signal

How would we compute the inverse transform of a given 1-D DCT transformed signal, in relation to the matrix form above?

2 Basic JPEG compression

The goal of this exercise is to understand the steps involved in JPEG compression that are described in DSP chapter 2, slide 23 .
Please find the attached file "birds.ppm" as a sample image, and blockproc.m for Matlab or Utility.py for Python in this exercise directory, and use them to implement the building blocks of JPEG compression.
Furthermore, add a pdf file "answers" or use the text / markdown area in ipynb file to answer the questions appearing in this exercise.

About *blockproc*:
It is common to implement the general function *blockproc*. This function splits the given matrix $I \in \mathbb{R}^{n \times m}$ into submatrices with size $b \in \mathbb{N}^2$ and applies the delivered function $fun : \mathbb{R}^b \to \mathbb{R}^b$ to each of the submatrices. Finally, it returns a matrix $O \in \mathbb{R}^{n \times m}$ out of all the submatrices. Find an example usage in the "blockproc.m" file.
(You can use "@" to deliver functions as reference; Matlab-Help: "function_handle")

```
function [O] = blockproc(I, b, fun)
```

For Python:

```
from Utility import blockproc
Output = blockproc(I, b, fun)
```

2.1 (1P) Optimal color space

The first step in JPEG compression is to apply a color transformation, which takes the image from RGB to YC_bC_r color space. The following matrix-vector multiplication transforms an RGB vector into a YC_bC_r vector.

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0 \\ 127.5 \\ 127.5 \end{pmatrix} + \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ -0,168736 & -0,331264 & 0,5 \\ 0,5 & -0,418688 & -0,081312 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2}$$

1. Start by reading the input sample image, and show the image using *imshow()* as well as print the image size.
2. Implement two functions *rgb2ycbcr* and *ycbcr2rgb*, that transform a given $RGB-$image to a YC_bC_r-image, and YC_bC_r-image to $RGB-$image, respectively.
3. Execute the function *rgb2ycbcr* to the input image and show the result.

2.2 (2P) Downsample

The next step in JPEG compression is downsampling of C_b and C_r channels separately by a factor of w in vertical and horizontal direction, while keeping the Y channel to its original size. Downsampling here means to assign the whole block of width and height w the average of all values in the given block.

1. Implement a function *downsample* that downsamples a given matrix given a blocksize $w \times w$. In addition, implement a function *upsample* that simply upsamples a matrix.
2. To test the function, execute *downsample* with $w = 8$ on the individual 3 channels of the color-transformed image, which you have obtained from task 2.1, show the resulting combined downsampled image, and print the image size.
3. Additionally, execute *upsample* with the same w value on the individual 3 channels of the downsampled image, show the resulting combined upsampled image, and print the image size.

2.3 (1P) DCT

The next important step in JPEG compression is to transform all 3 channels i.e. Y, downsampled C_b, and C_r channels separately using DCT.

- Implement 2-D DCT and inverse DCT functions, either by your own codes or using the 1-D "dct" and "idct" functions in Matlab or from scipy.fftpack of Python library.

2.4 (1P) Quantization

Another important step in the JPEG compression is to quantize blocks of a image. To do so, a function that quantizes a matrix $M \in \mathbb{R}^{8\times8}$ using a given matrix $Q \in \mathbb{R}^{4\times4}$, comes in handy. A quantization in this case is a block by component-wise division of the two mentioned matrices, i.e. the first 2×2 matrix of M ([0,0], [0,1], [1,0], [1,1]) gets divided by the element at [0,0] in matrix Q, and so on. The following quantization matrix is given:

$$Q = \begin{pmatrix} 8 & 19 & 26 & 29 \\ 19 & 26 & 29 & 34 \\ 22 & 27 & 32 & 40 \\ 26 & 29 & 38 & 56 \end{pmatrix}$$

- Implement a function *quanmat* to quantize, and a function *dequanmat* to dequantize a matrix M using the given Q matrix above.

2.5 (1P) Execution of DCT and Quantization

1. Use the method "blockproc" and a block-size of 8×8 pixels to first apply a blockwise DCT to Y channel and downsampled C_b and C_r channels, and continue to quantize the resulting matrices, which are called Y', C_b', and C_r'.

2. Display only the original Y channel, resulting Y', as well as the difference between Y and Y' in one window. For Python, you can use numpy hstack in the *imshow()*

3. What do you observe? How do these steps help with compression? Do we lose quality?

2.6 (1P) Encoding

The final step of JPEG compression is encoding the quantized DCT coefficients by applying Huffman coding on Y', C_b', and C_r' individually.

1. To apply Huffman coding, write two functions *encodemat* and *decodemat* for further utilization. You can use *huffmanenco,huffmandeco,huffmandict* in Matlab. For Python use *dahuffman* 0.2 (pip install dahuffman).
   Note: for simplicity, just flatten the matrix into a vector by concatenating all rows together and then apply Huffman encoding. You are not required to do the zig-zag ordering and RLE that are used in JPEG compression.

2. Write the length of each encoded channels Y', C_b', C_r' in answers.pdf or print them in the text / markdown area.

2.7 (1P) Decompression

Using the previously described methods, you have already applied a light weight JPEG compression, by first applying a colortransform, downsampling the C_b and C_r channels, applying DCT in all the channels, quantizing the DCT coefficients, and finally encoding the quantized DCT coefficients using Huffman encoding.
Now invert the whole process, show the resulting RGB image, and compare it with the input sample image. What do you observe?