Exercise 4: Image compression

Multimedia systems

2018/2019

Create a folder `exercise4` that you will use during this exercise. Unpack the content of `exercise4.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as the *Matlab/Octave* scripts to `exercise4` folder. In order to complete the exercise you have to present these files to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the presentation as well. The tasks that are marked with ★ are optional. Without completing them you can get at most 75 points for the exercise (the total number of points is 100 and results in grade 10). Each optional task has the amout of additional points written next to it, sometimes there are more optional exercises and you do not have to complete all of them.

Introduction

In this exercise, you will implement parts of the JPEG pipeline for image compression. The method is based on performing the discrete cosine transform (DCT) on 8x8 blocks, quantizing the resulting coefficients, then losslessly compressing them via entropy encoding. You will first familiarize yourself with the calculation and properties of the DCT by using it on 1D signals, then use its 2D variant to transform and reconstruct images. Finally, you will implement a rough version of the JPEG algorithm and check the reductions in space and quality of the compressed images.

Assignment 1: 1D DCT

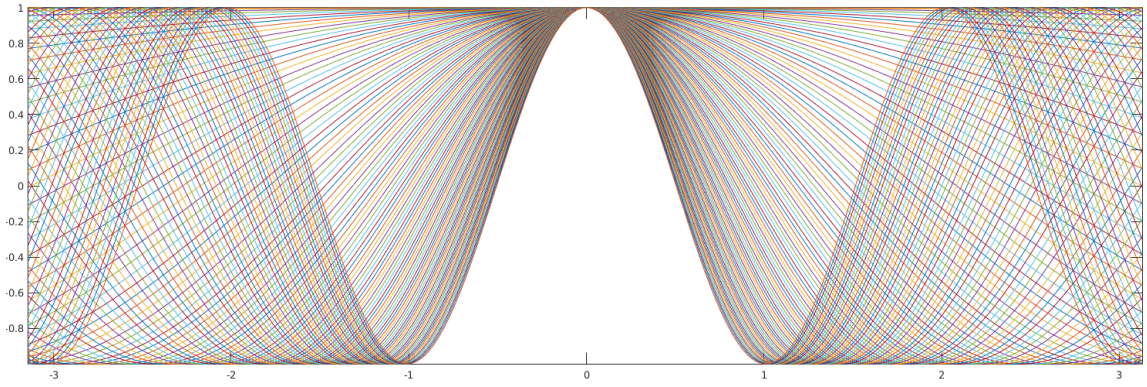
The discrete cosine transform used in JPEG standard is called DCT-II. The process expresses a (finite) sequence of data points as a sum of cosines with different frequencies (called basis functions). The result is a list of coefficients which can be used to reconstruct the original data. The most commonly used is DCT-II, also called "the DCT". Its inverse, DCT-III is called "the inverse DCT". Their formulas are as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right); \quad k = 0, \ldots, N-1,$$

$$x_k = \frac{1}{N}X_0 + \sum_{n=1}^{N-1} \frac{2}{N}\, X_n \cos\left(\frac{\pi}{N}\, n\, \left(k+\frac{1}{2}\right)\right); \quad k = 0, \ldots, N-1$$

(a) Implement 1D DCT transform and its inverse. Implement the functions `my_dct` and `my_idct` that receive a signal and return the DCT coefficients (the sizes must match). Use the above formulas. Take note that these are the basic formulas. In practice, scaling factors are used to allow the transform to be expressed as matrix multiplication (see point (d)). You can test the correctness of your implementation by transforming a signal to frequency domain and back again. The result should be equal to the input signal (up to machine precision).

(b) In DCT, the signal is expressed as a sum of cosines with differing frequencies. Take a sensible range of numbers (e.g. $[-\pi, \pi]$) and plot the basis functions. The number of basis functions must equal the number of data points in the chosen interval.



(c) ★ (10 points) Visualize the signal reconstruction from the DCT coefficients. Perform the DCT on a chosen signal (to make the process easier to follow, you should use a well known function), then multiply the resulting coefficient by the appropriate basis function for each of the data points and add it to the result. Plot the intermediate result at each step. If performed correctly, the signal should in the end be fully reconstructed. Also plot the error between the original and the reconstuction for each step.

(d) ★ (5 points) As mentioned above, the first term of the DCT is sometimes multiplied by $\sqrt{2}$ and the whole result is multiplied by $\sqrt{\frac{2}{N}}$[1]. This makes the matrix orthogonal and allows the DCT (and the IDCT) to be performed by matrix multiplication. Write a function `dct_coef` that takes a size N and returns a $N \times N$ matrix with the DCT coefficients. If the matrix is scaled correctly it should be orthogonal (i.e. $\mathbf{M}\mathbf{M}^\top = \mathbf{I}$). You can check the implementation details for scaling on Wikipedia or in the *Matlab/Octave* documentation. The DCT can then be calculated by multiplying a signal x by the DCT coefficient matrix $\mathbf{M}$. The same holds for the IDCT, except that it requires multiplication by the transpose of the matrix $\mathbf{M}$.

$$y = \mathbf{M}x^\top$$
$$\hat{x} = (\mathbf{M}^\top y)^\top$$

[1] `https://en.wikipedia.org/wiki/Discrete_cosine_transform#DCT-II`

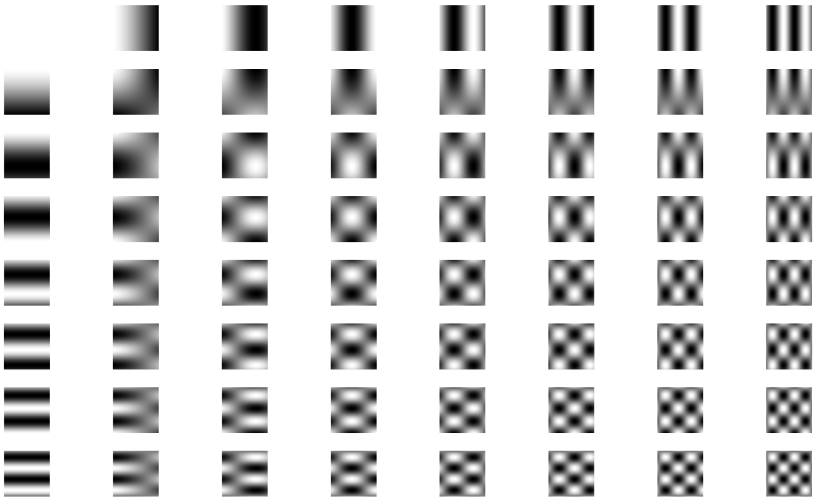
Assignment 2: 2D DCT

The DCT on two-dimensional data can be performed either by first calculating the 1D DCT on matrix rows and then on matrix columns (or vice versa) or by applying the formula directly.

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n1,n2} \cos\left(\frac{\pi}{N_1}\left(n_1 + \frac{1}{2}\right)k_1\right) \cos\left(\frac{\pi}{N_2}\left(n_2 + \frac{1}{2}\right)k_2\right) \tag{1}$$

(a) Implement the functions `my_dct2` and `my_idct2` that perform the DCT and IDCT on 2D matrices.

(b) ★ (5 points) Plot the 2D basis functions.



(c) ★ (10 points) Reconstruct an image from 2D DCT coefficients. Write a function `reconstruct_image` that loads the image `A.bmp` and calculates its 2D DCT. Then, add up the cosines multiplied with the appropriate coefficients. Visualize the reconstructed image at each step. Comment on the contribution of lower and higher frequency components to the recognizability of the image. Which frequencies are more important?

Assignment 3: JPEG pipeline

In this assignment you will implement the main part of the JPEG compression algorithm. The image needs to be split into 8x8 blocks and DCT must be performed on each of them. The coefficients are then quantized using a quantization matrix. The result is finally losslessly compressed using entropy encoding.

(a) Use the provided function `quantization_matrix` to generate different quantization matrices [2]. The function accepts an argument on the interval $[1, 100]$ which can be

[2] https://stackoverflow.com/questions/29215879/how-can-i-generalize-the-quantization-matrix-in-jpeg-compression

interpreted as the output image quality. How does the matrix change with different inputs?

(b) Split your input image into 8x8 blocks. Take care to pad the input image if its dimensions are not divisible by 8 (you can use the function `padarray`).

For each of the blocks, do:

- Subtract 128 (take care of the data type)
- Calculate the 2D DCT
- Perform element-wise division by the quantization matrix Q
- Save the quantized coefficients in a separate matrix
- Calculate the 2D IDCT
- Add 128

Display the image reconstructed from quantized blocks. Display the difference to the original. Comment on the distribution of the differences and the reduction in image quality (try different levels of quantization).

Note: Do not use the image `A.bmp` from the previous assignment.

Question: Why is the image `A.bmp` a bad candidate for JPEG compression?

(c) Use the provided function `norm2huff` to compress the entire matrix of quantized coefficients. Comment on the compression ratio for different levels of quantization. ***Question:*** Why can quantized images be stored using less space?

(d) ★ (10 points) Convert the image into YCbCr color space, then subsample the color channels and reconstruct the image. Find representative images and comment on the difference using different magnitudes of subsampling. Do the same with the Y channel. Use your JPEG implementation to compress a multi-channel image. Try compressing in YCbCr by subsapling color channels. How does that affect the compression ratio?