Image Compression with the Discrete Cosine Transform

Ferri Marco - 807130

Habbash Nassim - 808292

Università degli Studi di Milano-Bicocca

Course: *Methods of Scientific Computation*

1.

DISCRETE COSINE TRANSFORM IMPLEMENTATION

[github.com/mferri17/dct-image-compression/ PART1](github.com/mferri17/dct-image-compression/)

OBJECTIVE

This part of the project aims to implement the Discrete Cosine Transform Type-II, and compare its time complexity against an open-source library's implementation.

The DCT2 implementation is written in Python, and has been called **pyDCT**, while the comparison will be held against SciPy-FFTpack's implementation.

SOME NOTES

SciPy’s FFTpack module implements the first four types of DCTs, each allowing for normalization of the results.

SciPy also provides a range of utilities for linear algebra and matrix operations (through NumPy).

As such, NumPy matrices have been used in pyDCT.

NumPy’s Testing module has been used for testing.

SOURCE CODE (pyDCT.py)

```
# 1D DCT type-II
def dct1(f):
    f = np.ravel(f)
    c = []
    N = f.size
    alpha = np.pad([1/np.sqrt(N)], (0, N-1),
                   'constant',
                    constant_values = (np.sqrt(2/N)))
    for k in range(N):
        sum = 0.0
        for index, val in np.ndenumerate(f):
            i = index[0]
            sum += val*np.cos(np.pi*k*(2*i+1)/(2*N))
        sum = alpha[k] * sum
        c.append(sum)
    return c
```

```
# 2D DCT type-II
def dct2(f):

    # Applies DCT1 on both axises
      of the matrix to compute the 2D DCT
    c = np.apply_along_axis(dct1, 1,
        np.apply_along_axis(dct1, 0, f))

    return c
```

Also available on https://github.com/mferri17/dct-image-compression/_PART1

SOURCE CODE (test.py)

Tests have been conducted with the NumPy Testing library. All tests completed successfully.
Note: the relative tolerance used is of 1/100.

```
import pydct as p
import numpy as np
from scipy.fftpack import dctn

#
## Testing library for accuracy against assigned test data
#

test_dct1 = {
"in" : np.array([231, 32, 233, 161, 24, 71, 140, 245]),
"out" : np.array([4.01e+02, 6.60e+00, 1.09e+02, -1.12e+02,
6.54e+01, 1.21e+02, 1.16e+02, 2.88e+01])
}

dct1 = dctn(test_dct1["in"], type = 2, norm = 'ortho')
dct2 = dctn(test_dct2["in"], type = 2, norm = 'ortho')

# Check if the transform is identical to the given one
np.testing.assert_allclose(dct1, test_dct1["out"], rtol=1e-02)
np.testing.assert_allclose(dct2, test_dct2["out"], rtol=1e-02)
```

```
#
## Testing DCT1
#

dct1 = p.dct1(test_dct1["in"])
np.testing.assert_allclose(dct1,
                           test_dct1["out"],
                           rtol=1e-02)
```

TIME COMPLEXITY

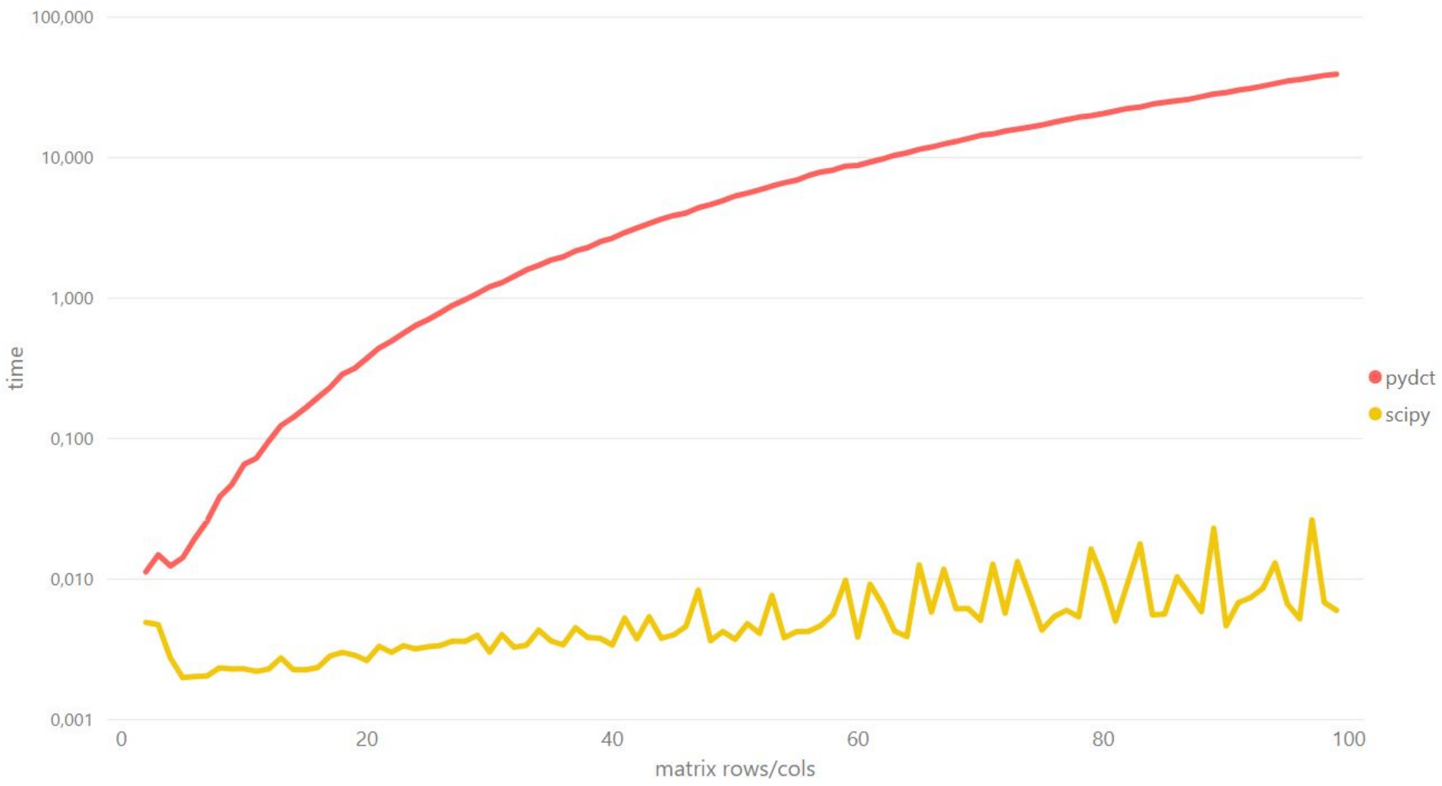
Time complexity has been measured through a series of applications of the DCT2 to matrices of increasing size, for both pyDCT and SciPy-FFTpack's Discrete Cosine Transform implementations.

Time complexity for pyDCT's unoptimized DCT-T2 is $\mathbf{O(N^3)}$

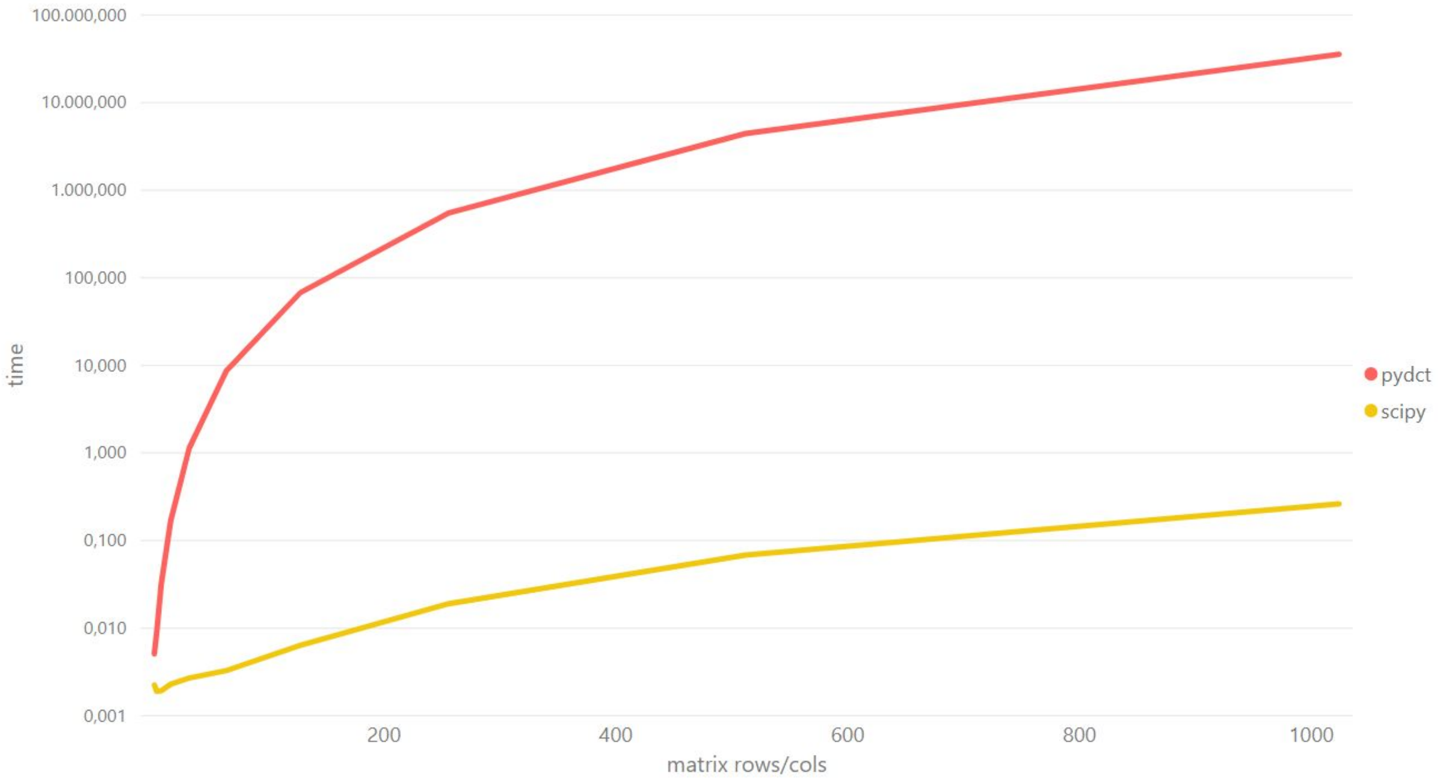
Time complexity for SciPy's DCT-T2 is $\mathbf{O(N^2 * \log(N))}$

- Recorded data:
  - Resolution time for:
    - **99** matrices of increasing size **(2x2 to 100x100)**
    - **10** matrices of increasing exponential size $\mathbf{(2^1 x 2^1 \text{ to } 2^{10} x 2^{10})}$
  - Each computation has been executed **10 times** and averaged to get accurate data

PERFORMANCE COMPARISON (2x2 to 100x100)



PERFORMANCE COMPARISON (2^1x2^1 to 2^{10}x2^{10})



CONSIDERATIONS

The unoptimized DCT is N times slower than the DCT used by SciPy's FFTPack.

FFTPack's DCT is actually computed with an FFT, as a type-II DCT is equivalent to a DFT of size 4N, as demonstrated by Narasimha & Peterson[1] and Makhoul[2], drastically reducing time complexity.

1. Narasimha, M.; Peterson, A. (June 1978). "On the Computation of the Discrete Cosine Transform". IEEE Transactions on Communications. 26

2. Makhoul, J. (February 1980). "A fast cosine transform in one and two dimensions". *IEEE Transactions on Acoustics, Speech, and Signal Processing*

2.
APP FOR IMAGES DCT2 COMPRESSION DEMONSTRATION

github.com/mferri17/dct-image-compression

OBJECTIVE

This part of the project aims to develop a GUI to demonstrate how the DCT2 can be used for image compression, implementing a simple JPEG-based algorithm.

The App has been developed using Python and the Django Web Framework. The application, fruible on any browser, can be found here:

https://dct-image-compression.herokuapp.com/

ALGORITHM

The compression algorithm takes a image and two integers (F and d) as input, then applies the following steps:

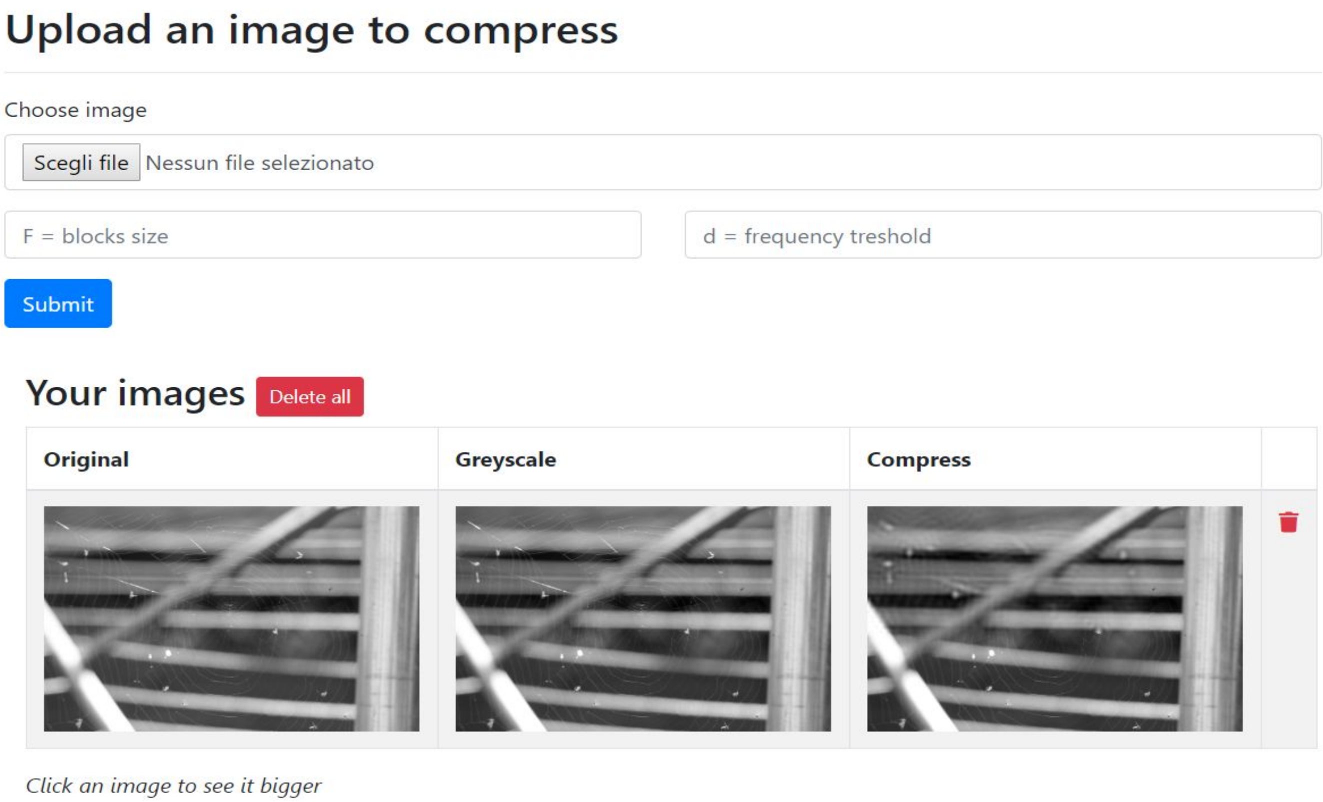
- splits the image into *FxF* pixel blocks
- for each block
  - applies the DCT2
  - filters frequencies where *row+col* >= *d*
  - applies the inverse DCT2
  - normalizes obtained values
- rebuilds the image as output

SOURCE CODE

```
bnimg = RGBtoGreyscale(imageio.imread(image)) # input image

for i in range(0, int(bnimg.shape[0] / F)): # splitting in FxF blocks
    for j in range(0, int(bnimg.shape[1] / F)):

        block = bnimg[(i*F):((i+1)*F), (j*F):((j+1)*F)]
        c = dctn(block, type=2, norm='ortho') # c = DCT2

        for k in range(0, block.shape[0] - 1):
            for l in range(0, block.shape[1] - 1):
                if(k + l >= d): # filtering frequences
                    c[k, l] = 0

        ff = idctn(c, type=2, norm='ortho') # ff = IDCT2(c)
        ff = np.round(ff) # normalization
        for index, value in np.ndenumerate(ff):
            if value < 0:       ff[index] = 0
            elif value > 255:   ff[index] = 255

        bnimg[(i*F):((i+1)*F), (j*F):((j+1)*F)] = ff

    imageio.imwrite(image_compress_path, bnimg)  # output
```

github.com/mferri17/dct-image-compression/webui/utils.py

USER INTERFACE



https://dct-image-compression.herokuapp.com/

WORKING NOTES

The App works both with color and grayscale images; the former are converted in grayscale applying the standard RGB conversion formula.

While the algorithm is designed to operate with .bmp images, our Python implementation also works properly providing .jpg images.

WARNING 1: on Firefox the HTML <input> accept attribute is bugged for .bpm images, you have to select the "all files" type filter when the browser prompts for the file system image selection.
WARNING 2: due to Heroku limits, the deployed app cannot convert an image if too big, or a small F is provided.

INPUT ARGUMENTS MEANING

As previously explained, the algorithm takes two integers as inputs:

- F determines how big are the blocks into which the image is split; the higher this value, the faster the algorithm but the lossier is the compression.
- d must have a value between *0* and *2F - 1* and it determines how many frequencies will be cut out; the lower this value, the more aggressive the compression. It doesn’t affect time performances.

EXAMPLES

f/2,8

1/50 sec.

ISO 1000

1365 x 2048 pixels

EXAMPLES

Original Grayscale

Compress   F=100, d = 50

EXAMPLES

Original Grayscale

Compress   F=100, d = 25

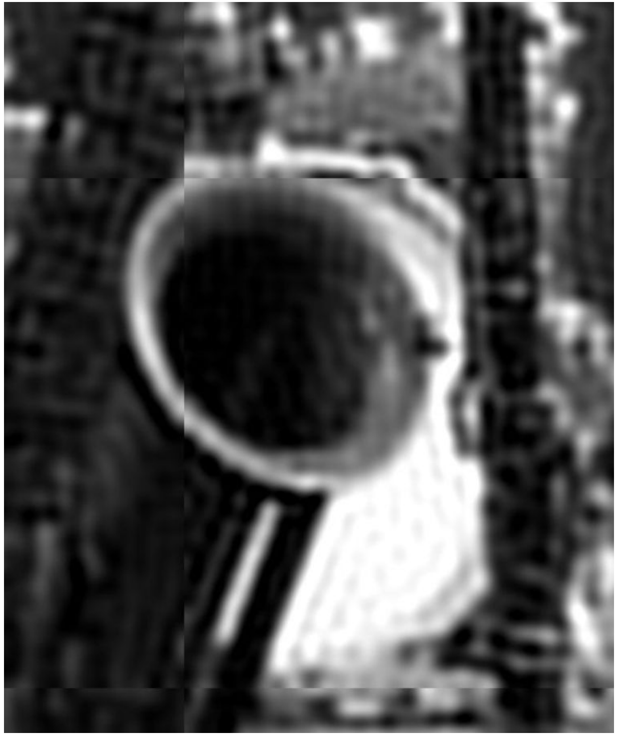
EXAMPLES

Original Grayscale

Compress F=100, d = 10

EXAMPLES

Original Grayscale



Compress F=500, d = 50

THANKS!

Any questions?

You can find us at:

m.ferri17@campus.unimib.it

n.habbash@campus.unimib.it

https://github.com/mferri17/dct-image-compression