UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica
Scuola di Scienze

Methods of Scientific Calculation
Project 2 - Image Compression via DCT

Candidati
Stranieri Francesco
Matr. 816551

Porto Francesco
Matr. 816042

Manan Mohammad Ali
Matr. 817205

Anno Accademico 2019–2020

Abstract

The DCT is a Fourier-related transform, that expresses a finite sequence of values in terms of a sum of cosine functions oscillating at different frequencies. It is vastly used in the image processing field, mostly for compression related tasks, such as in *JPEG*, a lossy format. Lossy compression permits reconstruction of an approximation of the original data, but reduces file size when compared to lossless formats, such as *BMP* (bitmap).

Contents

1 Introduction **4**

2 Theorical Background **4**
2.1 Fourier Series 4
2.2 DCT (and DCT2) 5
2.3 Gibbs Phenomenon 5

3 Environment **5**
3.1 DCT1 Implementation in Scipy 6
3.2 Our DCT1 Implementation 6
3.3 DCT2 Implementation in Scipy 7
3.4 Our DCT2 Implementation 7
3.5 Correctness of Our Implementations 7

4 Comparison Analysis **8**
4.1 Computer Specifications 8
4.2 Results Analysis 9

5 GUI Environment **10**
5.1 Interface Details 10
5.2 Interface Implementation 12
5.3 Compression Implementation 12

6 Result Analysis **13**
6.1 Images Comparison 13

A Code **17**
A.1 Our DCT1 (and IDCT1) Implementation 17
A.2 Our DCT2 (and IDCT2) Implementation 18
A.3 Correctness of Our Implementations 18
A.4 Comparison Analysis 20
A.5 Interface Implementation 20
A.6 Compression Implementation 22

1 Introduction

In this report we discuss our implementation of the **Discrete Cosine Transform** (DCT) in one and two dimension, called DCT1 and DCT2 respectively. We then compare our implementations in terms of *execution time* to the ones provided by Scipy, an open-source Python library. Finally, we present our GUI that compresses a greyscale bitmap image via a DCT2-based algorithm, while also providing a side-by-side comparison with the original image.

2 Theorical Background

In the next few subsections we will provide a brief introduction to key concepts that will be discussed upon in the report.

2.1 Fourier Series

Let $f : \mathbb{R} \to \mathbb{R}$ of period p can be approximated by a Fourier series:

$$f(x) = a_0 + \sum_{k=1}^{+\infty} \left(a_k \cdot \cos\left(\frac{2\pi k}{p} x\right) + b_k \cdot sen\left(\frac{2\pi k}{p} x\right)\right).$$

If f is integrable, then:

$$\begin{aligned} a_0 &= \frac{1}{p}\int_0^P f(x)dx, \\ a_k &= \frac{2}{p}\int_0^P f(x) cos\left(\frac{2\pi k}{p} x\right) dx, \\ b_k &= \frac{2}{p}\int_0^P f(x) sen\left(\frac{2\pi k}{p} x\right) dx, \end{aligned}$$

with $k \in \mathbb{N}, k \geq 1$.

Definition. *A function $f : \mathbb{R} \to \mathbb{R}$ is said to be periodic of period p ($p \in \mathbb{R}$, $p > 0$) if $f(x + p) = f(x)$, $\forall x \in \mathbb{R}$.*

Theorem (Dirichlet). *Let $f : \mathbb{R} \to \mathbb{R}$ p-periodic and piecewise-$\mathbb{C}^1$, then if f is continuous in a point $x \in R$, then the series converges in $f(x)$, otherwise it converges to the mean between the leftmost and rightmost value:*

$$\lim_{N \to +\infty} a_0 + \sum_{k=1}^{N} \left(a_k \cdot \cos\left(\frac{2\pi k}{p} x\right) + b_k \cdot sen\left(\frac{2\pi k}{p} x\right)\right) = f(x).$$

2.2 DCT (and DCT2)

Given a vector $v \in \mathbb{R}^n$, if we consider the canonical bases $\left\{e^j\right\}_{j=1}^N$ of $\mathbb{R}^n$, we can write:

$$v = \sum_{j=1}^{N} v_j \cdot e_j.$$

Let us now consider the w^k bases $\left\{w^k\right\}_{k=0}^{N-1}$, where $\left(w^k\right)_i = \cos\left(\pi k \frac{2i-1}{2N}\right)$. Suppose we want to represent the vector v with respect to w^k, that is we want to represent v as:

$$v = \sum_{k=0}^{N-1} c_k \cdot w^k,$$

where

$$c_k = \frac{1}{\alpha_k} \sum_{i=1}^{N} \cos\left(\pi k \frac{2i-1}{2N}\right),$$

and

$$\alpha_k = \begin{cases} N & if\, j = 0 \\ \frac{N}{2} & if\, j \neq 0. \end{cases}$$

This conversion represents the so called Discrete Cosine Transform. The inverse is also possible, and is called **Inverse Cosine Transform** (IDCT).

DCT2 is the DCT equivalent for matrices instead of vectors. In order to obtain the DCT2, the DCT should be applied first to each row of the matrix, and then to each column.

2.3 Gibbs Phenomenon

Gibbs phenomenon refers to the phenomenon where, in presence of a jump discontinuity in the function $f(x)$, its Fourier Series cannot accurately represent the function around the discontinuity itself. For example, this phenomenon can lead to graphical artefacts when compressing an image.

3 Environment

We used the Python open-source programming environment, with the SciPy library [1]. Scipy uses the Fast Fourier Transform in order to compute the DCT as efficiently as possible [2].

3.1 DCT1 Implementation in Scipy

The DCT1 of a given np.array can be obtained via the scipy.fftpack.dct [3]. This function returns a np.array containing the transform, and requires the following arguments:

- **x**: the input np.array.
- **type**: an integer representing the type of DCT to be computed (optional).
- **axis**: the axis over which to compute the transform (optional).
- **norm**: normalization mode (optional).
- **overwrite_x**: if the vector x should be destroyed after computing the transform (optional).

There are theoretically 8 types of the DCT but only 3 types are implemented in scipy. The DCT as we mean it refers to DCT type 2, while the IDCT refers to type 3. For a single dimension array x, dct(x, norm='ortho') is equal to the one from MATLAB dct(x), that is the one we used.

3.2 Our DCT1 Implementation

As seen in Algorithm 1, we compute iteratively both the a_k^i and c_k^i via two simple nested for loops, starting from the formulas in Section 2.2. We then return the vector containing all the c_k^i.

Algorithm 1 Our DCT1 implementation.

procedure OUR_DCT(vector)
  ▷ Variables initialization
  $N \leftarrow len(vector)$
  $c \leftarrow zeros(N)$
  for $k \in [0, N)$ **do**
    ▷ Compute a_k
    if $k = 0$ **then**
      $a_k \leftarrow \frac{1}{\sqrt{N}}$
    else
      $a_k \leftarrow \frac{1}{\sqrt{N/2}}$
    ▷ Compute c_k
    for $i \in [0, N)$ **do**
      $c_k \leftarrow c_k + vector[i] * \cos\left(\pi k \frac{2i-1}{2N}\right)$
    $c_k \leftarrow c_k * a_k$
return c

The full code is available in Appendix A.1.

3.3 DCT2 Implementation in Scipy

Even though Scipy has a native DCTN function [4], we decided to improve code reusability. So we decided to apply Scipy's DCT row-by-row and then column-by-column. Since SciPy's DCT works only column-by-column, we had to do the following:

1. Transpose the initial matrix.
2. Perform the DCT1 on each column (equivalent to the rows of the initial matrix).
3. Transpose the resulting matrix.
4. Perform the DCT1 on each column (equivalent to the columns of the initial matrix).

3.4 Our DCT2 Implementation

We obtain the DCT2 by simply applying our DCT1 function row-by-row first, and then column-by-column. No matrix transposition is therefore necessary.

The full code is available in Appendix A.2.

3.5 Correctness of Our Implementations

In order to verify that the DCT computed by our function is actually correct we compare its results to Scipy's DCT via the Algorithm 2.

Algorithm 2 Correctness of DCT and IDCT (SciPy vs Ours).

procedure DCT_AND IDCT_CORRECTNESS()

▷ Compare DCT
vectorExample ← *vector whose DCT is known*
vectorExampleDCT ← *scipy_dct*(*vectorExample*)
vectorExampleDCT1_OUR ← *our_dct*(*vectorExample*)
print *if vectorExampleDCT and vectorExampleDCT1_OUR are equivalent*

▷ Compare IDCT
vectorExampleIDCT ← *scipy_idct*(*vectorExampleDCT*)
vectorExampleIDCT1_OUR ← *our_idct*(*vectorExampleDCT*)
print *if vectorExampleIDCT and vectorExampleIDCT1_OUR are equivalent*

▷ Compare DCT2
matrixExample ← *matrix whose DCT2 is known*
matrixExampleDCT2 ← *scipy_dct*2(*matrixExample*)
matrixExampleDCT2_OUR ← *our_dct*2(*matrixExample*)
print *if matrixExampleDCT2 and matrixExampleDCT2_OUR are equivalent*

▷ Compare IDCT2
matrixExampleIDCT2 ← *scipy_idct*2(*matrixExampleDCT*2)
matrixExampleIDCT2_OUR ← *our_idct*2(*matrixExampleDCT*2)
print *if matrixExampleIDCT2 and matrixExampleIDCT2_OUR are equivalent*

The full code is available in Appendix A.3.

4 Comparison Analysis

4.1 Computer Specifications

The computer used for running the DCT2 comparison has the following specifications:

- **Device:** MacBook Pro
- **CPU:** 1,4 GHz Intel Core i5 quad-core
- **RAM:** 8 GB 2133 MHz LPDDR3
- **Storage:** SSD NVMe, 128 GB
- **OS:** macOS Catalina

4.2 Results Analysis

Since Scipy uses a FFT-based DCT we expect its efficiency to be much higher than our implementation's. More specifically, Scipy's DCT should have a time complexity of $O(logN * N^2)$, while ours should be around $O(N^3)$.

As a benchmark, we generate matrices starting from a 2x2 matrix up to a 100x100 one, with an increase of 1 in terms of size for each iteration, keeping also track of execution time.

The full code is available in Appendix A.4.

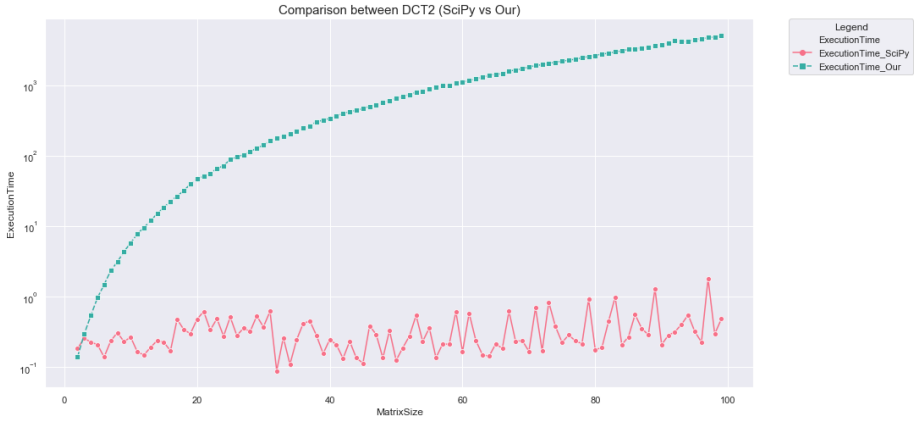


Figure 1: Comparison between DCT2 (SciPy vs Our Implementation).

From Figure 1, we notice that the results match our expectations. It's obvious to see that our implementation takes much more time than Scipy's, but does not exhibit its oscillating behavior.

In a professional environment, Scipy should obviously be preferred. Nonetheless, both implementations provide the same result, and could be used interchangeably.

5 GUI Environment

For the GUI creation, we used PyQt5 [5], the most popular Python framework for cross-platform development. PyQt provides Python bindings for Qt, which is a native C++ library for graphical user interfaces, used in many well-known programs such as Spotify.

5.1 Interface Details

In Figure 2 the interface of our application is shown. By default an image with a white "C" on a black background is displayed.

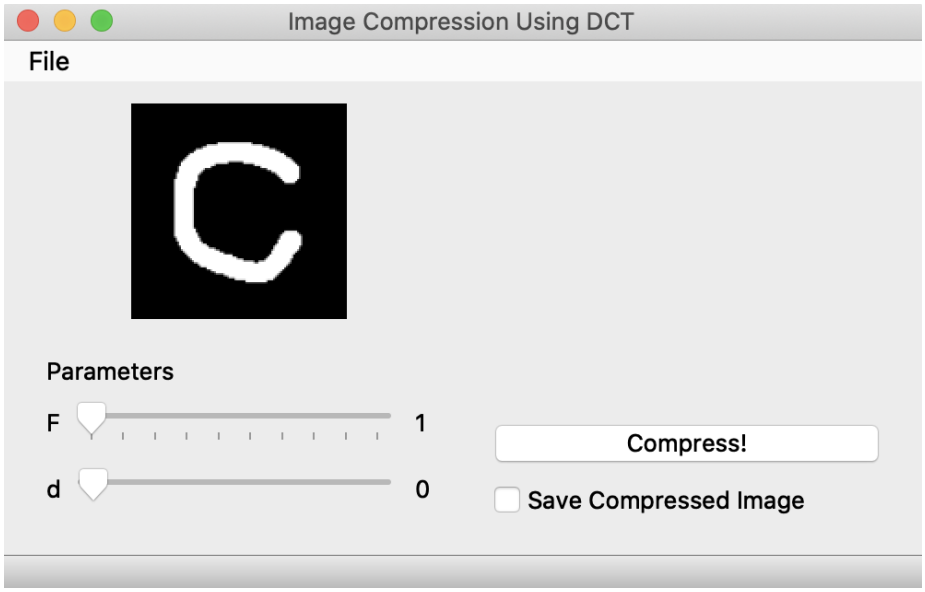


Figure 2: The main screen of our application.

From here, an image in BMP format can be loaded, while other image formats are ignored. Also, only greyscale images are supported, so that when the user tries to load a non-greyscale image, the error in Figure 3 is displayed.

Then, the user can choose the parameters that will be used for compression, F and d. Our program automatically determines the possible values for these parameters, according to the size of the image, so that no incorrect value can be given. These values will be explained in further details in the next few sections.

After clicking on 'Compress!' button, the compressed version of the image is shown next to the original one, so that they are easier to compare, as seen in Figure 4.

The resulting image can also be saved by selecting the checkbox. Note that the images displayed in the interface get scaled automatically according to the size of the window.

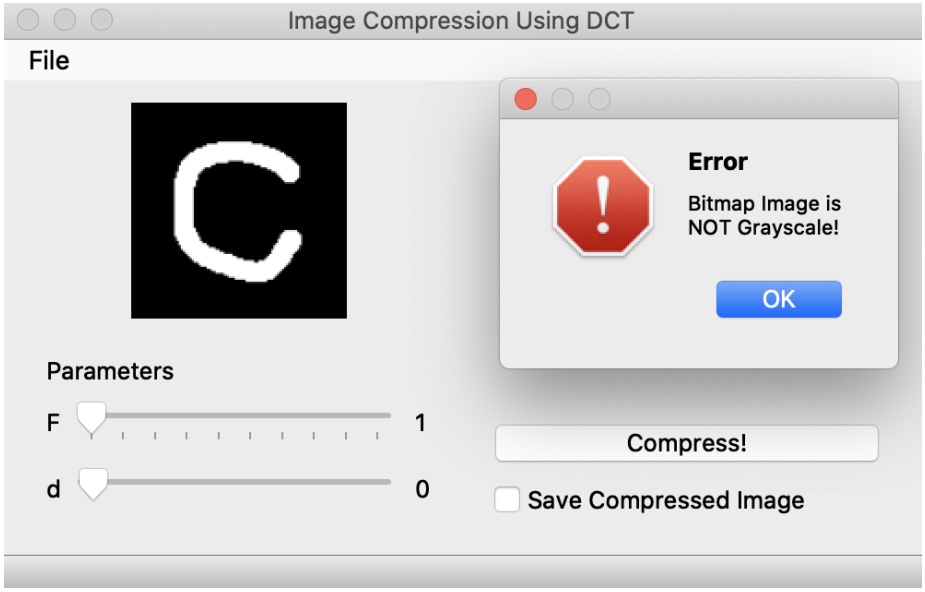


Figure 3: The error screen when a non-greyscale image is selected.

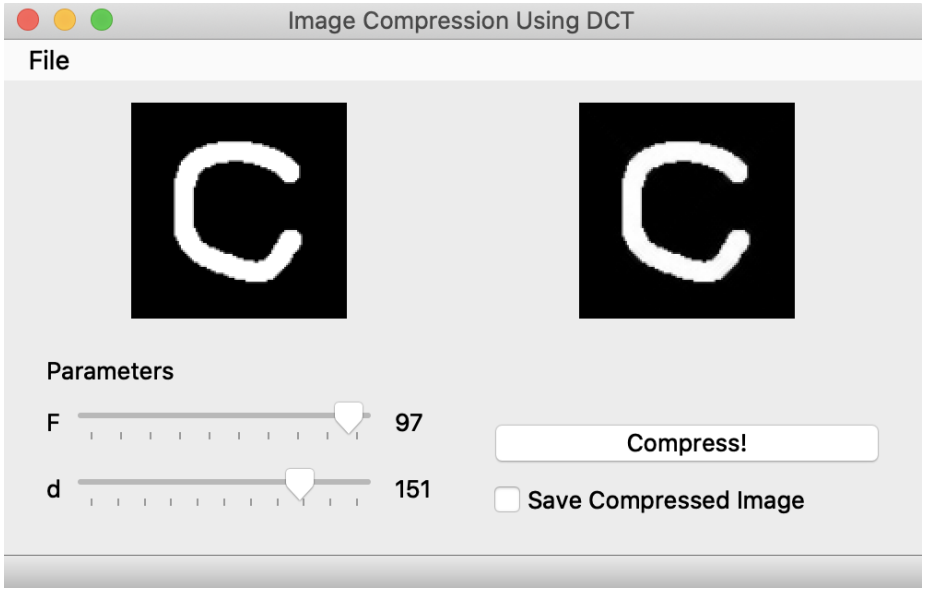


Figure 4: Obtaining the resulting compressed image.

5.2 Interface Implementation

The main content is displayed on a QGridLayout(). This component divides the interface in rows and columns, where QWidgets can be placed, by giving their position on the grid. A QWidget is a component of the interface, for example a picture, a button or a slider.

Our grid is structured like this:

- In the upper left (position 0,0), the image loaded by the user is displayed.
- In the lower left (position 1,0), a subgrid (also a QGridLayout()) is displayed:
  - In position (0,0), (1,0) and (2,0) QLabels for "Parameters", "F" and "d" are shown.
  - in position (1,1) a QSlider is present for selecting F. F can assume values between 1 and the minimum between width and height of the image. Once a new image is loaded, the value of F gets updated via a **lambda function** that makes use of the Qt's signals and slots mechanism.
  - in position (2,1) a QSlider is present for selecting d, whose value ranges between 0 and $2 * (F - 2)$. The maximum value changes according to the F chosen previously; this has been achieved by using another lambda function.
- In the lower right (position 1,1) another subgrid (also a QGridLayout()) is displayed:
  - In position (0,0) a blank QLabel is displayed to properly align the interface.
  - In position (1,0) a QPushButton is shown, which when clicked runs the compression on the loaded image.
  - In position (2,0) a QCheckBox is displayed, which allows the image to be saved for further analysis.

The full code is available in Appendix A.5.

5.3 Compression Implementation

Initially the loaded image is contained in a QPixmap object. In order to obtain the matrix representation of this image, we call the toImage() method. This method considers the image in the RGBA (Red, Green, Blue e Alpha) color model, where each pixel is represented as a 32-bit integer, each one of them occupies 8 bit (equivalent to a byte) for each channel. We decided to extract the first channel but any channel, other than alpha, could have been chosen. This is because for a greyscale image the RGB channels share the same value for each pixel, but this does not apply to alpha, which represents transparency.

Algorithm 3 Algorithm for image compression.

```
procedure compressImage(sourceImage)
    destinationImage ← sourceImage
    ▷ Get 'F' and 'd' value from GUI
    F ← F.value()
    d ← d.value()
    for every block FxF in destinationImage do
        tmp ← scipy_dct2(block)
        ▷ Remove frequencies over cutoff
        for k ∈ [0, F) do
            for l ∈ [0, F) do
                if k + l > d then
                    tmp[k,l] ← 0
        block ← scipy_idct2(tmp)
    return destinationImage
```

Finally, we can run the Algorithm 3 on this matrix.

The most crucial part of this algorithm consists in eliminating the frequencies c_{kl} with $k + l >= d$. This must be done for each block of size FxF starting from the top left.

Our destination image is initially a $1 : 1$ copy of the starting image. The remaining bytes which do not get compressed due to F's value maintain the same value as the starting image.

Note that we treat each pixel as float instead of uint8, in order to avoid problems due to truncation issues.

It is important to notice that in terms of file size the resulting image will not be smaller than the starting one, since we do not perform any final encoding (such as *Huffman's encoding* in *JPEG* [6]).

The code which describes the previous procedure is available in Appendix A.6.

6 Result Analysis

We now present the results of multiple runs of our program, with different starting images and configurations of parameters.

6.1 Images Comparison

As can be seen in Figure 5 by using big values for both F and d, the resulting image is almost indistinguishable from the starting one.

(a) The original image.

(b) The compressed image.

Figure 5: F=335 and d=425 as compression parameters.

In Figure 6, F and d were much smaller, thus resulting in a much lower quality image overall, which looks "noisy".

(a) The original image.

(b) The compressed image.

Figure 6: F=335 and d=425 as compression parameters.

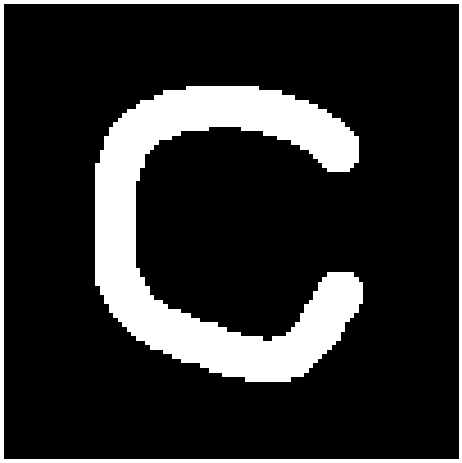
We also wanted to provide an example of a good balance between quality and compression. Some small artifacts can be noticed in Figure 7, however the overall image quality is quite good.

(a) The original image.

(b) The compressed image.

Figure 7: F=336 and d=330 as compression parameters.

Finally, as can be seen in Figure 8, due to a sharp contrast in border of the "C" from white to black, Gibbs phenomenon can be observed.



(a) The original image.

(b) The compressed image.

Figure 8: F=50 and d=49 as compression parameters.

References

[1] Scipy library. https://scipy.org/scipylib/. Accessed: 2020-06-07.

[2] J. Makhoul. A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34, 1980.

[3] scipy.fftpack.dct. https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.dct.html. Accessed: 2020-06-07.

[4] scipy.fftpack.dctn. https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.dctn.html. Accessed: 2020-06-07.

[5] Pyqt. https://wiki.python.org/moin/PyQt. Accessed: 2020-06-07.

[6] Jpeg huffman. http://www.robertstocker.co.uk/jpeg/jpeg_new_11.htm. Accessed: 2020-06-10.

A Code

A.1 Our DCT1 (and IDCT1) Implementation

```
def dct1_Our(vector):
    N = len(vector)
    coefficients = np.zeros(N)

    for k in range(0, N):
        alpha = 1/np.sqrt(N) if k == 0 else 1/np.sqrt(N/2)
        for i in range(0, N):
            coefficients[k] += vector[i] *
                np.cos(k * np.pi * (2*i+1) / (2*N))
        coefficients[k] *= alpha

    return coefficients

def idct1_Our(coefficients):
    N = len(coefficients)
    vector = np.zeros(N)

    for i in range(0, N):
        for k in range(0, N):
            alpha = 1/np.sqrt(N) if k == 0 else 1/np.sqrt(N/2)
            vector[i] += coefficients[k] * alpha * \
                np.cos(k * np.pi * (2*i+1) / (2*N))

    return vector
```

A.2 Our DCT2 (and IDCT2) Implementation

```
def dct2_Our(matrix):
    N = matrix.shape[0]
    M = matrix.shape[1]
    coefficients = np.zeros((N, M))

    for k in range(N):
        coefficients[k, :] = dct1_Our(matrix[k, :])
    for l in range(M):
        coefficients[:, l] = dct1_Our(coefficients[:, l])

    return coefficients


def idct2_Our(coefficients):
    N = coefficients.shape[0]
    M = coefficients.shape[1]
    matrix = np.zeros((N, M))

    for k in range(N):
        matrix[k, :] = idct1_Our(coefficients[k, :])
    for l in range(M):
        matrix[:, l] = idct1_Our(matrix[:, l])

    return matrix
```

A.3 Correctness of Our Implementations

```
vectorExample = np.array([231, 32, 233, 161, 24, 71, 140, 245])
print('VectorExample')
print(vectorExample)
print()

print('DCT1 with SciPy')
vectorExampleDCT = dct(vectorExample, type=2, norm='ortho')
print(vectorExampleDCT)

print('DCT1_Our')
vectorExampleDCT1_Our = dct1_Our(vectorExample)
print(vectorExampleDCT1_Our)

print('Are all elements equal?', np.allclose(
    vectorExampleDCT, vectorExampleDCT1_Our))
print()
```

```
print('IDCT1 with SciPy')
vectorExampleIDCT = idct(vectorExampleDCT, type=2, norm='ortho')
print(vectorExampleIDCT)

print('IDCT1_Our')
vectorExampleIDCT1_Our = idct1_Our(vectorExampleDCT)
print(vectorExampleIDCT1_Our)

print('Are all elements equal?', np.allclose(
    vectorExampleIDCT, vectorExampleIDCT1_Our))
print()
print()


print('MatrixExample')
matrixExample = np.array([
    [231, 32, 233, 161, 24, 71, 140, 245],
    [247, 40, 248, 245, 124, 204, 36, 107],
    [234, 202, 245, 167, 9, 217, 239, 173],
    [193, 190, 100, 167, 43, 180, 8, 70],
    [11, 24, 210, 177, 81, 243, 8, 112],
    [97, 195, 203, 47, 125, 114, 165, 181],
    [193, 70, 174, 167, 41, 30, 127, 245],
    [87, 149, 57, 192, 65, 129, 178, 228],
])
print(matrixExample)
print()

print('DCT2 with SciPy')
matrixExampleDCT2 = dct(dct(matrixExample.transpose(
), type=2, norm='ortho').transpose(), type=2, norm='ortho')
print(matrixExampleDCT2)

print('DCT2_Our')
matrixExampleDCT2_Our = dct2_Our(matrixExample)
print(matrixExampleDCT2_Our)

print('Are all elements equal?', np.allclose(
    matrixExampleDCT2, matrixExampleDCT2_Our))
print()
print()

print('IDCT2 with SciPy')
matrixExampleIDCT2 = idct(idct(matrixExampleDCT2.transpose(
), type=2, norm='ortho').transpose(), type=2, norm='ortho')
print(matrixExampleIDCT2)
```

```
print('IDCT2_Our')
matrixExampleIDCT2_Our = idct2_Our(matrixExampleDCT2)
print(matrixExampleIDCT2_Our)

print('Are all elements equal?', np.allclose(
    matrixExampleIDCT2, matrixExampleIDCT2_Our))
print()
print()
```

A.4 Comparison Analysis

```
low = 0
high = 255

iterations = 100
startIterations = 2
matrixSize = np.zeros(iterations - startIterations)
timeDCT_SciPy = np.zeros(iterations - startIterations)
timeDCT_Our = np.zeros(iterations - startIterations)

for i in range(startIterations, iterations):

    matrix = np.random.randint(low, high, size=(i, i))
    matrixSize[i-startIterations] = i
    # print(matrix)

    start = time.process_time()
    dct(dct(matrix.transpose(),
        type=2, norm='ortho').transpose(), type=2, norm='ortho')
    end = time.process_time()
    timeDCT_SciPy[i-startIterations] = "%.3f" % ((end - start) * 1000)

    start = time.process_time()
    dct2_Our(matrix)
    end = time.process_time()
    timeDCT_Our[i-startIterations] = "%.3f" % ((end - start) * 1000)
```

A.5 Interface Implementation

```
class MainContent(QWidget):
    def __init__(self):
        super(MainContent, self).__init__()

        mainGrid = QGridLayout()
```

```
# 0, 0
self.originalImage = ImageLabel('prova.bmp')
mainGrid.addWidget(self.originalImage, 0, 0)

# 1, 0
subGrid = QGridLayout()

# Parameters
subGrid.addWidget(QLabel('Parameters'), 0, 0, 1, 3)

imageSize = self.originalImage.pixmap.size()
minimumSize = min(imageSize.height(), imageSize.width())

# F
subGrid.addWidget(QLabel('F'), 1, 0)

self.F = QSlider(Qt.Horizontal)
self.F.setMinimum(1)
#self.f.setFocusPolicy(Qt.StrongFocus)
self.F.setTickPosition(QSlider.TicksBelow)
self.F.setSingleStep(1)

self.F.setMaximum(minimumSize)
self.F.setTickInterval(minimumSize/10)
self.F.valueChanged.connect(
    lambda: self.d.setMaximum(2 * self.F.value() - 2)
    or self.d.setTickInterval((2 * self.F.value() - 2) / 10)
    or self.Fvalue.setText(str(self.F.value())))
subGrid.addWidget(self.F, 1, 1)

self.Fvalue = QLabel('1')
subGrid.addWidget(self.Fvalue, 1, 2)

# d
subGrid.addWidget(QLabel('d'), 2, 0)

self.d = QSlider(Qt.Horizontal)
self.d.setMinimum(0)
#self.d.setFocusPolicy(Qt.StrongFocus)
self.d.setTickPosition(QSlider.TicksBelow)
self.d.setSingleStep(1)

self.d.setMaximum((2 * self.F.value() - 2))
self.d.valueChanged.connect(
    lambda: self.dvalue.setText(str(self.d.value())))
subGrid.addWidget(self.d, 2, 1)
```

```
        self.dvalue = QLabel('0')
        subGrid.addWidget(self.dvalue, 2, 2)

        innerWidget = QWidget()
        innerWidget.setLayout(subGrid)
        mainGrid.addWidget(innerWidget, 1, 0)

        # 1, 1
        buttonGrid = QGridLayout()

        buttonGrid.addWidget(QLabel(), 0, 0)

        # Button
        button = QPushButton('Compress!', self)
        button.setToolTip('Run Compression')
        button.clicked.connect(self.compressImage)
        buttonGrid.addWidget(button, 1, 0)

        # CheckBox
        self.checkbox = QCheckBox()
        self.checkbox.setText('Save Compressed Image')
        buttonGrid.addWidget(self.checkbox, 2, 0)

        buttonWidget = QWidget()
        buttonWidget.setLayout(buttonGrid)
        mainGrid.addWidget(buttonWidget, 1, 1)

        # 0, 1
        self.DCTImage = ImageLabel()
        mainGrid.addWidget(self.DCTImage, 0, 1)


        mainGrid.setColumnStretch(0, 1)
        mainGrid.setColumnStretch(1, 1)
        mainGrid.setRowStretch(0, 10)
        mainGrid.setRowStretch(1, 1)
        self.setLayout(mainGrid)
```

A.6 Compression Implementation

```
@pyqtSlot()
def compressImage(self):
    print('Running Compression')

    sourceImage = self.originalImage.pixmap.toImage()
```

```
sourceHeight = sourceImage.height()
sourceWidth = sourceImage.width()
sourceChannel = 4

sourceImageBits = sourceImage.bits()
sourceImageBits.setsize(sourceHeight * sourceWidth * sourceChannel)

sourceImageBytes = np.frombuffer(
    sourceImageBits, np.uint8).reshape((
        sourceHeight, sourceWidth, sourceChannel))
sourceImageBytes = sourceImageBytes[:,:,0].astype(float)

# np.savetxt("sourceImageBytes_Before.csv",
    sourceImageBytes, delimiter = ",", fmt = "%d")

fValue = self.F.value()
dValue = self.d.value()

for h in range(0, sourceHeight - fValue + 1, fValue):
    for w in range(0, sourceWidth - fValue + 1, fValue):
        tmp = sourceImageBytes[h:h+fValue, w:w+fValue]
        tmp = dctn(tmp, type=2, norm='ortho')

        for k in range(0, tmp.shape[1]):
            for l in range(0, tmp.shape[0]):
                if k + l >= dValue:
                    tmp[k, l] = 0

        tmp = idctn(tmp, norm='ortho')
        sourceImageBytes[h:h+fValue, w:w+fValue] = tmp

# np.savetxt("sourceImageBytes_After.csv",
    sourceImageBytes, delimiter = ",", fmt = "%d")

sourceImageBytes = sourceImageBytes.clip(0, 255).round().astype(np.uint8)
destinationImage = QtGui.QPixmap(
    QtGui.QImage(bytes(sourceImageBytes),
        sourceImageBytes.shape[1],
        sourceImageBytes.shape[0],
        int(sourceImageBytes.nbytes/sourceHeight),
        QtGui.QImage.Format_Grayscale8))

if self.checkbox.isChecked():
    formatOut = "%d-%m-%Y %H:%M:%S"
    date = datetime.now().strftime(formatOut)
```

```
        destinationImageName = date + '.bmp'
        destinationImage.save(destinationImageName, 'bmp')

    self.DCTImage.setPixmap(destinationImage)
```