# DCT - Progetto 2

Bianca Stan 816045

July 11, 2020

# Contents


**1 DCT - introduzione** **2**
- 1.1 DCT monodimensionale . . . 2
- 1.2 DCT bidimensionale . . . 2
  - 1.2.1 Normalizzazione . . . 3
- 1.3 IDCT . . . 3

**2 Progetto** **4**
- 2.1 Richieste . . . 4
- 2.2 Scelte di design . . . 4
- 2.3 Caratteristiche del hardware . . . 5

**3 Prima parte - implementazione DCT** **6**
- 3.1 Implementazione DCT non ottimizzata . . . 6
- 3.2 Implementazione ottimizzata . . . 8
- 3.3 Confronto con scipy.fft . . . 8

**4 Seconda parte - compressione immagini** **11**
- 4.1 Richieste . . . 11
- 4.2 Struttura del codice . . . 11
- 4.3 Risultati su immagini . . . 18
  - 4.3.1 F = 10, d = 4 . . . 18
  - 4.3.2 F = 50, d = 5 . . . 21
  - 4.3.3 F = 50, d = 50 . . . 23
  - 4.3.4 F = 200, d = 100, 200, 300 . . . 25

# Chapter 1

# DCT - introduzione

La Discrete Cosine Transform è una funzione che, presa in input una sequenza di dati finiti, ritorna in output una sua rappresentazione come somma di funzioni coseno con frequenze diverse.
È una tecnica che viene spesso applicata nella compressione delle immagini, in quanto il passaggio da un dominio spaziale ad uno di frequenze permette di identificare ridondanze. Per esempio, il formato JPEG utilizza la DCT per comprimere le immagini (in modo lossy).

## 1.1 DCT monodimensionale

Se la DCT è applicata ad un vettore monodimensionale di lunghezza N, allora la trasformazione del vettore V nel corrispondente vettore F (riscrittura dei coefficienti, da base canonica a base frequenziale) è regolata dalla seguente formula:

$$F(i) = c_i \sum_{x=0}^{N-1} V(x) cos(\frac{\pi i(2x+1)}{2N})$$

con

$$c_i = \begin{cases} \sqrt{\frac{1}{N}} & i = 0 \\ \sqrt{\frac{2}{N}} & i \neq 0 \end{cases}$$

## 1.2 DCT bidimensionale

Quando si tratta invece di vettori bidimensionali (matrici), ci sono due possibilità: ragionare direttamente in due dimensioni, oppure lavorare prima su una dimensione e poi sull'altra.
Nel primo caso, si procede applicando la DCT bidimensionale. Questa funzione trasforma direttamente la matrice V $\in \mathcal{R}^{NxN}$ in una matrice equivalente F nello spazio delle frequenze. Questi nuovi coefficienti si ottengono mediante la formula:

$$F(i,j) = c_i c_j \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} V(x,y) cos(\frac{\pi i(2x+1)}{2N}) cos(\frac{\pi j(2y+1)}{2N})$$

con

$$c_i, c_j = \begin{cases} \sqrt{\frac{1}{N}} & i, j = 0 \\ \sqrt{\frac{2}{N}} & i, j \neq 0 \end{cases}$$

Alternativamente, si può ottenere la matrice F applicando la DCT monodimensionale prima sulle righe (o colonne) di V per ottenere una matrice intermedia $F'$, e ripetendo l'operazione sulle colonne (o righe) di quest'ultima.

### 1.2.1 Normalizzazione

Da notare che si è presa in considerazione la DCT con normalizzazione ortogonale, ossia in cui le basi dello spazio vettoriale sono tra loro ortogonali.

## 1.3 IDCT

L'operazione inversa della DCT (appunto, IDCT) prende in input un array F (mono o bidimensionale) e lo riporta in base canonica. In particolare, attraverso la formula:

$$V(i) = \sum_{x=0}^{N-1} c_x F(x) cos(\frac{\pi x(2i+1)}{2N})$$

con

$$c_i = \begin{cases} \sqrt{\frac{1}{N}} & i = 0 \\ \sqrt{\frac{2}{N}} & i \neq 0 \end{cases}$$

applicata prima alle righe, e poi alle colonne (o viceversa), come per la DCT, si può tornare alla matrice originale V a partire dalla matrice trasformata F.

# Chapter 2

# Progetto

## 2.1 Richieste

Il progetto si compone di due parti:

1. implementazione della DCT bidimensionale
   (a) creazione di una funzione che, presa in input una matrice, ne restutisca la sua DCT
   (b) paragone in termini di tempo di esecuzione con un'implementazione ottimizzata di una libreria open source
2. sviluppo di una semplice applicazione GUI volta a comprimere immagini BMP a scala di grigio, utilizzando la DCT

## 2.2 Scelte di design

Per lo sviluppo di entrambe le parti si è scelto di utilizzare il linguaggio di programmazione Python. In particolare, per il punto 1.b, la libreria di controllo scelta è stata **scipy.fft**. Questa decisione è stata presa sulla base della popolarità della libreria, del suo mantenimento continuo (l'ultimo release stabile risale al 4 Luglio 2020), e dell'alta qualità della documentazione.
Si è considerato anche l'utilizzo di una seconda libreria, ossia **cvxopt**, ma l'istallazione di tale package è risultato impossibile (sia attraverso package manager - pip e Anaconda -, che manualmente a partire dai binari .whl).

Per quanto riguarda invece lo sviluppo della seconda richiesta, la GUI è stata sviluppata mediante **tkinter**. Questa libreria rappresenta lo standard per lo sviluppo di interfacce grafiche, ed è automaticamente inclusa con l'istallazione di Python in ambiente Windows, Unix e Mac OS X.
Non avendo esperienza pregressa di sviluppo di GUI in ambiente Windows/Python, questa è stata una scelta naturale - la libreria è ben documentata ed in rete sono disponibili molteplici tutorial introduttivi al suo utilizzo.

## 2.3 Caratteristiche del hardware

L'elaboratore utilizzato è un portatile di marca HP, con sistema operativo Windows 10.
Il processore è un Intel Core i5 di settima generazione, con frequenza base 2.70GHz (turbo fino a 3.40GHZ). Ha a disposizione 128KB per quanto riguarda la cache di primo livello, 512KB per quella di secondo livello ed infine 3MB di cache di terzo livello.
Per quanto riguarda la memoria, invece, il PC è munito di 8GB di RAM (velocità 2133MHz).

# Chapter 3

# Prima parte - implementazione DCT

## 3.1 Implementazione DCT non ottimizzata

Per svolgere la prima parte del progetto, si è preferito utilizzare il secondo modo di applicare la DCT ad array multidimensionale: ossia, lavorando prima per righe ed in seguito per colonne.
Il codice si articola in una classe che espone all'utente metodi statici - in particolare, **proposed_DCT** e **proposed_IDCT** sono i due metodi generale che ritornano rispettivamente la DCT o IDCT di qualsiasi array dato in input, a prescindere che esso sia mono o bidimensionale.
Sono gli unici resi "pubblici". Dato che non ci sono modificatori di visibilità in Python, i nomi degli altri metodi sono stati preceduti da un doppio underscore per indicare che il loro utilizzo dovrebbe restare interno alla classe.
**proposed_DCT1** e **proposed_IDCT1**, invece, sono i metodi specifici (di convenienza/appoggio) che svolgono la DCT/IDCT su array monodimensionali.
Infine, **alpha_factor** è il metodo di convenienza utilizzato per calcolare i coefficienti $c_i/c_x$. Si è preferito calcolarli una volta sola e salvarli in un array per evitare di doverli ricalcolare ad ogni ciclo (specie nella IDCT, dove ogni fattore sarebbe stato calcolato $N^2$ volte, in quanto viene utilizzato nel ciclo for innestato).

Listing 3.1: Proposed DCT

```
import numpy as np
import scipy as sp


class DCT:

    @staticmethod
    def proposed_DCT(matrix):
        """ Calculates DCT on given matrix or array.
        """
        if(type(matrix)!=np.ndarray):
            try:
                matrix = np.ndarray(matrix)
            except Exception:
                raise Exception("Cannot convert to numpy ndarray")
        if(len(matrix.shape) == 1):
            return DCT.__proposed_DCT1(matrix)
```

```
        elif(len(matrix.shape) == 2):
            n = matrix.shape[0]
            m = matrix.shape[1]

            dct = np.zeros(matrix.shape)
            for i in range(n):
                dct[i] = DCT.__proposed_DCT1(matrix[i])
            for j in range(m):
                dct[:, j] = DCT.__proposed_DCT1(dct[:, j])
            return dct
        else:
            raise Exception("Only 1D and 2D arrays accepted")

    @staticmethod
    def proposed_IDCT(matrix):
        if(type(matrix)!=np.ndarray):
            try:
                matrix = np.ndarray(matrix)
            except Exception:
                raise Exception("Cannot convert to numpy ndarray")
        if(len(matrix.shape) == 1):
            return DCT.__proposed_IDCT1(matrix)
        elif(len(matrix.shape) == 2):
            n = matrix.shape[0]
            m = matrix.shape[1]
            idct = np.zeros(matrix.shape)
            for i in range(n):
                idct[i] = DCT.__proposed_IDCT1(matrix[i])
            for j in range(m):
                idct[:, j] = DCT.__proposed_IDCT1(idct[:, j])
            return idct
        else:
            raise Exception("Only 1D and 2D arrays accepted")

    @staticmethod
    def __proposed_DCT1(array):
        length = array.shape[0]
        dct = np.zeros(length)
        alpha = DCT.__alpha_factor(length)
        for i in range(length):
            sum = 0
            for j in range(length):
                sum += array[j]*np.cos((np.pi*i*(2*j+1))/(2*length))
            dct[i] = alpha[i]*sum
        return dct

    @staticmethod
    def __proposed_IDCT1(array):
        length = array.shape[0]
        idct = np.zeros(length)
        alpha = DCT.__alpha_factor(length)
        for i in range(length):
            for j in range(length):
                idct[i] += array[j]*alpha[j]*np.cos((np.pi*j*(2*i+1))/(2*length))
        return idct

    @staticmethod
```

```
    def __alpha_factor(N):
        alpha = np.zeros(N)
        alpha[0] = np.sqrt(1/N)
        alpha[1:] = np.sqrt(2/N)
        return alpha
```

## 3.2 Implementazione ottimizzata

La libreria utilizzata utilizza la seguente formula di calcolo quando si sceglie la normalizzazione ortogonale:

$$F(i) = 2f_i \sum_{x=0}^{N-1} V(x) cos(\frac{\pi i(2x+1)}{2N})$$

con

$$f_i = \begin{cases} \sqrt{\frac{1}{4N}} & i = 0 \\ \sqrt{\frac{1}{2N}} & i \neq 0 \end{cases}$$

Per quanto riguarda, invece, la IDCT (DCT-III, normalizzazione ortogonale), viene usata la formula:

$$V(i) = \frac{V(0)}{\sqrt{N}} + \sqrt{\frac{2}{N}} \sum_{x=1}^{N-1} F(x) cos(\frac{\pi x(2i+1)}{2N})$$

## 3.3 Confronto con scipy.fft

Per confrontare i tempi di esecuzione tra il metodo chiamato "proposed" e il metodo ottimizzato della libreria scipy, si è creato un modulo ad hoc. Questo espone un metodo chiamato **compare_execution_times** che prende in input un intero (n). A partire da una dimensione minima 8x8 vengono generate n matrici randomiche, ciascuna di lato un'unità più grande della precedente, fino ad arrivare ad una matrice di lato 8 + x.
Utilizzando la libreria **timeit**, vengono misurati i tempi di elaborazione della DCT sia per il metodo proposto che per quello ottimizzato della libreria scipy, sulla stessa matrice. L'operazione viene fatta 10 volte, per poi mediare il tempo totale onde ovviare a possibili inaccuratezze nella misurazione dovute all'interferenza di altri processi.
Alla fine, i tempi vengono plottati grazie a **pyplot.matplotlib**.

Listing 3.2: Time Comparison

```
xfrom DCT import DCT
from scipy import fftpack as fft
import timeit
import numpy as np
from matplotlib import pyplot as plt

def generate_matrix(N):
    """
    Creates a random square matrix of pixels, size NxN.
```

```
    Parameters
    ----------
    N: int

    Returns
    -------
    matrix: ndarray
    """
    matrix = np.random.random_integers(0, 255, (N, N))

    return matrix



def compare_execution_times(executions):
    """
    Runs both proposed DCT and library DCT ten times. Then averages out the
    times and plots the results.
    Parameters
    ----------
    executions: int
            units by which to increment matrix size

    """

    #setup array for graphing execution times. Toggle comments in order to
    #test either home made or library fft
    fftDCTTime = np.zeros(executions+8)


    ourDCTTime = np.zeros(executions+8)

    #declare as global in order for timeit to be able to use it
    global matrix
    n3 = np.zeros_like(ourDCTTime)
    for i in np.arange(8, 8+executions):
        n3[i] = np.power(i, 3)
        matrix = generate_matrix(i)
        print(i)
        setupHomeMade = '''from DCT import DCT
        '''
        setupFFT = '''from scipy import fftpack as fft
        '''
        # homeMadeDCT = np.zeros((i, i))
        fftDCT = np.zeros((i, i))

        ourCode = '''homeMadeDCT = DCT.proposed_DCT(matrix)
        '''


        fftCode = '''fftDCT = fft.dctn(matrix, norm='ortho')
        '''
        ourDCTTime[i] = timeit.timeit(ourCode, setupHomeMade,
        globals = globals(), number=10)/10

        fftDCTTime[i] = timeit.timeit(fftCode, setupFFT,
        globals = globals(), number=10)/10
```

```
library = plt.semilogy(fftDCTTime, label="Library DCT")
proposed = plt.semilogy(ourDCTTime, label="Proposed DCT")
n3_plot = plt.semilogy(n3, label="N^3")
plt.legend()
plt.xlim(xmin=8, xmax = executions+8)
plt.xlabel("Matrix dimension")
plt.ylabel("Execution Time")

plt.show()
```

Gli ordini di grandezza della complessità dei due metodi è $O(N^3)$ per quanto riguarda il metodo proposto, non ottimizzato. Invece, **fftpack** ha una complessità $O(N^2 \log(N))$ - come atteso, l'andamento è molto irregolare.

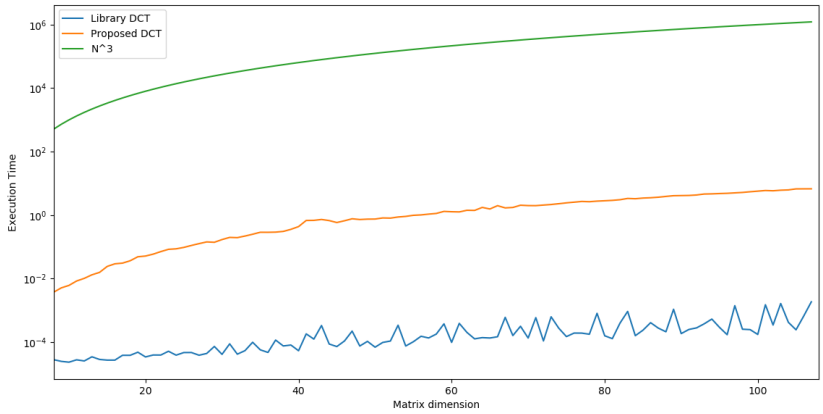


Figure 3.1: Grafico semilogaritmico dei tempi di esecuzione

# Chapter 4

# Seconda parte - compressione immagini

## 4.1 Richieste

La seconda parte del progetto prevedeva lo sviluppo di un'applicazione che permettesse il caricamento di un'immagine BMP da comprimere.
L'utente può allora scegliere il lato F dei blocchi in cui suddividere l'immagine, e una soglia d per le frequenze da eliminare. Una volta inserito un input valido (lato minore o uguale alla dimensione minima dell'immagine, soglia minore uguale a 2F-1), l'immagine viene suddivisa in blocchi della dimensione desiderata a cui viene applicata la DCT bidimensionale.
Dalla matrice così ottenuta vengono scartate le frequenze desiderate (ossia, qualsiasi frequenza in posizione (i, j) tale che d $\leq$ i+j). Finalmente, si applica la DCT inversa per tornare in base canonica e si ricompongono i blocchi nell'ordine originale per ottenere l'immagine compressa.

## 4.2 Struttura del codice

Il codice è suddiviso in due file: il primo si occupa della creazione della finestra e della gestione dell'elaborazione. Nel secondo, invece, sono inserite varie classi, ciascuna rappresentante di una pagina dell'applicazione: la prima che gestisce l'input dell'immagine, la seconda che gestisce l'input dei parametri d ed F e la validazione dell'input, ed infine la terza che mostra a schermo l'originale e l'immagine compressa una di fianco all'altra.

Listing 4.1: GUI application

```
import tkinter as tk
import pages
from tkinter.filedialog import askopenfilename
from matplotlib.pyplot import imshow
from matplotlib import pyplot as plt

import cv2
from scipy import fftpack as fft
import numpy as np


class Application():
```

```
    root_window = None
    main_view = None
    image = None


####SETUP METHODS FOR FIRST VIEW OF APP####
    def __init__(self):
        self.root_window = tk.Tk()
        self.setup()
        self.layout()


    def setup(self):
        self.main_view = pages.FileSelection(self.root_window, self.open_file)

    def layout(self):
        self.root_window.title("JPEG compression")
        self.root_window.minsize(width = 500, height = 500)

        for i in range(3):
            self.root_window.rowconfigure(i, weight = 1)
            self.root_window.columnconfigure(i, weight = 1)

        self.main_view.place(column = 1, row = 1, sticky ="ew")


####BUTTON COMMAND FUNCTIONS AND NAVIGATION####
    def open_file(self):
        """Open a file for editing, and navigate to next page"""
        file_path = askopenfilename(filetypes=[("BMP Images", "*.bmp")])
        if not file_path:
            return
        self.image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
        self.navigate_to_input_view()

    def navigate_to_input_view(self):
        self.main_view.destroy();
        self.main_view = pages.ParameterInput(self.root_window,
                         self.submit, self.image.shape)
        self.layout()

    def submit(self):
        parameters = self.main_view.get_parameters()
        compressed_image = self.compress(parameters)

        self.show_comparison(compressed_image)


    def compress(self, parameters):
        block_size = parameters["block_size"]
        cutoff = parameters["frequence_cutoff"]
        row_blocks = self.image.shape[0] // block_size
        column_blocks = self.image.shape[1] // block_size

        compressed_image = np.zeros((row_blocks*block_size, column_blocks*block_size))
        #create index arrays
        x, y = np.mgrid[0:block_size, 0:block_size]
```

```
        #keep only the frequences where the indices sum to less than cutoff
        eliminated_frequencies = x + y >= cutoff

        for i in np.arange(start = 0, stop = row_blocks):
            row_start = i*block_size
            row_end = row_start + block_size
            for j in np.arange(start = 0, stop = column_blocks):
                column_start = j*block_size
                column_end = column_start + block_size
                #extract block
                f = self.image[row_start:row_end, column_start:column_end]
                c = fft.dctn(f, 2, norm='ortho')

                c[eliminated_frequencies] = 0
                ff = fft.idctn(c, 2, norm = 'ortho')
                rounded_ff = np.rint(ff).astype(np.int)

                #replace invalid values with valid ones
                rounded_ff.clip(min=0, max=255)

                compressed_image[row_start:row_end, column_start:column_end] = rounded_


        return compressed_image


    def show_comparison(self, compressed):
        self.main_view.destroy()
        self.main_view = pages.Comparison(self.root_window,
                          self.image, compressed)
        self.layout()
        cv2.imwrite("compressed.bmp", compressed)
    def run(self):
        self.root_window.mainloop()

if __name__ == "__main__":
    app = Application()
    app.run()
```

Listing 4.2: Various views of the application

```
import tkinter as tk
from abc import ABC, abstractmethod
from PIL.ImageTk import PhotoImage
from PIL import Image


class View(ABC):
    """General class for application pages"""
    container = None
    frm_self = None

    def __init__(self, container):
        self.container = container

    @abstractmethod
    def setup(self):
```

```
        pass

    @abstractmethod
    def layout(self):
        pass

    def place(self, **kwargs):
        self.frm_self.grid(kwargs)

    def destroy(self):
        self.frm_self.destroy()

class FileSelection(View):
    """Class for the image input view"""
    btn_open_file = None
    lbl_instruction = None
    open_file = None

    def __init__(self, container, open_file):
        super().__init__(container)
        self.open_file = open_file
        self.setup()
        self.layout()

    def setup(self):
        self.frm_self = tk.Frame(self.container)
        self.lbl_instruction = tk.Label(self.frm_self)
        self.btn_open_file = tk.Button(self.frm_self, command = self.open_file)

    def layout(self):
        self.frm_self.columnconfigure(0, minsize = 100, weight = 1)
        self.frm_self.rowconfigure(1, minsize = 60, weight = 1)

        self.lbl_instruction["text"] = """
        Please load a grayscale .bmp image (color images will be converted)"""
        self.btn_open_file["text"] = "Open..."
        self.lbl_instruction.grid(row = 0, column = 0, sticky = "ew")
        self.btn_open_file.grid(row = 1, column = 0, sticky = "ew")




class ParameterInput(View):

    lbl_block_dimension = None
    ent_block_dimension = None

    lbl_frequence_cutoff = None
    ent_frequence_cutoff = None

    image_dimensions = None

    btn_submit = None
    submit = None

    lbl_error = None
```

```
def __init__(self, container, submit, image_dimensions):
    super().__init__(container)
    self.submit = submit
    self.image_dimensions = image_dimensions
    self.setup()
    self.layout()

def setup(self):

    self.frm_self = tk.Frame(self.container)

    self.lbl_block_dimension = tk.Label(self.frm_self)
    self.ent_block_dimension = tk.Entry(self.frm_self)

    self.lbl_frequence_cutoff = tk.Label(self.frm_self)
    self.ent_frequence_cutoff = tk.Entry(self.frm_self)

    self.lbl_error = tk.Label(self.frm_self)

    self.btn_submit = tk.Button(self.frm_self, command = self.submit)

    self.ent_block_dimension.bind("<KeyRelease>", self.on_block_dimension_entry_char
    self.ent_frequence_cutoff.bind("<KeyRelease>", self.on_frequence_cutoff_entry_ch

def layout(self):
    self.frm_self.columnconfigure(0, minsize = 100, weight = 1)
    self.frm_self.columnconfigure(1, minsize = 100, weight = 1)
    self.frm_self.rowconfigure([0, 1, 2], minsize = 50, weight = 1)

    self.lbl_block_dimension["text"] = """Choose a block dimension greater
    than 0, lesser than """
    self.lbl_block_dimension["text"] = self.lbl_block_dimension["text"] + str(min(se

    self.ent_block_dimension.insert(0, "0")


    self.lbl_frequence_cutoff["text"] = """Choose a frequency cutoff point"""
    self.ent_frequence_cutoff.insert(0, "0")

    self.btn_submit["text"] = "Submit"

    #disable submit button until valid input is provided
    self.btn_submit.config(state = tk.DISABLED)

    #disable frequency cutoff input until valid block dimension is provided
    self.ent_frequence_cutoff.config(state = tk.DISABLED)

    self.lbl_block_dimension.grid(row = 0, column = 0, sticky = "ew")
    self.ent_block_dimension.grid(row = 0, column = 1, sticky = "ew")


    self.lbl_frequence_cutoff.grid(row = 1, column = 0, sticky = "ew")
    self.ent_frequence_cutoff.grid(row = 1, column = 1, sticky = "ew")

    self.btn_submit.grid(row = 2, column = 1, sticky = "e")
```

```
        self.lbl_error.grid(row=2, column=0, sticky = "w")
        self.lbl_error.grid_remove()


        self.lbl_error["fg"] = "red"


    def get_parameters(self):
        block_dimension_string = self.ent_block_dimension.get()
        frequence_cutoff_string = self.ent_frequence_cutoff.get()
        parameters = {}
        parameters["block_size"] = int(block_dimension_string)
        parameters["frequence_cutoff"] = int(frequence_cutoff_string)
        return parameters



    def on_block_dimension_entry_change(self, event):
        """Input validation"""
        block_dimension_string = self.ent_block_dimension.get()

        try:

            block_dimension = int(block_dimension_string)

            #Remove previous limits to frequence cutoff
            self.lbl_frequence_cutoff["text"] = """Choose a frequency cutoff point"""

            #Remove any eventual previous error messages
            self.lbl_error.grid_remove()
            self.ent_frequence_cutoff.config(state = tk.NORMAL)

            if block_dimension < 1 or block_dimension > min(self.image_dimensions):
                raise Exception()

            self.max_cutoff = 2*block_dimension - 2
            self.lbl_frequence_cutoff["text"] = self.lbl_frequence_cutoff["text"] + "\n
            self.check_frequence_cutoff_validity()
        except:
            #show error message, disable frequency cutoff and submit button
            self.lbl_error.grid()
            self.ent_frequence_cutoff.config(state = tk.DISABLED)
            self.btn_submit.config(state = tk.DISABLED)
            self.lbl_error["text"] = "Please enter a valid block dimension"
            self.lbl_frequence_cutoff["text"] = "Choose a frequency cutoff point"

    def on_frequence_cutoff_entry_change(self, event):
        """Input validation"""
        self.check_frequence_cutoff_validity()


    def check_frequence_cutoff_validity(self):
        """Input validation"""
        frequence_cutoff_string = self.ent_frequence_cutoff.get()
        try:
            if len(frequence_cutoff_string) == 0:
```

```
                raise Exception()

            frequence_cutoff = int(frequence_cutoff_string)

            if frequence_cutoff < 0 or frequence_cutoff > self.max_cutoff:
                raise Exception()

            #Remove any residual messages
            self.lbl_error.grid_remove()
            self.btn_submit.config(state = tk.NORMAL)

        except:
            #show error message, disable frequency cutoff and submit button
            self.lbl_error.grid()
            self.lbl_error["text"] = "Please enter a valid frequency cutoff"
            self.btn_submit.config(state = tk.DISABLED)

class Comparison(View):
    """Class for viewing side by side comparison of original and compressed"""
    resized_original = None
    resized_compressed = None

    original = None
    compressed = None
    lbl_original = None
    lbl_compressed = None
    def __init__(self, container, original, compressed):
        super().__init__(container)

        self.original = Image.fromarray(original)
        self.compressed = Image.fromarray(compressed)
        self.setup()
        self.layout()

    def setup(self):
        self.frm_self = tk.Frame(self.container)
        self.lbl_original = tk.Label(self.frm_self)
        self.lbl_compressed = tk.Label(self.frm_self)

    def layout(self):
        width, height = self.original.size


        #figure out initial scale of images
        scale = 500/max((height, width))


        #resize image to fit scaled down aspect
        resized_compressed = self.compressed.resize(size =(int(width*scale), int(height*
        resized_original = self.original.resize(size =(int(width*scale), int(height*scal


        #keep reference to resized picture in order to be able to display it
        self.resized_compressed = PhotoImage(image = resized_compressed)
        self.resized_original = PhotoImage(image = resized_original)
```

```
        #set minsize so it fits images scaled down to 500px along max dimension
        self.frm_self.columnconfigure([0,1], minsize = (width*scale), weight = 1)
        self.frm_self.rowconfigure(0, minsize = (height*scale), weight = 1)

        self.lbl_original["image"] = self.resized_original

        self.lbl_original.grid(row = 0, column = 0, sticky = "nsew")
        self.lbl_original.config(width = (width*scale), height = (height*scale))

        self.lbl_compressed["image"] = self.resized_compressed
        self.lbl_compressed.grid(row = 0, column = 1, sticky = "ew")
        self.lbl_compressed.config(width = (width*scale), height = (height*scale))
```

## 4.3 Risultati su immagini

Si sono prese in esame tre immagini:

- **deer.bmp**: 1011px x 661px
- **bridge.bmp**: 2749px x 4049px
- **cathedral.bmp**: 2000px x 30008px

Si sono scelti valori di F e d secondo questa strategia: finestra piccola, ma soglia molto bassa (alta compressione); finestra grande, e soglia molto bassa; finestra grande, soglia uguale al lato della finestra; finestra grande, soglia alta.
In particolare, per tutte e tre le immagini si sono utilizzate le seguenti combinazioni:

- F = 10, d = 4. Compressione molto alta con finestre piccole
- F = 50, d = 5. Compressione molto alta con finestre più grandi
- F = 50, d = 50

Vista la differenza di dimensione tra le immagini, per le ultime due si sono inoltre fatte le seguenti prove:

- F = 200, d = 100
- F = 200, d = 200
- F = 200, d = 300

### 4.3.1 F = 10, d = 4

Per quanto riguarda l'immagine del cervo, gli artefatti dovuti alla compressione sono subito visibili quando si osserva l'immagine intera. Tuttavia, nelle altre due - dove il rapporto tra lato della finestra e dimensioni dell'immagine è molto più basso -, a prima vista non si vedono grossi disturbi.
Ingrandendo l'immagine, tuttavia, in tutti e tre i casi si notano i singoli blocchetti che compongono l'immagine, particolarmente in casi di parti dell'immagine che presentano molta variazione. Invece, in zone dove il valore dei pixel è più o meno costante (come sul

Figure 4.1: Deer con F = 10, d = 4

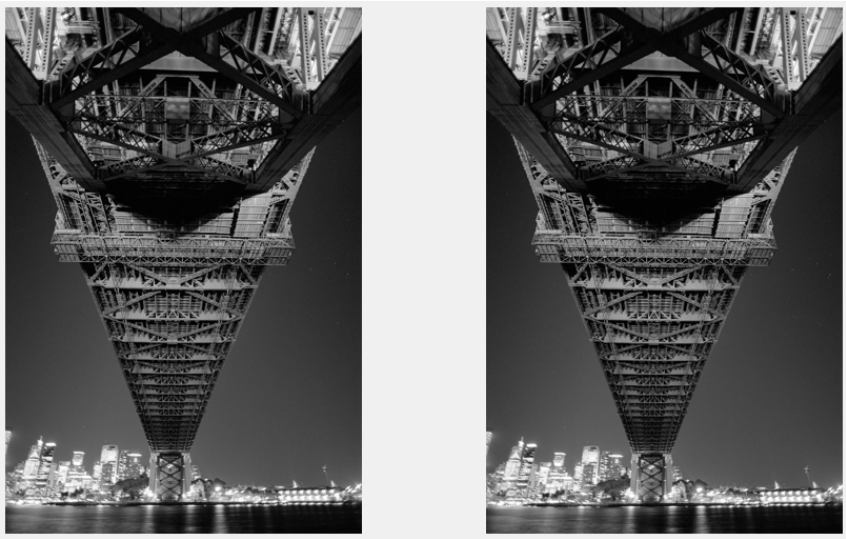

Figure 4.2: Bridge con F = 10, d = 4

Figure 4.3: Cathedral con F = 10, d = 4

muso del cervo, come si vede in figura) questo delineamento delle finestre è molto meno marcato.
Un tale risultato è in linea con le aspettative, vista la soglia molto bassa per le frequenze.

Figure 4.4: Dettaglio dell'immagine "deer" compressa

Figure 4.5: Dettaglio dell'immagine "bridge" compressa

Figure 4.6: Dettaglio dell'immagine "cathedral" compressa

### 4.3.2 F = 50, d = 5

Il fenomeno evidenziato con i precedenti parametri è accentuato ulteriormente aumentando le dimensioni della finestra. In questo caso, infatti, la suddivisione in blocchi altamente compressi risulta immediatamente visibile all'occhio nudo in tutte e tre le immagini.
Per l'immagine "deer", dato il rapporto tra le dimensioni della finestra e quelle dell'immagine, questo è marcato al punto da dare all'immagine compressa un aspetto cubista.
Per le rimanenti due, invece, non si capisce in modo così immediato la causa degli artefatti, ma è apparente che la qualità dell'immagine abbia subito un abbassamento.

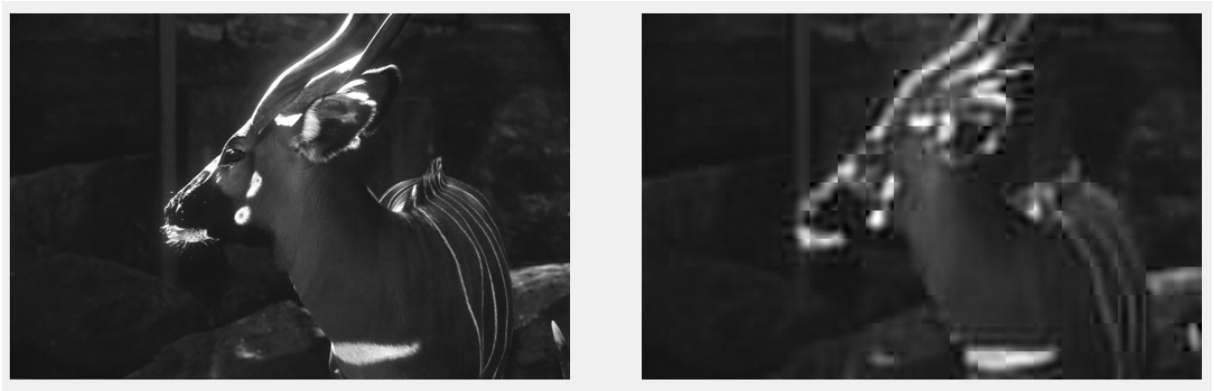

Figure 4.7: Deer con F = 50, d = 5

Figure 4.8: Bridge con F = 50, d = 5

Figure 4.9: Cathedral con F = 50, d = 5

### 4.3.3 F = 50, d = 50

In questo caso, nelle due immagini più grandi la compressione non causa una perdita significativa della qualità. Anche ingrandendo il più possibile le immagini, non si riescono a distinguere blocchi o blur evidenti.
Invece, per quanto riguarda l'immagine del cervo, nella zona del muso (dove c'è un passaggio repentino da chiaro a scuro) si nota il fenomeno di Gibbs - risultato aspettato, vista la discontinuità che un tale salto di valori produce.

Figure 4.10: Deer con F = 50, d = 50

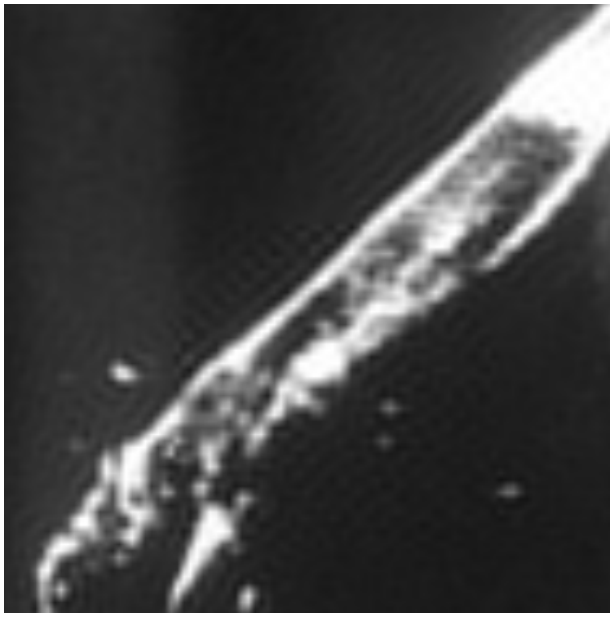

Figure 4.11: Deer con F = 50, d = 50

Figure 4.12: Bridge con F = 50, d = 50

Figure 4.13: Cathedral con F = 50, d = 50

### 4.3.4 F = 200, d = 100, 200, 300

Anche in questo caso, per le immagini più grandi a colpo d'occhio non si nota nessuna perdita di qualità dell'immagine.
Ingrandendo il più possibile, nel caso della soglia posta a 100, si notano piccoli artefatti. In particolare, nell'immagine del ponte si possono osservare i bordi delle finestre e un tasso di sgranatura più alto rispetto all'originale: Questo tipo di artefatti non si riescono

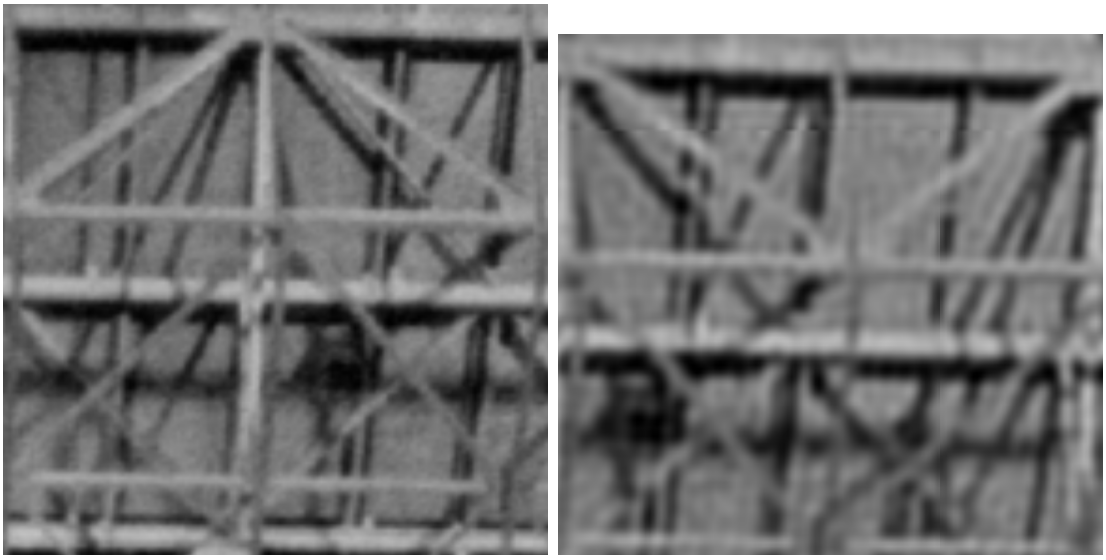

Figure 4.14: Bridge con F = 200, d = 100 vs originale

a trovare nella foto "Cathedral".
A maggior ragione, con soglie maggiori, la qualità dell'immagine compressa aumenta ulteriormente. Già con d posto a 200, infatti, gli artefatti spariscono anche dall'immagine del ponte.