

## Progetto 2: Compressione di immagini tramite la DCT

**Corso:** Metodi del Calcolo Scientifico

**Relazione di:**  
Alessandro Capelli 816302  
Damiano Dovico 816682  
Davide Bucci 816067  
Emilio Brambilla 816538

**Anno Accademico 2019-2020**

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Prima parte</b>	<b>1</b>
2.1	DCT2	1
2.2	Implementazione DCT2	2
2.3	Implementazione FFT	5
2.4	Tempi di esecuzione	6
<b>3</b>	<b>Seconda parte</b>	<b>7</b>
3.1	Descrizione del SW	7
3.2	Implementazione	8
3.3	Analisi dei risultati	13
<b>4</b>	<b>Struttura della directory</b>	<b>20</b>

# 1 Introduzione

Questo progetto riguarda lo sviluppo e utilizzo della DCT2 in ambiente open source. Come ambiente di sviluppo è stato scelto *Python*, per consentire di sfruttare la velocità, la dinamicità sintattica e semantica del linguaggio (interpretato) e la sua caratteristica di essere cross-platform. Il progetto è suddiviso in due fasi:

- inizialmente è stata sviluppata una libreria "homemade" (chiamata "**DCT2 homemade**") che implementa una versione non ottimizzata di DCT2 (e  $\bar{DCT}$ ) normalizzata, che è stata confrontata con una libreria ottimizzata (chiamata "**scipy.fft**") che implementa la versione fast (FFT) della DCT2 (anch'essa normalizzata).
- successivamente è stato sviluppato un software (chiamato "**imageCompressor.py**") che implementa un semplice algoritmo di compressione delle immagini in toni di grigio, in particolare, per immagini in formato **BMP**.

## 2 Prima parte

### 2.1 DCT2

Gli algoritmi implementati fanno largo uso del concetto di DCT (in particolare nella sua forma in 2 dimensioni - DCT2); si evidenzia che la tipologia presa in esame è quella **normalizzata**, ovvero quella in cui le basi  $\{e_{ij}\}$  e  $\{\tilde{w}_{kl}\}$  sono ortonormali.

Formalmente la **DCT2** (Discrete Cosine Transform) è definita come:

$$c_{kl} = \alpha_{kl} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} f_{ij} \cos(k \pi \frac{2i+1}{2N}) \cos(l \pi \frac{2j+1}{2M}) \quad (1)$$

Mentre la sua inversa, la **IDCT2** (Inverse Discrete Cosine Transform), è definita formalmente come:

$$f_{ij} = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} c_{kl} \alpha_{kl} \cos(k \pi \frac{2i+1}{2N}) \cos(l \pi \frac{2j+1}{2M}) \quad (2)$$

$$\text{Con } \begin{cases} \alpha_{00} = \frac{1}{\sqrt{NM}} \\ \alpha_{k0} = \alpha_{0l} = \sqrt{\frac{2}{NM}} \\ \alpha_{kl} = \frac{2}{\sqrt{NM}}, \quad k, l \geq 1 \end{cases}$$

Tali funzioni permettono la compressione (e decompressione, in quanto invertibili) spaziale del segnale, tramite una trasformazione dell'input da spazio vettoriale delle basi canoniche a quello in frequenza; se affiancate a tecniche adeguate, consente di restringere il dominio dei valori (e.g., tagliando le alte frequenze) ottenendo formati di compressione lossy (e.g., JPEG).

## 2.2 Implementazione DCT2

```
1 import numpy as np
2 import math
3
4 def dct(f):
5     """
6         Return the Discrete Cosine Transform of arbitrary array f.
7
8     Parameters
9     -----
10    f : array of integers
11        The input array.
12
13    Returns
14    -----
15    c : array of floats
16        The transformed input array.
17    """
18
19    if(len(f.shape) != 1):
20        raise Exception('Dimension is different from 1')
21
22    N = f.size
23    c = np.zeros(N)
24    alpha = np.zeros(N)
25
26    alpha[0] = math.sqrt(1 / N)
27    alpha[1:] = math.sqrt(2 / N)
28
29    for k in range(N):
30        for i in range(N):
31            c[k] += f[i] * math.cos(k * math.pi * ((2 * i + 1) / (2 * N)))
32        c[k] = alpha[k] * c[k]
33
34    return c
35
36 def idct(c):
37     """
38         Return the Inverse Discrete Cosine Transform of arbitrary array c.
39
40     Parameters
41     -----
42     c : array of floats
43        The input array.
44
45     Returns
46     -----
47     f : array of integers
48        The transformed input array.
49    """
50
51    if(len(c.shape) != 1):
52        raise Exception('Dimension is different from 1')
53
54    N = c.size
55    f = np.zeros(N)
```

```

56     alpha = np.zeros(N)
57
58     alpha[0] = math.sqrt(1 / N)
59     alpha[1:] = math.sqrt(2 / N)
60
61     for j in range(N):
62         for k in range(N):
63             f[j] += c[k] * alpha[k] * math.cos(k * math.pi * ((2 * j +
64             1) / (2 * N)))
65
66     return f
67
68 def dct2(f):
69     """
70     Return the Discrete Cosine Transform in 2D of arbitrary matrix f.
71
72     Parameters
73     -----
74     f : matrix of integers
75         The input matrix.
76
77     Returns
78     -----
79     c : matrix of floats
80         The transformed input matrix.
81
82     """
83
84     if(len(f.shape) != 2):
85         raise Exception('Dimension is different from 2')
86
87     N = f.shape[0]
88     M = f.shape[1]
89     c = np.zeros((N, M))
90
91     # DCT on columns
92     for j in range(M):
93         c[:, j] = dct(np.squeeze(np.asarray(f))[:, j])
94
95     # DCT on rows
96     for i in range(N):
97         c[i, :] = dct(np.squeeze(np.asarray(c))[i, :])
98
99     return c
100
101 def idct2(c):
102     """
103     Return the Inverse Discrete Cosine Transform in 2D of arbitrary
104     matrix c.
105
106     Parameters
107     -----
108     c : matrix of floats
109         The input matrix.
110
111     Returns
112     -----
113     f : matrix of integers
114         The transformed input matrix.

```

```

112 """
113
114     if(len(c.shape) != 2):
115         raise Exception('Dimension is different from 2')
116
117     N = c.shape[0]
118     M = c.shape[1]
119     f = np.zeros((N, M))
120
121     # IDCT2 on columns
122     for j in range(M):
123         f[:, j] = idct(np.squeeze(np.asarray(c))[:, j])
124
125     # IDCT2 on rows
126     for i in range(N):
127         f[i, :] = idct(np.squeeze(np.asarray(f))[i, :])
128
129     return f

```

Listing 1: DCT2\_hOMEMADE.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import datetime
4 import scipy.fft as DCT2_library
5 import DCT2_hOMEMADE
6
7 def main():
8     # DCT1 on the test array
9     # f = np.array([231, 32, 233, 161, 24, 71, 140, 245])
10    # c = DCT2_hOMEMADE.dct(f)
11    # print(c)
12    # f = DCT2_hOMEMADE.idct(c)
13    # print(f)
14
15    # DCT2 on the test matrix
16    # f = np.matrix('231 32 233 161 24 71 140 245; 247 40 248 245 124
17    # 204 36 107; 234 202 245 167 9 217 239 173; 193 190 100 167 43 180 8
18    # 70; 11 24 210 177 81 243 8 112; 97 195 203 47 125 114 165 181; 193
19    # 70 174 167 41 30 127 245; 87 149 57 192 65 129 178 228')
20    # c = DCT2_hOMEMADE.dct2(f)
21    # print(c)
22    # f = DCT2_hOMEMADE.idct2(c)
23    # print(f)
24    # c = DCT2_library.dctn(f, 2, norm='ortho')
25    # print(c)
26    # f = DCT2_library.idctn(c, 2, norm='ortho')
27    # print(f)
28
29
30    max_value = 255
31    times_DCT2_hOMEMADE = []
32    times_DCT2_library = []
33
34    n_min = 50
35    n_max = 500
36    steps = 50
37    for n in range(n_min, (n_max+1), steps):
38        f = np.matrix(np.random.randint(max_value, size=(n, n))).astype
39        (float)

```

```

35
36     begin_time = datetime.datetime.now()
37     DCT2_hOMEMADE.dct2(f)
38     time = datetime.datetime.now() - begin_time
39     times_DCT2_hOMEMADE.append(time.seconds*1000 + time.
40     microseconds/1000)
41     print("Execution time of DCT2_hOMEMADE (n={}): {}".format(n,
42     time))
43
44     begin_time = datetime.datetime.now()
45     DCT2_LIBRARY.dctn(f, 2, norm='ortho')
46     time = datetime.datetime.now() - begin_time
47     times_DCT2_LIBRARY.append(time.seconds*1000 + time.microseconds
48     /1000)
49     print("Execution time of DCT2_LIBRARY (n={}): {}".format(n,
50     time))
51
52     palette = plt.get_cmap('Set1')
53     plt.xlabel('N')
54     plt.ylabel('Time (milliseconds in log scale)')
55     plt.yscale('log')
56     plt.grid(True, alpha=0.2)
57     plt.plot(range(n_min, (n_max+1), steps), times_DCT2_hOMEMADE, color
58     =palette(1), linewidth=2)
59     plt.plot(range(n_min, (n_max+1), steps), times_DCT2_LIBRARY, color=
      palette(4), linewidth=2)
60     plt.legend(['DCT2_hOMEMADE', 'DCT2_LIBRARY'], loc=2, ncol=2)
61     plt.show()
62
63 if __name__ == "__main__":
64     main()

```

Listing 2: DCT2\_resourceUsage.py

### 2.3 Implementazione FFT

La libreria `scipy.fft` implementa la versione FFT (Fast Fourier Transform) per calcolare in modo efficiente la DFT (Discrete Fourier Transform), sfruttando le simmetrie nei termini calcolati. "La simmetria è massima quando  $n$  è una potenza di 2 e la trasformazione è quindi più efficiente per queste dimensioni. Per dimensioni poco fattorizzabili, `scipy.fft` utilizza l'algoritmo di *Bluestein* e quindi non è mai superiore a  $O(n \log n)$  [riferito al caso monodimensionale]."<sup>1</sup>. Formalmente, nel caso monodimensionale, la FFT di una sequenza  $x[n]$  di lunghezza  $N$  è:

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n] \quad (3)$$

Con la trasformata inversa definita come segue:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi j \frac{kn}{N}} y[k] \quad (4)$$

---

<sup>1</sup>Fourier Transforms (`scipy.fft`). <https://docs.scipy.org/doc/scipy/reference/tutorial/fft.html>

## 2.4 Tempi di esecuzione

Gli algoritmi sono stati valutati sulla seguente macchina:

- **OS:** MacOS 10.15.4
- **CPU:** 2.9 GHz Quad-Core Intel Core i7
- **RAM:** 16 GB 2133 MHz LPDDR3
- **Memoria secondaria:** SSD

Il confronto è avvenuto in termini di tempo di esecuzione (misurato in millisecondi e posto in un grafico in scala semi-logaritmica) al variare di  $N$  (ovvero avendo in input matrici di dimensione  $N \times N$ ); in particolare, la figura 1 rappresenta il confronto tra le librerie "DCT2\_hOMEMADE" e "scipy.fft" con  $N$  che varia da 50 a 500, con step uguali a 50 (le matrici vengono generate in modo randomico, mantenendole invariate ad ogni confronto così da poter ridurre al minimo differenze date da agenti esterni alle librerie). Si tenga conto che le complessità computazionali dei due algoritmi sono nell'ordine delle seguenti grandezze:

- DCT2\_hOMEMADE:  $O(N^3)$
- scipy.fft:  $O(N^2 \log(N))$

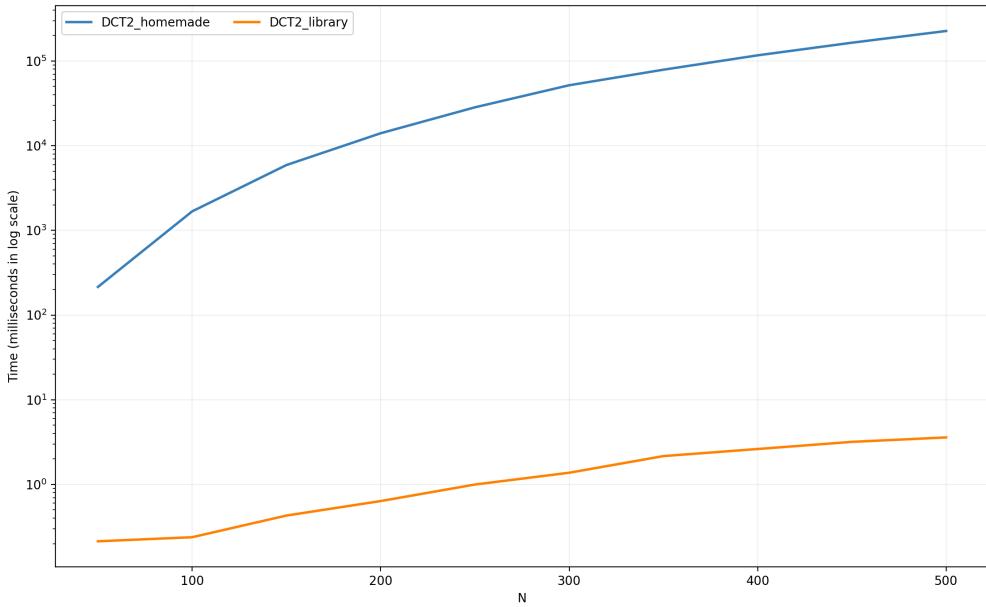


Figura 1: Comparazione delle tempistiche

Per porre maggiore enfasi sull'alta efficienza della libreria che implementa la versione FFT, è stato deciso di effettuare un test ad hoc, generando matrici che raggiungono la dimensione di  $N = 10000$  (quindi aventi  $10^8$  elementi da elaborare).

I risultati ottenuti sono rappresentati nella figura 2 (sempre utilizzando il grafico in scala semi-logaritmica).

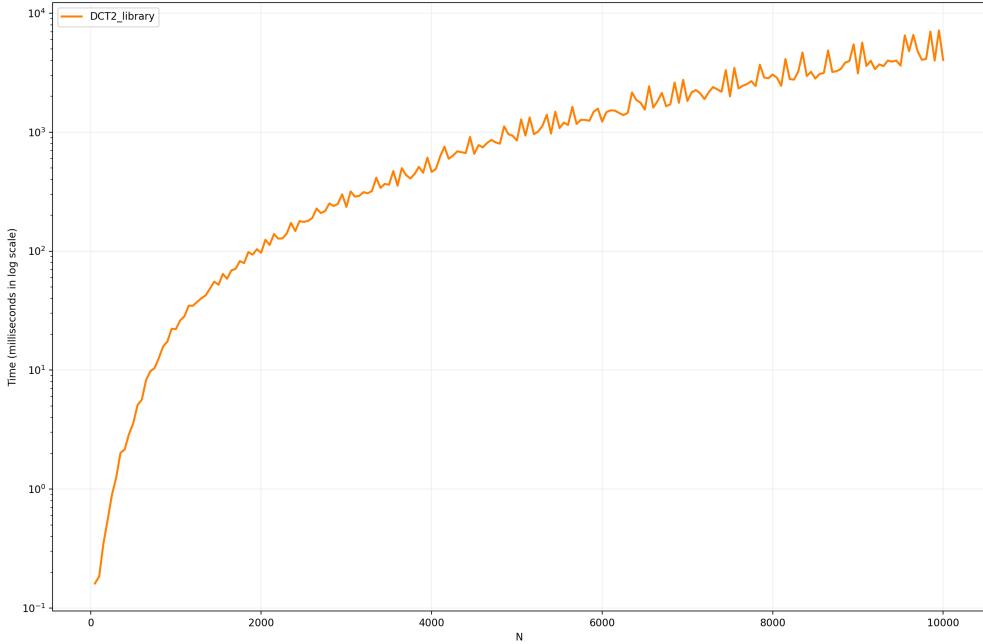


Figura 2: Tempistiche della libreria FFT

### 3 Seconda parte

#### 3.1 Descrizione del SW

Il software sviluppato (chiamato **imageCompressor.py**) ha lo scopo di implementare un algoritmo di compressione (di tipo *lossy*) di immagini in toni di grigio tramite l'utilizzo della DCT2 (sfruttando l'implementazione efficiente della libreria `scipy.fft`), in concomitanza con tecniche di eliminazione delle alte frequenze.

Il software è dotato di una semplice interfaccia grafica (figura 3) implementata utilizzando la libreria *Tkinter*: cross-platform, "essenziale", versatile e parte integrante di Python stesso. Per rendere più gradevole alla vista la GUI e per migliorare l'UX ("user experience") è stato scelto di rendere l'interfaccia grafica "responsive".

Le funzionalità rese disponibili dal software sono le seguenti:

- scelta dell'immagine in input tramite selezione diretta dal filesystem dell'utente. I file in input devono necessariamente essere delle immagini in formato bitmap (.bmp) in toni di grigio
- scelta della directory di output (tramite navigazione del filesystem) del risultato ottenuto, ovvero dell'immagine compressa secondo i parametri scelti dall'utente

- scelta del parametro **F**: rappresenta la dimensione dei macro blocchi su cui verrà effettuata la DCT2. Deve essere un numero intero con valore maggiore di 0
- scelta del parametro **d**: rappresenta la soglia di taglio delle frequenze. Deve essere un numero intero con valore compreso tra 0 e  $(2F - 2)$
- Risultato finale: rappresentazione a schermo dell'immagine originale affiancata all'immagine compressa ottenuta in output

In conformità alle specifiche, è possibile che durante la fase di suddivisione dell'immagine in macro blocchi di dimensione  $F \times F$  (effettuata partendo dal primo pixel in alto a sinistra dell'immagine in input), vengano scartati dei pixel che rappresentano gli avanzi dettati dalla suddivisione stessa.

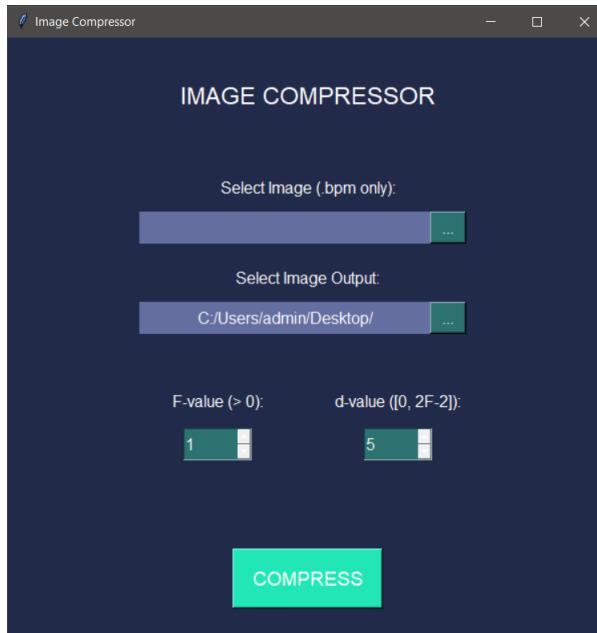


Figura 3: Schermata principale dell'interfaccia grafica

### 3.2 Implementazione

```

1 import numpy as np
2 from numpy import r_
3 import PIL.Image
4 import PIL.ImageTk
5 import scipy.fft as DCT2_library
6 import tkinter
7 from tkinter import filedialog
8 from tkinter.messagebox import showinfo
9 import os
10
11 # Global variables
12 img_directory = ""
13 output_directory = ""
14 output_name = "result"
15 HEIGHT, WIDTH = 600, 600
16 colors = {'blue': '#212a49', 'white': 'white', 'violet': '#646e9f', 'green': '#21e7b6', 'green2': '#2c7270'}
```

```

17 font = 'Sans-Serif'
18
19 def select_image():
20     global img_directory
21
22     img_path = filedialog.askopenfilename( title='Please select a
23 picture', initialdir=os.getcwd(), filetypes=[('Image File', ['.bmp',
24 ])])
25         if (len(img_path) > 0) and os.path.isfile(img_path):
26             img_directory = img_path
27             lbl_img_directory.config(text=img_directory)
28
29
30 def select_output_directory():
31     global output_directory
32
33     output_directory_path = filedialog.askdirectory()
34     if (len(output_directory_path) > 0) and os.path.isdir(
35 output_directory_path):
36         output_directory = output_directory_path + "/"
37         lbl_output_directory.config(text=output_directory)
38
39
40 def display_output(img_original, img_compressed):
41     x = window.winfo_x() / (1.5)
42     y = window.winfo_y()
43     WIDTH = window.winfo_width()
44     if(WIDTH < 900):
45         WIDTH = window.winfo_width() * 2
46     HEIGHT = window.winfo_height()
47     img_w, img_h = 420, 420
48
49     window_display_output = tkinter.Toplevel(window)
50     window_display_output.title('Original vs. Compressed')
51     window_display_output.geometry("%dx%d+%d+%d" % (WIDTH, HEIGHT, x, y))
52     window_display_output.minsize(WIDTH, HEIGHT)
53
54     # GUI: Left frame
55     left_frame = tkinter.Frame(window_display_output, bg=colors['blue'])
56     left_frame.place(relx=0.25, rely=0, relwidth=0.5, relheight=1,
57     anchor='n')
58
59     lbl_OriginalImage = tkinter.Label(left_frame, text='Original image',
60     font=(font, 18), fg=colors['white'], bg=colors['blue'])
61     lbl_OriginalImage.place(relx=0.5, rely=0, relwidth=0.6, relheight
62     =0.2, anchor='n')
63
64     # GUI: Left frame image
65     image_frame_left = tkinter.Frame(left_frame, bg=colors['violet'])
66     image_frame_left.place(relx=0.5, rely=0.20, width=img_w+10, height=
67     img_h+10, anchor='n')
68
69     resized_image = img_original.resize((img_w, img_h), PIL.Image.
70     ANTIALIAS)
71     pi_img_original = PIL.ImageTk.PhotoImage(image=resized_image)
72
73     lbl_img_original = tkinter.Label(image_frame_left, image=
74     pi_img_original)

```

```

64     lbl_img_original.place(relwidth=1, relheight=1)
65
66     # GUI: Right frame
67     right_frame = tkinter.Frame(window_display_output, bg=colors[‘
68     green2’])
69     right_frame.place(relx=0.75, rely=0, relwidth=0.5, relheight=1,
70     anchor='n')
71
72     lbl_OriginalImage = tkinter.Label(right_frame, text='Compressed
73     image', font=(font, 18), fg= colors[‘white’], bg=colors[‘green2’])
74     lbl_OriginalImage.place(relx=0.5, rely=0, relwidth=0.6, relheight
75     =0.2, anchor='n')
76
77     # GUI: Right frame image
78     image_frame_right = tkinter.Frame(right_frame, bg=colors[‘green2’])
79     image_frame_right.place(relx=0.5, rely=0.20, width=img_w+10, height
80     =img_h+10, anchor='n')
81
82     resized_image_right = img_compressed.resize((img_w, img_h), PIL.
83     Image.ANTIALIAS)
84     pi_img_compressed = PIL.ImageTk.PhotoImage(image=
85     resized_image_right)
86
87     lbl_img_compressed = tkinter.Label(image_frame_right, image=
88     pi_img_compressed)
89     lbl_img_compressed.place(relwidth=1, relheight=1)
90
91     window_display_output.mainloop()
92
93
94
95 def save_image(img, output_directory, output_name):
96     path = output_directory + output_name + ‘.bmp’
97     img.save(path)
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912

```

```

113 def image_to_matrix(img):
114     img_grey = img.convert('L') # 'L': 8-bit pixels, black and white
115     img_matrix = np.array(img_grey)
116
117     return img_matrix
118
119 def dct2(sub_img):
120     return DCT2_library.dctn(sub_img, 2, norm='ortho')
121
122 def idct2(sub_img):
123     return DCT2_library.idctn(sub_img, 2, norm='ortho')
124
125 def compress(img_matrix, F, d):
126     img_sizes = img_matrix.shape
127     outcome_matrix = np.zeros(img_sizes)
128
129     # split, dct2, frequencies elimination, idct2, reconstruct
130     for row in r_[:img_sizes[0]:F]:
131         for col in r_[:img_sizes[1]:F]:
132             if(((row+F) > img_sizes[0]) or (col+F > img_sizes[1])):
133                 continue
134
135             sub_img_dct2 = dct2(img_matrix[row:(row+F), col:(col+F)])
136
137             for i in range(0, F):
138                 for j in range(0, F):
139                     if((i+j) >= d):
140                         sub_img_dct2[i][j] = 0
141
142             outcome_matrix[row:(row+F), col:(col+F)] = idct2(
143             sub_img_dct2)
144
145     # round, normalize
146     outcome_matrix = np.around(outcome_matrix)
147     outcome_matrix = outcome_matrix.clip(0, 255)
148
149     return outcome_matrix
150
151 def main():
152     global img_directory
153     global output_directory
154
155     if(check_parameters(img_directory, output_directory) == False):
156         showinfo("Error", "Wrong parameters!")
157     else:
158         F = int(value_F.get())
159         d = int(value_d.get())
160
161         img = PIL.Image.open(img_directory)
162         img_matrix = image_to_matrix(img)
163         img_outcome = compress(img_matrix, F, d)
164         img_outcome = PIL.Image.fromarray(img_outcome.astype('uint8'))
165         save_image(img_outcome, output_directory, output_name)
166         display_output(img, img_outcome)
167
168 # GUI
169 window = tkinter.Tk()
170 window.title("Image Compressor")

```

```

170 window.minsize(WIDTH, HEIGHT)
171 background = tkinter.Label(window, bg=colors['blue'])
172 background.place(x=0, y=0, relwidth=1, relheight=1)
173 window.geometry("%d%d" % (window.winfo_screenwidth() / 2 - WIDTH / 2,
174                             window.winfo_screenheight() / 2 - HEIGHT / 2))
175 # GUI: Title
176 title_frame = tkinter.Frame(window)
177 title_frame.place(relx=0.5, rely=0.05, relwidth=0.5, relheight=0.1,
178                   anchor='n')
178 lbl_title = tkinter.Label(title_frame, text="IMAGE COMPRESSOR", font=(
179                         font, 18), fg=colors['white'], bg=colors['blue'])
180 lbl_title.place(relwidth=1, relheight=1)
181 # GUI: Body
182 font_size = 12
183 body_frame = tkinter.Frame(window, bg=colors['blue'])
184 body_frame.place(relx=0.5, rely=0.20, relwidth=0.75, relheight=0.6,
185                   anchor='n')
186 # GUI: Input
187 lbl_select_image_input = tkinter.Label(body_frame, text="Select Image
188 (.bpm only):", font=(font, font_size), fg=colors['white'], bg=colors
189 ['blue'])
190 lbl_select_image_input.place(relx=0.5, rely=0.05, anchor='n')
191 lbl_img_directory = tkinter.Label(body_frame, text="", font=(font,
192 font_size), fg=colors['white'], bg=colors['violet'])
193 lbl_img_directory.place(relx=0.45, rely=0.15, relwidth=0.65, relheight
194 =0.09, anchor='n')
195 btn_select_image_input = tkinter.Button(body_frame, text="...", font=(
196 font, font_size), fg=colors['white'], bg=colors['green2'],
197 highlightbackground=colors['green2'], highlightthickness= 500,
198 command=select_image)
199 btn_select_image_input.place(relx=0.81, rely=0.15, relwidth=0.08,
200 relheight=0.09, anchor='n')
201 # GUI: Output
202 y_offset = 0.25
203 lbl_select_image_output = tkinter.Label(body_frame, text="Select Image
204 Output:", font=(font, font_size), fg=colors['white'], bg=colors[
205 blue])
206 lbl_select_image_output.place(relx=0.5, rely=0.05+y_offset, anchor='n')
207 lbl_output_directory = tkinter.Label(body_frame, text="", font=(font,
208 font_size), fg=colors['white'], bg=colors['violet'])
209 lbl_output_directory.place(relx=0.45, rely=0.15+y_offset, relwidth
210 =0.65, relheight=0.09, anchor='n')
211 btn_select_image_output = tkinter.Button(body_frame, text="...", font=(
212 font, font_size), fg=colors['white'], bg=colors['green2'],
213 highlightbackground=colors['green2'], highlightthickness= 500,
214 command=select_output_directory)
215 btn_select_image_output.place(relx=0.81, rely=0.15+y_offset, relwidth
216 =0.08, relheight=0.09, anchor='n')
217 # GUI: F/d value

```

```

208 y_offset = y_offset * 2 - 0.05
209 lbl_F = tkinter.Label(body_frame, text="F-value (> 0):", font=(font,
    font_size), fg=colors['white'], bg=colors['blue'])
210 lbl_F.place(relx=0.30, rely=0.18+y_offset, relwidth=0.30, relheight
    =0.09, anchor='n')
211 value_F = tkinter.Spinbox(body_frame, from_=1, to=100, width=10, font=(
    font, font_size), fg=colors['white'], bg= colors['green2'])
212 value_F.place(relx=0.30, rely=0.30+y_offset, relwidth=0.15, relheight
    =0.09, anchor='n')
213
214 lbl_d = tkinter.Label(body_frame, text="d-value ([0, 2F-2]):", font=(
    font, font_size), fg=colors['white'], bg=colors['blue'])
215 lbl_d.place(relx=0.70, rely=0.18+y_offset, relwidth=0.30, relheight
    =0.09, anchor='n')
216 value_d = tkinter.Spinbox(body_frame, from_=0, to=100, width=10, font=(
    font, font_size), fg=colors['white'], bg= colors['green2'])
217 value_d.place(relx=0.70, rely=0.30+y_offset, relwidth=0.15, relheight
    =0.09, anchor='n')
218
219 # GUI: Compress
220 lower_frame = tkinter.Frame(window)
221 lower_frame.place(relx=0.5, rely=0.85, relwidth=0.25, relheight=0.1,
    anchor='n')
222 btn_compress = tkinter.Button(lower_frame, text="COMPRESS", fg=colors['
    white'], bg=colors['green'], font=("Sans-Serif", font_size+2),
    highlightbackground=colors['green'], highlightthickness=500, command
    =main)
223 btn_compress.place(relwidth=1, relheight=1)
224
225 if __name__ == "__main__":
226     window.mainloop()

```

Listing 3: imageCompressor.py

### 3.3 Analisi dei risultati

L’analisi dei risultati consiste nello studio delle immagini prodotte in output dal software a fronte di differenti immagini in input e differenti parametri scelti. In particolare, è stato deciso di confrontare immagini diverse per risaltare il funzionamento della compressione, minimizzando gli effetti causati dalla scelta di una specifica immagine in input (su cui l’algoritmo potrebbe agire in modo “anomalo”). Sono stati effettuati ulteriori test sui parametri  $F$  e  $d$ , per evidenziare come la scelta degli stessi sia da intendere come un *trade-off* tra qualità desiderata e costo in termini di risorse (spazio occupato e tempo computazionale richiesto per la compressione). Si ricorda che la compressione utilizzata rientra tra quelle di tipo *lossy* e quindi la perdita di informazione è intrinseca all’algoritmo stesso.

Il primo confronto, mostrato in figura 4, rappresenta a sinistra l’immagine originale di un ponte e a destra la stessa immagine compressa con  $F = 8$  e  $d = 1$ ; questo indica che la dimensione scelta per i macro-blocchi è di 8 e che le frequenze mantenute sono quelle inferiori a 1 (assumendo che partano da 0). Si noti che la dimensione dei macro-blocchi è tale da dividere l’immagine in sezioni sufficientemente piccole da rendere minimo l’effetto di divisione (quello che si potrebbe definire effetto “Minecraft”), anche a fronte di un consistente taglio di frequenze che ha introdotto sgranatura e perdita di definizione nell’immagine compressa. La scelta di  $d = 1$  è per definizione la compressione massi-

ma (considerando  $F$  costante) e cioè quella che elimina il maggior numero di frequenze, escludendo la scelta di  $d = 0$ , che porterebbe ad avere un'immagine completamente nera.



Figura 4: Sinistra: originale - Destra:  $F = 8$ ,  $d = 1$

Successivamente è stato effettuato un esperimento (rappresentato in figura 5) che mantiene il valore di  $F = 8$  ma che aumenta le frequenze non eliminate, applicando il valore di  $d = 4$ . Come si può osservare, l'immagine ottenuta è estremamente fedele all'originale pur eliminando molte informazioni. Per osservare questa perdita di informazione è necessario analizzare più nel dettaglio i particolari, come mostrato in figura 6.

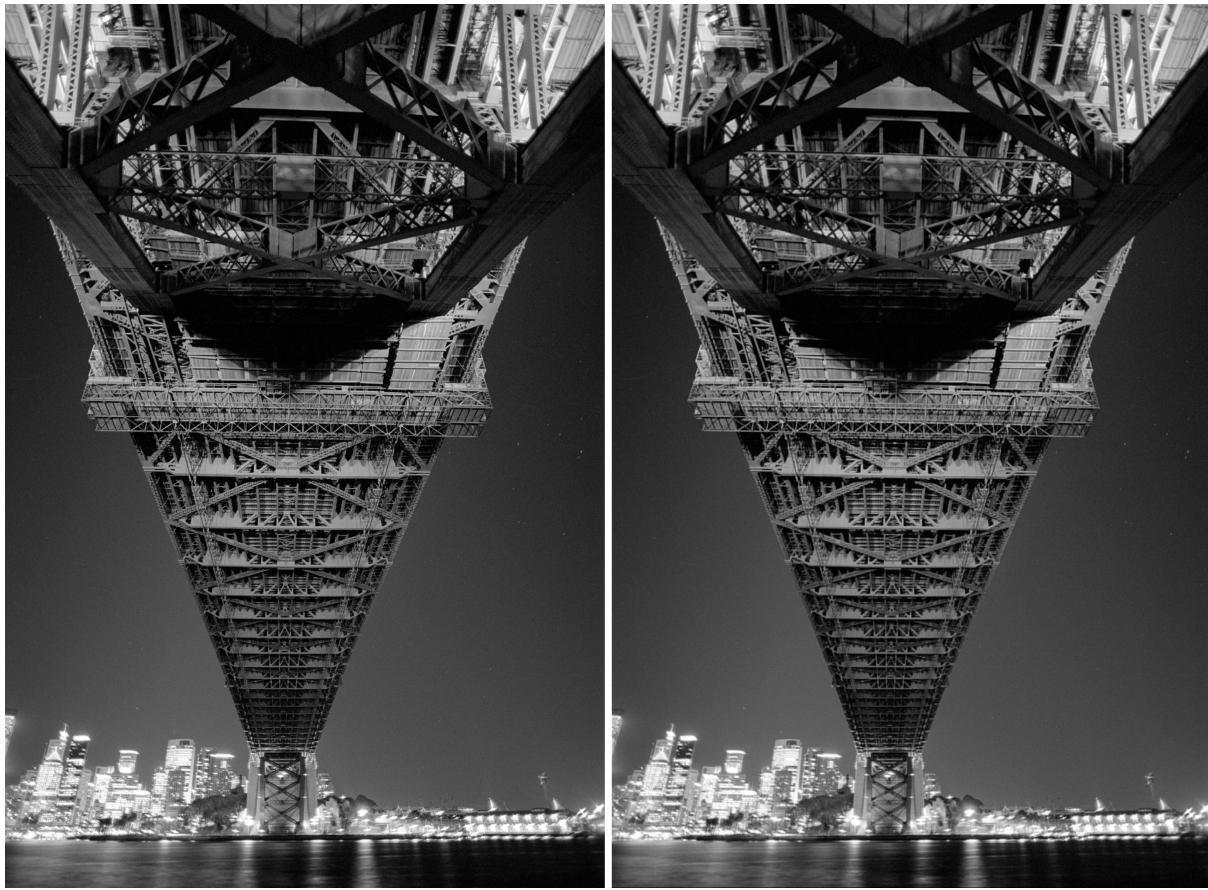


Figura 5: Sinistra: originale - Destra:  $F = 8$ ,  $d = 4$

In molti contesti tale perdita di informazioni è accettabile ed è molto complicato riuscire a percepire considerevoli differenze. Per poter cogliere notevoli sgranature ed osservare l'effetto di divisione in macro-blocchi, è necessario effettuare un ulteriore ingrandimento (rappresentato in figura 7).

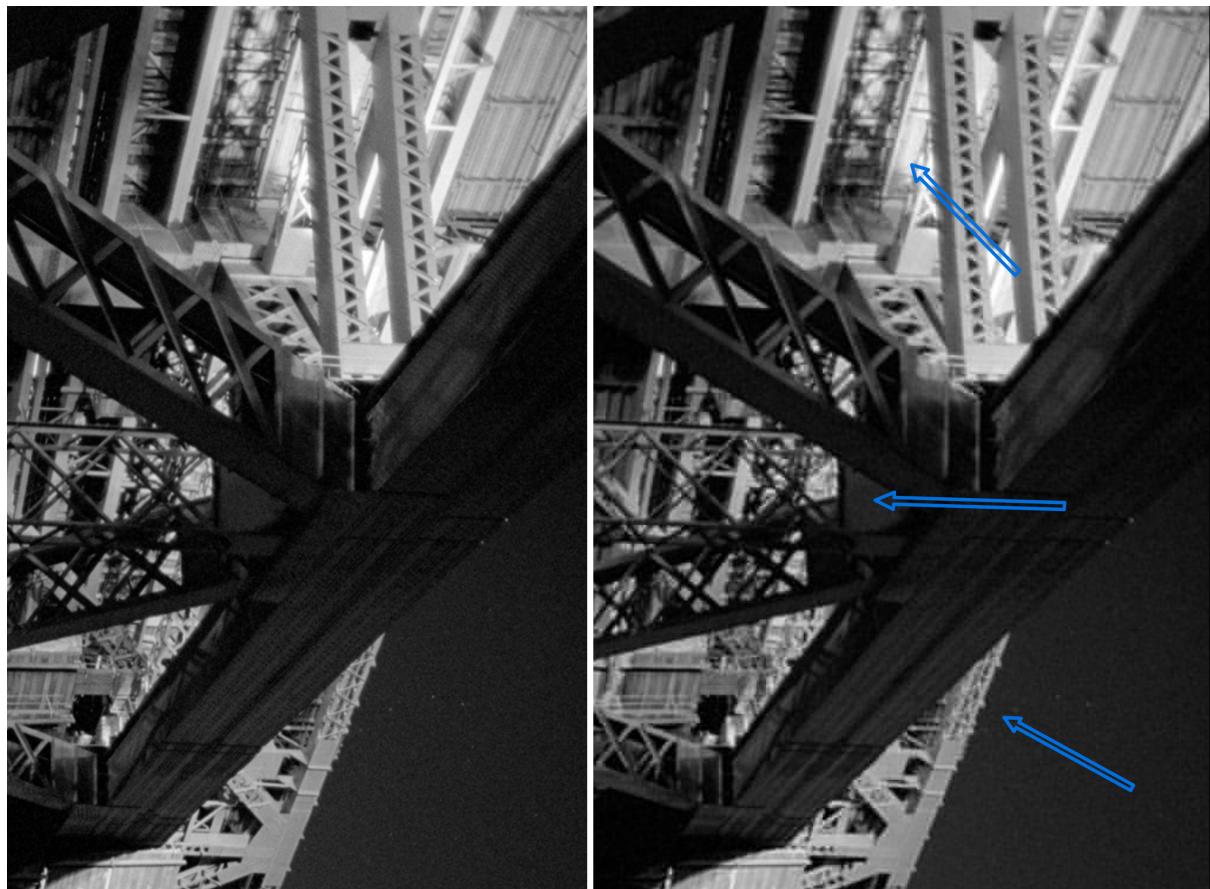


Figura 6: Sinistra: originale (ingrandita) - Destra:  $F = 8$ ,  $d = 4$  (ingrandita)



Figura 7: Sinistra: originale (ingrandita ulteriormente) - Destra:  $F = 8$ ,  $d = 4$  (ingrandita ulteriormente)

In figura 8 sono rappresentati una serie di esperimenti effettuati sull'immagine del ponte, in particolare vengono rappresentate (da sinistra a destra):

- l'immagine originale
- l'immagine compressa con  $F = 8$  e  $d = 8$
- l'immagine compressa con  $F = 8$  e con  $d = 14$  (quindi con medesima dimensione dei macro-blocchi ma più definita rispetto a quella precedente)
- l'immagine compressa con  $F = 50$  e con  $d = 5$  (vengono messi in risalto i dettagli poco definiti delle luci dei palazzi in lontananza)

Dell'ultima sequenza di immagini, viene presa in considerazione l'ultima (figura 9). Si evidenzia in particolare una sezione ingrandita che risalta la dimensione dei macro-blocchi, evidentemente troppo grande per essere trascurata ad occhio nudo. Dall'applicazione dell'algoritmo sull'immagine è evidente il funzionamento dei tagli di frequenze nei blocchi (percepito come una sequenza di output caotici e poco conformi tra loro).



Figura 8: Da sinistra a destra: (originale  $F = 8$ ,  $d = 8$ ), ( $F = 8$ ,  $d = 14$ ), ( $F = 50$ ,  $d = 5$ )

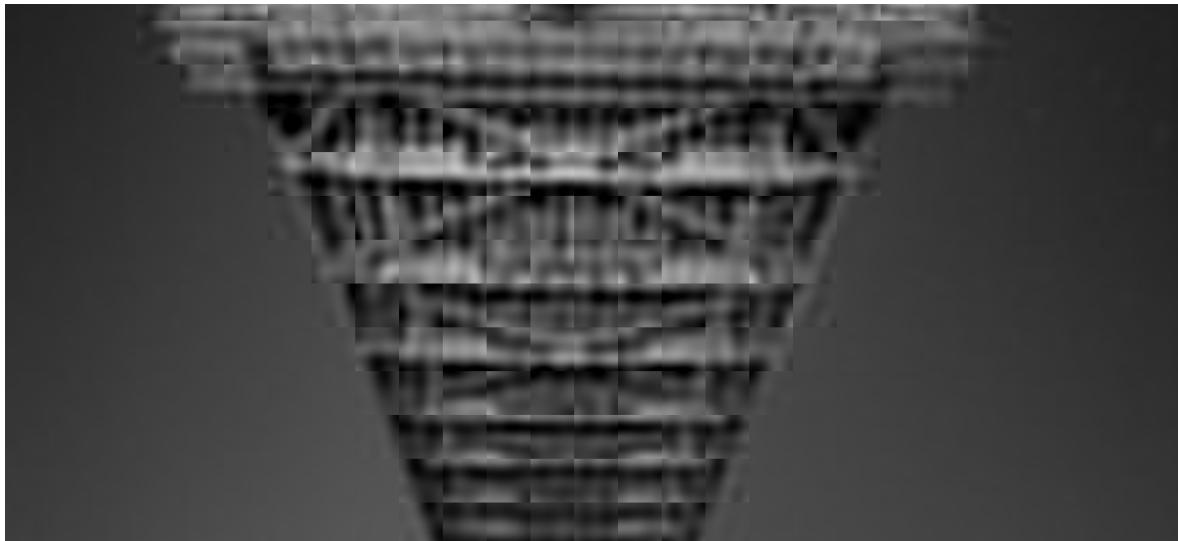


Figura 9: Ingrandimento dell'immagine compressa con  $F = 50$ ,  $d = 5$

Gli ultimi confronti, rappresentati nelle figure 10 e 11, contengono una serie di immagini in cui viene posta enfasi su quella centrale (con  $F = 8$  e  $d = 1$ ) che è quella in cui i particolari degradano maggiormente. Si tenga presente che la DCT2 è poco efficace quando le immagini contengono zone con molto contrasto, in quanto emerge il fenomeno di Gibbs. L' utilizzo di macro-blocchi riduce questo effetto e ne limita la propagazione e utilizzando la matrice di quantizzazione si avrebbe un ulteriore miglioramento.



Figura 10: Da sinistra a destra: originale -  $F = 8$ ,  $d = 1$  -  $F = 4$ ,  $d = 4$



Figura 11: Da sinistra a destra: originale -  $F = 8$ ,  $d = 1$  -  $F = 4$ ,  $d = 4$

## 4 Struttura della directory

- /Progetto 2/: root
  - /Progetto 2/**DCT2\_hOMEMADE.py**: libreria "homemade" che implementa DCT2 e DCT
  - /Progetto 2/**DCT2\_resourceUsage.py**: programma per il confronto computazionale delle librerie
  - /Progetto 2/**imageCompressor.py**: software per la compressione di immagini bitmap
  - /Progetto 2/Pics/: cartella delle immagini di test
    - \* /Progetto 2/Pics/bridge.bmp:  $1375 \times 2025$  pixel (width  $\times$  height), 8.4 MB
    - \* /Progetto 2/Pics/cathedral.bmp:  $1000 \times 1504$  pixel (width  $\times$  height), 4.5 MB
    - \* /Progetto 2/Pics/deer.bmp:  $2022 \times 1321$  pixel (width  $\times$  height), 8 MB
    - \* /Progetto 2/Pics/fireworks.bmp:  $3136 \times 2352$  pixel (width  $\times$  height), 7.4 MB