Università degli Studi di Milano– Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Anno Accademico 2021 – 2022

Metodi del Calcolo Scientifico

Raffaele Cerizza 845512
Giacomo Savazzi 845372
Andrea Assirelli 820149

22 Luglio 2022

Secondo Progetto – Compressione JPEG

Compressione di Immagini tramite la DCT

Linguaggio e Librerie Utilizzate

- ***SciPy.fft*** *– implementazioni DCT, IDCT, DCT2 e IDCT2 tramite Fast Fourier Transform;*
- ***Matplotlib*** *– creazione di grafici;*
- ***TkInter*** *– implementazione dell'interfaccia grafica;*
- ***OpenCV, Pillow, Imageio*** *– caricamento, gestione e salvataggio di immagini.*

Prima Parte – DCT e DCT2 Semplice (1/5)

```python
def my_dct1(f):
    '''
    Implementazione della DCT1.
    :param f: vettore dei valori puntuali
    :return c: vettore delle frequenze
    '''
    # Imposta i valori di f in formato float in 64 bit
    f = f.astype('float64')

    # Calcola il numero di elementi di f
    n = len(f)

    # Calcola la matrice di trasformazione
    D = compute_D(n)

    # Moltiplica la matrice di trasformazione e f
    c = np.dot(D, f)

    return c
```

DCT $\vec{c} = D\vec{f}$

DCT2

```python
def my_dct2(f):
    '''
    Implementazione della DCT2.

    :param f: matrice dei valori puntuali
    :return c: matrice delle frequenze
    '''

    # Calcola il numero di righe (n) e colonne (m)
    n, m = np.shape(f)

    # Copia f in c utilizzando valori float in 64 bit
    c = np.copy(f.astype('float64'))

    # Esegue la DCT1 sulle colonne
    for j in range(m):
        c[:,j] = my_dct1(c[:,j])

    # Esegue la DCT1 sulle righe
    for i in range(n):
        c[i,:] = my_dct1(c[i,:])

    return c
```

Prima Parte – Calcolo Matrice D (2/5)

```python
def compute_D(n):
    '''
    Funzione per calcolare la matrice di trasformazione D.

    :param n: numero di elementi del vettore f per il calcolo della DCT1
    :return D: matrice di trasformazione
    '''

    # Calcola i valori alfa utilizzati per la matrice di trasformazione
    alpha_vect = np.zeros(n)
    alpha_vect[0] = 1.0 / sqrt(n)
    alpha_vect[1:n] = sqrt(2.0 / n)

    # Calcola la matrice di trasformazione
    D = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            D[i,j] = alpha_vect[i] * cos(pi * (2 * j + 1) * i / (2 * n))
    return D
```

$$D_{l,j} = \alpha_l^N \cos\left(l\pi \frac{2j+1}{2N}\right),$$

$$\alpha_l^N = \begin{cases} \frac{1}{\sqrt{N}} \; se \; l = 0 \\ \sqrt{\frac{2}{N}} \; altrimenti \end{cases}$$

Prima Parte – DCT e DCT2 di SciPy.fft (3/5)

```python
def scipy_dct1(f):
    '''
    Implementazione della DCT1 tramite scipy.fft.
    '''
    return dct(f.T, norm='ortho')
```

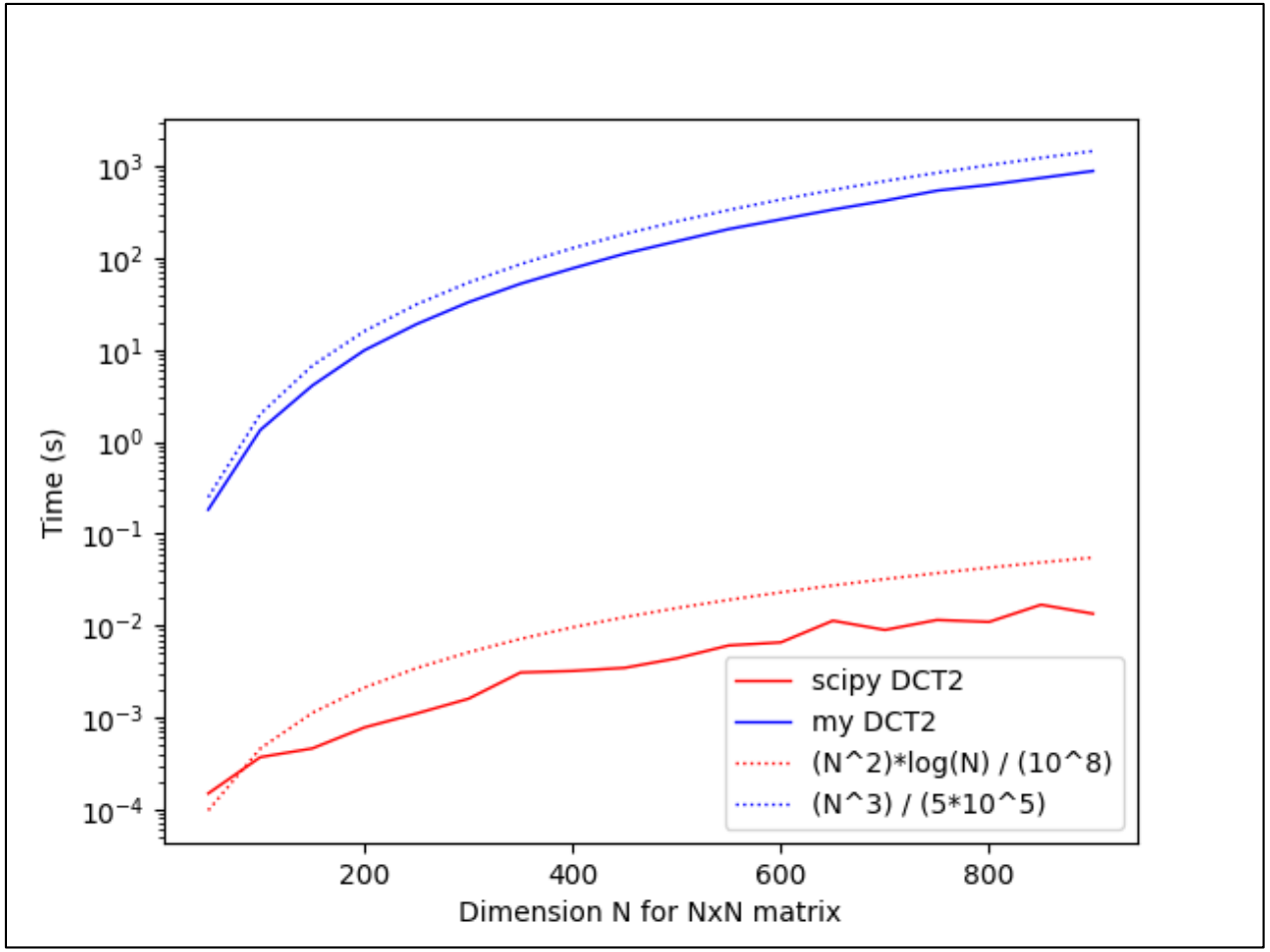
```python
def scipy_dct2(f):
    '''
    Implementazione della DCT2 tramite scipy.fft.
    '''
    return dct(dct(f.T, norm='ortho').T, norm='ortho')
```

Prima Parte – Verifica della Correttezza (4/5)

```python
def test():
    '''
    Funzione per verificare se le funzioni implementate per
    calcolare DCT1 e DCT2 rispettano i valori specificati
    nella consegna.
    '''

    # Array con i valori utilizzati per il test
    test_dct1 = np.array([231,  32, 233, 161,  24,  71, 140, 245])
    test_dct2 = 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]])

    # Stampa dei risultati. Il controllo della correttezza va svolto
    # manualmente
    print("Verifica correttezza di scipy_dct1: ")
    result_scipy_test_dct1 = scipy_dct1(test_dct1)
    print(result_scipy_test_dct1)
    print()
    print("Verifica correttezza di my_dct1: ")
    result_my_test_dct1 = my_dct1(test_dct1)
    print(result_my_test_dct1)
    print()
    print("Verifica correttezza di scipy_dct2: ")
    result_scipy_test_dct2 = scipy_dct2(test_dct2)
    print(result_scipy_test_dct2)
    print()
    print("Verifica correttezza di my_dct2: ")
    result_my_test_dct2 = my_dct2(test_dct2)
    print(result_my_test_dct2)
    print()
```

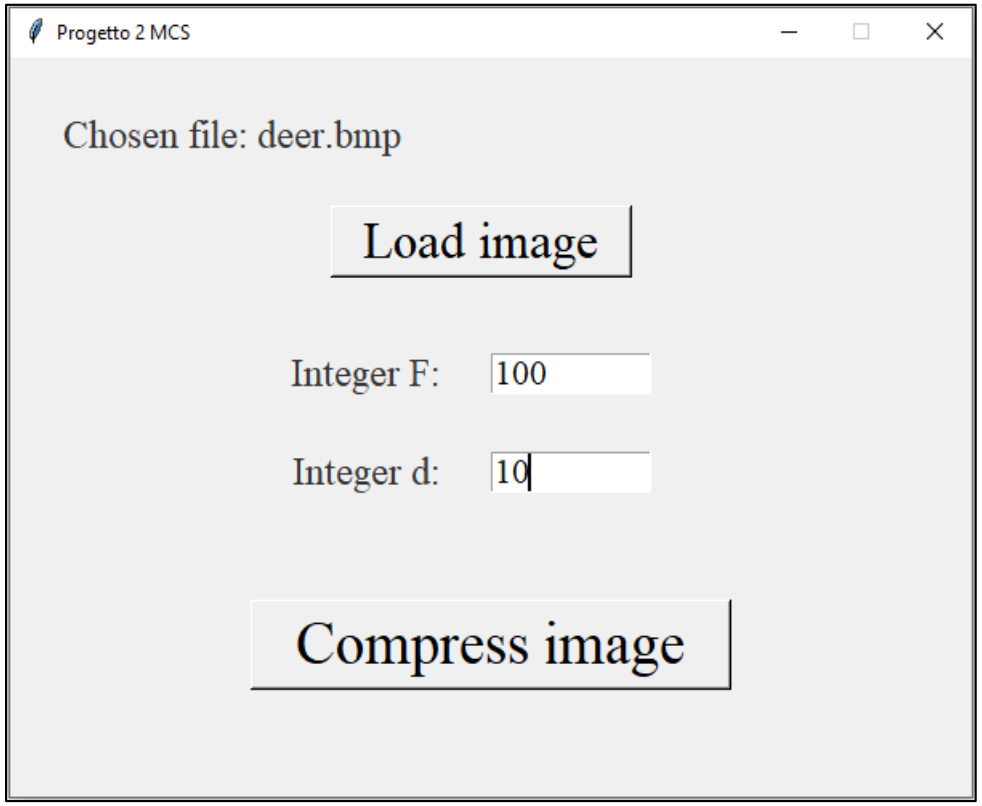
Funzione di verifica della correttezza di DCT e DCT2 nella versione semplice e di libreria

Prima Parte – Confronto dei Risultati (5/5)



Confronto dei tempi di esecuzione della DCT2 semplice e di libreria (basata su FFT)

Seconda Parte – Interfaccia Grafica (1/6)



- ***F:*** *ampiezza macro blocchi;*
- ***d:*** *soglia di taglio delle frequenze, compresa tra 0 e 2F-2.*

Seconda Parte – Funzione Principale (2/6)

```
# Legge l'immagine originale dal path memorizzato
image = cv2.imread(self.chosen_image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Converte l'immagine in scala di grigi
image_pil = Image.fromarray(image)
image_greyscale = image_pil.convert('L')
image_matrix = np.array(image_greyscale)

# Recupera altezza e larghezza dell'immagine
height = image_matrix.shape[0]
width = image_matrix.shape[1]

# Se l'altezza non è multiplo di f, si tronca al maggior valore multiplo di f
if (height % self.f_value != 0):
    height = height - (height % self.f_value)

# Se la larghezza non è multiplo di f, si tronca al maggior valore multiplo di f
if (width % self.f_value != 0):
    width = width - (width % self.f_value)

# Comprime l'immagine in base ad altezza e larghezza troncate
image_matrix_truncated = image_matrix[0:height , 0:width]
height = image_matrix_truncated.shape[0]
width = image_matrix_truncated.shape[1]

# Crea blocchi F x F
blocks = self.create_blocks(height, width, image_matrix_truncated)

# Trasforma i blocchi con DCT2 e IDCT2
compressed_blocks = self.compress_blocks(blocks)

# Ricostruisce la matrice con i valori calcolati
image_matrix_compressed = self.rebuild_image_matrix(height, width, image_matrix_truncated, compressed_blocks)

# Trasforma la matrice in un'immagine
image_compressed = Image.fromarray(image_matrix_compressed.astype('uint8'))
image_compressed = image_compressed.convert('L')

# Salva l'immagine compressa
index = self.chosen_image_path.rfind("/") + 1
path = self.chosen_image_path[:index]
self.compressed_image_path = path + "compressed_image.bmp"
imageio.imsave(self.compressed_image_path, image_compressed)

# Mostra a schermo l'immagine originale e quella compressa
self.show_images()
```

caricamento dell'immagine

troncamento dell'immagine

creazione dei macro-blocchi

applicazione DCT2 e IDCT2

ricostruzione dell'immagine

visualizzazione dell'imagine originale e di quella ricostruita

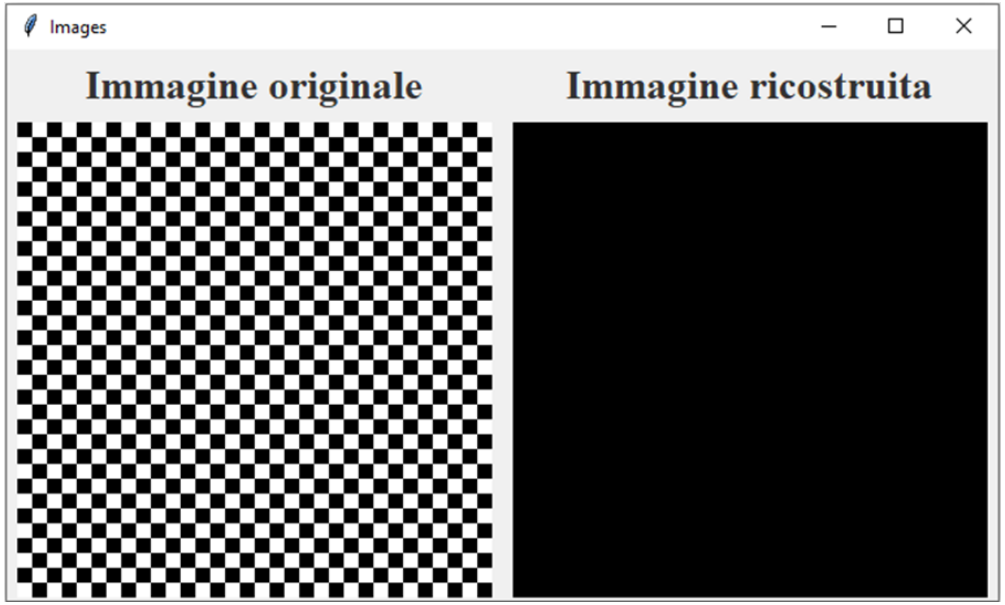
Seconda Parte – Funzioni di Supporto (3/6)

```python
def create_blocks(self, height, width, image_matrix_truncated):
    '''
    Funzione per la creazione di blocchi F x F.

    :param self: variabili e funzioni della classe App
    :param height: altezza dell'immagine originale
    :param width: larghezza dell'immagine originale
    :param image_matrix_truncated: matrice dell'immagine troncata
    :return blocks: blocchi F x F
    '''

    # Crea blocchi F x F
    blocks = []
    for i in range(0, height, self.f_value):
        for j in range(0, width, self.f_value):

            # Prende i valori di un blocco F x F
            values = image_matrix_truncated[i:i+self.f_value,
                                            j:j+self.f_value]

            # Crea un blocco F x F con i precedenti valori
            block = np.array(values).reshape(self.f_value,
                                             self.f_value)

            # Aggiunge il blocco alla lista dei blocchi
            blocks.append(block)

    return blocks
```

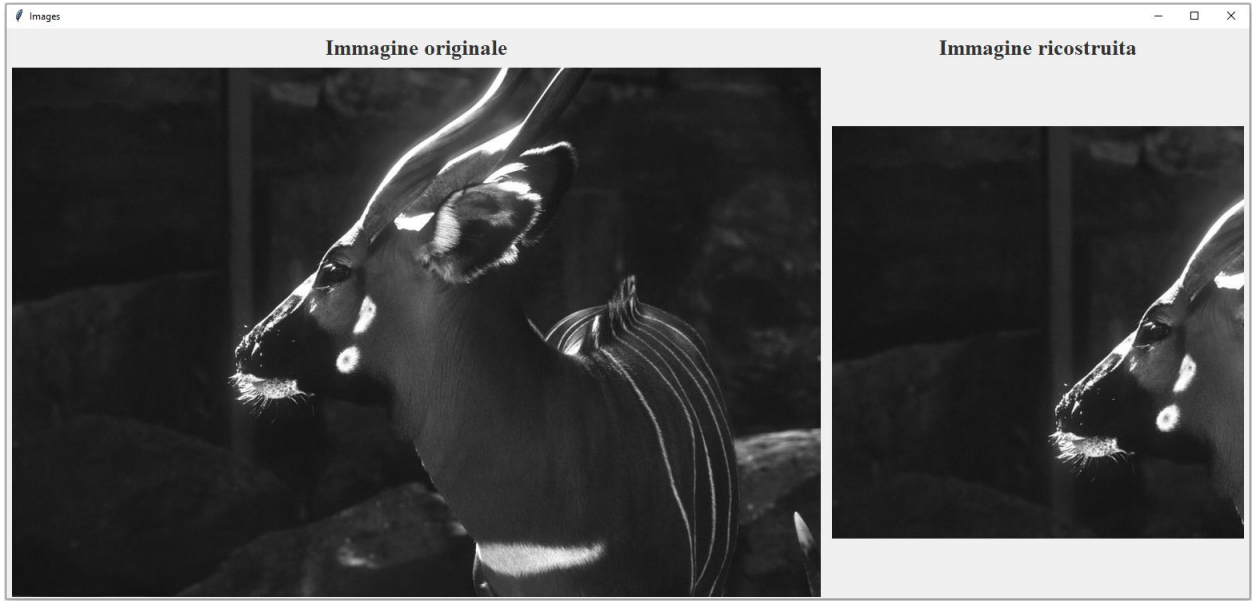
```python
# Trasforma i blocchi con DCT2 e IDCT2
compressed_blocks = []
for k in range(len(blocks)):

        # Recupera un singolo blocco dalla lista dei blocchi
        block = blocks[k]

        # Esegue la DCT2 sul singolo blocco
        c = dct(dct(block.T, norm='ortho').T, norm='ortho')

        # Elimina le frequenze per i + j >= d
        for i in range(0, self.f_value):
            for j in range(0, self.f_value):
                if (i + j >= self.d_value):
                    c[i][j] = 0

        # Esegue la IDCT2 sul singolo blocco
        f = idct(idct(c.T, norm='ortho').T, norm='ortho')

        # Arrotonda i valori di f all'intero più vicino
        f = np.rint(f)

        # Mette a 0 i numeri negativi e a 255 quelli maggiori
        di 255
        for i in range(0, self.f_value):
            for j in range(0, self.f_value):
                if (f[i][j] < 0):
                    f[i][j] = 0
                if (f[i][j] > 255):
                    f[i][j] = 255

        # Aggiunge il blocco compresso alla lista dei blocchi
        compressi
        compressed_blocks.append(f)

return compressed_blocks
```

Seconda Parte – Esperimenti (4/6)

Casi particolari di utilizzo dei valori d e F



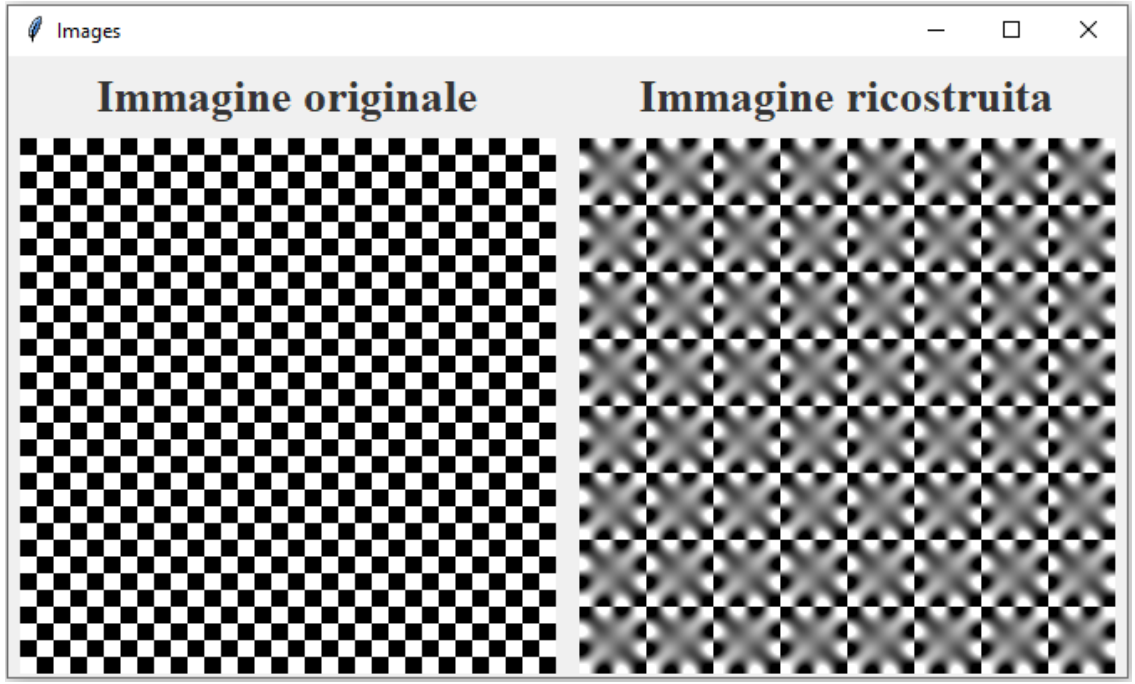
- ***Size:*** *320 X 320;*
- ***F:*** *320;*
- ***d:*** *0.*



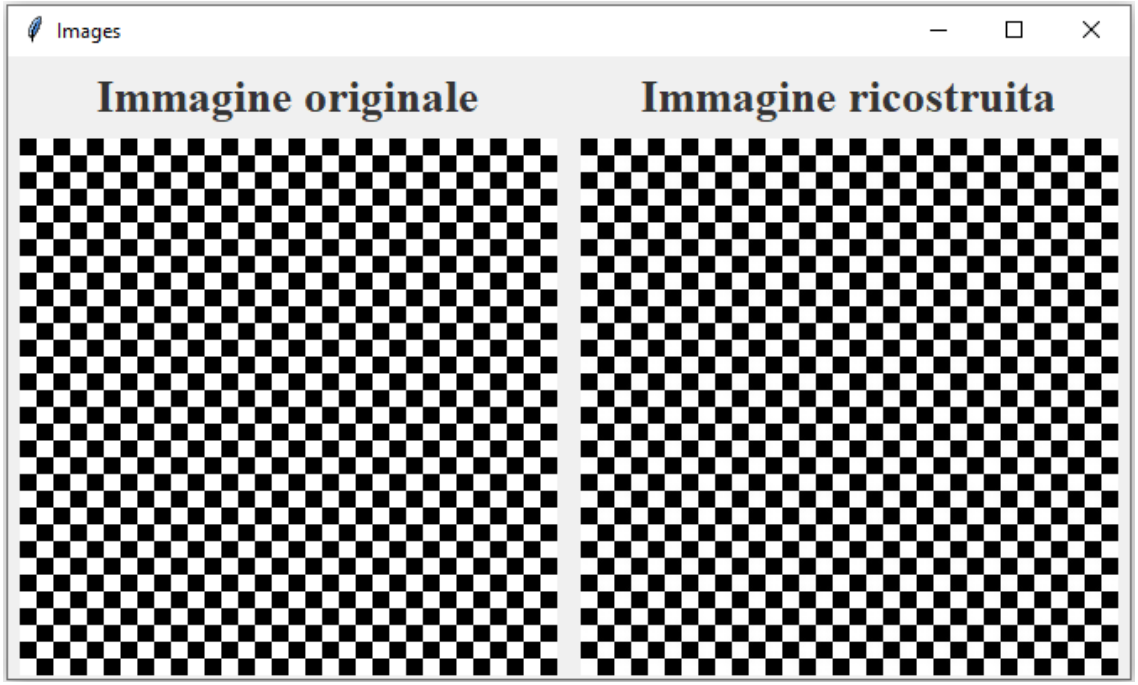
- ***Size:*** *1011 X 661;*
- ***F:*** *515;*
- ***d:*** *515.*

Seconda Parte – Esperimenti (5/6)

Immagini con Contrasti Forti



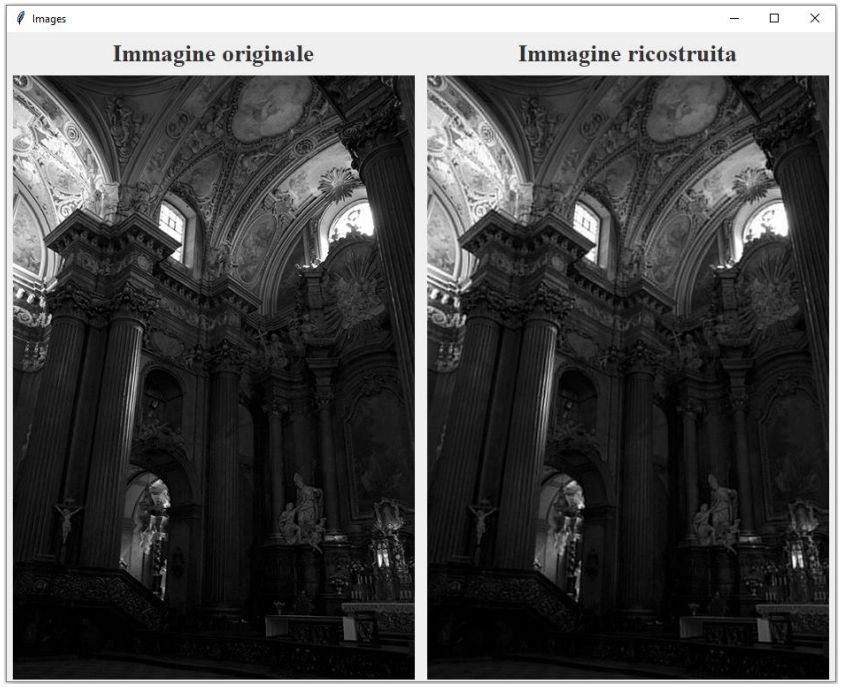
- ***Size:*** *320 X 320;*
- ***F:*** *40;*
- ***d:*** *7.*



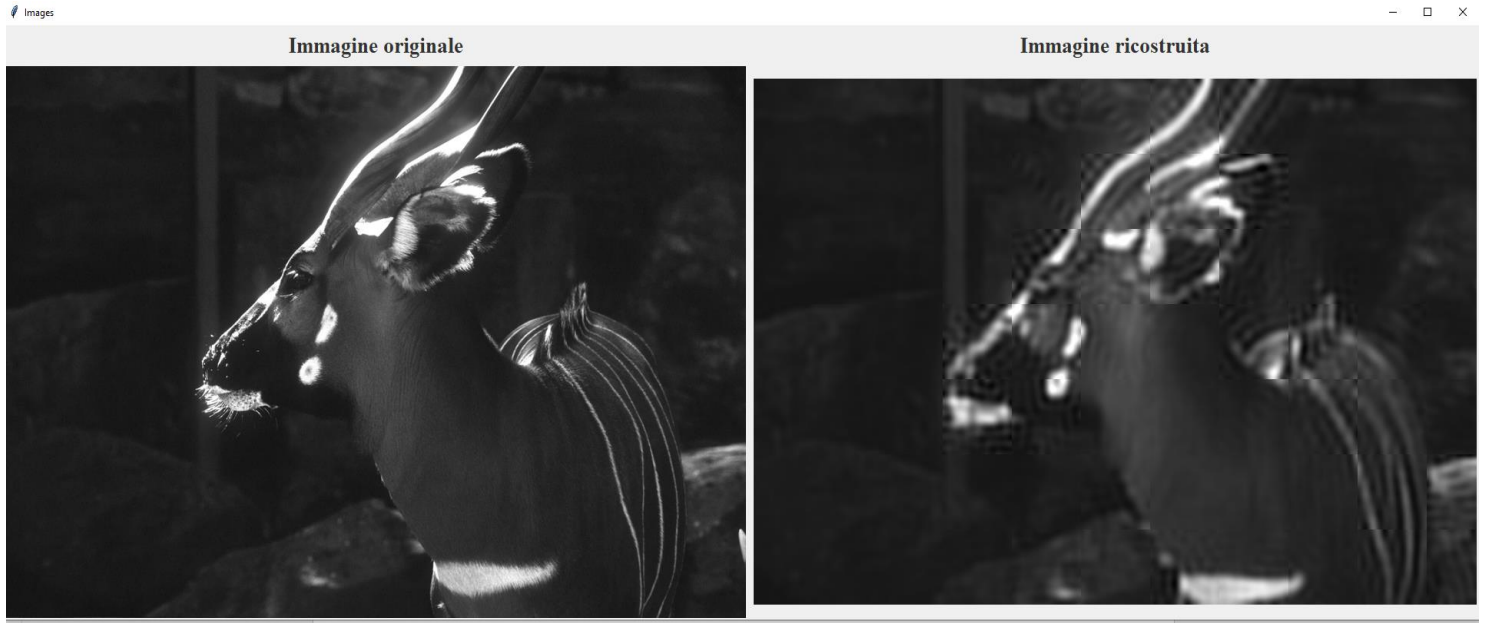
- ***Size:*** *320 X 320;*
- ***F:*** *40;*
- ***d:*** *70.*

Seconda Parte – Esperimenti (6/6)

Immagini con Contrasti Deboli e Misti



- ***Size:*** *2000 X 3008;*
- ***F:*** *40;*
- ***d:*** *7.*



- ***Size:*** *1011 X 661;*
- ***F:*** *90;*
- ***d:*** *15.*

GRAZIE PER L’ATTENZIONE