

Compressioni delle immagini con la Trasformata Discreta del Coseno

Metodi del calcolo scientifico

Matamoros Ricardo 807450

r.matamorosaragon@campus.unimib.it

19 giugno 2019

Introduzione

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

L'obiettivo di questo progetto è quello di analizzare gli effetti della compressione di tipo Jpeg su immagini in scala di grigio.

Tale studio è stato realizzato utilizzando la libreria open source SciPy-FFTpack fornita in Python.

Inoltre l'intero progetto si divide in due parti:

- La prima parte prevede il confronto tra l'implementazione propria della DCT2 e quella fornita nella libreria.
- La seconda parte consiste in eseguire un insieme di task per lo studio della compressione.

Descrizione SciPy-FFTPack

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

- La documentazione per la funzione DCT è stata realizzata adeguatamente.
- Questa libreria implementa 3 degli 8 tipi di DCT esistenti.
- Ogni implementazione permette la normalizzazione dei risultati.
- SciPy fornisce anche una gamma di utility per algebra lineare e per operazioni su matrici tramite NumPy.

Implementazione DCT2

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

```
def dct_one(vector):
    lista=[]
    size = len(vector)

    for k in range(size):
        ck = 0.0
        if k==0:
            alfa_k_N=1.0/math.sqrt(size)
        else :
            alfa_k_N=math.sqrt((2.0/size))

        for i in range(size):
            fi=vector[i]
            coseno=math.cos(float(math.pi*(2.0*i+1)*k)/float(2.0*size))
            ck+=float(fi*coseno)
        lista.append(round(float(alfa_k_N*ck),8))

    return np.array(lista)

def my_dct(block):
    tmp=[]
    for row in block:
        tmp.append(dct_one(row))
    return np.array(tmp)

def my_dct2(block):
    result_row_dct=my_dct(block)
    tras=result_row_dct.T
    result=my_dct(tras)
    return result.T
```

Test correttezza DCT

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

E' stato verificato che la DCT2 fornita da SciPy utilizzasse lo scaling richiesto.

```
#verifica richiesta
matrix = 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]])


#verifica della dct per la prima riga
test_row1=dct(matrix[0], norm= 'ortho')

#verifica di my_dct per la prima riga
mydct=dct_one(matrix[0])

#verifica della dct2 per la matrice
matrix_dct2=np.array(dct2(matrix))

print('row1: ',matrix[0])
print('DCT(row1): ',test_row1)
print('My DCT(row1): ',mydct)
print(matrix_dct2)
```

Analisi del tempo computazionale

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

La complessità del tempo è stata misurata attraverso una serie di applicazioni della DCT2 a matrici di dimensioni crescenti.

- La prima fase di applicazione della DCT2 è stata realizzata su 100 matrici di dimensione NxN, con N=2,..,100.
- La seconda fase di applicazione della DCT2 è stata realizzata su 10 matrici di dimensione NxN, con N=100*i dove i=1,..,10.

Il tempo computazionale per MyDCT2 non ottimizzata è proporzionale a $O(N^3)$.

Il tempo computazionale per la DCT2 fornita da SciPy è proporzionale a $O(N^2 * \log(N))$.

Prima applicazione della DCT2

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

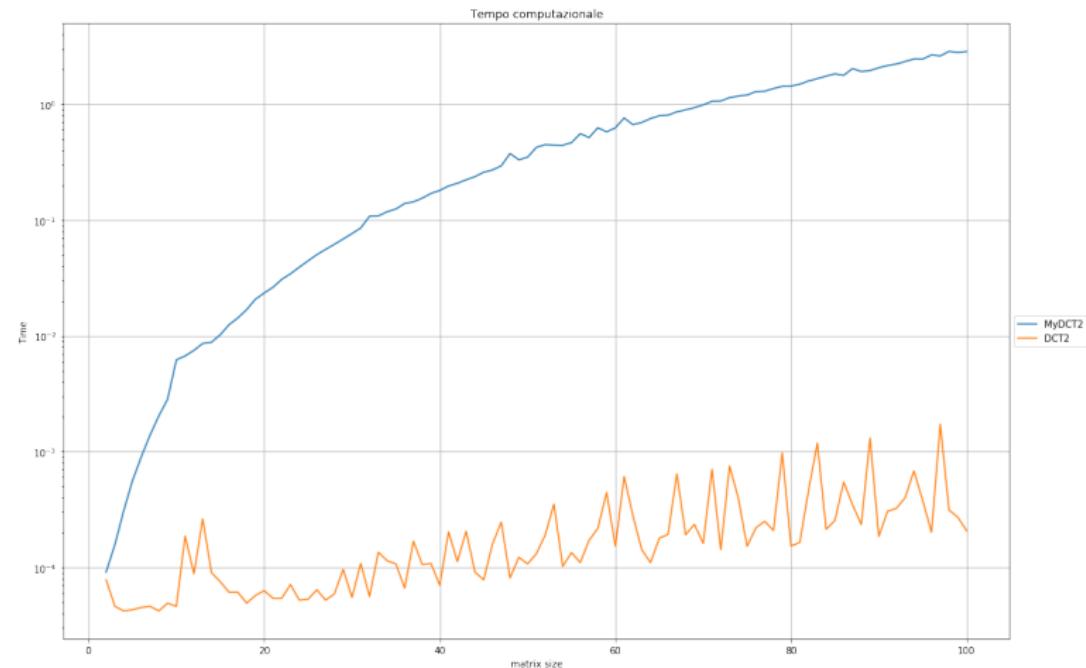
Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples



Seconda applicazione della DCT2

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

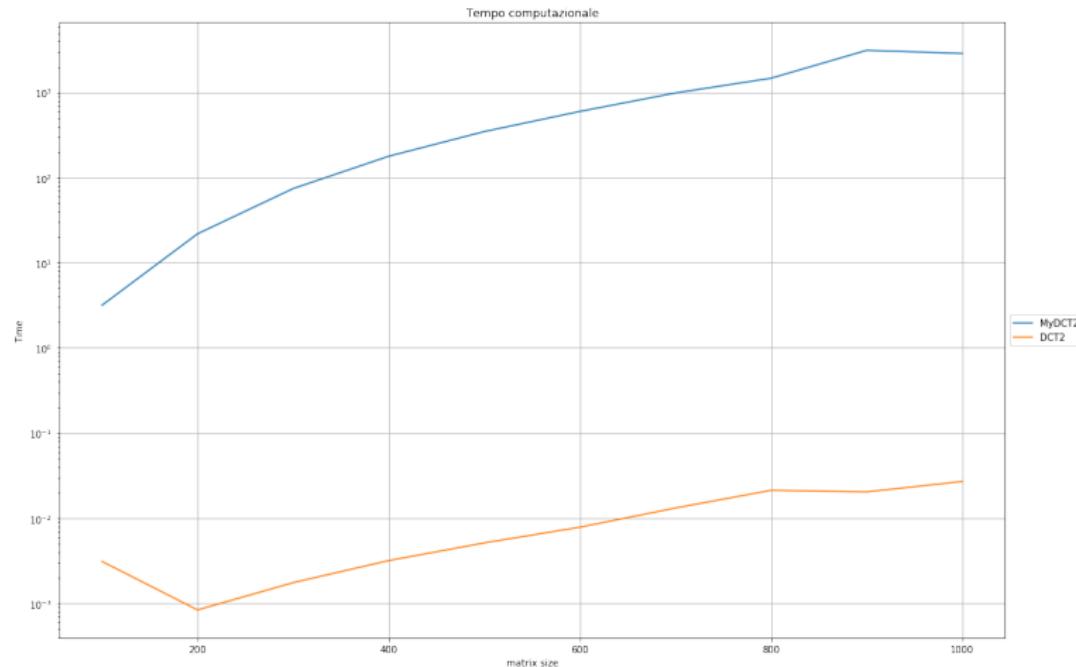
Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples



Considerazioni

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

Dall'analisi dei grafici si nota come la DCT2 non ottimizzata sia N volte più lenta della DCT2 fornita da SciPy-FFTpack, dato che quest'ultima viene calcolata con una FFT (Fast Fourier Transform).

Dunque per questo motivo la DCT2 presenta una drastica riduzione della complessità del tempo rispetto a MyDCT2.

Descrizione Processo

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

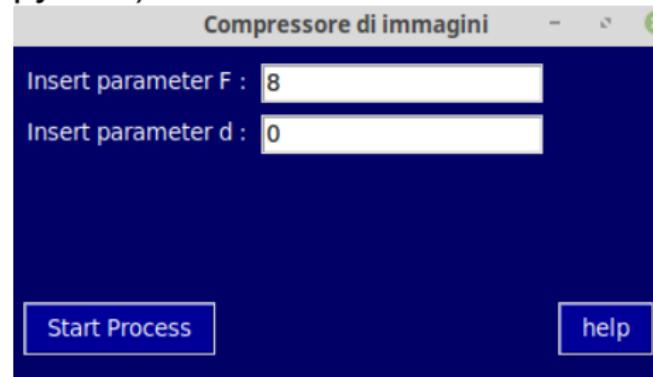
descrizione

Algoritmo simple_JPEG

Codice

Examples

Per questa parte del progetto è stata sviluppata una GUI per l'interazione col processo di compressione delle immagini, attraverso l'applicazione della DCT2. L'algoritmo basato su JPEG è stato implementato in python e la GUI è stata realizzata con Tkinter (lib di python).



Descrizione algoritmo

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

- **Input:** l'algoritmo riceve in input tre parametri, di cui due sono interi 'F' e 'd', e un'immagine.
- **Output:** l'algoritmo genera in output l'immagine iniziale ma compressa.

L'applicazione funziona con immagini a colori e in scala di grigi, le immagini vengono convertite in scala di grigi applicando la formula di conversione RGB standard. L'algoritmo è progettato per funzionare con .bmp.

Descrizione algoritmo

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

La pipeline del processo è la seguente:

- Split dell'immagine in blocchi di dimensione FxF
- Per ogni blocco :
 - Si applica DCT2
 - Filtraggio delle frequenze, $\text{row} + \text{col} \geq d$
 - Si applica IDCT2
 - Normalizzazione dei valori
- costruzione della matrice finale

Codice

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

```
#metodo che calcola l'idct2 di ogni sotto matrice
def idct2_process(block):
    block=idct2(block)
    for i in range(len(block)):
        for j in range(len(block)):
            if(block[i][j] < 0):
                block[i][j]=0
            if(block[i][j] > 255):
                block[i][j]=255
            tmp=round(block[i][j])
            block[i][j]=int(tmp)
    return block

#metodo che converte un matrice in immagine.bmp
def build_final_image(blocks,size_rows,size_cols):
    blocks=split_list(blocks,size_cols)
    rows=[]
    for block in blocks:
        tmp= np.concatenate(block, axis=1)
        rows.append(tmp)

    image=np.concatenate(rows, axis=0)
    scipy.misc.imsave('..\\immagini\\outfile.bmp', image)

#metodo principale, richiama le funzioni descritte in precedenza
def DCT_process(image):
    size_rows, size_cols = image.shape
    dim_blocco=parameter_F.get()
    size_rows = size_rows - (size_rows%dim_blocco)
    size_cols = size_cols - (size_cols%dim_blocco)
    image_new=image[0:size_rows,0:size_cols]
    sub_matrixs= blockshaped(image_new, dim_blocco, dim_blocco)
    mtxs_dct2=[]
    for i in range(len(sub_matrixs)):
        mtxs_dct2.append(dct2(sub_matrixs[i]))
    ff_m=[]
    for i in range(len(mtxs_dct2)):
        ff_m.append(delete_frequence(mtxs_dct2[i]))

    idct2_mtx=[]
    for i in range(len(ff_m)):
        idct2_mtx.append(idct2_process(ff_m[i]))

    build_final_image(idct2_mtx, size_rows, size_cols)

#funzione di controllo dei parametri 'F' e 'd'
def process():
    flag=True
    if parameter_F.get() <= 0:
        message(1)
        flag=False

    if parameter_d.get() < 0 or parameter_d.get() > (2*parameter_F.get() - 2):
        message(3)
        flag=False

    return flag

#metodo che elimina le frequenze in base al parametro d
def delete_frequence(block):
    if parameter_d.get()==0:
        for i in range(len(block)):
            for j in range(len(block)):
                block[i][j]=0

    elif parameter_d.get()=(2*parameter_F.get() - 2):
        block[-1][-1]=0

    else:
        for i in range(len(block)):
            for j in range(len(block)):
                if i+j > parameter_d.get():
                    block[i][j]=0

    return block
```

Example F=10, d=18, size=1930x2800

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

original image



compressed image



Example F=100, d=50, size=1930x2800

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

original image



compressed image



Example F=100, d=10, size=1930x2800

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples

original image



compressed image



Example F=300, d=50, size=1930x2800

Introduzione

Prima Parte

MyDCT2

Tempo computazionale

Considerazioni

Seconda Parte

descrizione

Algoritmo simple_JPEG

Codice

Examples



GRAZIE PER L'ATTENZIONE



Matamoros Ricardo 807450