Assignment NO.4 Solutions

Digital Image Processing | Fall 1400 | Dr.Mohammadi

Teacher Assistant : Samin Heydarian

Student name : **Amin Fathi**

Student id : **400722102**

Problem 1

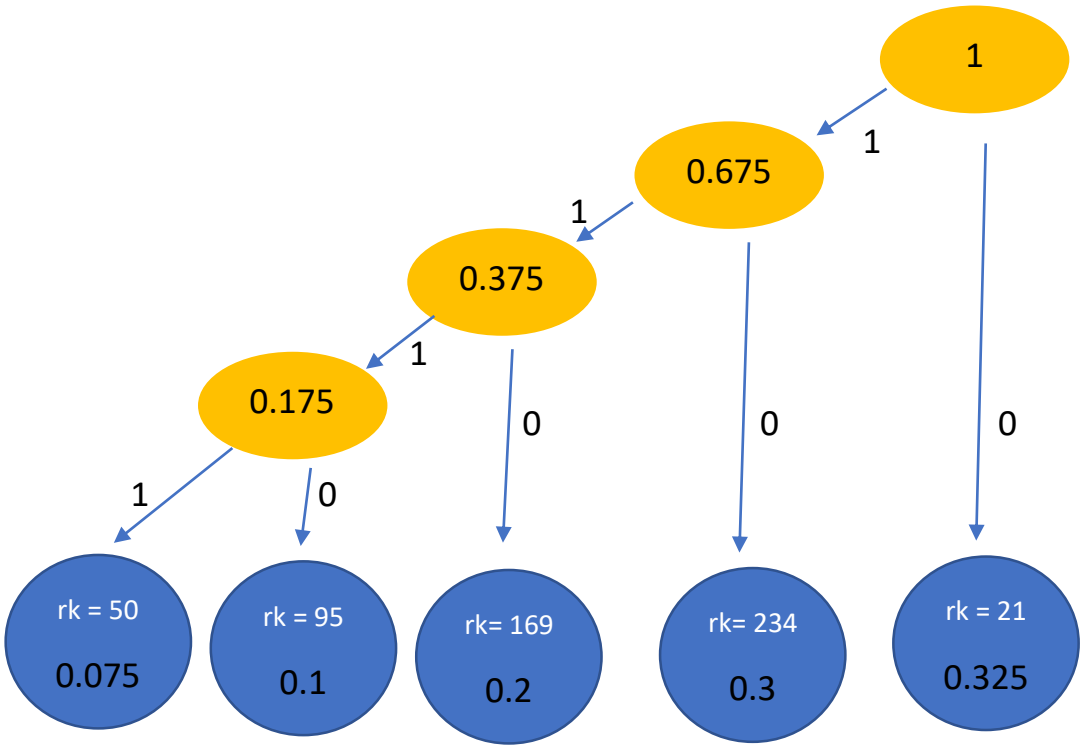
تصویر زیر را به دو روش عادی (8بیت برای هر سطح روشنایی) و هافمن کدگذاری کنید. سپس نرخ فشرده سازی Cو افزونگی نسبی داده Rرا محاسبه کنید

rk	Pr(rk)	Bit 8	Lr	Huffman	lr
21	0.325	00010101	8	0	1
50	0.075	00110010	8	1111	4
95	0.1	01011111	8	1110	4
169	0.2	10101001	8	110	3
234	0.3	11101010	8	10	2

Lavg = 0.325*1 + 0.3*2 + 0.2*3 + 0.1*4 + 0.075*4 = 2.225 bit

C = Lr 8 bit / Lr Huffman = 8 / 2.225 = 3.59 = 3.6

R = 1 – 1/C = 0.72



Problem 2

اگر از روش فشرده سازی پیش گویانه استفاده کنیم، پخش زنده اینترنتی کدام یک از دو ورزش در هر مورد، حجم بیشتری از داده را مصرف خواهد کرد؟ دلیل خود را فقط برای مورد الف توضیح دهید .

الف) کارتینگ یا شطرنج؟

ب) تیراندازی یا اسب سواری؟

ج) دو 100 متر یا یوگا؟

الف) کارتینگ ، این روش فشرده سازی در فشرده سازی ویدیو ها بسیار پر کاربرد است ، از آنجا که اساس عملکرد آن بر مبنای اختلاف بین فریم هاست و در شطرنج نسبت به کارتینگ بین فریم های ویدیو اختلاف کمتری داریم ، بنابراین در شطرنج حجم داده کمتری استفاده خواهد شد .

ب) اسب سواری

ج) دو 100 متر

Problem 3

ابتدا کتابخانه های مورد نیاز را import میکنیم .

```
#### Import library ####
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import scipy.fftpack as fftpack
from scipy.fft import ifft, idct
from PIL import Image as im
```

در این قسمت ، بردار های مورد نیاز و طرح شده در سوال تعریف شده اند .

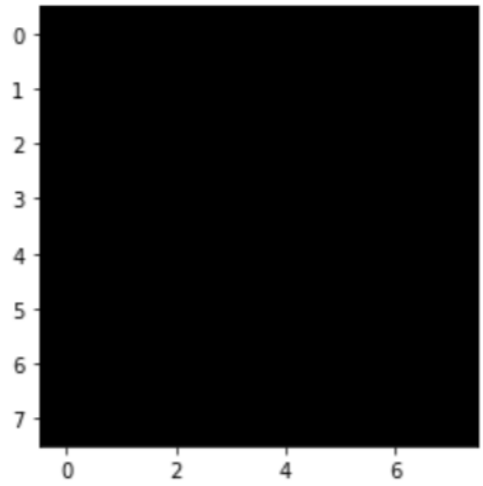
```
#### Vectors ####

a = [-27, -17, -15, -3, 11, -4, -1, 0, 0, -1, -1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 'EOB']
b = [-27, 17, -15, -3, -11, -4, 1, 0, 0, -1, -1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 'EOB']
c = [-27, -17, 15, -3, -11, -4, -1, 0, 0, 1, -1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 'EOB']
d = [-27, 17, 15, -3, 11, -4, 1, 0, 0, 1, -1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 'EOB']
```

در این قسمت هم ماتریس Z تعریف شده است .

```
#### Matrix Quantization ####

Z = np.array([[16,11,10,16,24,40,51,61],
              [12,12,14,19,26,58,60,55],
              [14,13,16,24,40,57,69,56],
              [14,17,22,29,51,87,80,62],
              [18,22,37,56,68,109,103,77],
              [24,35,55,64,81,104,113,92],
              [49,64,78,87,103,121,120,101],
              [72,92,95,98,112,100,130,99]])
```

سپس تابع show را جهت تبدیل array به عکس تعریف می کینیم ، این تابع آرایه را به عنوان ورودی گرفته و در خروجی عکس متناظر با آن را تحویل می دهد ، از ان جا که ابعاد تصاویر در این سوال 8*8 می باشد ، به عنوان تصویر اولیه ، یک تصویر 8*8 مشکی تعریف کرده که مقادیر پیکسل های آن را در دو حلقه for متناظر با مقدار آرایه تغییر می دهیم ، کد این قسمت و تصویر مشکی اولیه را در زیر مشاهده می کنید .

```
#### Define Image Show Function ####
def show(array):
  img = np.zeros([8,8],dtype=np.uint8)
  img.fill(0)
  for i in range(8):
    for j in range(8):
      img[i][j] = array [i][j]
  return img
```



سپس در قسمت بعدی ، تابع Vecmat را تعریف می کنیم ، وظیفه این تابع تبدیل آرایه صورت سوال به ماتریس T-hat با در نظر گرفتن الگوی زیگ زاگ است ، این تابع در ورودی vector را دریافت می کند و خروجی آن ماتریسی است به ابعاد 8 * 8 که همان ماتریس T-hat مورد نیاز برای فشرده سازی است .

```
#### Define vector_to_matrix Function ####
def Vecmat(vector):
  zeros = [[0 for row in range(8)] for row in range(8)]
  i = 0
  j = 0
  sum2 = 14
  k =0
  for sum in range(0, sum2 +1):
    if sum %2 == 0 :
      for i in range( 0,sum+1):
        if a[k] != 'EOB':
          zeros[ sum - i][i] = vector[k]
          k = k+1
        else :
          break
    else :
      for i in range( 0,sum+1):
        if vector[k] != 'EOB':
          zeros[i][sum -i] = vector[k]
          k = k+1
        else :
          break
  return np.array(zeros)
```

همانطور که در شکل بالا مشاهده می شود ، ابتدا یک ماتریس صفر 8 * 8 به نام zeros ایجاد شده و سپس مقادیر vector به این ماتریس به صورت زیگ زاگ منتقل می شوند (شیوه انتقال اقتباس شده و به نوعی معکوس یافته شیوه ای هست که در لینک زیر استفاده شده است) مقدار sum2 هم بیانگر مقدار مجموع درایه های اخرین سطر و ستون ماتریس است که برابر 14 می باشد .

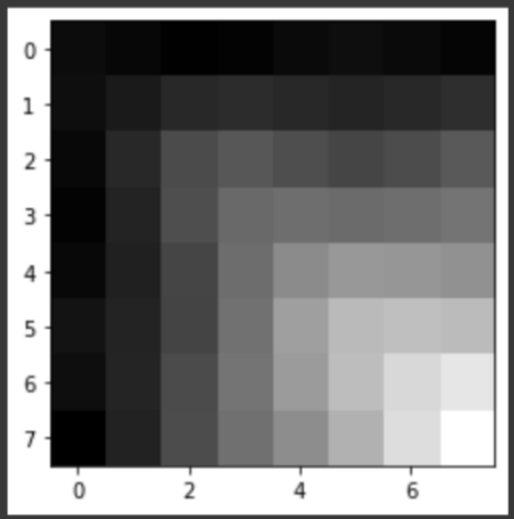
Print matrix in zig-zag fashion - GeeksforGeeks

در ادامه تابع jpeg_decompress را تعریف میکنیم ، این تابع وظیفه پیدا کردن T-dot بر اساس روابط موجود در فرایند Compression را بر عده دارد .

```
#### Define JPEG Decompression Function #####

def jpeg_decompress(T_hat, Z):

  T_dot = [[0 for row in range(8)] for row in range(8)]
  b = T_hat * Z
  idctb = cv.idct(np.float32(b))
  T_dot = np.around(idctb+ 128)
  print(T_dot)
  print('idct ortho ' , np.around(idct(b , norm= 'ortho')+128)  )


  # Please set norm='ortho' in Discrete Cobine Transform Function in fftpack

  return T_dot
```

حال در قسمت A مقدار غیر فشرده سازی شده بردار a را حساب می کنیم

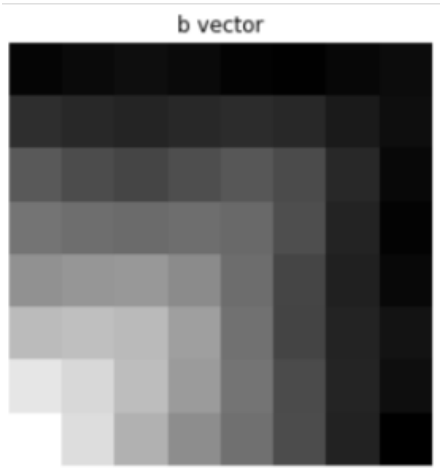
```
####  Decompress Vector "a"  ####
#### Show Decompressed Image ####
image1 = show(jpeg_decompress(Vecmat(a) , Z))
imgplot = plt.imshow(image1 , cmap = 'gray')
plt.show()
```

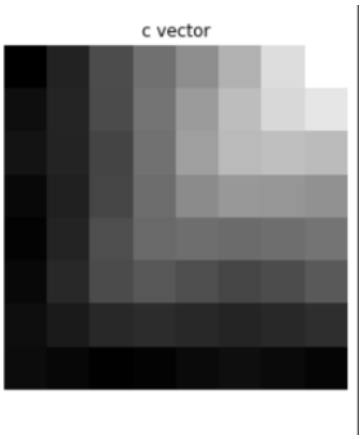


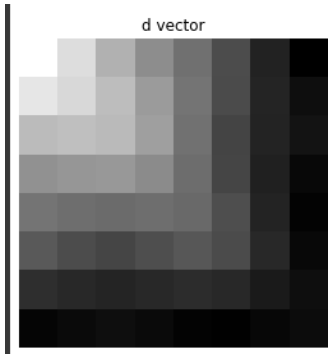
این کار را برای بردار های b ,c ,d هم تکرار می کنیم که به ترتیب در ادامه مشاهده می کنید .

```
#### Decompress Vector "b", "c" And "d" ####
from mpl_toolkits.axes_grid1 import ImageGrid
image2 = show(jpeg_decompress(Vecmat(b) , Z))
image3 = show(jpeg_decompress(Vecmat(c) , Z))
image4 = show(jpeg_decompress(Vecmat(d) , Z))

fig = plt.figure(figsize=(16,16))
grid = ImageGrid(fig, 111,
                 nrows_ncols=(3, 1),
                 axes_pad=1,
                 )


grid[0].imshow(image2,cmap='gray')
grid[0].set_title('b vector')
grid[0].axis('off')
grid[1].imshow(image3, cmap='gray')
grid[1].set_title('c vector')
grid[1].axis('off')
grid[2].imshow(image4 , cmap = 'gray')
grid[2].set_title('d vector')
grid[2].axis('off')
```

تصاویر حاصل :



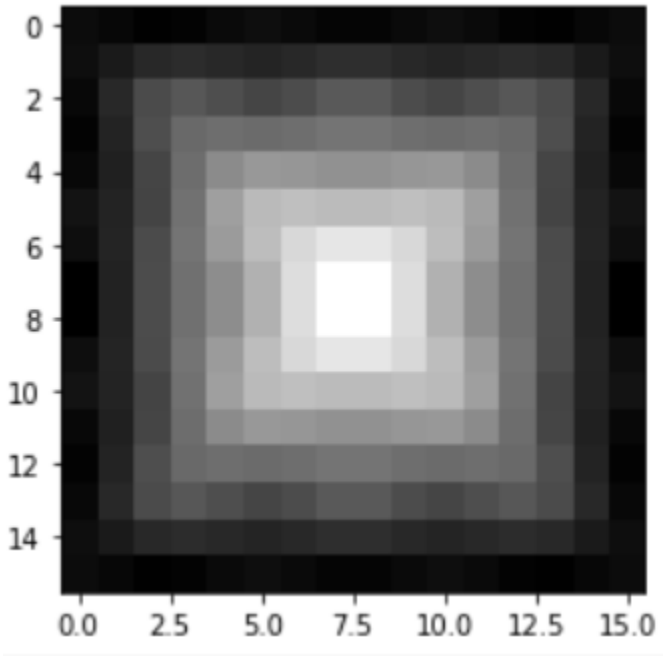




و در قسمت آخر عکس ها را به ترتیبی که در روی سوال ذکر شده است در کنار هم قرار می دهیم ، برای این منظور ابتدا یک عکس 16*16 سیاه تعریف کرده و سپس مقادیر پیکسل های هر یک از قسمت هایش را برابر با مقادیر تصاویر به دست امده در قسمت قبل می کنیم .

```
#### Concatenate Decompressed Images ####
####     Show Concatenated Image     ####
imag = np.zeros([16,16],dtype=np.uint8)
imag.fill(255)
for i in range(8):
  for j in range(8):
      imag[i][j] = image1[i][j]
for i in range(8):
  for j in range(8 , 16):
      imag[i][j] = image2[i][j-8]
for i in range(8 , 16):
  for j in range(8):
      imag[i][j] = image3[i-8][j]
for i in range(8 , 16):
  for j in range(8 , 16):
      imag[i][j] = image4[i-8][j-8]

mgplot = plt.imshow(imag , cmap = 'gray')
plt.show()
```

تصویر حاصل:



Problem 4

ابتدا کتابخانه ها را import کرده و تصاویر را به عنوان ورودی در نوتبوک ذخیره می کنیم .

```
#### Import library ####
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
from scipy.fftpack import fft2, ifft2
from google.colab import files
uploaded = files.upload()
uploaded = files.upload()
```

```
Choose Files 1.bmp
• 1.bmp(image/bmp) - 1300 bytes, last modified: 11/25/2021 - 100% done
Saving 1.bmp to 1 (1).bmp
Choose Files stone.jpg
• stone.jpg(image/jpeg) - 24165 bytes, last modified: 11/25/2021 - 100% done
Saving stone.jpg to stone (1).jpg
```

تابع read را جهت خواندن عکس ها در مود خاکستری تعریف می کنیم .

```
#### Define Image And Sensor Read Function ####
def read(path):
  output = cv.imread('{0}'.format(path) , 0)
  return output
```

همچنین تابع show جهت نمایش عکس ها را نیز تعریف می کنیم ، که تصاویر را در مود خاکستری نشان می دهد .

```
[143] #### Define Image Show Function ####
      def show(image):
        plt.figure(figsize = ( 8 ,8))
        plt.imshow(image , cmap = 'gray'  )
        plt.show()
```

حال عکس stone را با استفاده از دو تابع قبلی ، خوانده و به نمایش می گزاریم و آن را در متغیر ReadImage ذخیره می کنیم .

```
#### Read And Show Image ####
ReadImage = read('stone.jpg')
show(readimage)
```

در ادامه سنسور را می خوانیم و در متغیر ReadSensor ذخیره میکنیم

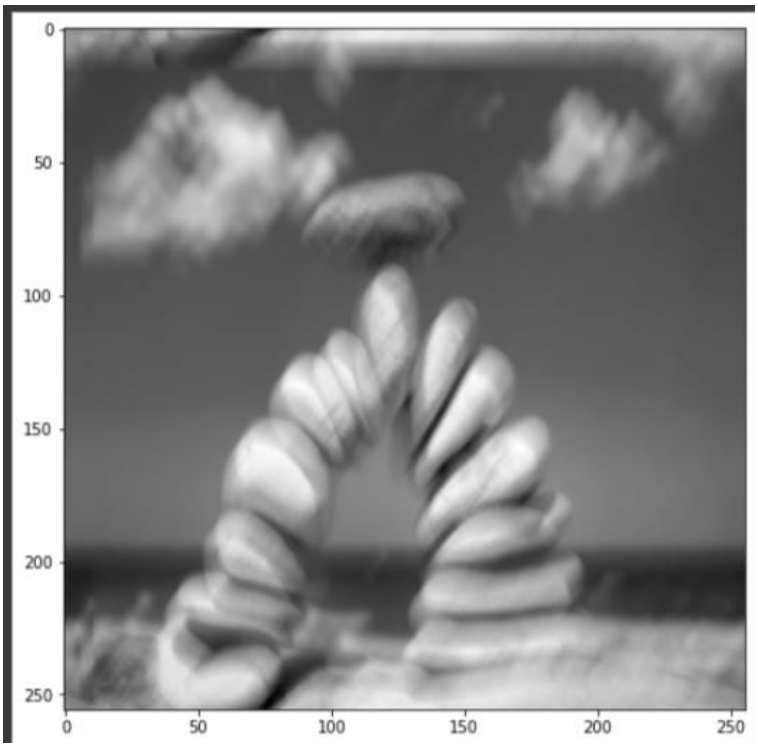
```
#### Read Sensor ####
ReadSensor = read('1.bmp')
```

در قسمت بعد ، تابع degrade را تعریف می کنیم که وظیفه آن خروجی دادن تصویر با فیلتر motion است ، این تابع عکس اصلی و سنسور را به عنوان وروی گرفته و سپس مقدار zero پدینگ یافته ان در جهت پایین و راست (تا هم اندازه شدن با تصویر) را در متغیر psfpadding ذخیره کرده و سپس مقدار فوریه دو بعدی آن را در مقدار فوریه دو بعدی تصویر ضرب کرده و از حاصل ضرب آن دو فوریه دو بعدی معکوس گرفته و پس از نرمالایز کردن تصویر به عنوان عکس نهایی خروجی می دهد .

```python
#### Define Degradation Function ####
def degrade(image, psf):
    imw = image.shape[1]
    imh = image.shape[0]
    pw = psf.shape[1]
    ph = psf.shape[0]
    psfpadding = cv.copyMakeBorder(psf, 0, imh - ph , 0, imw - pw, cv.BORDER_CONSTANT,value=0)

    PSF = fft2(psfpadding)
    IMAGE = fft2(image)
    out = ifft2(IMAGE*PSF).real
    normout = (out - np.min(out)) / (np.max(out) - np.min(out))

    # Rescale Your Result to [0-255], Then Return Output
    return normout
```

و در انتها نتیجه نهایی نمایش داده می شود

```python
####    Degrade Image    ####
#### Show Degraded Image ####
final = degrade(ReadImage ,ReadSensor)
show(final)
```



Problem 5

دو روش فشردهسازی مبتنی برDCT وPCA را مقایسه کنید (مزایا و معایب آنها را توضیح دهید.) به منظور آشنایی بیشتر با کاربردهای PCAدر پردازش تصویر میتوانید در رابطه با EigenFace مطالعه کنید

PCA یک تکنیک کاهش ابعاد خطی است که مجموعه ای از متغیرهای همبسته و وابسته به هم به تعداد p را به تعداد k (k<p) از متغیرهای غیرهمبسته به نام مؤلفه های اصلی تبدیل می کند و در عین حال در تلاش است تا در حد ممکن تنوع در داده های اصلی خفظ شود . از مزایای PCA میتوان گفت که مقداری از فیچر ها را حذف میکند و در نتیجه به تحیل ارتباط بین آن ها نیازی نیست و این عمل به استقلال اجزای اصلی در داده کمک میکند . این عمل باعث افزایش سرعت عملکرد الگوریتم می شود ، همچنین PCA با کاهش ابعاد داده به 2 بعد ، به فهم و درک داده ها و مساله کمک شایانی می کند همچنین PCA با حذف متغیر های زیادی از داده ، از مشکل overfitting هم جلوگیری می کند ، اگر چه این قضیه می تواند چنانچه اگر اصولی به اجرا در نیاید ، منجر به حذف اطلاعات شود ، همچنین بنابه آن چه در الگوریتم PCA ذکر شده است ، باید قبل از اعمال ان به داده عمل استاندارد سازی داده رخ دهد .

DCT یا تبدیل کسینوسی گسسته به ما کمک میکند تا تصویر را با حفظ حداکثر کیفیت ممکن جهت اعمال تغییراتی در ان به فضای فرکانسی منتقل کنیم و عمل پردازش تصویر راحت تری داشته باشیم .

از معایب dct میتوان نیاز به کوانتیزه شدن در طی مراحل اشاره کند ، خروجی ماتریس آن برای ماتریس هایی که تغییرات زیادی در آن ها رخ نداده است دارای مقادیر زیادی صفر است که کار محاسبات را راحت تر می کند ، انرژی عکس در کل حفظ می شود در فرایند تبدیل و تبدیل معکوس idct اما ممکن است که به خاطر کوانتیزه شدن مقداری از اطلاعات از دست برود . مهمترین نقطه ضعف طرح‌های فشرده‌سازی تصویر که مبتنی بر DCT هستند، این واقعیت است که در تبدیل به فضای فرکانسی ، اطلاعاتی از مکان یابی پیکسل ها منتقل نمی شود به همین دلیل، تصاویر باید در بلوک هایی که به طور جداگانه تبدیل می شوند، تقسیم شوند.که در نسبت‌های فشرده‌سازی بالا، تقریب‌های انجام‌شده در مرحله کوانتیزاسیون می‌تواند تفاوت‌های مهمی بین پیکسل‌های همسایه در لبه بلوک‌ها ایجاد کند که میتواند منجر به از بین رفتن آن حس طبیعی بودن تصاویر شود

از تفاوت های پایه ای روش PCA و DCT میتوان به نحوه محاسبه ماتریس کورلیشن در این دو روش تبدیل اشاره کرد .در PCA این ماتریس باید با توجه به دیتای ارایه داده شده تنظیم شود (تطبیقی) ولی در DCT این ماتریس مستقل از داده تعریف می شود . هر چند PCA با توجه به حذف وابستگی فیچر ها میتواند کار بردی باشد اما برای رمز گشایی تصویر با این روش هم می بایست ماتریس تبدیل را داشته باشیم که ممکن است رمزگشا به دیتا هایی که PCA با استفاده از آن ها ماتریس تبدیل را تعریف کرده است دسترسی نداشته باشد . اما در DCT ماتریس تبدیل هم برای رمزگزار و هم رمزگشا از پیش تعریف شده است .

منابع

.1

https://towardsdatascience.com/image-compression-using-principal-component-analysis-pca-253f26740a9f

.2

The Discrete Cosine Transform (DCT) (cf.ac.uk)

.3

What are the Pros and cons of the PCA? | i2tutorials

.4

https://www.whydomath.org/node/wavlets/dct.html

.5

image processing - Relationship between DCT and PCA - Signal Processing Stack Exchange