Ukrainian Catholic University

Faculty of Applied Sciences

Computer Science Programme

An Application of Linear Algebra to Image Compression

Linear Algebra second interim project report

Authors:
Yuliia Hapunovych
Vitalii Papka

08 May 2020

Abstract

With rapid demand for storing a large amount of multimedia content, the space to store it is getting less compared to the mass of data which needs to be stored. In this context, this paper addresses the problem of the lossy compression of images. Main proposed methods are based on SVD and DCT techniques. We implemented the mentioned algorithms of compression and performed experiments using images as input data. The code for implementation is available on :
`https://github.com/YulitaGap/LinearAlgebraProject`.

1 Introduction

Due to the high demand for transmitting data in the form of images and graphics, fast compression techniques are particularly required. This is precisely the role of linear algebra in data compression: to convert arrays of bits into shorter ones for more economical and profitable transmission, storage and processing.

Not surprising that intense mathematical research in this direction has been going on and many methods are available for compression of still images. However, the most widely used image compression technique today is JPEG which uses DCT for compression of images, capable of achieving data compression ratios from 8:1 to 14:1.

Even though DCT gives high energy compaction as compared to SVD which gives optimal energy compaction, SVD performs better than DCT in case of images having high standard deviation (i.e. higher pixel quality) Our goal is to make the research on the elegant application of linear algebra and concepts to the problem of image compression, and illustrate how certain theoretical mathematical results can be effectively used in applications that include transformations in the image representation.

2 Problem setting

2.1 Notations

A^T — transposed matrix σ - singular value Re- real part of number
$\mathcal{F}$ - forward Fourier transform $\mathcal{F}^{-1}$ - inverse Fourier transform;

2.2 Problem formulation

The main goal of image compression is to reduce the storage quantity as much as possible while ensuring that the decoded image displayed in the monitor can be visually similar to the original image as much as it can be. Therefore, in this paper, we will make a review of two main techniques of lossy image compression, where a detail of the input image can be lost, but ultimately, the stored data size is drastically reduced.

The focus of this project is to research the application of Singular-Value Decomposition and Discrete Cosine Transform to the solution for the problem of reducing the redundancy of the image and the transferring data in an efficient form. Based on research conclusions, to summarize how tools of linear algebra, including matrix manipulation, would be useful outside of mathematics, namely to the image compression process.

3 Overview of approaches

Our project undertakes the following approaches as a base:

3.1 Singular-Value Decomposition

Singular value decomposition (SVD) is a generalization of the eigen-decomposition used to analyze rectangular matrices. The main goal of application the SVD to an image (matrix of m $\times$ n) is to create approximations of an image using the least amount of the terms of the diagonal matrix in the decomposition. Where U and V are orthogonal, Σ is a diagonal matrix, and $U\Sigma V^T$ is the closest rank-k approximation to A (see Figure 1).

The purpose of transforming matrix A into $U\Sigma V^T$ is to approximate matrix A by using far fewer entries than in the original matrix. By using the rank of a matrix, we remove the redundant information (the dependent entries). SVD can be used to provide the best lower-rank approximation to matrix A and thus be used for image compression.

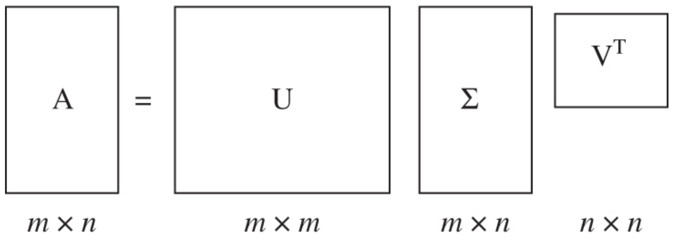


Figure 1.

3.2 Discrete Cosine Transformation

Compression of an image can also be done via a discrete cosine transformation. The original image is transformed in 8-by-8 blocks, and for pixel values of each of them we compute DCT. After this, the quantization of DCT coefficients for all image blocks is carried out. The larger quantization step, the larger compression ratio received, and simultaneously, it leads to more significant losses.

After that, the division of quantized DCT coefficients into bit-planes is carried out. The obtained bit-planes are coded in the order of the descending bits. While plane coding iterates, the values of bits of previously coded planes are taken into account. A coded bit is referred to one of the groups of bits according to the values of already coded bits. At the image decoding stage, all steps are repeated in reverse order.

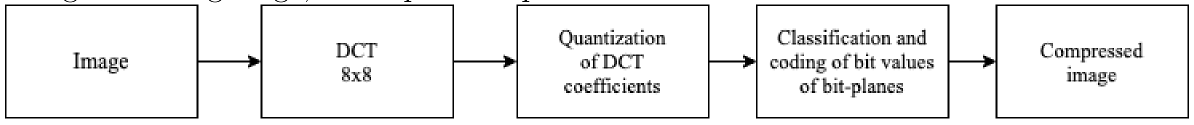


Image
DCT 8x8
Quantization of DCT coefficients
Classification and coding of bit values of bit-planes
Compressed image

Figure 2.

4 Algorithm analysis

4.1 Singular-Value Decomposition

What the SVD does is split a rectangular m x n matrix into three important sub matrices to represent the data:

- U — matrix of left singular vectors in the columns
- Σ — diagonal matrix with singular values
- V^T — matrix of right singular vectors in the columns

Calculating the SVD consists of finding the eigenvalues and eigenvectors of AA^T and A^TA. The eigenvectors of A^TA make up the columns of V , the eigenvectors of AA^T make up the columns of U.

$AA^T = (U\Sigma V^T)(V\Sigma^T U^T) = U\Sigma\Sigma^T U^T$ and,similarly, $A^TA = V\Sigma^T\Sigma V^T$.
Also, the singular values in Σ are square roots of eigenvalues from AA^T or A^TA. The singular values are places as the diagonal entries of the Σ matrix and are arranged in descending order. The singular values are always real numbers. If the matrix A is a real matrix, then U and V are also real.

The method of image compression using singular value decomposition is based on the idea that after the SVD is known, some of the singular values σ are significant while the others are small and insignificant. Therefore, if the significant values are kept, and the small values are discarded, then only those columns of U and V that correspond to the singular values are used.
The reason why the SVD is used is that you can use only the first components of these matrices to give you a close to real matrix approximation.

In the implementation below, we apply SVD to the image of a size 900x474. Suppose we know the SVD. The key is in the singular values (in Σ). Typically, some 's are significant and others are extremely small. If we keep 20 and throw away other 454, then we send only the corresponding 20 columns of U and V. The other 880 columns are multiplied in $U\Sigma V^T$ by the small σ 's that are being ignored. We can do the matrix multiplication as columns times rows:

$A = U\Sigma V^T = u_1\sigma_1 v_1 + u_2\sigma_2 v_2 + ... + u_r\sigma_r v_r$.
Any matrix is the sum of r matrices of rank 1. If only 20 terms are kept, we send 20 times 1800 numbers instead of a 900x474=426 600 (12 to 1 compression).

The pictures are really striking, as more and more singular values are included(from 10 to 50). At first, you see nothing, and suddenly - recognize everything. The cost is in computing the SVD - this has become much more efficient, but it is still expensive for a big matrix size.

4.2 Discrete Cosine Transformation

The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain but can approximate lines well with fewer coefficients. The general DCT for a 1D (N data items) is obtained with the following algorithm:

- **Step 1:** Generate a sequence $y[m]$ from the given sequence $x[m]$:

$$\begin{cases} y[m] = x[2m] \\ y[N-1-m] = x[2m+1] \quad (i = 0, \cdots, N/2-1) \end{cases}$$

- **Step 2:** Obtain discrete Fourier transform $Y[n]$ of $y[m]$ using fast Fourier transform. (As $y[m]$ is real, $Y[n]$ is symmetric and only half of the data points need be computed.)

$$Y[n] = \mathcal{F}[y[m]]$$

- **Step 3:** Obtain DCT $X[n]$ from $Y[n]$ by

$$X[n] = Re[\ e^{-jn\pi/2N} Y[n]\ ]$$

To encode the data, we use the inverse of DCT - IDCT - reverse the order and the mathematical operations: these three steps are mathematically equivalent to the previous steps:

- **Step 1:** Generate a sequence $Y[n]$ from the given DCT sequence $X[n]$:

$$Y[n] = X[n]e^{jn\pi/2N} \qquad (n = 0, \cdots, N-1)$$

- **Step 2:** Obtain $y[m]$ from $Y[n]$ by inverse DFT also using FFT. (Only the real part need be computed.)

$$y[m] = Re[\mathcal{F}^{-1}[Y[n]]]$$

- **Step 3:** Obtain $x[m]'s$ from $y[m]'s$ by

$$\begin{cases} x[2m] = y[m] \\ x[2m+1] = y[N-1-m] \quad (i = 0, \cdots, N/2-1) \end{cases}$$

The general equation for a 2D (N by M image) DCT is defined by the following equation:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(i).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] f(i)$$

and the corresponding inverse 2D DCT transform is simple $F^{-1}(u, v)$ where :

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

5 Implementation

5.1 Singular-Value Decomposition

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
```

In [2]:

```
# Transform image to grayscale
img = Image.open('example.jpg')
imggray = img.convert('LA')
```

In [3]:

```
# Show grayscaled image form
plt.figure(figsize=(9, 6))
plt.imshow(imggray)
plt.show()
```

In [4]:

```
# Transform image to numpy matrix
imgmat = np.array(list(imggray.getdata(band=0)), float)
imgmat.shape = (imggray.size[1], imggray.size[0])
imgmat = np.matrix(imgmat)
```

In [ ]:

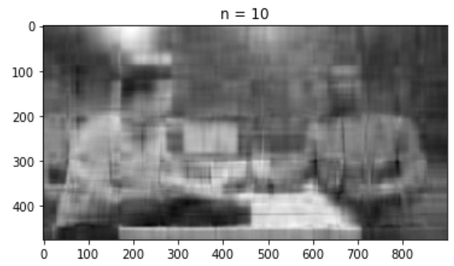
```
#SVD decomposition
U, sigma, V = np.linalg.svd(imgmat)
```

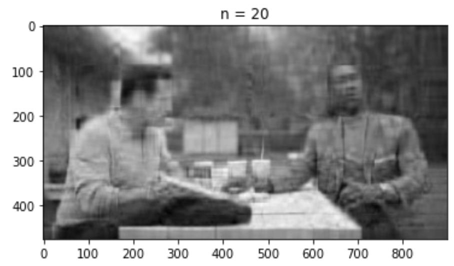
In [ ]:

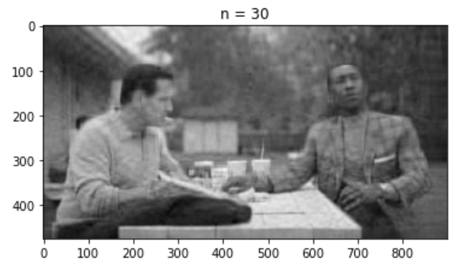
```
#How resulting image looks using from 10-50 vectors for reconstruction after decomposition
for i in range(10, 60, 10):
    reconstimg = np.matrix(U[:, :i]) * np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()
```

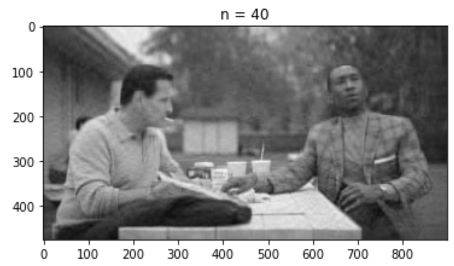
In [ ]:

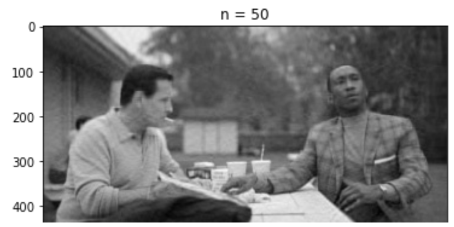
In [6]:











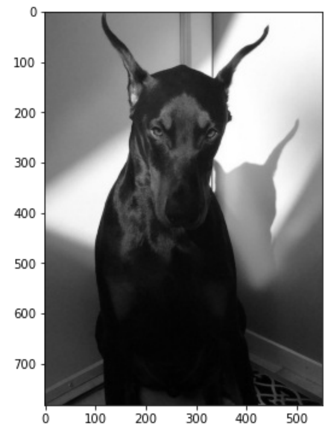
5.2 Discrete Cosine Transformation

In [1]:

```
# Import functions and libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import scipy
from PIL import Image
from numpy import r_
import scipy.fftpack
```

In [2]:

```
# Transform image to grayscale
img = Image.open('example2.jpg')
img_gray = img.convert('LA')
plt.figure(figsize=(9, 6))
plt.imshow(img_gray)
plt.show()
```



In [3]:

```
#Define 2D DCT and IDCT
def dct2(a):
    return scipy.fftpack.dct( scipy.fftpack.dct( a, axis=0, norm='ortho' ), axis=1, norm='ortho' )

def idct2(a):
    return scipy.fftpack.idct( scipy.fftpack.idct( a, axis=0 , norm='ortho'), axis=1 , norm='ortho'
)
```

In [4]:

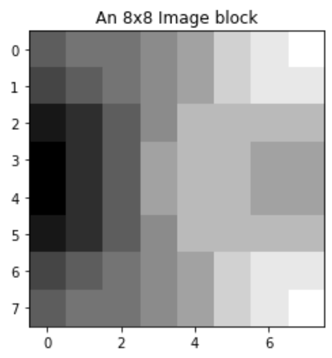
```
# Transform image to numpy matrix
img_matrix = np.array(list(img_gray.getdata(band=0)), float)
img_matrix.shape = (img_gray.size[1], img_gray.size[0])
img_matrix = np.matrix(img_matrix)
```

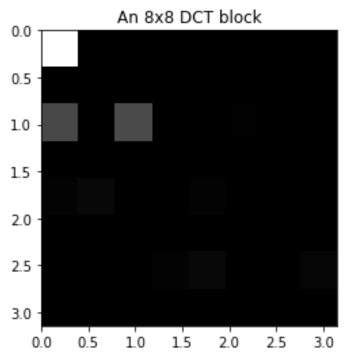
In [5]:

```
imsize = img_matrix.shape
img_dct = np.zeros(imsize)

# Do 8x8 DCT on image (in-place)
for i in r_[:imsize[0]:8]:
    for j in r_[:imsize[1]:8]:
        img_dct[i:(i+8),j:(j+8)] = dct2( img_matrix[i:(i+8),j:(j+8)] )
```

In [6]:

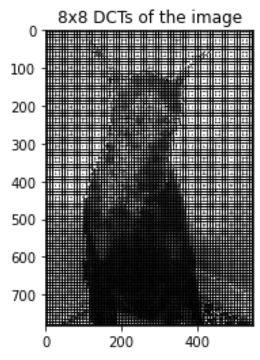
```
pos = 128
# Extract a block from image
plt.figure()
plt.imshow(img_matrix[pos:pos+8,pos:pos+8],cmap='gray')
plt.title( "An 8x8 Image block")

# Display the dct of that block
plt.figure()
plt.imshow(img_dct[pos:pos+8,pos:pos+8],cmap='gray',vmax= np.max(img_dct)*0.01,vmin = 0, extent=[0,
np.pi,np.pi,0])
plt.title( "An 8x8 DCT block")
plt.show()
```





In [7]:

```
# Display entire DCT
plt.figure()
plt.imshow(img_dct,cmap='gray',vmax = np.max(img_dct)*0.01,vmin = 0)
plt.title( "8x8 DCTs of the image")
plt.show()
```

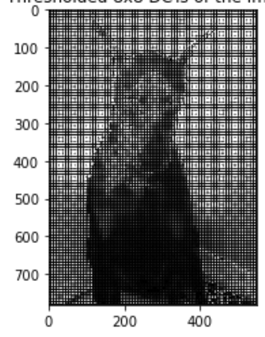


In [8]:

```
# Threshold
thresh = 0.012
dct_thresh = img_dct * (abs(img_dct) > (thresh*np.max(img_dct)))


plt.figure()
plt.imshow(dct_thresh,cmap='gray',vmax = np.max(img_dct)*0.01,vmin = 0)
plt.title( "Thresholded 8x8 DCTs of the image")

percent_nonzeros = np.sum( dct_thresh != 0.0 ) / (imsize[0]*imsize[1]*1.0)

print("Keeping only %f%% of the DCT coefficients" % (percent_nonzeros*100.0))
```

```
Keeping only 3.254359% of the DCT coefficients
```

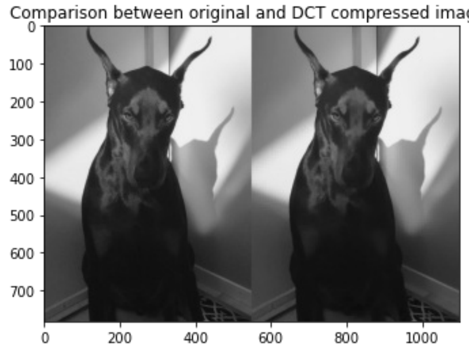
Thresholded 8x8 DCTs of the image



In [9]:

```
img_idct = np.zeros(imsize)

for i in np.r_[:imsize[0]:8]:
    for j in np.r_[:imsize[1]:8]:
        img_idct[i:(i+8),j:(j+8)] = idct2( dct_thresh[i:(i+8),j:(j+8)] )


plt.figure()
plt.imshow( np.hstack( (img_matrix, img_idct) ) ,cmap='gray')
plt.title("Comparison between original and DCT compressed images" )
plt.show()
```

Comparison between original and DCT compressed images



6 Implementation Pipeline

What is done :

- Theoretical research and analyzing of SVD technique
- Providing Python code for SVD implementation
- Theoretical research and analyzing of DCT technique
- Providing Python code for DCT

To be done :

- Testing on various image sets
- Algorithms comparison
- Visualization of comparison results

References

[1] Gilbert Strang, *Introduction to Linear Algebra*, 5th Edition, Wellesley–Cambridge Press, 2016.

[2] Gilbert Strang, *Linear Algebra and Its Applications*, 4th Edition, Cengage Learning, 2018.

[3] Ee-Leng Tan, Woon-Seng Gan,*Perceptual Image Coding with Discrete Cosine Transform*, Springer, 2015.