Projeto

August 22, 2021

Compressão de Imagens em JPEG e Grayscale Através da Segmentação e da Transformada Discreta do Cosseno de Fourier

August 22, 2021

1 Compressão de Imagens

O termo Compressão se refere ao processo de reduzir a quantidade de dados requerida para representar uma dada quantidade de informação [1]. Uma das características mais comuns de uma imagem é que os pixels vizinhos tem similaridadedes e portanto contém informação redundante[1].

Através de diversas técnicas podemos encontrar e remover informações desnecessárias de uma imagem, diminundo o espaço ocupado em disco por esta mas mantendo uma qualidade bem próxima da original, com diferenças na maioria das vezes impercepitiveis. A transformada discreta de Fourier e a segmentação podem ser usadas como ferramentas para se atingir este objetivo.

Há basicamente dois tipos de técnicas de compressão de imagens: Com Perda e Sem Perda. As técnicas sem perda são usadas quando se quer temporariamente reduzir a informação, Enquanto as técnica com perda são usadas quando se quer permanente reduzir o tamanho de uma imagem. [10]

1.1 O Formato JPEG

O termo JPEG é um acrônimo para Joint Photographic Experts Group, um comitê que tem uma longa tradição na criação de padrões de codificação de imagens sem movimento [10] .

JPEG é um formato padrão de compressão para imagens digitais, O padrão JPEG apresenta quatro modos de compressão sendo um deles o JPEG com perdas (Lossy JPEG) também conhecido como JPEG-DCT pois utiliza como ferramenta a Transformada do Cosseno Discreta (Discrete Cosine Transform) para comprimir a imagem [4][8][3].

1.2 A Transformada do Cosseno Discreta de Fourier (DCT)

A DCT é uma das transformadas discretas de Fourier, ela transforma um sinal do domínio do espaço para o domínio da frequência. Ela ajuda a separar a imagem em partes (faixas expectrais) de diferente importâncias em relação a qualidade da imagem[11][12]. A fórmula da DCT é dada por:

$$X[k] = \alpha[k] \sum_{n=0}^{N-1} x[n] cos\left(\frac{k\pi(2n+1)}{2N}\right)$$

e a fórmula da DCT inversa é dada por:

$$x[n] = \sum_{k=0}^{N-1} \alpha[k] X[k] cos\left(\frac{k\pi(2n+1)}{2N}\right)$$

onde:

$$\alpha[k] = \frac{1}{\sqrt{N}}\ se\ k = 0$$
$$\alpha[k] = \sqrt{\frac{2}{N}}\ caso\ contrrio$$

Como a imagem é um objeto bidimensional é necessário usar a transformada em duas dimensões do cosseno que é dada por:

$$F(u,v) = C(u)C(v) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f(m,n) cos\left(\frac{u\pi(2m+1)}{2N}\right) cos\left(\frac{v\pi(2n+1)}{2N}\right)$$

E sua transformada inversa por:

$$F(m,n) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) cos\left(\frac{u\pi(2m+1)}{2N}\right) cos\left(\frac{v\pi(2n+1)}{2N}\right)$$

onde:

$$C(\gamma) = \frac{1}{\sqrt{N}}\ se\ k = 0$$
$$C(\gamma) = \sqrt{\frac{2}{N}}\ caso\ contrrio$$

[14][15]

1.3 Usando DCT na Compressão de Imagens JPEG em Grayscale

No projeto a compressão de imagens foi definida de acordo com os seguintes passos:

- Passo 1: Aplica-se o DCT bidimensional em segmentos 8 por 8 da imagem
- Passo 2: Descobre-se o valor da média da magnitude coeficientes de cada segmento 8 por 8 e se iguala a zero todos os coeficientes que tiverem o módulo menor que a média (este método se chama thresholding).
- Passo 3: Aplica-se o DCT inverso bidimensional em cada segmento 8 por 8 da imagem no domínio da frequência para retorná-la ao domínio do espaço

O DCT apresenta um melhor desempenho que a transformada Discreta de Fourier pois fornece um maior acúmulo dos coeficientes mais significativos da imagem, proporcionando uma melhor capacidade de compressão [13], porém como é um método de compressão com perda ele não permite a recuperação da imagem original (sem nenhum coeficiente igualado a zero).

O projeto trabalha com imagens em Grayscale em vez de RGB ou BGR porque imagens coloridas precisam de três dimensões para ser representadas, pois cada pixel da imagem é representado por

três valores, que representam as intensidades das cores vermelha (Red), verde (Green) e azul (Blue). Enquanto as imagens em Grayscale apresentam apenas duas dimensões, pois é necessário apenas a intensidade de cinza para representar cada pixel. É possível trabalhar com a compressão de imagens coloridas com a transformada de fourier, mas o método teria que sofrer uma forte adaptação.

2 Setup Inicial

2.1 Bibliotecas Usadas no Projeto

[127]:
```
import numpy as np
import matplotlib.pyplot as plt
#Funções que implementam a transformada discreta do cosseno de fourier
from scipy.fftpack import dct, idct
import cv2 # Biblioteca usada para importar imagens
from math import floor,log10
from IPython.display import Latex
```

2.2 Funções Auxiliares

O módulo scipy.fftpack fornece funções apenas para transformadas de uma dimensão, a função abaixo converte uma função de transformada discreta de fourier unidimensional para bidimensional.

[128]:
```
def transform2D(img,function):
    return np.transpose(function(np.transpose(function(img,norm =␣
 ↪"ortho")),norm = "ortho"))
```

Função responsável por segmentar uma imagem e aplicar uma função sobre os segmentos de tamanho **size X size** da imagem.

[129]:
```
def segmentation(img,size,function=transform2D,function2 =dct):
    copy = np.zeros(img.shape)

    for i in range(copy.shape[0]//size):
        for j in range(copy.shape[1]//size):
            copy[i*size:(i+1)*size,j*size:(j+1)*size] = function(img[i*size:
 ↪(i+1)*size,j*size:(j+1)*size],function2)

    return copy
```

A função abaixo é uma função auxiliar que é utilizada para o melhor entendimento da função **thresholdingPC**. Ela retorna o index que representa **n%** da matriz que representa a imagem quando esta é convertida para um array unidimensional.

[130]:
```
def percent(img,n):
    return floor((img.shape[0]*img.shape[1])*(n/100))
```

Função responsável por filtrar através do método thresholding de segmentação, a imagem no domínio da frequência. Esta função iguala a zero os valores de frequência que estão abaixo de

um certo limiar, que é cálculado através do enésimo valor de maior magnitude da matriz que representa a imagem.

[131]:
```
def thresholdingPC(img,pc):
    imgCopy = img.copy()
    sortedCts = np.sort(abs(imgCopy.ravel()))
    threshold = sortedCts[-percent(imgCopy,pc)]
    imgCopy[abs(imgCopy) < threshold] = 0;
    return imgCopy
```

Função auxiliar para a função **thresholdingMean**. Ela recebe um segmento de uma imagem **img** que já passou pela tranformada de fourier, e baseado no critério dado pela função **function** calcula o limiar (threshold) para filtrar os coeficientes redundantes e igualá-los a zero.

[132]:
```
def filterImg(img,function):
    imgCopy = img.copy();
    threshold = function(np.abs(imgCopy))
    #print(threshold)
    imgCopy[abs(imgCopy) < threshold ] = 0
    return imgCopy
```

Função que filtra os segmentos da imagem, aplicando a média dos valores da matriz de tamanho **size X size** como método estatistico para cálcular o limiar (threshold)

[133]:
```
def thresholdingMean(img,size):
    imgCopy = img.copy()
    imgCopy = segmentation(imgCopy,size,filterImg,np.mean)
    return imgCopy;
```

Função que é um compilado das funções acima. Ela basicamente recebe uma imagem JPEG e algumas informações adicionais e executa a compressão desta. O passo a passo de como ela funciona será explicado com mais detalhes futuramente.

[134]:
```
def compress(img, thresholding = thresholdingMean,argThres = 8, seg = 8, save =␣
↪False, path = "imgCMP.jpg"):
    compImg = img.copy()
    compImg = segmentation(compImg,seg)
    compImg = thresholding(compImg,argThres)
    compImg = segmentation(compImg,seg,function2 = idct)
    if(save):
        cv2.imwrite(path,compImg)
    return compImg
```

Função que cálcula a Raiz do Erro Quadrático Médio (Root Mean Square Error).

[135]:
```
def RMSE(original, compressed):
    result = 0
```

```
    M = original.shape[0]
    N = original.shape[1]

    for i in range(M):
        for j in range(N):
            result += (compressed[i][j] - original[i][j])**2
    #print(result)
    return (result/(M*N))**(1/2)
```

Função que Cálcula a Relação Sinal-Ruído de Pico (Peak Signal Noise Ratio).

[136]:
```
def PSNR(original,compressed):
    result = 20*log10(255/RMSE(original,compressed))
    return result
```

3 Compressão de Imagens Na Prática

Primeiro é necessário importar imagem e salvar em uma variável em python.

[137]:
```
path = "../BANCO-DE-IMAGENS/img6.jpg"
imgBGR = cv2.imread(path)
```

[138]:
```
#Imagem é convertida de BGR para RGB para poder  ser mostrada
imgRGB = cv2.cvtColor(imgBGR,cv2.COLOR_BGR2RGB)
plt.imshow(imgRGB)
cv2.imwrite("generated-files/imgBGR.jpg",imgBGR)
```

[138]: True

Converte-se a imagem para Grayscale Para que o método de compressão funcione.

[139]:
```
imgGRAY = cv2.cvtColor(imgBGR,cv2.COLOR_BGR2GRAY)
plt.imshow(imgGRAY,cmap = "gray")
```

[139]: <matplotlib.image.AxesImage at 0x7ff5a9c67700>

A imagem é salva para comparações futuras.

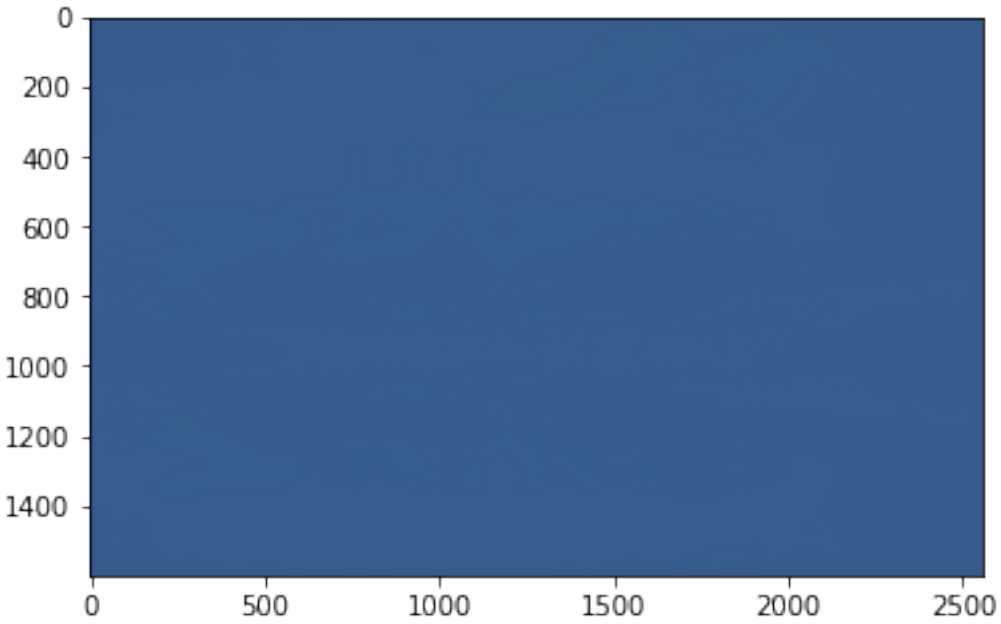
[140]:
```
cv2.imwrite("generated-files/imgGRAY.jpg",imgGRAY)
```

[140]:
```
True
```

Aplica-se a **Transformada Discreta do Cosseno de Fourier** em segmentos 8 por 8 da imagem. (Tamanho do segmento escolhido com base no padrão JPEG)

[141]:
```
imgCMP = segmentation(imgGRAY,8,transform2D,dct)
plt.imshow(imgCMP)
```

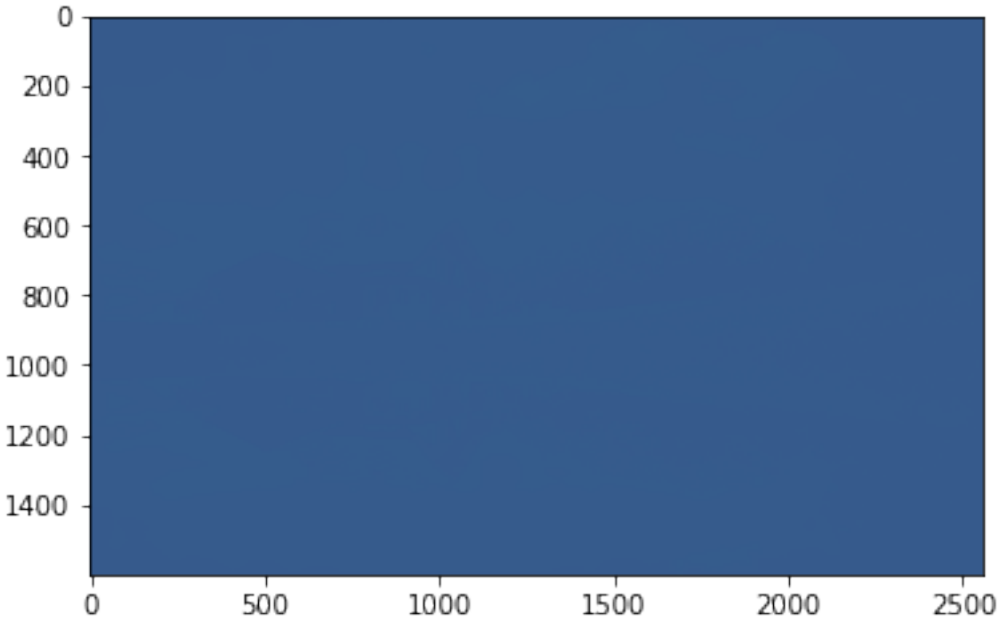
[141]:
```
<matplotlib.image.AxesImage at 0x7ff5a9c50d60>
```



Aplica-se um filtro na imagem para eliminar os coeficientes de menor relevância, usando a **média** como método estatístico para cálcular o limiar.

[142]:
```
imgCMP = thresholdingMean(imgCMP,8)
plt.imshow(imgCMP)
```

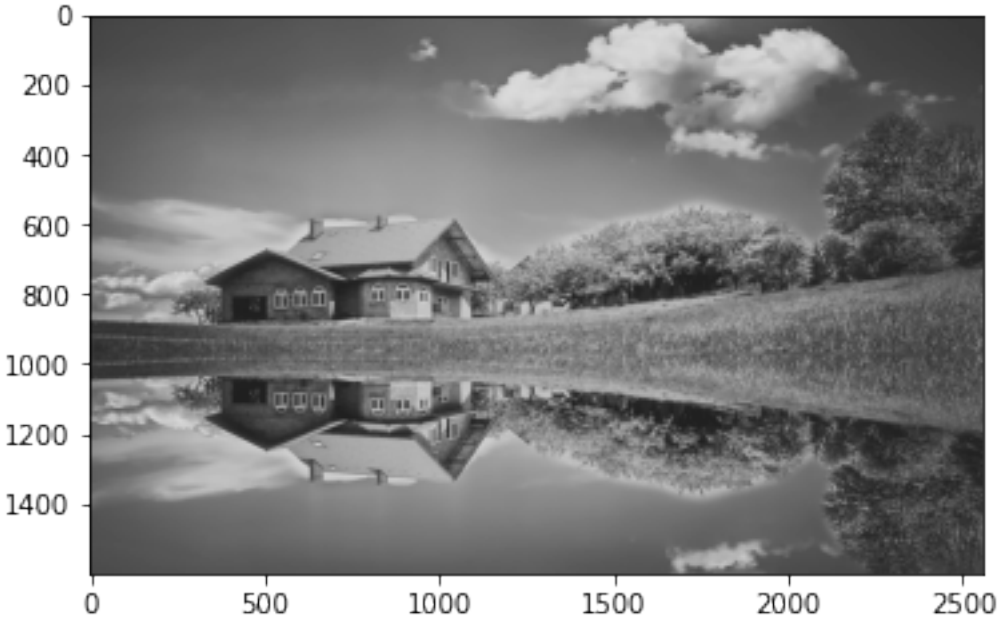
[142]:
```
<matplotlib.image.AxesImage at 0x7ff5a9bb8c40>
```



Aplica-se a **Tranformada Discreta do Coseno de Fourier Inversa** sobre segmentos de tamanho 8 por 8 da imagem para se recuperar a imagem original

[143]:
```
imgCMP = segmentation(imgCMP,8,function2 = idct)
plt.imshow(imgCMP, cmap = "gray")
```

[143]:
```
<matplotlib.image.AxesImage at 0x7ff5a82efeb0>
```



Salva-se a imagem comprimida para comparações futuras.

[144]:
```
cv2.imwrite('generated-files/imgCMP.jpg',imgCMP)
```

[144]:
```
True
```

O Mesmo resultado poderia ser obtido através da função **compress** que unifica todos os passos acima

[145]:
```
test = compress(imgGRAY,thresholding = thresholdingMean, argThres = 8,save =␣
↪True)
plt.imshow(test, cmap = "gray")
```

[145]:
```
<matplotlib.image.AxesImage at 0x7ff5a9d17550>
```

3.1 Comparações entre a Imagem Comprimida e a Imagem Original

A imagem original (em Grayscale) ocupa 1.1 MB de espaço em disco (1,131,392 bytes). Enquanto a imagem comprimida ocupa 698.0 kB (697,977 bytes).

A taxa de compressão de uma imagem pode ser obitida através da seguinte fórmula:

$$C = \frac{n_c}{n_o}$$

onde:

- C = Taxa de Compressão da imagem
- n_c = número de bytes (ou bits) da imagem comprimida
- n_o = número de bytes (ou bits) da imagem original

[1]

Para o caso acima a taxa de compressão será dada por:

$$C = \frac{697,977 bytes}{1,131,392 bytes} = 0.00532273$$

A taxa de compressão da imagem foi de um pouco mais de 5 por cento, o que é um valor pequeno para apenas uma imagem, mas caso precissassemos armazenar milhares de imagens essa diferença se acumularia fazendo o processo valer a pena.

A diferença de qualidade entres as duas imagens é quase imperceptível a olho nu, porém para melhor análise será cálculado o PSNR das duas imagens.

PSNR (Peak Signal to Noise Ratio) se refere a razão entre a potência máxima de um sinal e a potência do ruído que afeta fidelidade da representação de um sinal. Ele é um dos métodos mais comumente usado como medida de qualidade da imagem resconstruída [1]. Quanto maior for o PSNR melhor a imagem foi reconstruida.[16]

A fórmula de Peak Signal to Noise Ratio é dada por:

$$PSNR = 20 log_{10} \left(\frac{MAX_f}{RMSE} \right)$$

onde:

- MAX_f: O valor máximo que o sinal pode atingir na imagem original, que no nosso caso será igual a 255.
- $RMSE = \sqrt{\frac{1}{M*N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (g(i,j) - f(i,j))^2}$

sendo que:

- M e N são as dimensões da imagem
- $g(i,j)$ é a imagem original
- $f(i,j)$ é a imagem reconstruida

[16]

O PSNR das imagens é dado por:

[146]:
```
print(PSNR(imgGRAY, imgCMP), "db")
```

```
33.36912844685023 db
```

33.36 db de PSNR é um valor típico para imagens comprimidas.

References

[1] M. Kanaka Reddy , V. V. Haragopal and S. A. Jyothi Rani "Statistical Image Compression using Fast Fourier Coefficients" , Dec. 2016

[2] J.Feydy "Part 6: Fourier analysis and JPEG compression", Feb.2019. [Online]. Available: http://www.jeanfeydy.com/Teaching/MasterClass_Radiologie/Part%206%20-%20JPEG%20compression.html

[3] S. W. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing",[Online]. Available: http://www.dspguide.com/ch27/6.htm

[4] J. F. Neto, "Compressão Sem Perdas de Imagens Digitais", [Online]. Available: http://www.dpi.inpe.br/~carlos/Academicos/Cursos/Pdi/SemPerdas.htm

[5] O. Hampiholi, "Image Compression — DCT Method", Mar.21. [Online]. Available: https://towardsdatascience.com/image-compression-dct-method-f2bb79419587

[6] Johnsy,A. "2-D Discrete Cosine Transform", [Online]. Available: https://www.imageeprocessing.com/2013/03/2-d-discrete-cosine-transform.html

[7] S.Thayammal and D.Selvathi "A Review On Segmentation Based Image Compression Techniques",2013

[8] L. Wake, "What is a JPEG file?", Apr.2019, [Online]. Available: https://digitalcommunications.wp.st-andrews.ac.uk/2019/04/08/what-is-a-jpeg-file/

[9] Joint Photographic Experts Group, "ABOUT JPEG",[Online]. Available: https://jpeg.org/about.html

[10] Krita Manual, "Lossy and Lossless Image Compression", [Online]. Available: https://docs.krita.org/en/general_concepts/file_format/lossy_lossless.html

[11] E. Roberts, "The Discrete Cosine Transform (DCT)", [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/dct.htm

[12] D. Marshal, "The Discrete Cosine Transform (DCT)", Apr.2001. [Online]. Available: https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html

[13] S. Cárceres, "Processamento de Imagem". [Online]. Available: http://sheilacaceres.com/courses/pi/aula9/PI-9-Compressao.pdf

[14] A. Conci, "Transformada de Discreta de Cosenos DCT"

[15] W. R. Schwartz and H. Pedrini, "Aspectos Teóricos das Transformadas de Imagens"

[16] National Instruments, "Peak Signal-to-Noise Ratio as an Image Quality Metric". Dec.2020. [Online]. Available: https://www.ni.com/pt-br/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html