camila-atividade-ass195173ncrona

August 10, 2023

1 Tópico I - Filtro Blur

[132]:
```python
# Importações
from skimage import data
import numpy as np
import cv2
import matplotlib.pylab as pylab
```

[133]:
```python
# Carregando a imagem
imagem = data.chelsea()
```

1.1 Enunciado:

A) Implemente o filtro Blur e aplique este mesmo filtro na imagem chelsea, empregue o kernel 3x3. Plote a imagem original e a imagem resultante.

Resultado Esperado:

[134]:
```python
# Filtragem
filtragem = cv2.blur(imagem,ksize=(3,3))

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```

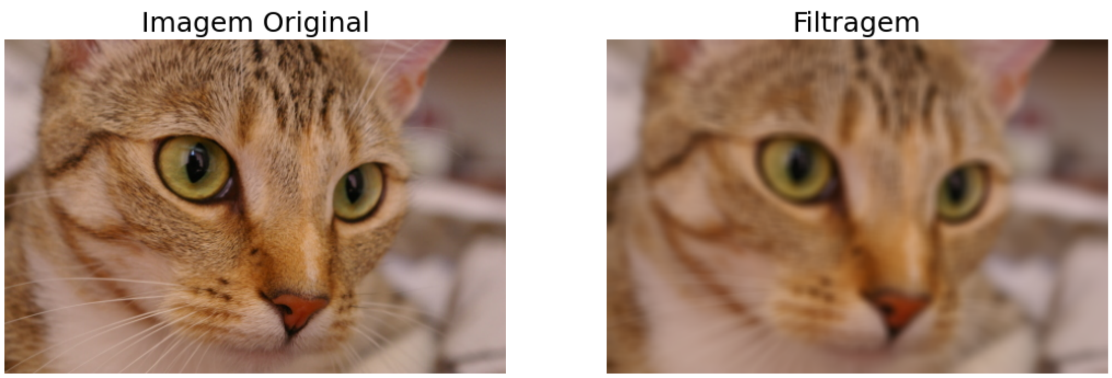
1.2 Enunciado:

B) Implemente o filtro Blur e aplique o filtro Blur na imagem chelsea. Porém, empregue o kernel 9x9. Plote a imagem original e a imagem resultante.

Resultado Esperado:

[135]:

```python
# Filtragem
filtragem = cv2.blur(imagem,ksize=(9,9))

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```



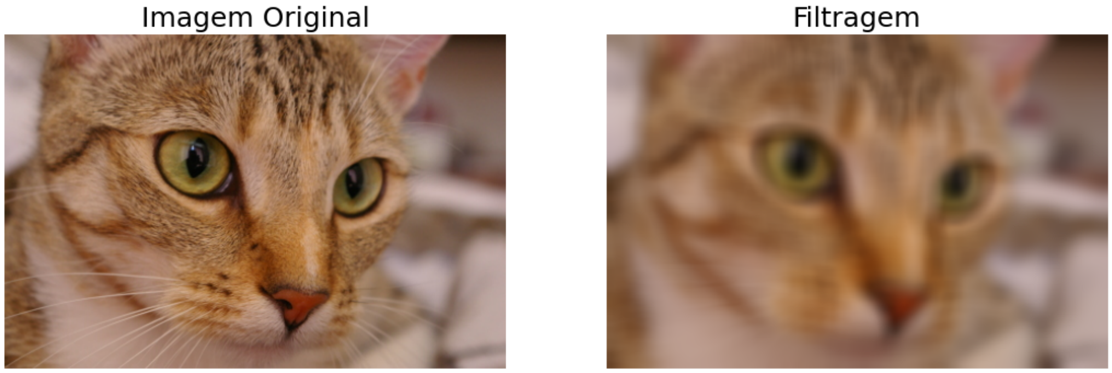
1.3 Enunciado:

C) Repita o procedimento do item B). No entanto, empregue o kernel 17x17. Plote a imagem original e a imagem resultante.

Resultado Esperado:

[136]:

```
# Filtragem
filtragem = cv2.blur(imagem,ksize=(17,17))

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```



D) Compare os resultados obtidos nos itens A), B) e C) e discuta o efeito que um kernel de dimensionalidade maior, para o filtro Blur, promove em uma imagem qualquer.

Completar a célula de texto abaixo com a sua resposta:

A filtragem blur é utilizada para reduzir o ruído da imagem, tornando os detalhes da imagem menos nítidos. A dimensão do kernel utilizado no filtro afeta a quantidade de suavização aplicada à imagem.

Kernel 3x3: *A imagem tem traços mais suaves e com menos ruídos, no entanto, ainda preservando detalhes significativos da imagem.*

Kernel 9x9: *A imagem perderá mais detalhes finos e a textura será mais uniforme. O ruído também será reduzido de maneira mais eficaz em relação a imagem que foi ap-licada o kernel 3x3, resultando em uma imagem mais limpa, mas com menos nitidez.*

Kernel 17x17: *Com os detalhes drasticamente reduzidos e com uma aparência mais abstrata e desfocada. O kernel 17x17 causa um efeito de suavização significativamente maior e com uma textura mais homegenizada.*

2 Tópico II - Filtro Bilateral

[137]:
```
# Importações
from skimage import data
import numpy as np
import cv2
import matplotlib.pylab as pylab
```

[138]:
```
# Carregando a imagem
imagem = data.chelsea()
```

[139]:
```
# d  - Diâmetro da vizinhança do pixel a ser considerado durante o desfoque

# sigmaColor:
```

```
# Filtro sigma no espaço de cores. Um valor maior do parâmetro significa
# que as cores mais distantes dentro da vizinhança do pixel
# serão misturadas.

# sigmaSpace:

# Filtro sigma no espaço de coordenadas.
# Um valor maior do parâmetro significa que pixels
# mais distantes influenciarão uns aos outros
```

2.1 Enunciado:

A) Implemente o filtro Bilateral e aplique este filtro na imagem chelsea, empregue d=5, sigmaColor=0, sigmaSpace=0. Plote a imagem original e a imagem resultante.

Resultado Esperado:

[140]:

```python
# Filtragem
filtragem = cv2.bilateralFilter(imagem,d=5,sigmaColor=0,sigmaSpace=0)

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```

2.2 Enunciado:

B) Repita o procedimento do item A), no entanto, empregue d=10.

[141]:

```
# Filtragem
filtragem = cv2.bilateralFilter(imagem,d=10,sigmaColor=0,sigmaSpace=0)

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```

Ainda neste mesmo item. Ao alterar d=5 para d=10, houve modificação na imagem após a filtragem?

Completar a célula de texto abaixo com a sua resposta:

A segunda imagem, onde d=10, a suavização é maior do que na primeira imagem, onde d=5. Assim, concluímos que a segunda imagem terá uma aparência mais homogênia e suave do que a primeira imagem.

2.3 Enunciado:

C) Repita o procedimento do item A), no entanto, empregue d=3, sigmaColor=150, sigmaSpace=150. Plote a imagem original e a imagem resultante.

[142]:
```
# Filtragem
filtragem = cv2.bilateralFilter(imagem,d=3,sigmaColor=150,sigmaSpace=150)

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```

Ao empregar d=3, sigamColor=150 e sigmaSpace=150, houve modificação na imagem após a filtragem?

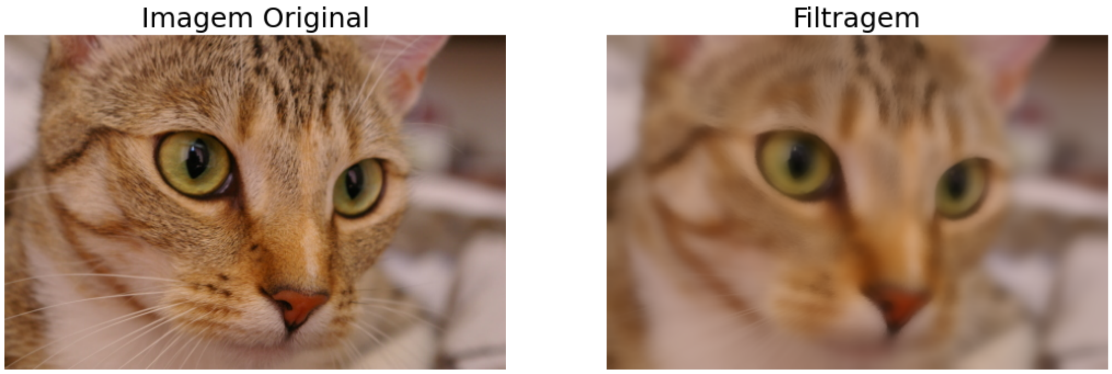
Completar a célula de texto abaixo com a sua resposta:

Na imagem acima está mais evidente que ela foi suavizada, tanto nas cores quanto nos traços. A imagem está mais uniforme, com menos contrastes em relação aos traços.

2.4 Enunciado:

D) Repita o procedimento do item A), no entanto, empregue d=17, sigmaColor=200, sigmaSpace=200. Plote a imagem original e a imagem resultante.

[143]:
```
# Filtragem
filtragem = cv2.bilateralFilter(imagem,d=17,sigmaColor=200,sigmaSpace=200)

# Plotando o resultado
fig, axes = pylab.subplots(ncols=2, sharex=True, sharey=True, figsize=(13,13))

# Configurando o plot da imagem original
axes[0].set_title('Imagem Original', size=18)
axes[0].imshow(imagem)

# Configurando o plot da imagem filtrada
axes[1].set_title('Filtragem', size=18)
axes[1].imshow(filtragem)

for ax in axes:
    ax.axis('off')

pylab.show()
```



Neste caso, houve modificação na imagem após a filtragem?

Completar a célula de texto abaixo com a sua resposta:

A imagem acima está bem mais suavizada e os seus detalhes estão reduzidos ao ponto do desfoque. Além disso, as cores apresentam uma tonalidade mais uniforme quando comparada a imagem original.

3 Tópico III: Transformada de Fourier

Temos a sequência:

$$x[n] = \cos\left(2\pi rn \ / \ N\right), \quad \text{com } 0 \leq n \leq N-1, \quad \text{onde } 0 \leq r \leq N-1$$

A DFT de $x[n]$ é:

$$X[k] = \begin{cases} N/2, & \text{para } k = r \\ N/2, & \text{para } k = N-r \\ 0, & \text{c.c.} \end{cases}$$

Plote a parte real de $X[k]$, para N=17 e r=3.

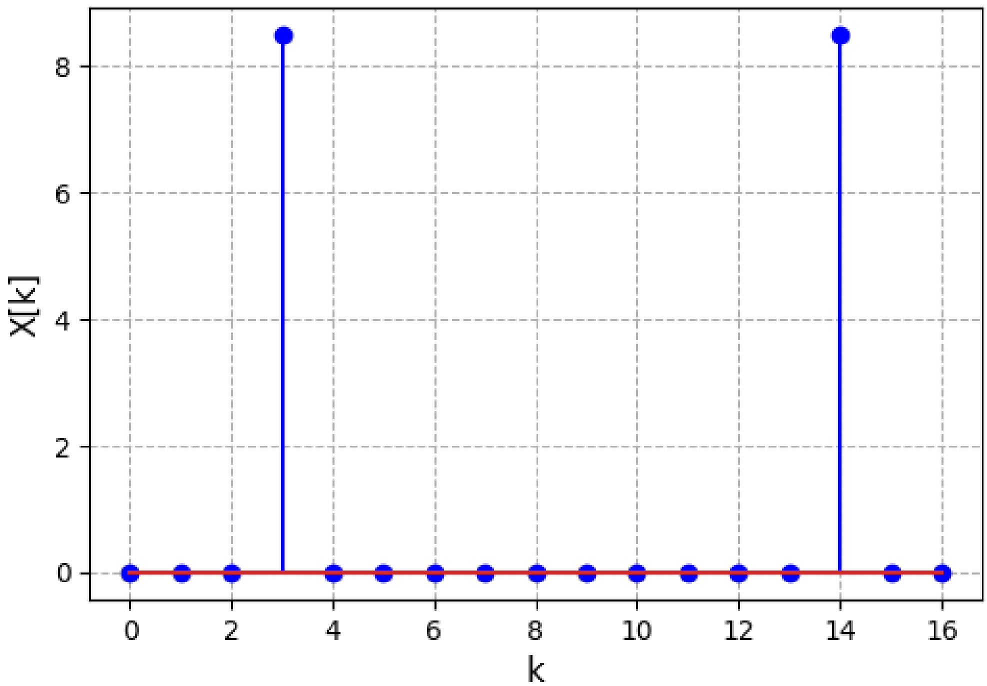
Resultado esperado:

[144]:
```
# Completar esta célula com o seu código

# Importações
import numpy as np
import matplotlib.pyplot as plt

N = 17  # Número de pontos da sequência
r = 3  # Frequência da função cosseno
n = np.arange(N)  # Vetor de índices de tempo discreto

x_n = np.cos(2*np.pi*r*n/N)  # Sequência de tempo discreto x[n] = cos(2 rn/N)

X_k = np.fft.fft(x_n)  # Cálculo da DFT da sequência x[n]

print('X[k]:')
print(X_k)
```

```
X[k]:
[-2.88657986e-15+0.00000000e+00j -1.11022302e-15-3.67481573e-16j
 -8.88178420e-16-1.04460620e-15j  8.50000000e+00-5.13415629e-15j
  2.66453526e-15+1.02567764e-15j -2.22044605e-16-4.70676026e-16j
  2.33146835e-15-2.37941579e-16j -8.88178420e-16+1.29141438e-15j
  3.33066907e-16-5.91432970e-16j  3.33066907e-16+5.91432970e-16j
 -8.88178420e-16-1.29141438e-15j  2.33146835e-15+2.37941579e-16j
 -2.22044605e-16+4.70676026e-16j  2.66453526e-15-1.02567764e-15j
```

```
 8.50000000e+00+5.13415629e-15j -8.88178420e-16+1.04460620e-15j
-1.11022302e-15+3.67481573e-16j]
```

[145]:
```
plt.figure(figsize=(6,4))
plt.stem(X_k.real, 'b');
plt.xlabel('k', fontsize=12)
plt.ylabel('X[k]', fontsize=12)
plt.grid(True, linestyle='dashed')
plt.show();
```



Em relação ao uso da Transformada de Fourier no campo da visão computacional, é importante entender que através dela que é possível a análise de padrões e estruturas em imagens por meio da decomposição em componentes de frequência. Ela possibilita a filtragem para remoção de ruído e detecção de bordas, compactação de imagens, correlação para reconhecimento de padrões e correspondência, aplicação de transformações geométricas, e são essenciais para o processamento de vídeo, capacitando os computadores a compreender e manipular informações visuais com eficácia.