Processamento Digital de Imagens - Filtragem *High Boost*

ALUNOS: SANDRO HENRIQUE ULIANA CATABRIGA - RA983917

VINÍCIUS DA COSTA REGATIERI - RA104016

Filtragem espacial

O termo *filtragem* está diretamente relacionado com o domínio da frequência, que veremos a seguir, e se refere a aceitar (passar) ou rejeitar certos componentes de frequência. Um filtro espacial consiste em:

1. Uma vizinhança (normalmente definida por uma **máscara**)
2. Uma operação pré definida realizada sobre os pixels da imagem incluídos na vizinhança.

A filtragem cria um novo pixel com coordenadas iguais às coordenadas do centro da vizinhança, e cujo valor é o resultado da operação de filtragem. Se a operação realizada sobre os pixels da imagem for linear, o filtro é chamado de *filtro linear*, caso contrário, o filtro é *não linear*.

Para realizar uma filtragem espacial linear, podemos utilizar tanto o conceito de **correlação** quanto de **convolução.** A correlação é um filtro linear que consiste em mover uma máscara pela imagem e calcular a soma dos produtos ponto-a-ponto entre a imagem e a máscara. A convolução segue o mesmo conceito, mas com a máscara rotacionada 180°.

Filtro gaussiano

Em algumas aplicações, temos uma função contínua de duas variáveis que é utilizada para se obter uma máscara de filtragem espacial com base nessa função. Uma máscara muito utilizada é a **máscara gaussiana**, que é definida pela função gaussiana de duas variáveis, sendo essa: $h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$, em que σ é o desvio padrão.

Filtros espaciais de suavização

Filtros espaciais de suavização são utilizados para borramento da imagem e redução de ruído. A saída de um filtro espacial linear de suavização é simplesmente a média dos pixels contidos na vizinhança da máscara de filtragem. Esses filtros são geralmente chamados de **filtros passa-baixa**.

A ideia por trás dos filtros de suavização é bem simples. Ao substituir o valor de cada pixel de uma imagem pela **média** dos níveis de intensidade da vizinhança definida pela máscara, o processo resulta em uma imagem com perda da nitidez. Podemos destacar como principal efeito colateral da utilização de um filtro de suavização o borramento das bordas de uma imagem. Além disso, quanto maior a dimensão da máscara, maior o borramento.

Filtros espaciais de aguçamento

O principal objetivo do aguçamento é salientar transições de intensidade para o aumento da nitidez de uma imagem.

Filtragem *High Boost*

Um processo muito comum para aumento de nitidez em imagens é a filtragem *High Boost* utilizando máscara de nitidez, que consiste nos seguintes passos:

1. Borrar a imagem original.
2. Subtrair a imagem borrada da original (a diferença resultante é chamada de **máscara**) - não confundir com a máscara da convolução.
3. Adicionar a máscara à imagem original.

Matematicamente, podemos denotar a máscara por:

$$g_{mascara}(x, y) = f(x, y) - \bar{f}(x, y)$$

Em que $\bar{f}(x, y)$ é a imagem borrada. Após isso, adicionamos uma porção ponderada da máscara (utilizando um valor *k*) de volta à imagem original:

$$g(x, y) = f(x, y) + k * g_{mascara}(x, y)$$

O valor de *k* deve ser maior ou igual a zero, sendo que com $k = 1$, temos a máscara de nitidez, e quando $k > 1$, temos a filtragem *High Boost*. Na Figura 1 podemos observar o passo a passo do processo.

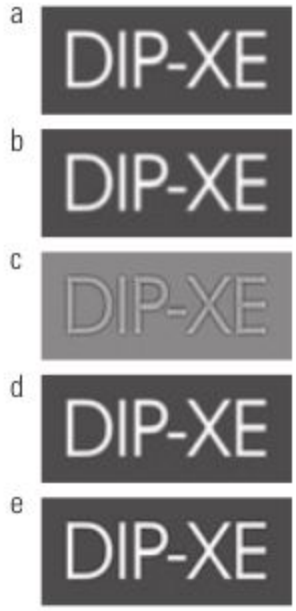

FIgura 1 (a) Imagem original. (b) Resultado do borramento utilizando filtro gaussiano. (c) Máscara de nitidez. (d) Resultado da utilização de uma máscara de nitidez. (e) Resultado da filtragem *High Boost*.

Algoritmo desenvolvido

O objetivo principal desta parte do trabalho era implementar e testar o método de filtragem conhecido como *High Boost*, utilizado para o aumento de nitidez de imagens digitais, utilizando a filtragem passa-baixa no domínio espacial. No Algoritmo 1 podemos ver a solução desenvolvida.

```
import cv2
from skimage import io
import matplotlib.pylab as pylab

```

```
# carrega a imagem
 image = cv2.imread(input("Digite o caminho para a imagem de
entrada: "))[:, :, 0]

# lê a dimensão do kernel
n = int(input("Digite a dimensão da máscara: "))

# lê o valor de k
k = int(input("Digite o valor de k: "))

# realiza borramento
gauss = cv2.GaussianBlur(image, (n, n), 3)

# obtem a máscara (arestas para aguçamento)
mask = cv2.subtract(image, gauss)
mask = k * mask

# obtem imagem final
final_image = cv2.add(image, mask)
 cv2.imwrite(input("Digite o caminho para a imagem de saída: "),
final_image)
print("Imagem final salva!")
```

Algoritmo 1 *High Boost* utilizando passa-baixa gaussiano no domínio espacial

Em um primeiro momento, nota-se que a dimensão da máscara de convolução e o valor valor de ponderação k são variáveis definidas pelo usuário. Nossa primeira escolha foi trabalhar apenas com funções da biblioteca *OpenCV*, para que não houvesse qualquer problema na manipulação dos dados e possíveis erros em operações entre eles. Explicaremos, então, os passos de nossa solução:

1. **Filtro passa-baixa gaussiano:** nosso primeiro passo na solução é realizar o borramento da imagem original, utilizando um filtro passa-baixa. O filtro gaussiano é um filtro linear muito utilizado para borramento e redução de ruído, por isso optamos por ele. Além disso, esse filtro foi escolhido para que uma comparação fosse feita no domínio da frequência, já que a transformada de um filtro gaussiano continua sendo um filtro gaussiano. A biblioteca *OpenCV* nos oferece uma função para se aplicar um filtro gaussiano em uma imagem, necessitando apenas especificar a dimensão da máscara de convolução e o valor de σ, o qual tomamos como sendo 3.
2. **Obtenção da máscara:** de forma simples, utilizamos a função *cv2.subtract* para realizarmos a subtração ponto-a-ponto da imagem original com a imagem borrada. Essa função não apresenta seus resultados de subtração em módulo ou com "corte" em zero.
3. **Obtenção da imagem final:** ao final, adicionamos a máscara de nitidez, com ou sem *High Boost*, na imagem original utilizando a função *cv2.add*. O resultado é uma imagem aguçada nas bordas ou arestas, com aumento de nitidez.

Optamos, ainda, por exibir alguns passos intermediários ao usuário, como apresentamos no Algoritmo 2.

```
############### Mostra todos os passos da convolução #############
pylab.figure(figsize=(30, 25))
pylab.gray()

pylab.subplot(2, 3, 1), pylab.imshow(image), pylab.title(
  'Imagem original', size=10), pylab.axis('off')

pylab.subplot(2, 3, 3), pylab.imshow(gauss)
pylab.title('Imagem após convolução', size=10), pylab.axis('off')

pylab.subplot(2, 3, 5), pylab.imshow(mask)
pylab.title('Máscara', size=10), pylab.axis('off')
```

```
pylab.subplots_adjust(wspace=0.2, hspace=1)
pylab.show()
#########################################################################
```

Algoritmo 2 Apresentação dos passos intermediários ao usuário

Na Figura 2 podemos ver uma comparação entre uma imagem original e seu resultado após a utilização de uma máscara de nitidez ($k = 1$) de tamanho $3x3$. Percebe-se, de forma suave, o aumento de nitidez em várias arestas da lua.

Figura 2 Figura original (esquerda) e seu aguçamento com máscara de nitidez (esquerda) $3x3$

Na Figura 3 fazemos uma comparação entre máscara de nitidez e *High Boost*, com uma máscara de convolução de dimensão $3x3$. Notamos que realmente há um considerável aumento na nitidez quando utilizamos o *High Boost*, pois a adição das arestas de aguçamento com pesos maiores na imagem torna os contornos mais nítidos, ou até mesmo saturados, próximos ou igual ao branco.

Figura 3 Figura com máscara de nitidez (esquerda) e figura com *High Boost,* com $k = 3$ (direita), ambas com máscara $3x3$

Podemos ver, ainda, na Figura 4, uma comparação entre uma máscara de convolução $3x3$ com $k = 3$, e uma máscara de convolução $11x11.$ Notamos que o realce foi parecido, mas a figura da direita realçou mais pontos, ou arestas, enquanto que nos pontos em que ambas realizaram o realce, a intensidade foi muito parecida.

Figura 4 Figura figura com *High Boost,* com $k = 3$ e máscara $3x3$, e figura com máscara de nitidez e máscara $11x11$

Transformada de Fourier

De um ponto de vista matemático, uma transformada mapeia uma função f de seu domínio original X em um outro domínio Y, pois a manipulação de algumas propriedades de f pode ser mais fácil de se executar no novo domínio Y que no domínio original X. O mapeamento de f de volta ao seu domínio original é dado pela transformada inversa.

De acordo com Fourier, qualquer função periódica com o período, pode ser expressa como a soma dos senos e cossenos multiplicada por coeficientes apropriados. Essa soma, conhecida como série de Fourier.

Quando as funções não são periódicas mas suas integrais são finitas, elas podem ser expressas por uma integral de senos e cossenos multiplicada por um função de ponderação. Essa é a transformada de Fourier, que decompõe uma função temporal (um sinal) em frequências. De forma geral, essa transformada é chamada de representação do domínio da frequência do sinal original.

Para imagens digitais, utilizamos a transformada discreta de Fourier (DFT). Descrevemos a DFT, para uma sequência discreta de sinais X de N termos, como sendo $\overline{X}$:

$$\overline{X}_k = \sum_{m=0}^{N-1} x_m W^{mk}$$
$$k = 0, ..., N-1$$
$$W = e^{-2j\pi/N}$$

Filtragem no Domínio da Frequência

A filtragem no domínio da frequência consiste em modificar a transformada de Fourier de uma imagem, realizar um processamento, e depois calcular a transformada inversa para obter o resultado processado no domínio espacial. Assim, dada uma imagem digital, $f(x, y)$ de tamanho $M \times N$, a equação básica de filtragem na qual estamos interessados tem a seguinte forma:

$$g(x, y) = \Im^{-1} [H(u, v)F(u, v)]$$

Isto é, o resultado é a transformada inversa de $H(u, v)F(u, v)$, em que $F(u, v)$ é a transformada de uma imagem de entrada, e $H(u, v)$ é uma função de filtro. Por exemplo, um filtro que aceita baixas frequências é chamado de filtro passa-baixa. O efeito final produzido por um filtro passa-baixa é borrar (suavizar) uma imagem.

Algo muito importante para a filtragem no domínio da frequência é o **teorema da convolução**. Esse teorema nos diz que fazer a convolução no domínio espacial é equivalente a multiplicar ponto-a-ponto a transformada da imagem e um filtro no domínio da frequência.

Filtro gaussiano no domínio da Frequência

Filtros gaussianos são filtros passa baixa que geralmente deixam a imagem muito atenuada. Um filtro gaussiano bidimensional é dado por:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

Em que $D(u, v)$ é a distância até o centro, e $\sigma = D_0$ é uma medida de dispersão ao redor do centro. Quanto maior o valor de D_0 maior o borramento.

Filtro High Boost no domínio da Frequência

Os conceitos relacionados à máscara de nitidez e ao filtro *High Boost* são os mesmos do domínio espacial, no entanto, os métodos para sua aplicação são diferentes. Dessa vez temos que a máscara é definida por:

$$g_{\text{máscara}}(x, y) = f(x, y) - f_{\text{LP}}(x, y)$$

$$f_{\text{LP}}(x, y) = \Im^{-1}[H_{\text{LP}}(u, v)F(u, v)]$$

Em que $H_{LP}(u, v)$ é um filtro passa-baixa e $F(u, v)$ é a transformada de Fourier de $f(u, v)$. Ao final, a imagem filtrada é definida como:

$$g(x, y) = f(x, y) + k * g_{\text{máscara}}(x, y)$$

Algoritmo desenvolvido

O objetivo principal desta parte do trabalho era implementar e testar a filtragem *High Boost*, utilizada para o aumento de nitidez de imagens digitais, utilizando a filtragem passa-baixa no domínio da frequência. Podemos ver no Algoritmo 3 a solução desenvolvida pelos autores.

```
import scipy.fftpack as fp
from scipy import signal
import numpy as np
import cv2
import matplotlib.pylab as pylab
```

```
# carrega a imagem
img = cv2.imread(input("Digite o caminho para a imagem de entrada:
"))[:, :, 0]

# lê o valor de D
D = int(input("Digite o valor de D para o kernel: "))

# obtém o filtro gaussiano no domínio espacial
gauss_kernel = np.outer(signal.gaussian(
   img.shape[0], D), signal.gaussian(img.shape[1], D))

# obtém a imagem e o filtro no domínio da frquência
freq_img = fp.fft2(img)
freq_kernel = fp.fft2(fp.ifftshift(gauss_kernel))
assert(freq_img.shape == gauss_kernel.shape)

# aplica convolução (borramento) utilizando teorema da convolução
convolved_img = freq_img * freq_kernel
blured_img = fp.ifft2(convolved_img).real

# normalização
blured_img = np.abs(blured_img)
blured_img = blured_img * 255 / blured_img.max()
blured_img = blured_img.astype(np.uint8)

# lê o valor de k
k = int(input("Digite o valor de k: "))

# obtem a máscara (arestas)
mask = cv2.subtract(img, blured_img)
mask = k * mask
```

```
# obtem imagem final
final_image = cv2.add(img, mask)
cv2.imwrite(input("Digite o caminho para a imagem de saída: "),
final_image)
print("Imagem final salva!")
```

Algoritmo 3 *High Boost* utilizando passa-baixa gaussiano no domínio da frequência

Primeiramente, é importante dizer que a solução desenvolvida poderia ser implementada de diversas maneiras, no entanto, optamos por aquela que teria menor custo computacional.

Observando o algoritmo desenvolvido, nota-se que o valor de D_0 utilizado no filtro gaussiano e o valor de ponderação k são variáveis definidas pelo usuário. Assim como na solução anterior, nossa primeira escolha foi trabalhar apenas com funções da biblioteca *OpenCV*, para que não houvesse qualquer problema na manipulação dos dados e possíveis erros em operações entre eles. No entanto, optamos por utilizar as funções da biblioteca *scipy* para realização das transformadas, pois já havíamos trabalhado com ela. Explicaremos, então, os passos de nossa solução:

1. **Filtro passa-baixa gaussiano:** primeiramente, obtemos nosso filtro gaussiano no domínio espacial. O filtro é construído com as mesmas dimensões da imagem, para que no domínio da frequência possa ser multiplicado ponto-a-ponto, e com a medida de dispersão D_0 especificada pelo usuário. O resultado será uma matriz centrada na origem, isto é, com uma distribuição a partir do centro da matriz.
2. **Transformada da imagem:** nesse passo apenas realizamos a transformada de Fourier da imagem original, obtendo a correspondente no domínio da frequência. Fazemos a ressalva de que, nesse momento, a periodicidade não está concentrada no centro da imagem.
3. **Adequação do filtro:** para que a multiplicação ponto-a-ponto fosse realizada corretamente, tivemos de fazer o *shift* inverso da matriz gaussiana, removendo a periodicidade do centro da matriz. Além disso, aplicamos a transformada de Fourier

no filtro, sendo que um filtro gaussiano continua sendo um filtro gaussiano após a transformada, mas agora com valores complexos adequados.

4. **Aplicação da convolução:** a partir do teorema da convolução, realizamos a multiplicação ponto-a-ponto da imagem no domínio da frequência e do filtro gaussiano para realizar a filtragem. Após isso, realizamos a transformada inversa para obtermos, agora, a imagem filtrada no domínio espacial.
5. **Normalização:** aplicamos uma normalização nos valores da imagem filtrada, para se adequarem corretamente à escala de cinza [0, 255].
6. **Obtenção da máscara:** de forma simples, utilizamos a função *cv2.subtract* para realizarmos a subtração ponto-a-ponto da imagem original com a imagem borrada normalizada obtida no passo anterior.
7. **Obtenção da imagem final:** por fim, adicionamos a máscara de nitidez, com ou sem *High Boost*, na imagem original utilizando a função *cv2.add*, obtendo a imagem aguçada.

Optamos, ainda, por exibir alguns passos intermediários da filtragem na frequência ao usuário, como apresentamos no Algoritmo 4. A Figura 5 nos mostra um exemplo.

```
############## Mostra todos os passo intermediários #############
pylab.figure(figsize=(75, 75))
pylab.gray()

pylab.subplot(2, 2, 1), pylab.imshow(img), pylab.title(
   'Imagem original', size=10), pylab.axis('off')

pylab.subplot(2, 2, 4), pylab.imshow(blured_img)
pylab.title('Imagem após convolução', size=10), pylab.axis('off')

pylab.subplot(2, 2, 2), pylab.imshow(
   (20*np.log10(0.1 + fp.fftshift(freq_kernel))).astype(int))
pylab.title('Espectro do filtro', size=10), pylab.axis('off')

pylab.subplot(2, 2, 3), pylab.imshow(
   (20*np.log10(0.1 + fp.fftshift(convolved_img))).astype(int))
```

```
pylab.title('Espectro da imagem borrada',
           size=10), pylab.axis('off')

pylab.subplots_adjust(wspace=0.2, hspace=1)
pylab.show()
#######################################################################
```

Algoritmo 4 Apresentação dos passos intermediários da filtragem na frequência ao usuário

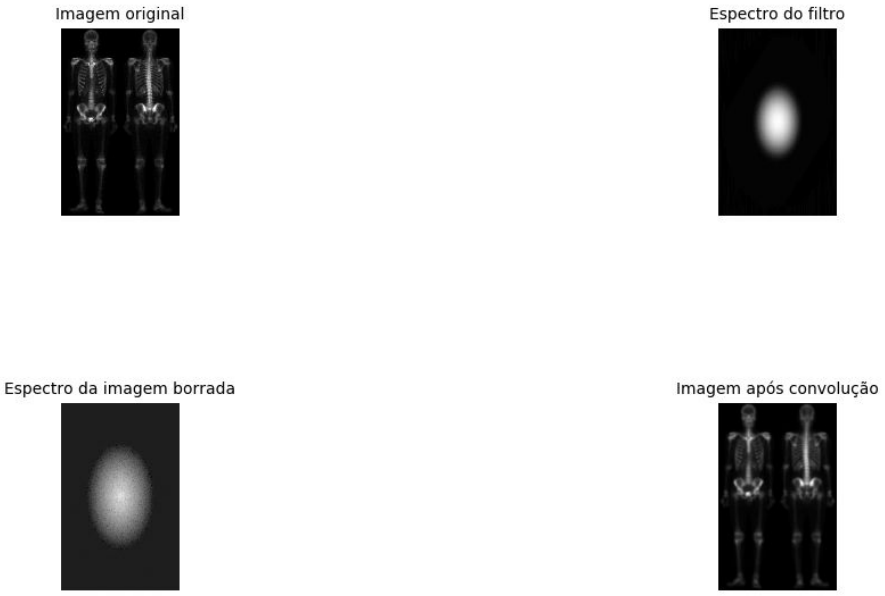

Figura 5 Etapas intermediárias da filtragem no domínio da frequência

Na Figura 6 podemos observar alguns resultados utilizando a filtragem passa-baixa na frequência. É possível notar que mesmo com um valor pequeno de dispersão, $D_0 = 3$, a imagem fica mais aguçada que para máscaras $11x11$ no domínio espacial. É interessante notar, também, que quanto maior o valor de dispersão, menos a filtragem deixa passar, permitindo que apenas regiões já preenchidas com branco sejam mais aguçadas.

Figura 6 FIltragem no domínio da frequência utilizando filtro gaussiano com D_0 igual a 3, 10 e 50, respectivamente.

Referências bibliográficas

GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens** . Pearson Educación, 2009.

NUMPY. **Numpy.outer**. Disponível em: https://numpy.org/doc/stable/reference/generated/numpy.outer.html. Acesso em: 29 nov. 2020.

OpenCV. **Cv2.** Disponível em:https://docs.opencv.org/. Acesso em 29 nov. 2020.

SCIPY. **Scipy.fftpack.** Disponível em: https://docs.scipy.org/doc/scipy/reference/fftpack.html. Acesso em: 29 nov. 2020.

SCIPY. **Scipy.signal**. Disponível em: https://docs.scipy.org/doc/scipy/reference/signal.html. Acesso em: 29 nov. 2020.