Trabalho MC920 1:
Filtragens no Domínio Espacial e de Frequências

Esdras R. Carmo - RA 170656

17 de abril de 2019

1 Introdução

Exploraremos nesse trabalho alguns algoritmos de filtragem de imagens. Um dos algoritmos irá implementar a operação de convolução em imagem bidimensional no domínio espacial com diferentes filtros, como passa-alta, passa-baixa e de detecção de bordas a partir do gradiente.

Um segundo algoritmo será reponsável por implementar um filtro gaussiano no domínio de frequência, sendo necessário aplicar a transformada de Fourier da imagem e aplicar o filtro com uma multiplicação na matriz, tornando-se assim uma operação muito mais barata computacionalmente que a convolução no domínio espacial.

2 Filtragem no Domínio Espacial

2.1 Especificação do Problema

A filtragem no domínio espacial é feita a partir da operação de convolução entre a imagem e um *kernel* [4] seguindo a expressão:

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$

Sendo $g(x,y)$ a imagem filtrada, $f(x,y)$ a imagem de entrada e ω o *kernel*. Um exemplo de convolução utilizando um filtro de Sobel para detecção de bordas é mostrado na Figura 1.

Os filtros utilizados nessa seção foram:

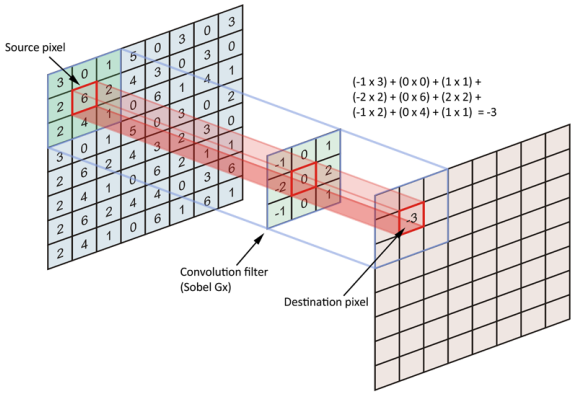


Figura 1: Exemplo de Convolução utilizando filtro de Sobel Gx

- Filtro h_1: Consiste em um filtro passa-alta, responsável por atenuar baixas frequências, realçando detalhes da imagem;

$$h_1 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

- Filtro h_2: Consiste em um filtro passa-baixa, responsável por atenuar altas frequências, atenunado ruídos e detalhes da imagem, dando um aspecto borrado;

$$h_2 = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- Filtros h_3 e h_4: Filtros de Sobel G_x e G_y respectivamente, que aproxima o valor do gradiente da imagem em x e em y.

$$h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} h_4 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

  Também será feito a combinação dos filtros h_3 e h_4 por meio da expressão:

$$g(x,y) = \sqrt{(h_3 * f(x,y)^2 + (h_4 * f(x,y))^2} \tag{1}$$

  Aproximando assim a magnitude do gradiente $\nabla f(x,y)$, dado por:

$$\nabla f(x,y) = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

2.2 Entrada de Dados

O script para aplicação dos filtros foi escrito utilizando Python 3.7 com as bibliotecas *scikit-image*, *scipy* e *numpy*. Para a execução do código basta rodar o seguinte comando:

```
python spacial_filters.py input_image.png output_image.png hx
```

Sendo os parâmetros:

- *spacial_filters.py*: Nome do script;
- *input_image.png*: Caminho da imagem de entrada do filtro, aceitando imagens em png em escala de cinza;
- *output_image.png*: Caminho da imagem de saída do filtro, no formato png em escala de cinza;
- *hx*: Filtro a ser aplicado, aceitando os valores: *h1*, *h2*, *h3*, *h4* e *h34*, sendo esse último a combinação entre os filtros h_3 e h_4.

2.3 Detalhes de Implementação

Para a leitura e escrita da imagem em tons de cinza foi utilizado o módulo *io* da biblioteca *scikit-image* [2] e a entrada foi convertida para o tipo *float32* do *numpy* a fim de não ter problemas de overflow com o tipo *uint8*.

A operação de convolução foi realizada utilizando a função *convolve* da biblioteca *scipy* [3] com *mode* = *constant*, o que faz com que as bordas sejam preenchidas com 0.

Após a convolução é feito uma normalização na imagem para o intervalo $(0, 255)$ utilizando as seguintes equações:

$$g(x,y) := f(x,y) - min(f(x,y))$$
$$f_{norm}(x,y) := \frac{255 * g(x,y)}{max(g(x,y))}$$

2.4 Resultado e Discussão

2.4.1 Filtro h_1

A aplicação do filtro pode ser observada na Figura 2. Como se trata de um filtro passa-alta, percebe-se que apenas os detalhes da imagem foram mantidos e regiões homogêneas foram zeradas, sendo essa característica devido o fato do *kernel* possuir pesos que se anulam. O filtro é interessante para ressaltar bordas, detalhes e ruídos da imagem.

2.4.2 Filtro h_2

A aplicação do filtro pode ser observada na Figura 3. Esse filtro se trata de um passa-baixa, tendo como principal característica ter um *kernel* em que os pesos fazem uma média ponderada dos pixels vizinhos. Dessa forma, a aplicação desse filtro resulta em uma imagem com menos detalhes, menos ruídos e mais borrada. A maior aplicação do filtro é para remoção de ruídos.

2.4.3 Filtros h_3 e h_4

A aplicação do filtro h_3 pode ser observada na Figura 4b. Como se trata de um filtro de Sobel em x, temos como resultado as variações em x, identificando dessa forma bordas paralelas ao eixo y.

Na Figura 4c observa-se a aplicação do filtro h_4, que complementa h_3 com Sobel em y, identificando assim as bordas paralelas ao eixo x.

Quando combina-se os dois filtros utilizando a expressão (1), temos o resultado na Figura 4d. Percebe-se que apenas as bordas da imagem original 4a foram mantidas após a aplicação do filtro de Sobel. Com isso, basta aplicar um bom limiar na imagem resultado e teremos um mapa de bordas.

3 Filtragem Gaussiana no Domínio de Frequência

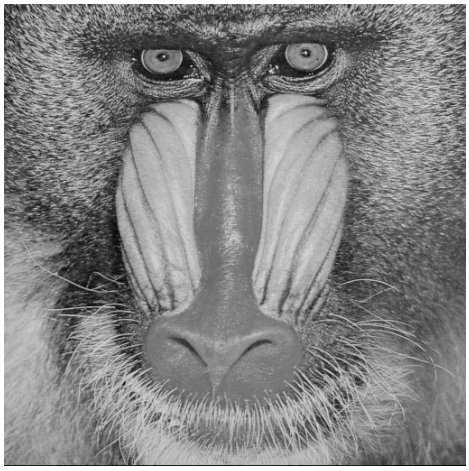
3.1 Especificação do Problema

Nesta etapa a filtragem será feita no domínio de frequência, portanto, em primeiro momento é necessário aplicar a transformada de Fourier na imagem de entrada e obter a imagem no domínio da frequência através da expressão:

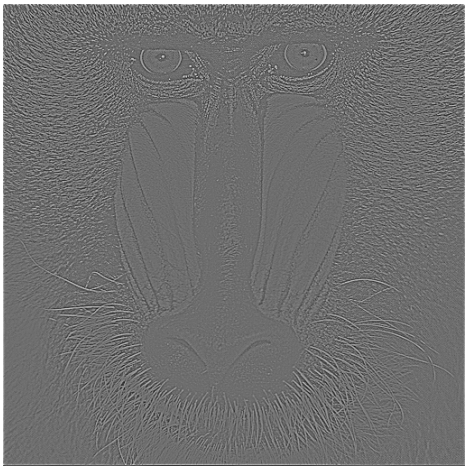
$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi(ux/M + vy/N)}$$

No domínio espacial os filtros são realizados através da convolução da imagem de entrada com um *kernel*. No domínio de frequência, a operação de convolução se converte em uma multiplicação da transformada do filtro com a transformada da imagem, segundo o Teorema da Convolução 2D:

$$f(x,y) \circledast h(x,y) \iff F(u,v)H(u,v)$$



(a) Imagem de entrada



(b) Resultado da convolução com h_1

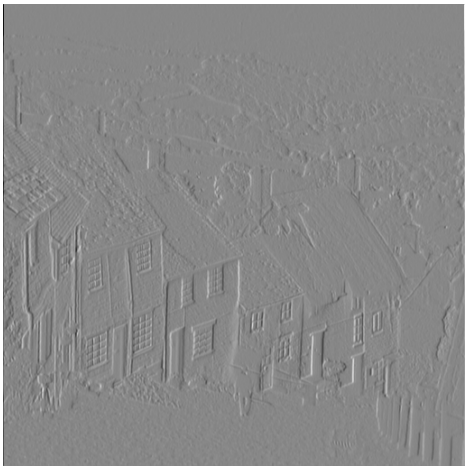
Figura 2: Aplicação do filtro h_1

(a) Imagem de entrada

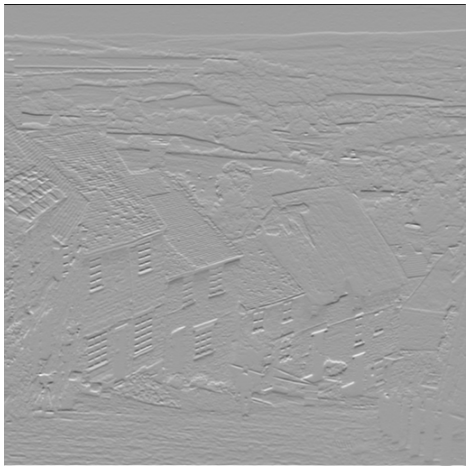
(b) Resultado da convolução com h_2

Figura 3: Aplicação do filtro h_2

(a) Imagem de entrada



(b) Resultado da convolução com h_3



(c) Resultado da convolução com h_4

(d) Resultado da combinação dos filtros h_3 e h_4

Figura 4: Aplicação dos filtros h_3, h_4 e a combinação dos dois

Para realizar um filtro passa-baixa Gaussiano, precisamos da função de transferência Gaussiana, dada pela expressão:

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

Em que $D^2(u,v) = u^2 + v^2$ e D_0 é a frequência de corte do filtro.

3.2 Entrada de Dados

O script para execução da transformada de Fourier e filtragem em domínio de frequência foi escrito em Python 3.7 utilizando as bibliotecas *skimage*, *numpy* e *scipy*.

Para a execução do código basta executar:

```
python freq_gaussian_filter.py input_image.png output_image.png d0
```

Sendo os parâmetros:

- *freq_ gaussian_ filter.py*: Nome do script;
- *input_ image.png*: Caminho da imagem de entrada do filtro, aceitando imagens em png em escala de cinza;
- *output_ image.png*: Caminho da imagem de saída do filtro, no formato png em escala de cinza;
- *d0*: Frequência de corte utilizada na geração da função de transferência gaussiana.

3.3 Detalhes de Implementação

Para a transformada de Fourier da imagem de entrada foi utilizado as funções *fft2* e *fftshift* do módulo *fftpack* [1] da biblioteca *scipy*, gerando assim a imagem transladada com a frequência zero no centro.

O filtro gaussiano foi gerado através da fórmula (2) recebendo D_0 como parâmetro do script e com as distâncias sendo calculadas referentes ao centro da imagem.

As imagens no domínio de frequência foram salvas aplicando o log da magnitude da imagem: $20 * \log |F(u,v)|$

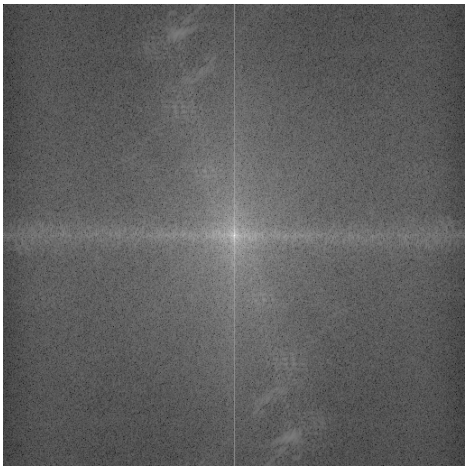
3.4 Resultado e Discussão

A imagem de entrada está representada na Figura 5 no domínio espacial e de frequência. Percebe-se que o pixel com maior valor da Figura 5b está posicionado no centro da imagem, sendo essa a frequência zero.

Após a aplicação do filtro gaussiano com $d_0 = 20000$ no domínio de frequência percebe-se que quanto mais longe do centro da imagem, menor o valor do pixel (vide Figura 6b). Esse é o comportamento esperado de um filtro passa-baixa, já que ele atenua altas frequências, i. e., pixels com maior distância do centro.

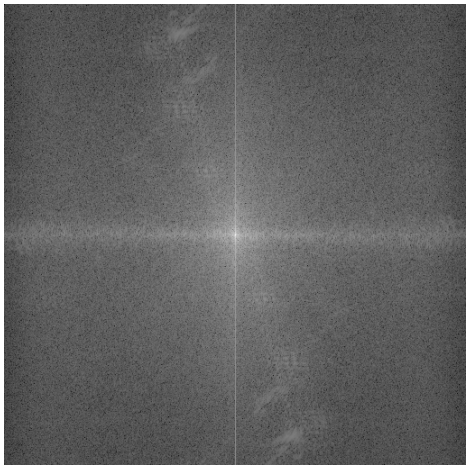
O resultado da filtragem no domínio espacial pode ser observado na Figura 7 para diferentes frequências de corte. Percebe-se que quanto menor a frequência de corte d_0, mais borrada a imagem final fica, devido ao fato que o filtro com menor d_0 acaba atenuando os valores de pixels mais próximos ao centro do domínio de frequência, cortando ainda mais frequências altas.

(a) Imagem no domínio espacial

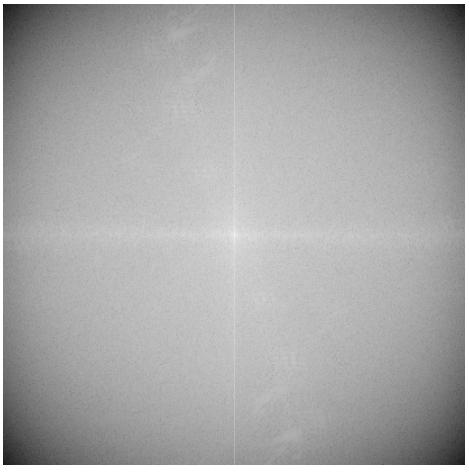


(b) Imagem no domínio de frequência

Figura 5: Entrada da filtragem gaussiana no domínio da freqência



(a) Entrada no domínio de frequência



(b) Aplicação do filtro no domínio de frequência

Figura 6: Comparação da entrada e saída do filtro no domínio de frequência

(a) $d_0 = 10000$

(b) $d_0 = 5000$

(c) $d_0 = 2000$

(d) $d_0 = 500$

Figura 7: Comparação da saída do filtro no domínio espacial

4 Conclusão

Neste trabalho foi aplicado diversos filtros em imagens no domínio espacial através da convolução. Com ela observou-se as diferenças entre os filtros passa-baixa e passa-alta, além da aplicação do filtro de Sobel, muito utilizado para a detecção de bordas.

Utilizando das propriedades da transformada de Fourier, foi possível aplicar um filtro passa-baixa gaussiano de forma mais eficiente, já que a operação de convolução se transformou em uma mera multiplicação pixel a pixel entre a imagem e a função de transferência.

Referências

[1] Scipy FFTPack. Discrete fourier transforms. `https://docs.scipy.org/doc/scipy/reference/fftpack.html`. [Online; acessado em 16 de Abril de 2019].

[2] Scikit Image. Module io documentation. `https://scikit-image.org/docs/dev/api/skimage.io.html`. [Online; acessado em 14 de Abril de 2019].

[3] Scipy NdImage. Multidimensional convolution. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.convolve.html`. [Online; acessado em 14 de Abril de 2019].

[4] Wikipédia. Kernel (image processing). `https://en.wikipedia.org/wiki/Kernel_(image_processing)`, 2019. [Online; acessado em 14 de Abril de 2019].