# Relatório do trabalho 1

Allan Nozomu Fukasawa RA:163527

15/04/2019

## 1 Introdução

O objetivo deste trabalho é implementar alguns filtros de imagens no domínio espacial e de frequência. "A filtragem aplicada a uma imagem digital é uma operação que alter os valores de intensidade dos pixels da imagem levando-se em conta o valor do pixel em questão quanto valores de pixels vizinhos." [1]

## 2 Componentes

Está sendo enviado junto a este relatório, o arquivo Trabalho 1.ipynb onde contém todo o código executado durante este trabalho, 5 imagens de exemplo (sendo a que está sendo usada no programa é, por padrão, a house.png, Figura 1.) e uma pasta results contendo todos os resultados finais e intermediários.

Figure 1: Imagem original

### 2.1 O Programa

O programa foi implementado com Jupyter Notebooks, usando Python 3.7.1. As bibliotecas utilizadas no desenvolvimento do programa foram, com suas respectivas versões:

- numpy (1.15.4): para manipulação dos vetores.
- matplotlib (3.0.2): visualização dos dados, resultados finais e intermediários.
- opencv (3.4.2): realização da leitura e escrita das imagens, transformação das cores e normalização dos dados.
- scipy.ndimage (1.1.0): transformações das imagens (Transformada de Fourier e sua inversa), aplicação dos filtros.

### 2.2 Formato das imagens

As imagens de entrada e de saída serão no formato PNG (Portable Network Graphics) em tom de cinza. Todas as imagens serão salvas na pasta results para melhor organização do projeto.

## 3 Solução

### 3.1 Leitura das imagens

A imagem de entrada é lida com função **cv2.imread** que armazena a imagem em um **numpy.ndarray** de 3 dimensões (MxNx3).

Depois de lida, a imagem é convertida para níveis de cinza pela função **cv2.cvtColor** e também tem seus valores convertidos para float, dividindo seu valor por 255, o maior número. Dessa forma, não é necessário se preocupar com overflow e underflow dos números nas operações dos diferentes filtros aplicados.

### 3.2 Escrita das imagens

Também foi feito uma função auxiliar para facilitar na saída das imagens. Antes de tudo, a imagem é normalizada utilizando **cv2.normalize**, transformando a imagem que antes estava como float em inteiro novamente, variando de 0 a 255.

Após a normalização, a escrita da imagem é feita utilizando a função **cv2.imwrite**

Há também algumas imagens salvas a partir do **matplotlib** correspondente aos resultados das imagens no domínio de frequência. Estas foram salvas utilizando a função **matplotlib.pyplot.savefig**.

### 3.3 Plotagem das imagens

Foi utilizado para visualização dos resultados as funções de plotagem de imagens em **matplotlib.pyplot**, utilizando a paleta de cores em escalas de cinza. Não foi

necessário fazer a normalização para a visualização dos dados, porque a função já realiza uma normalização linear.

## 3.4 Filtragem em domínio espacial

Para a aplicação do filtro em domínio espacial utiliza-se uma operação de convolução de uma máscara pela imagem. Este processo é equivalente a percorrer a imagem modificando seus valores conforme os pesos da máscara e as intensidades da imagem.

Para realizarmos a função de convolução foi utilizado **ndimage.convolve**. Para o tratamento das bordas, foi feito uma moldura formada por 0, dessa forma, serão ignorados na operação de convolução.

Foi feito um truncamento dos resultados obtidos entre 0 e 255, principalmente nos filtros passa-alta onde valores podem ser muito grandes ou pequenos. Dessa forma, obteu-se uma diferente maneira e também mais fácil na detecção de bordas. Além disso, os valores nessas imagens foram invertidos nesses casos. Desse modo, as bordas se destacariam com a cor preta e o fundo, cor branca.

Ao todo foram feitos 5 filtros: um filtro passa alta, um filtro passa baixa, dois filtros para detecção de borda (horizontal e vertical) e a combinação dos dois anteriores.

### 3.4.1 Filtro Laplaciano

O primeiro filtro aplicado foi um filtro passa-alta conhecido como Filtro Laplaciano. Seu uso mais comum é para detecção de bordas em imagens. O resultado obtido está na Figura 2.

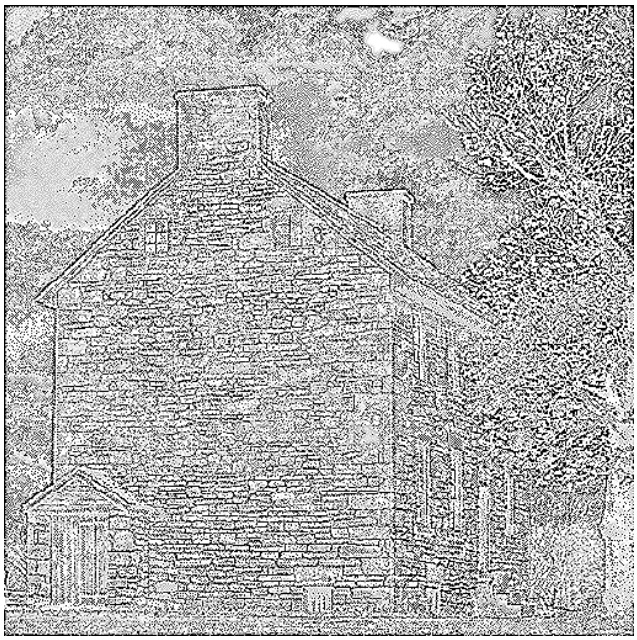

Figure 2: Filtro laplaciano, valores truncados

Podemos notar algumas limitações desse algoritmo quanto ao resultado com ruídos, provocados pela imagem original. Para melhores resultados, foi passado antes um filtro que que eliminasse ou diminuísse o ruído (o filtro gaussiano utilizado na segunda parte do exercício) antes de aplicar o filtro laplaciano para detecção de bordas.

Com isso, temos o seguinte resultado na Figura 3 . Nota-se uma melhora notável entre as duas imagens principalmente perto do telhado onde a imagem original apesentava bastante ruído e chuviscos.

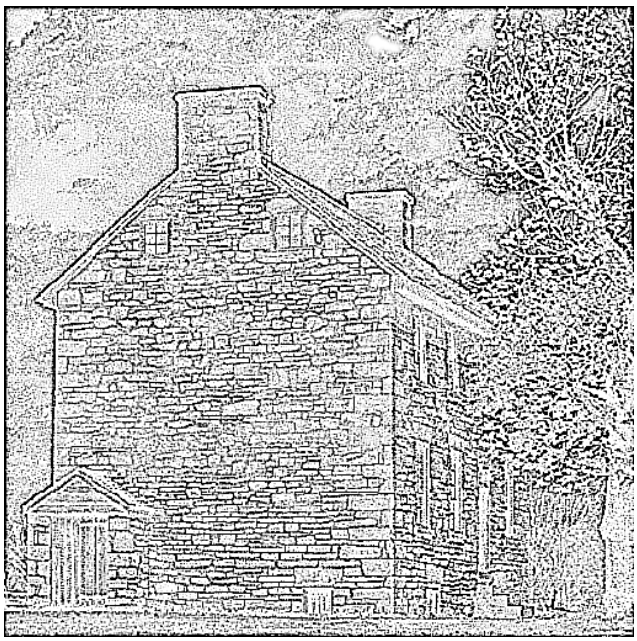

Figure 3: aplicação do filtro laplaciano após gaussiano

### 3.4.2 Filtro passa-baixa Gaussiano

O segundo filtro aplicado foi um filtro passa-baixa gaussiano. Seu resultado causa um borramento, muitas vezes utilizado para redução de ruído e detalhamento.

Figure 4: aplicação do filtro passa-baixa gaussiano

Por se tratar de um filtro separável, ele foi sepa-

rado em dois filtros: linha e coluna. Dessa forma, há um ganho computacional pois o número de operações de multipliação será bem menor para cada pixel da imagem. Para a tranforação dos filtros tanto em linha quanto coluna, foi usado a função **reshape** presentes nos **numpy.ndarray**.

Depois de realizado a filtragem na linha e coluna, o resultado de ambos foi somado, obtendo a seguinte imagem filtrada (Figura 4). Podemos notar uma redução no ruído (principalmente nas nuvens e também nas janelas da casa).

### 3.4.3 Filtro Sobel

Os três seguintes filtros são os de Sobel, utilizados sobretudo, para detecção de bordas. Dois desses filtros são para calcular as variações verticais e horizontais, logo, cada um deles vai detectar contornos na vertical e horizontal como observados na Figura 5.

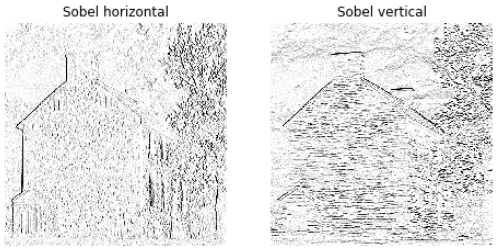


Figure 5: Detecção de bordas verticais e horizontais com valores truncados

Com ambos os filtros calculados, o filtro resultante de Sobel (magnitude) se dá pela raiz quadrada da soma dos quadrados de cada pixel obtidos anteriormente, resultando na Figura 6. Podemos notar uma detecção de borda que, apesar do ruído da imagem, apresentou resultados satisfatórios em comparaçõ ao primeiro filtro laplaciano 2.

Comparado ao filtro Laplaciano, este apresentou uma tolerância maior em relação aos ruídos e na detecção de bordas.

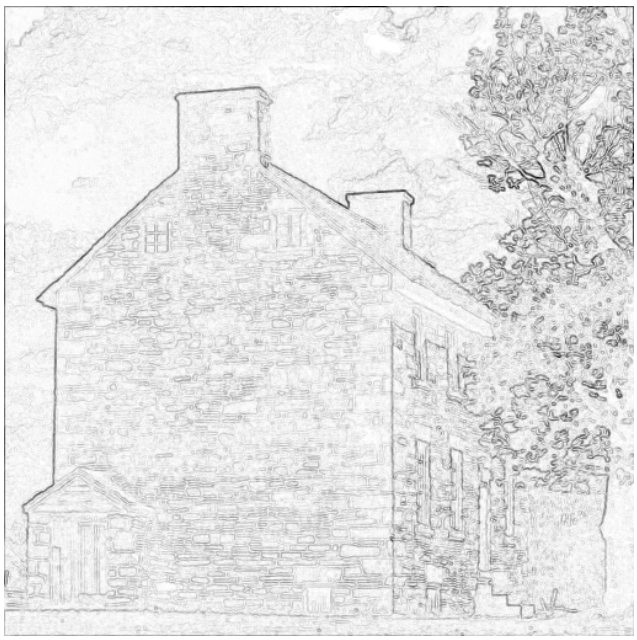

Figure 6: Detecção de bordas po Sobel

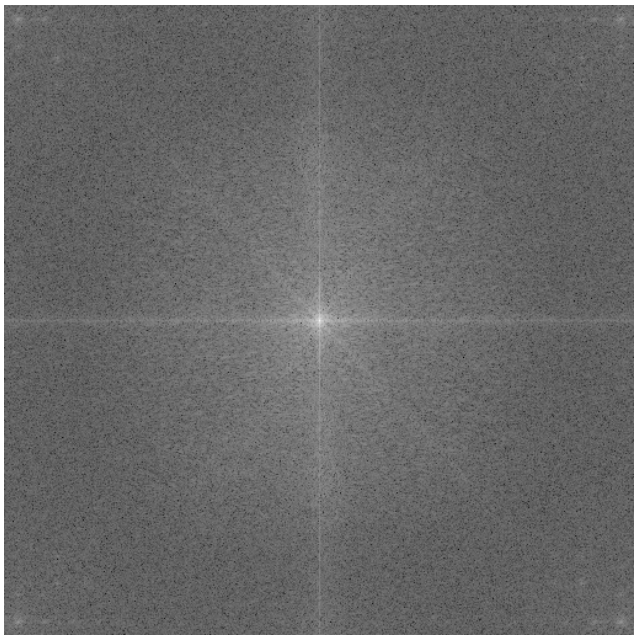

Figure 7: Espectro de frequência

## 3.5 Filtragem domínio de frequência

Para a aplicação do filtro em domínio de frequência, foi necessário antes de tudo, tranformar a imagem do domínio espacial para o de frequência. Para isso, foi utilziado a função **numpy.fft.fft2** que implementa a Tranformada Rápida de Fourier (FFT). Também foi feito a translação da frequência zero para o centro do espectro utilizando a função **numpy.fft.fftshift**.

Foi também realizado um pequeno tratamento para uma melhor compreensão do espectro de frequência. Foi feito uma operação logarítmica **numpy.log** depois de calculado o valor absoluto do espectro de frequência **numpy.abs**, obtendo o resultado na 7.

Para a aplicação do filtro, basta definirmos uma função de máscara em torno do espectro de frequência que permite a passagem (parcial ou inteira) ou não das faixas de frequência. Filtros passa alta são definidos nas frequências mais altas (nas bordas), enquanto os filtros passa-baixa nas frequências mais ao centro, porque a imagem foi transladada a partir do centro.

Para a contrução da máscara, um **numpy.array** de indices começando a partir do centro da imagem e crescendo conforme aproximando das bordas foi criado. Dessa forma, foi calculado as distâncias dos filtros.

Para tranformar a imagem de volta para o domínio espacial a partir do domínio de frequência, após aplicar os filtros, é necessário realizar o processo inverso. É realizado primeiramente a translação da frequência zero utilizando sua inversa **numpy.fft.ifftshift** e depois a inversa da Tranformada de Fourier **numpy.fft.ifft2**.

No domínio de frequência, foram realizados 4 diferentes tipos de filtro gausiano: passa-alta, passa-baixa, passa-faixa e rejeita-faixa.

### 3.5.1 Passa alta gaussiano

Para o filtro passa-alta, foram feito 4 filtragens com os respectivos valores de corte: 2, 10, 25 e 200. Conforme o número de corte aumenta, diminui o número de detalher da imagens, permanecendo apenas os contornos.

Em números altos de corte, é impossível distinguir elementos da imagem original, como na frequência de corte 200. O resultado pode ser conferido na Figura 8.

### 3.5.2 Passa baixa gaussiano

Para o filtro passa-baixa, foram feito 4 filtragens com os respectivos valores de corte: 10, 25, 100 e 200. Conforme o número de corte diminui, aumenta o nível de borramento da imagem

Em números baixos de corte, a imagem apresenta um borramento muito alto, perdendo o nível de detalhamento da imagem. O resultado pode ser conferido na Figura 9.

### 3.5.3 Passa faixa gaussiano

Para o filtro passa-faixa, foram feito 4 filtragens com os respectivos valores em par, tanto para corte quanto para a largura da faixa respectivamente: (25, 50), (50, 100), (100, 200) e (200, 400). Tem-se um resultado similar ao filtro passa-alta gaussiano, acentuando os contornos da imagem conforme diminuímos o raio de corte e da largura da faixa. O resultado pode ser conferido na Figura 10.

### 3.5.4 Rejeita faixa gaussiano

Para o filtro rejeita-faixa, foram feito 4 filtragens com os respectivos valores em par, tanto para corte quanto para a largura da faixa respectivamente: (25, 50), (50, 100), (100, 200) e (200, 400). Tem-se um resultado similar ao filtro passa-baixa gaussiano, apresentando um borramento com efeitos de ruído (chuviscos). O resultado pode ser conferido na Figura 11.

## References

[1] H. Pedrini, *Trabalho 1.* Introdução ao Processamento de Imagens (MC920 / MO443), 2019.

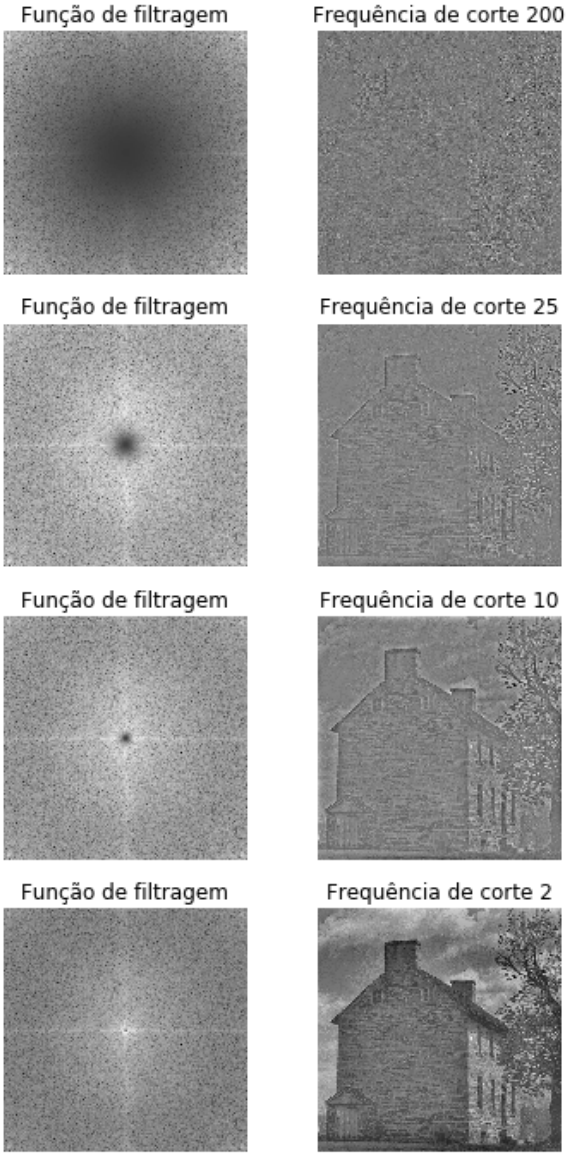


Figure 8: Passa-alta gaussiano

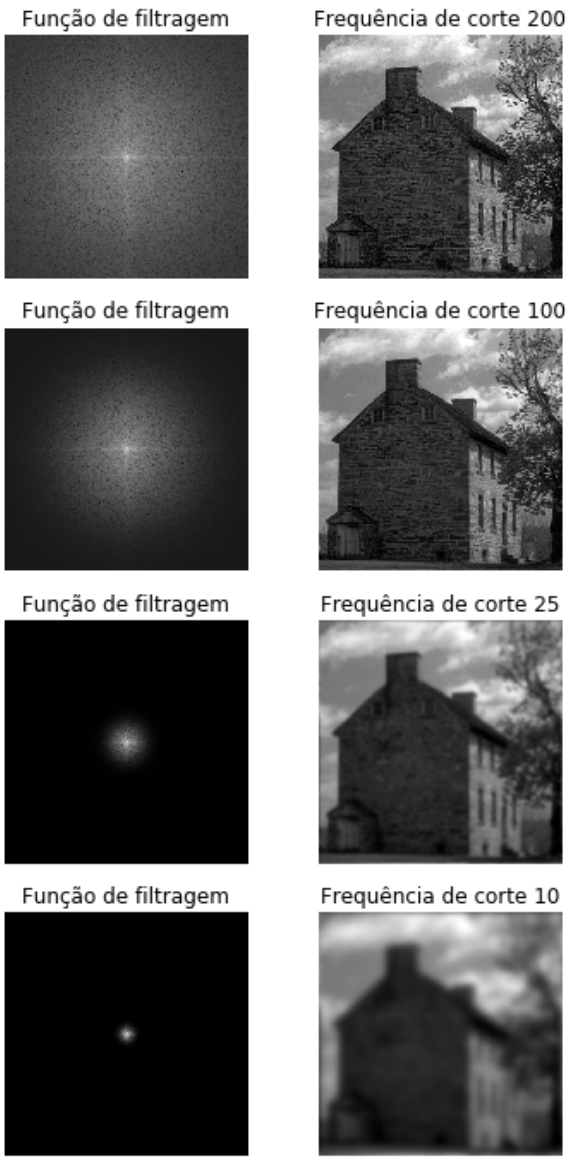


Figure 9: Passa-baixa gaussiano

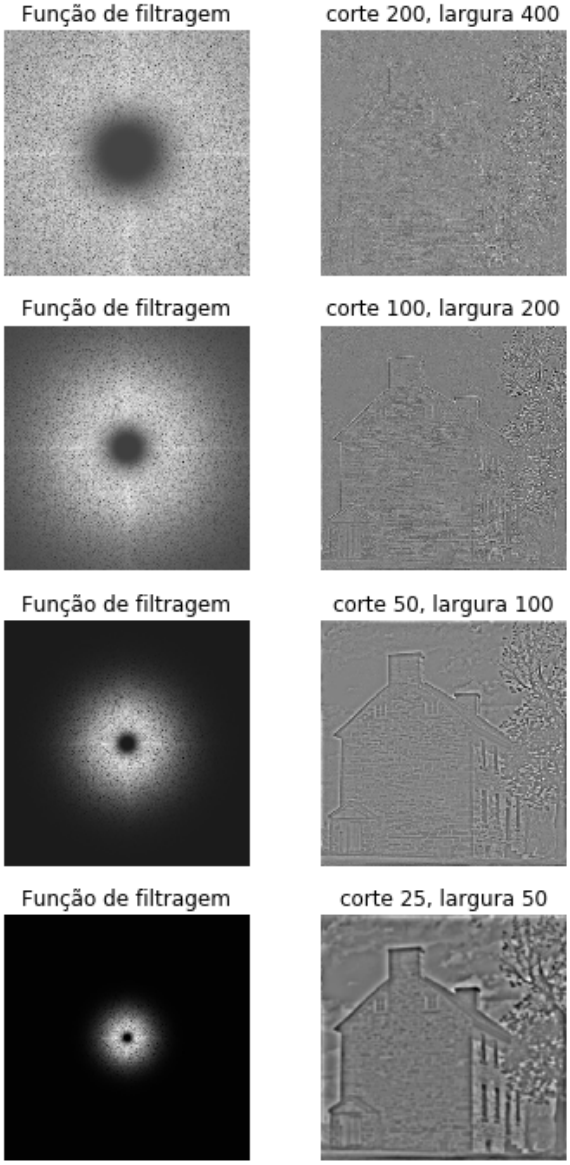


Figure 10: Passa-faixa gaussiano

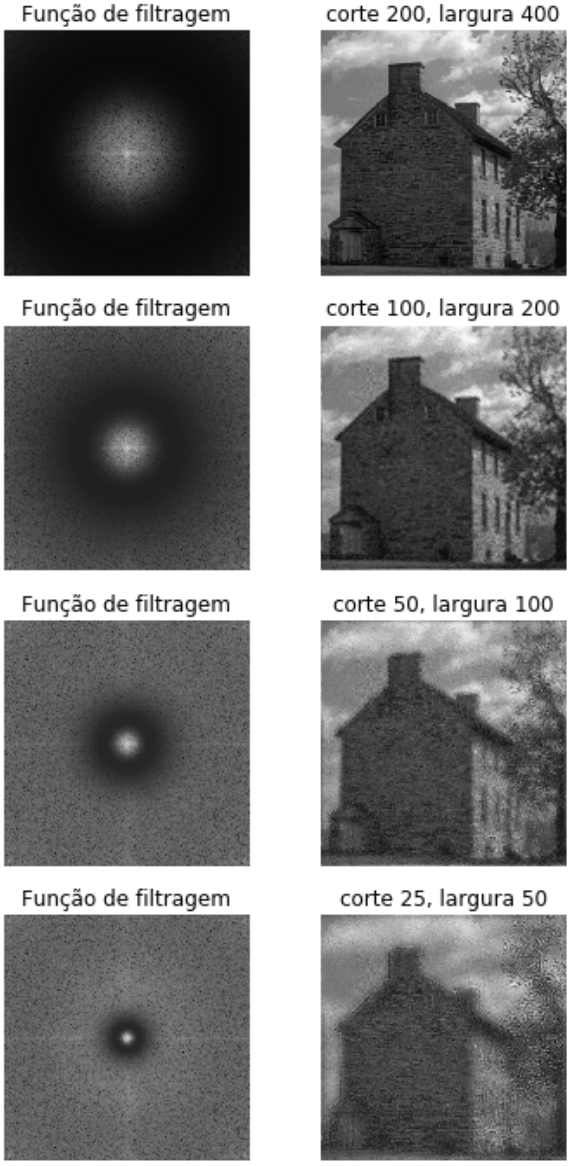


Figure 11: Rejeita-faixa gaussiano