

Trabalho 1

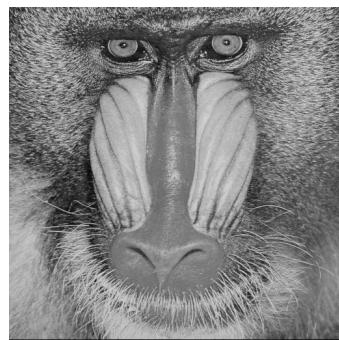
Tiago de Paula Alves (187679)
tiagodepalves@gmail.com

31 de março de 2025

1 Introdução

Este trabalho teve como objetivo a implementação e a análise de alguns filtros de imagem no domínio espacial. A filtragem é feita pela convolução da imagem com uma máscara utilizando bibliotecas de processamento de imagens.

A figura 1 apresenta as imagens base deste trabalho, usadas para análise e discussão dos filtros, todas monocromáticas. Os filtros utilizados serão apresentados ao longo do relatório, à medida que forem necessários pelo texto.



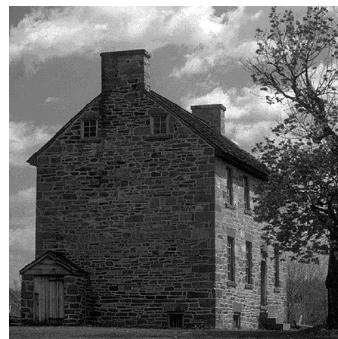
(a) `imagens/baboon.png`



(b) `imagens/butterfly.png`



(c) `imagens/city.png`



(d) `imagens/house.png`

Figura 1: Imagens base da comparação dos filtros.

2 O Programa

Além das bibliotecas padrão de Python, foram utilizados os pacotes SciPy¹ e OpenCV.²

2.1 Código Fonte

O programa foi desenvolvido em Python 3.8, mas deveria funcionar com as versões 3.6 e 3.7 também. Além disso, o código fonte foi separado nos seguintes arquivos:

¹ SciPy. URL: <https://www.scipy.org/>.

² OpenCV – Open Source Computer Vision Library. URL: <https://opencv.org/>.

main.py É o corpo do programa, responsável por processar os comandos e as opções da linha de comando.

lib.py Operações de convolução de imagens e operações auxiliares.

filtro.py Definição das máscaras de convoluções (*kernels*).

inout.py Funções que tratam da entrada e saída do programa, como leitura e escrita de arquivos de imagem e também da apresentação da imagem em uma janela gráfica.

tipos.py Definição dos tipos para checagem estática com `mypy`.³

Todas as figuras base utilizadas neste relatório podem ser encontradas na pasta `imagens` do código fonte, como descrito nos rótulos da figura 1. Além disso, foi disponibilizado também um *script* em `bash`, `run.sh`, que realiza todos os processamentos requeridos em cada uma das imagens na pasta.

2.2 Execução

A execução deve ser feita através do interpretador de Python 3.6+. A única entrada obrigatória é o caminho para a imagem PNG que será processada. As entradas seguintes devem ser os *kernels* de convolução, no formato `h1` até `h11`. Ao final da execução, a imagem resultante será exibida na tela. Por exemplo, o comando abaixo apresenta a figura 2 em uma nova janela gráfica.

```
$ python3 main.py imagens/baboon.png h1 h2
```

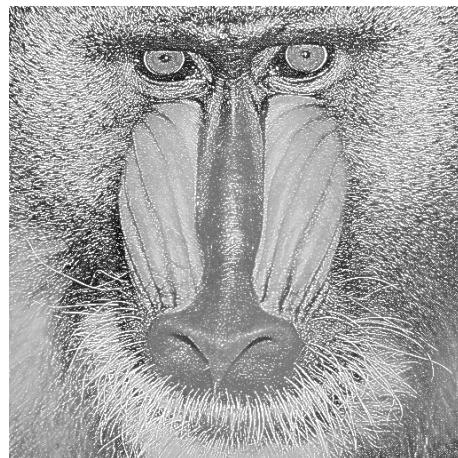


Figura 2: Aplicação de alguns processamentos na `baboon.png`.

Além das entradas posicionais, existem alguns argumentos opcionais, que podem ser vistos com `$ python3 main.py --help`. A mais importante das opções é `--output`, ou `-o`, que salva o resultado em um arquivo PNG em vez de exibir na tela. Se é desejável tanto a exibição da imagem quanto o salvamento no arquivo, o argumento `--force-show` ou `-f` pode ser usado. As outras opções são referentes ao *kernel* de convolução e serão descritas nas seções 3 e 4.

³ Mypy: Optional Static Typing for Python. URL: <http://mypy-lang.org/>.

3 Implementação

3.1 Teoria: Correlação e Convolução

A correlação é uma operação em que os pesos são aplicados na ordem em que aparecem visualmente. Assim, considerando o produto Hadamard (\cdot), a correlação (\circ) de uma região R da imagem por uma máscara M (de dimensões $N \times N$) será:⁴

$$R \circ M = \sum_{i=1}^N \sum_{j=1}^N R_{i,j} M_{i,j} = \sum_{i=1}^N \sum_{j=1}^N (R \cdot M)_{i,j}$$

Portanto, a implementação dessa etapa em NumPy poderia ser:

```
def correlacao_Regiao(R: np.ndarray, M: np.ndarray) -> float:
    return np.sum(R * M)
```

Podemos ver isso mais visualmente com o seguinte exemplo de um filtro 3×3 em uma região genérica.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 \\ 2 & 3 & 1 \\ 0 & 0 & 0 \end{bmatrix} = b + 2d + 3e + f$$

Apesar disso, a problema pedia a implementação de uma convolução. Nesse processamento a ordem dos elementos de um dos sinais era percorrido de forma contrária, de modo que os sinais fossem combinados na mesma ordem em que aparecem visualmente. Podemos ver isso na figura 3, onde a reflexão do eixo em $g(-\tau)$ faz com que os primeiros pontos de $g(t)$ sejam combinados antes, isto é, $g(t = 1)$ aparece antes de $g(t = 4)$ na varredura apresentada nos três últimos gráficos.

Com a mesma região R e máscara M acima, a convolução discreta resulta em.⁵

$$R * M = \sum_{i=1}^N \sum_{j=1}^N R_{i,j} M_{N-i,N-j} \neq R \circ M$$

Observando o exemplo matricial anterior, teremos o seguinte comportamento.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 2 & 3 & 1 \\ 0 & 0 & 0 \end{bmatrix} = d + 3e + 2f + h = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 3 & 2 \\ 0 & 1 & 0 \end{bmatrix} \quad (1)$$

A convolução é comumente usada devido a familiaridade das vastas propriedades dessa operação. Por outro lado, a correlação também aparece em vários processamentos de imagem graças à sua aproximação com a visualização de matrizes.

⁴ David Jacobs. *Correlation and Convolution*. URL: <https://www.cs.umd.edu/~djacobs/CMSC426/Convolution.pdf>.

⁵ Ibid.

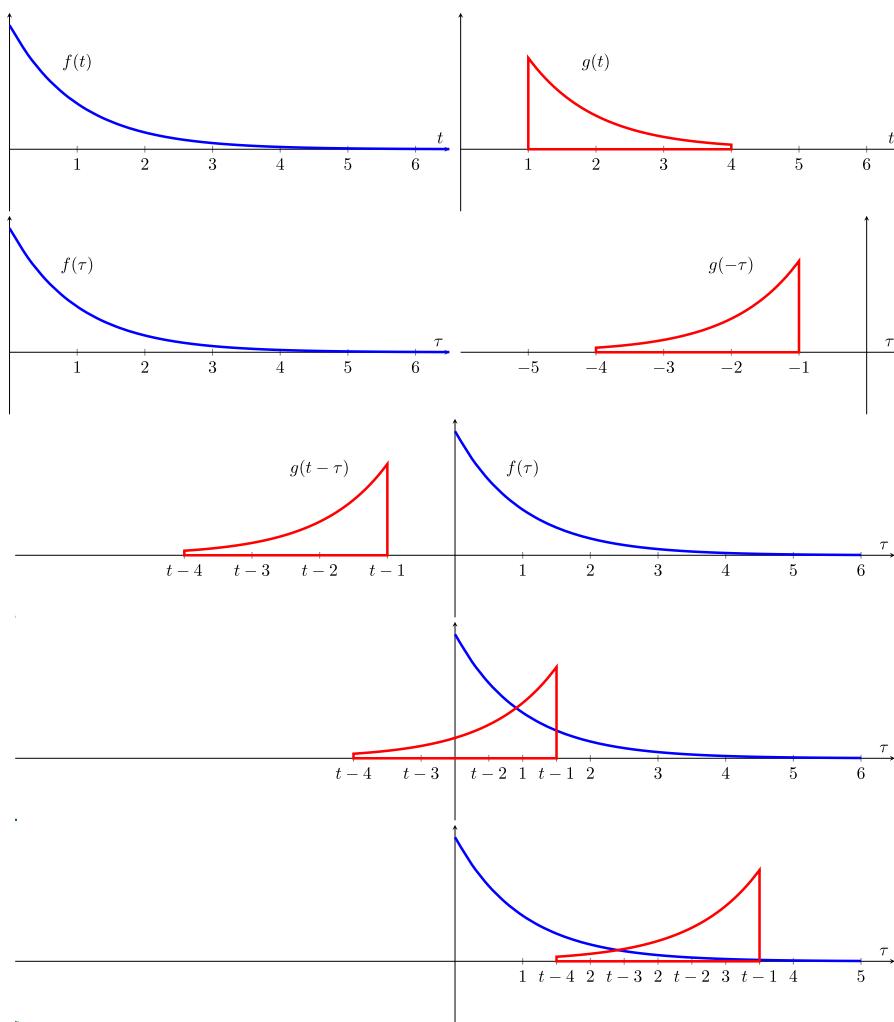


Figura 3: Exemplo visual da convolução em sinais unidimensionais contínuos.

3.2 Código da Convolução

A convolução foi implementada em dois *beckends* distintos, com as funções `convolve` do SciPy⁶ e `filter2D` do OpenCV.⁷ A função do SciPy realiza uma convolução, como definida matematicamente, e foi implementada como no código 3.1, ignorando o tratamento das bordas.

```
import numpy as np
from scipy import ndimage

def scipy_convolve(input: Image, kernel: Kernel, borda: ...) -> np.ndarray:
    # mudança para float
    input = input.astype(np.float64)
    # convolução
    output = ndimage.convolve(input, kernel, ...)
    return output
```

Código 3.1: Convolução com o SciPy, sem o tratamento de bordas.

⁶ *scipy.ndimage.convolve – SciPy v1.5.3 Reference Guide*. URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.convolve.html> (acesso em 20/10/2020).

⁷ *OpenCV: Image filtering – cv::filter2D*. URL: https://docs.opencv.org/4.5.0/d4/d86/group__imgproc__filter.html#ga27c049795ce870216ddfb366086b5a04.

A biblioteca OpenCV não tem uma operação de convolução, considerando a definição formal. Assim, a função `cv2.filter2D` faz na realidade uma correlação.

Como podemos ver na equação (1), as duas operações são similares, bastando reverter a posição dos elementos de uma das matriz. A máscara normalmente é bem menor em relação à imagem, então ela que foi invertida. No caso do OpenCV, esse tipo de inversão das posições, de (i, j) para $(N - i, N - j)$, pode ser feito pelo `cv2.flip`,⁸ com o argumento `flipCode` negativo. A implementação da convolução, nesse caso, ficou como no código 3.2 .

```
import numpy as np
import cv2

def opencv_convolve(input: Image, kernel: Kernel, borda: ...) -> np.ndarray:
    # flip do kernel para a correlação equivalente
    kernel = cv2.flip(kernel, flipCode=-1)
    # convolução
    output = cv2.filter2D(input, cv2.CV_64F, kernel, ...)
    return output
```

Código 3.2: Convolução com o OpenCV, sem o tratamento de bordas.

Para selecionar entre os *backends*, existem dois argumentos opcionais. O `--scipy` garante a execução com a biblioteca SciPy, por mais que já seja o padrão. Podemos então conseguir a mesma imagem da figura 2 com o comando:

```
$ python3 main.py imagens/baboon.png h1 h2 --scipy
```

Do mesmo modo, o OpenCV pode ser escolhido com o argumento `--opencv`.

```
$ python3 main.py imagens/baboon.png h1 h2 --opencv
```

Os dois *backends* não apresentam diferenças significativas nos resultados.

3.3 Tratamento de Bordas

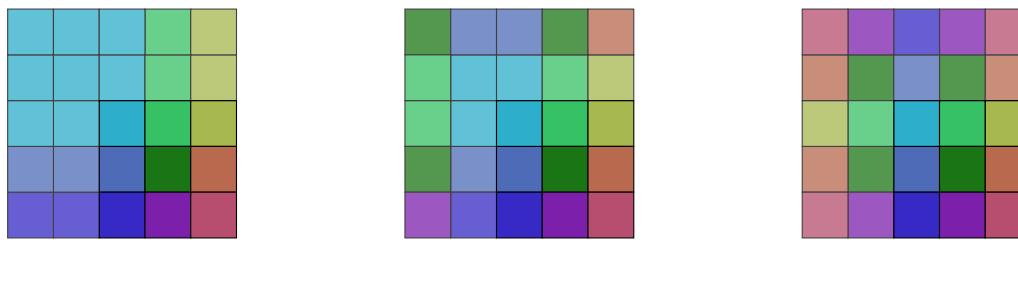


Figura 4: Argumentos válidos para `--borda` ou `-b`.

Para que a convolução possa ser feita nas bordas da imagem, existem vários modos de tratamento. Na ferramenta foram implementadas três formas de dentre as várias possíveis: a extensão do último pixel (4a), a reflexão dos pixels de borda (4b) e a mesma reflexão, mas sem repetir o pixel mais externo (4c).

⁸ OpenCV: Operations on arrays – `cv::flip`. URL: https://docs.opencv.org/4.5.0/d2/de8/group__core__array.html#gaca7be533e3dac7feb70fc60635adf441.

A flag `--borda`, ou `-b`, serve para controlar o tratamento de borda. As opções devem ser passadas como aparecem na figura 4. Por padrão, o tratamento é feito como na figura 4c, como se fosse a opção `-b reflexao_pula_ultimo`. Ambos *backends*, SciPy e OpenCV, funcionam com os três tipos de bordas.

Apesar de interessante, e em alguns casos importante, o três tratamentos de bordas não alteram muito no resultado da convolução. A razão disso é que os *kernels* desse trabalho são relativamente pequenos.

3.4 Discretização

Toda o processo de convolução é feito com operações de ponto flutuante, buscando evitar *overflow* e problemas de arredondamento. Então, para que a matriz volte a representar uma imagem, é preciso discretizar os valores, para os níveis 0 a 255.

A forma mais comum é arredondando os valores para os inteiros mais próximos no intervalo [0, 255]. Dessa forma, os valores negativos se tornam 0 e valores maiores que 255 vão para 255. No código, esse método foi implementado na função `lib.trunca`. Essa é a opção padrão do programa.

Uma forma alternativa também foi implementada, baseada no mapeamento linear do menor valor da imagem para 0 e do maior para 255. Na ferramenta em Python, esse método de discretização está implementado na função `lib.transforma_limites` e pode ser selecionada com a opção `-t` na linha de comando. Essa opção não foi muito utilizada neste relatório.

3.5 Combinação das Imagens

Vários filtros podem ser passados como argumento, como no exemplo da seção 2.2. Os filtros são aplicados, um por vez, na mesma imagem de entrada, discretizados e só então combinados em uma imagem final. A combinação é feita pela raiz da soma quadrática, em ponto flutuante, e discretizada novamente.

Para padronizar a implementação, a etapa de combinação é feita mesmo com apenas uma imagem. Por causa disso, a discretização é feita antes e depois da combinação, mantendo o resultado esperado da convolução. No entanto, isso pode ser alterado com a opção `-n`, fazendo com que a discretização seja aplicada apenas no final.

Para um filtro apenas, isso faz com que a convolução seja tratada pelo valor absoluto, fazendo as imagens ficarem com regiões mais claras, onde anteriormente seria preto. Esse modo não foi utilizado neste relatório.

4 Resultados

4.1 Blur (h₂ e h₆)

Os filtros mais simples de identificar foram os de *blur*, que fazem um tipo de média ponderada da vizinhança do pixel.

1	4	6	4	1
4	6	24	6	4
6	24	36	24	6
4	6	24	6	4
1	4	6	4	1

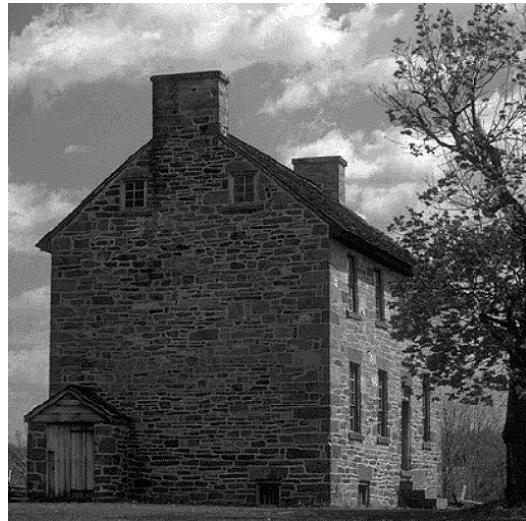
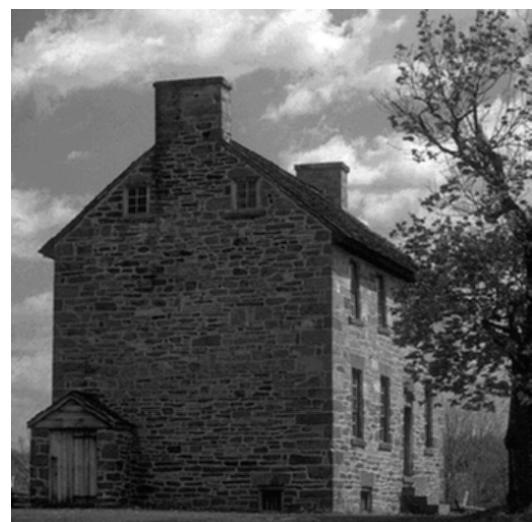
(a) h_2

1	1	1
1	1	1
1	1	1

(b) h_6

Figura 5: Máscaras de *blur*.

O primeiro deles é o filtro gaussiano 5×5 ,⁹ que pode ser visto na figura 5a. Essa matriz é ponderada de acordo com as distâncias 4 do pixel central, reduzindo o peso para pixels mais distantes. No espaço das frequências, esse filtro funciona como um passa-baixas, reduzindo a influência de frequências muito altas, o que pode servir para melhorar o resultado de outros filtros, como o laplaciano, ou para evitar problemas de *aliasing*.¹⁰ Na figura 6b, podemos ver o resultado de aplicar esse filtro na imagem `house.png` (6a).

(a) Original: `house.png`.(b) Convolução com h_2 (5a).(c) Convolução com h_6 (5b).Figura 6: Aplicação do *blur*.

O segundo filtro é chamada de *box blur*, que faz uma média simples da vizinhança 8 do pixel. Isso pode ser visto no *kernel* do filtro (figura 5b). Apesar de ser diferente do gaussiano, o resultado ficou bem parecido, como pode ser visto na figura 6. O motivo disso é que os pesos mais externos do filtro gaussiano 5×5 são bem baixos, fazendo com a região 3×3 seja mais presente, sendo essa a mesma região do filtro *box*.

No programa, esses filtros podem ser aplicados com:

```
$ python3 main.py imagens/house.png h2 # ou h6
```

⁹ *Spatial Filters – Gaussian Smoothing*. URL: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm> (acesso em 22/10/2020).

¹⁰ *Gaussian Blur – Wikipedia*. URL: https://en.wikipedia.org/wiki/Gaussian_blur#Low-pass_filter.

4.2 Motion Blur (h9)

Além dos filtros de *blur* apresentados na seção anterior, temos também um filtro de *blur* direcionado, muitas vezes usado para alcançar um efeito de *motion blur*. Para o *kernel* da figura 7, o *blur* está direcionado a 45° , como se a câmera estivesse se movendo da esquerda-superior para a direita-inferior, ou vice-versa.

$$\frac{1}{9} \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

Figura 7: Máscara de *motion blur*: h_9 .



(a) Original: `city.png`.



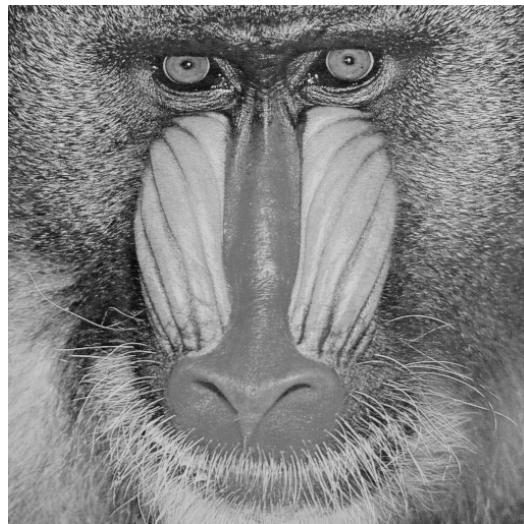
(b) Convolução com h_9 (7).

Figura 8: Aplicação de *motion blur*.

Para a execução pelo programa, o comando deve ser no formato abaixo, resultando em algo como a figura 8b.

```
$ python3 main.py imagens/city.png h9
```

4.3 Detecção de Borda (h_1 e h_5)



(a) Original: baboon.png.

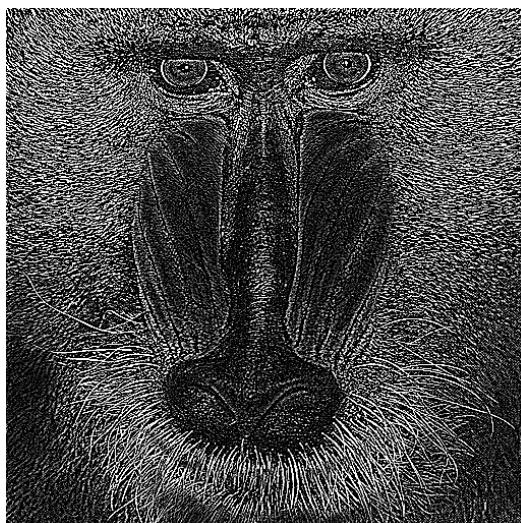
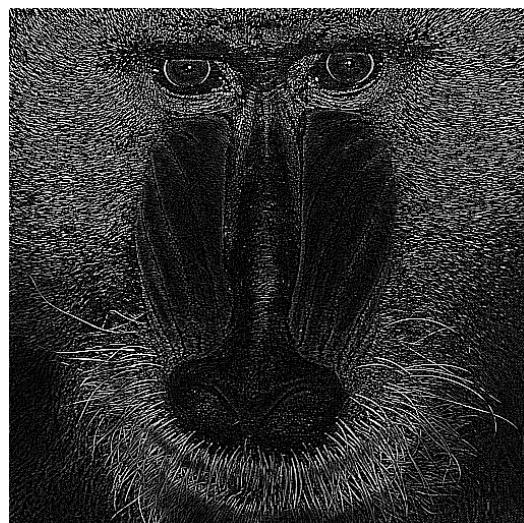
(b) Convolução com h_1 (10a).(c) Convolução com h_5 (10b).

Figura 9: Aplicação do laplaciano discreto.

Os filtros de detecção ou realce de bordas mais comuns são os chamados não-direcionais. Normalmente, eles são desenvolvidos a partir do operador laplaciano discreto.¹¹ Uma característica importante dos filtros de detecção de bordas é que seus elementos somam zero.

O operador laplaciano serve como uma medida de quanto uma função espacial no ponto p é diferente da média dos pontos em um raio $[p - \delta r, p + \delta r]$. No caso discreto bidimensional, como imagens, existem duas principais medidas de distância, considerando as vizinhanças 4 e 8. Elas resultam em dois tipos de laplacianos diferentes, como pode ser visto na figura 10.

Nesse trabalho temos o *kernel* h_1 (10a), com vizinhança 4 de raio 2, e o h_5 , com vizinhança 8 de raio 1. Os resultados podem ser vistos nas figuras 9b e 9c, respectivamente. Podemos ver que as bordas nas regiões de baixa frequência são bem detectadas, como no centro da imagem, mas regiões de alta frequência resultam em muita informação e podem acabar atrapalhando a análise das bordas. Uma forma de contornar esse problema é aplicando um filtro passa-baixas, como o *blur gaussiano*, discutido na seção 4.1.

¹¹ *Spatial Filters – Laplacian / Laplacian of Gaussian*. URL: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm> (acesso em 22/10/2020).

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

(a) h_1

-1	-1	-1
-1	8	-1
-1	-1	-1

(b) h_5

Figura 10: Filtros laplacianos discretos

A execução pode ser feita com:

```
$ python3 main.py imagens/baboon.png h1
# ou
$ python3 main.py imagens/baboon.png h5
```

4.4 Operador de Sobel (h_3 e h_4)

Além dos detectores de borda não-direcionais, existem detectores que se importam com qual direção está sendo feita a análise. Muitos desses operadores são baseados na noção de gradiente, que representa matematicamente o conceito de variação.

Dentre eles, existe o operador de Sobel,¹² que faz o gradiente nas direções x e y do plano cartesiano. Em específico, considerando a operação de convolução, o filtro h_3 (11a) aponta na direção $-x$ (para a esquerda) e o h_4 (11b), na $-y$ (para cima).

-1	0	1
-2	0	2
-1	0	1

(a) h_3

-1	-2	-1
0	0	0
1	2	1

(b) h_4

Figura 11: Máscaras de Sobel.

Observando os resultados na figura 12, podemos ver que quase não existe separação do corpo da borboleta com as asas na figura 12d, isso é porque ela quase não apresenta variações verticais nessa região. Nessa mesma figura, no entanto, podemos ver claramente a separação do topo da asa, que não acontece na figura 12c, já que é uma região com pouca variação horizontal.

Além disso, podemos ver a presença do direcionamento $-x$ em vez do $+x$ na 12c. Em especial, na parte inferior da asa esquerda, logo abaixo do abdômen do inseto, é bem visível a borda do componente, o que não acontece na mesma região da asa direita. No tórax da borboleta também fica visível uma separação bilateral, que não é aparente na imagem original (12a).

Muitas vezes, os operadores de Sobel são combinados para extrair informações importantes do gradiente. O mais comum, é o módulo da gradiente, que no caso é encontrado por $\sqrt{(h_3)^2 + (h_4)^2}$. Ele serve como um bom detector de borda, medindo o quanto forte é a variação em cada região da imagem. O resultado pode ser visto na figura 12b.

¹² Sobel operator – Wikipedia. URL: https://en.wikipedia.org/wiki/Sobel_operator.



(a) Original: butterfly.png.

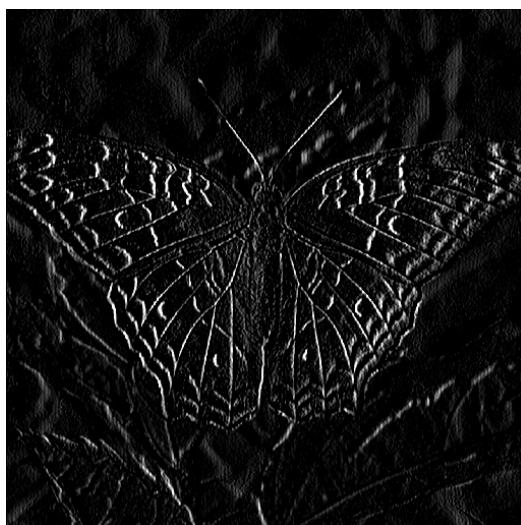
(b) Convolução com h_3 e h_4 .(c) Convolução com h_3 (11a).(d) Convolução com h_4 (11b).

Figura 12: Aplicação dos operadores de Sobel.

A aplicação do operador de Sobel, combinado nas duas direções, pode ser feita no programa por:

```
$ python3 main.py imagens/butterfly.png h3 h4
```

4.5 Operador de Prewitt (h11)

Além do operador de Sobel, existem vários outros operadores baseados em gradiente. Um deles é o de Prewitt,¹³ caracterizado por dois *kernels* 3×3 , com apenas 0, 1 e -1 . No caso da figura 13a, o gradiente está apontando para a direção $(-1, -1)$, isto é, da direita-inferior para a esquerda-superior.

Na figura 13b, podemos ver claramente o direcionamento pelas antenas da borboleta. A antena esquerda, que está alinhada com o filtro, fica quase indetectada pelo filtro, enquanto a antena direita aparece de forma bem marcada. Além disso, na antena direta, apenas uma das bordas fica marcada, isso porque o filtro não depende apenas da angulação, mas do direcionamento positivo ou negativo naquele ângulo.

¹³ Prewitt operator – Wikipedia. URL: https://en.wikipedia.org/wiki/Prewitt_operator.

O operador de Prewitt, quando usado com direções ortogonais, também pode ser combinado como o de Sobel (seção 4.4).

-1	-1	0
-1	0	1
0	1	1

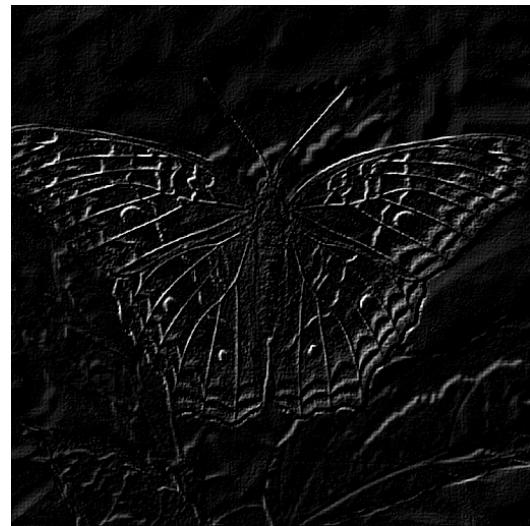
(a) Máscara h_{11} .(b) Convolução com h_{11} .

Figura 13: Operador de Prewitt, com exemplo.

A aplicação do filtro pode ser feita por:

```
$ python3 main.py imagens/butterfly h11
```

4.6 Detecção de Linha (h_7 e h_8)

Um último tipo de detecção de borda apresentado neste trabalho são os detectores de linha.¹⁴ Assim como os operadores baseado em gradiente (seções 4.4 e 4.5), esses filtros dependem de uma certa angulação. No entanto, o direcionamento dentro daquela angulação não é importante. Em especial, o filtro h_7 (14a) detecta diagonais na linha $y = -x$ e o h_8 (14b), na linha $y = x$.

-1	-1	2
-1	2	-1
2	-1	-1

(a) h_7

2	-1	-1
-1	2	-1
-1	-1	2

(b) h_8

Figura 14: Máscaras de detecção de linhas.

Podemos ver a diferença com os dois tipos de detectores direcionais comparando com operador Prewitt (seção 4.5). No caso, as listras das asas da borboleta (veja a original, figura 12a), ficam com a duas bordas bem aparentes. Na figura 15a, isso acontece na asa esquerda, enquanto na figura 15b, é na asa direita. Agora no operador Prewitt (figura 13b), isso não fica aparente em nenhuma das asas, mostrando muitas vezes apenas umas das bordas de cada componente.

¹⁴ Line Detection. URL: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/linedet.htm> (acesso em 22/10/2020).

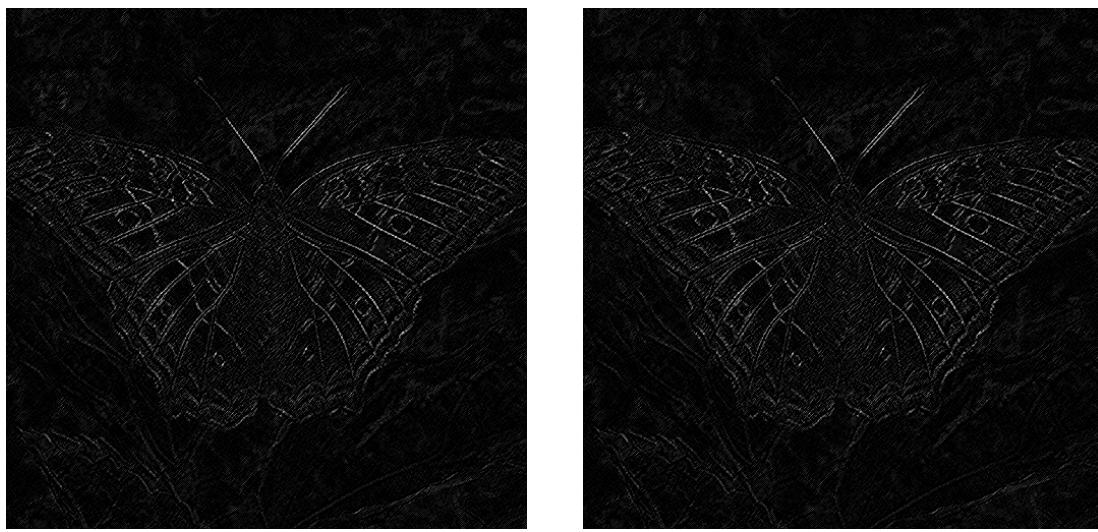
(a) Convolução com h_7 (14a).(b) Convolução com h_8 (14b).

Figura 15: Aplicação dos filtros de detecção de linhas.

A aplicação desse filtro pode ser feita com:

```
$ python3 main.py imagens/butterfly h7
# ou
$ python3 main.py imagens/butterfly h8
```

4.7 Nitidez (h_{10})

O último filtro deste trabalho é o de nitidez, também chamado de *sharpen* ou agudização. No espaço das frequências, esse tipo de filtro é responsável por enfraquecer frequências mais baixas, por isso é chamado de filtro passa-altas. Isso faz com que esse filtro seja um tipo de processo "inverso" do filtro de *blur* (seção 4.1).

Assim como os filtros de *blur*, filtros de nitidez têm a soma de seus elementos igual a 1. Isso faz com que o aspecto geral das intensidades da imagem seja mantido. Outra característica interessante desses filtros, é que eles são basicamente uma soma do resultado de um filtro de detecção de borda não direcional com a imagem original. Podemos ver isso comparando as figuras 17e, que é a aplicação do laplaciano em *city.png*, e 17f, a subtração de figura 17d por 17a.

$$\frac{1}{8} \begin{array}{|c|c|c|c|c|} \hline -1 & -1 & -1 & -1 & -1 \\ \hline -1 & 2 & 2 & 2 & -1 \\ \hline -1 & 2 & 8 & 2 & -1 \\ \hline -1 & 2 & 2 & 2 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 \\ \hline \end{array}$$
Figura 16: Máscara de nitidez: h_{10} .



(a) Original: city.png.

(b) Convolução com h_2 , depois com h_{10} .(c) Convolução com h_2 (5a).(d) Convolução com h_{10} (16).(e) Convolução com h_5 (10b).(f) Convolução com h_{10} , removido da original.Figura 17: Filtros de *blur* e de nitidez.

Podemos ver nas figuras 17c e 17d, comparando com a original (17a), que o *blur* (h_2) reduz a nitidez, enquanto o *sharpen* (h_{10}) aumenta. Aplicando o filtro h_2 e o h_{10} , nessa ordem, resulta

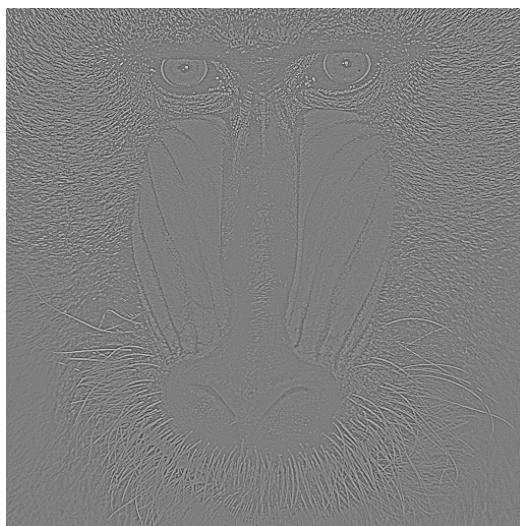
na figura 17b, que tem nitidez comparável com a original.

A execução desse filtro é pode ser feita com:

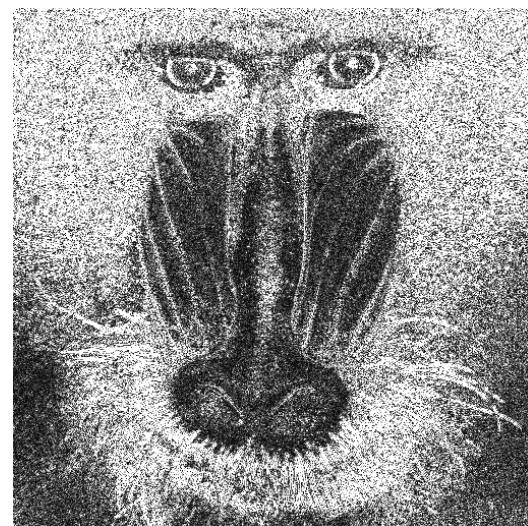
```
$ python3 main.py imagens/city.png h10
```

4.8 Argumentos Opcionais

Na figura 18 podemos ver o resultado da mesma operação da figura 9b, que é o filtro h_1 da seção 4.3, mas com os argumentos opcionais $-t$ (seção 3.4) e $-n$ (seção 3.5). Os resultados acabaram ficando mais difíceis de analisar, por isso as opções padrões foram utilizadas em todo o projeto.



(a) Opção $-t$.



(b) Opção $-n$.

Figura 18: Resultados com os argumentos opcionais.

5 Conclusões

Neste trabalho podemos ver como convoluções de filtros espaciais simples podem ser usados para extrair algumas informações importantes da imagem e para aplicações de efeitos de imagem.

Podemos ver diferentes tipos de detecção de borda, baseado em técnicas matemáticas diferentes. Encontramos aqui também como detectar linhas na imagem (seção 4.6), medições de gradiente em direções diferentes (seções 4.4 e 4.5) e aplicação do laplaciano discreto (seção 4.3).

Por outro lado, conseguimos também alguns efeitos interessantes na imagem, como o *blur* (seção 4.1), efeitos de movimentação (seção 4.2) e agudização da imagem (seção 4.7).