

Universidade Estadual de Campinas - UNICAMP
Instituto de Computação
Introdução ao Processamento de Imagem Digital (MC920 / MO443)

Trabalho 1

Aluno: Pedro Brasil Barroso - RA 260637

Professor: Dr. Hélio Pedrini

Campinas
1º Semestre de 2025

Sumário

1	Introdução	2
2	Estrutura do trabalho	2
2.1	Requisitos	2
2.2	Organização dos arquivos	2
2.3	Como executar os exercícios	3
3	Exercícios	3
3.1	Esboço a Lápis	3
3.1.1	Implementação inicial	3
3.1.2	Experimentos	4
3.1.3	Discussão dos resultados e limitações	6
3.2	Ajuste de Brilho	6
3.3	Mosaico	8
3.3.1	Implementação e resultado	8
3.3.2	Limitações	9
3.4	Alteração de Cores	10
3.4.1	Explicação do conceito	10
3.4.2	Implementação e resultado	10
3.5	Transformação de Imagens Coloridas	11
3.5.1	Item (a)	11
3.5.2	Item (b)	12
3.6	Planos de Bits	13
3.7	Combinação de Imagens	15
3.7.1	Implementação e resultado	15
3.7.2	Limitações	15
3.8	Transformação de Intensidade	16
3.8.1	Item (b)	16
3.8.2	Item (c)	16
3.8.3	Item (d)	17
3.8.4	Item (e)	17
3.8.5	Item (f)	17
3.8.6	Resultado final	18
3.9	Quantização de Imagens	18
3.9.1	Implementação	18
3.9.2	Resultado	19
3.10	Filtragem de Imagens	20
3.10.1	Implementação	20
3.10.2	Resultados e discussão	21
4	Conclusão	32
5	Referências	32

1 Introdução

O objetivo deste trabalho é aplicar e analisar técnicas básicas de processamento digital de imagens, explorando suas possibilidades de implementação, suas limitações práticas e os fundamentos por trás das transformações realizadas.

2 Estrutura do trabalho

2.1 Requisitos

As versões de Python e das bibliotecas utilizadas no projeto são as que seguem:

- Python: 3.12.3
- NumPy: 2.2.4
- OpenCV: 4.11.0.86

2.2 Organização dos arquivos

O arquivo .zip submetido apresenta a seguinte organização:

- images/
Contém as imagens de entrada utilizadas nos exercícios.
- results/
Armazena as imagens processadas (resultados).
- helper_functions.py
Script com funções auxiliares, que permitem salvar, ler e obter o caminho de imagens.
- arrays.py
Script com a definição das matrizes utilizadas nos exercícios 4, 5 e 10, as quais podem ser carregadas por meio das funções `get_array` e `get_kernels`.
- run.py
Script principal que executa os exercícios. Permite especificar quais imagens e exercícios executar, além de opções para salvar e/ou exibir os resultados (explicado na seção 2.3).
- requirements.txt
Lista de dependências do projeto (bibliotecas Python necessárias).

2.3 Como executar os exercícios

O script `run.py` é responsável pela execução dos exercícios presentes nesse trabalho. Ele deve ser executado via terminal com os seguintes argumentos:

- `-i`: Lista de nomes das imagens (localizadas no diretório `images`) a serem processadas, ou `all` para usar todas as imagens da pasta. Deve ser incluída a extensão ao fim do nome da imagem. Caso a lista não seja especificada, são utilizadas imagens padrão.
- `-e`: Lista dos números dos exercícios a serem executados (de 1 a 10). Caso a lista não seja especificada, todos os exercícios são executados.
- `-s`: Salva as imagens resultantes no diretório `results`. As imagens são salvas seguindo o padrão: `<numero_exercicio>/ex<numero>_<titulo>[_<identificador>].png`
Exemplos: ‘01/ex01_esboco_lapis.png’, ‘04/ex04.Alteracao_cores_1.png’
- `-d`: Exibe as imagens processadas. Para interromper o programa, basta pressionar ”q”; para interromper a execução do exercício atual e iniciar a do próximo, ”n”, e para exibir a próxima próxima imagem, qualquer outra tecla.

Exemplo de uso:

```
python run.py -i imagem1.png imagem2.png -e 1 4 5 -s -d
```

Esse comando executa os exercícios 1, 4 e 5 nas imagens `imagem1.png` e `imagem2.png`, salvando e exibindo os resultados.

O exercício 7 (Combinação de Imagens) é uma exceção a esse padrão, pois recebe duas imagens como entrada. Nesse caso, as imagens listadas serão combinadas duas a duas na ordem em que foram especificadas (ou uma imagem padrão quando o tamanho da lista for ímpar). Por exemplo: usar `-i image1.png image2.png image3.png image4.png` fará as combinações de `image1.png` com `image2.png` e de `image3.png` com `image4.png`.

3 Exercícios

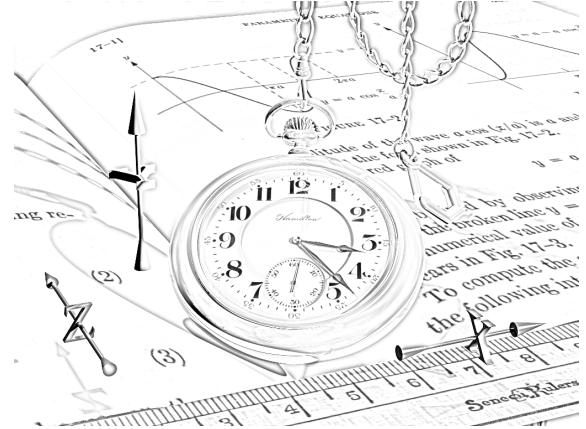
3.1 Esboço a Lápis

3.1.1 Implementação inicial

Inicialmente, foi realizado o procedimento recomendado no enunciado, utilizando `watch.png` com uma máscara de 21x21 pixels para o desfoque gaussiano e um σ (desvio padrão) de 0 em ambos os eixos, uma vez que, segundo a documentação da biblioteca OpenCV, isso faz com que o desvio padrão seja calculado a partir da fórmula $\sigma = 0,3 * ((ksize - 1) * 0,5 - 1) + 0,8$, onde `ksize` é o tamanho da máscara (portanto, $\sigma = 3,5$). O resultado foi o esperado, como pode ser observado na Figura 1.



(a) Imagem original



(b) Imagem após processamento

Figura 1 – Esboço de lápis da imagem `watch.png`

Contudo, essa solução, apesar de simples, não é capaz de detectar bordas em regiões de baixa frequência, uma vez que o filtro de desfoque gaussiano se comporta como um filtro passa-baixas. Um exemplo disso pode ser visto na Figura 2, onde as bordas das frentes de avanço e de recuo das ondas possuem contornos menos intensos, assim como na região entre as ondas e o céu.



(a) Imagem original



(b) Imagem após processamento

Figura 2 – Esboço de lápis da imagem `waves.png`

3.1.2 Experimentos

Visando compreender o problema citado acima, foram testados diferentes kernels e valores de σ para analisar o comportamento desses parâmetros. Primeiro, foi fixado $\sigma = 5$ e então testados diferentes tamanhos de máscara; em seguida, foi fixada uma máscara de 21x21 e testados diferentes valores de σ .

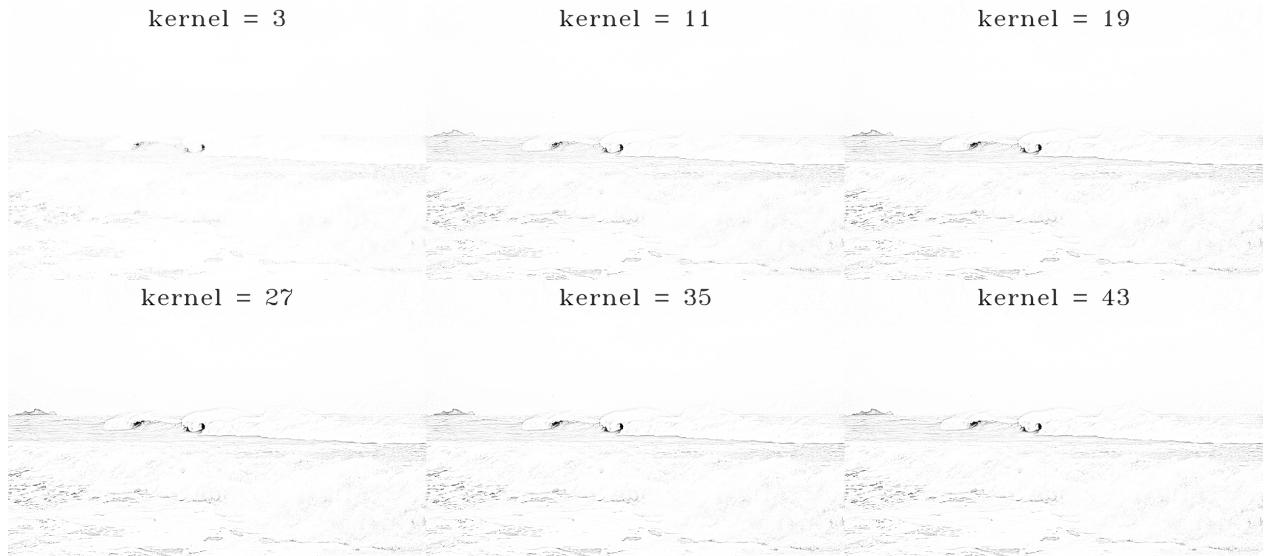


Figura 3 – Esboço de lápis gerado a partir de diferentes tamanhos de máscara (o título $kernel = x$ indica que foi usada uma máscara de dimensões (x, x))

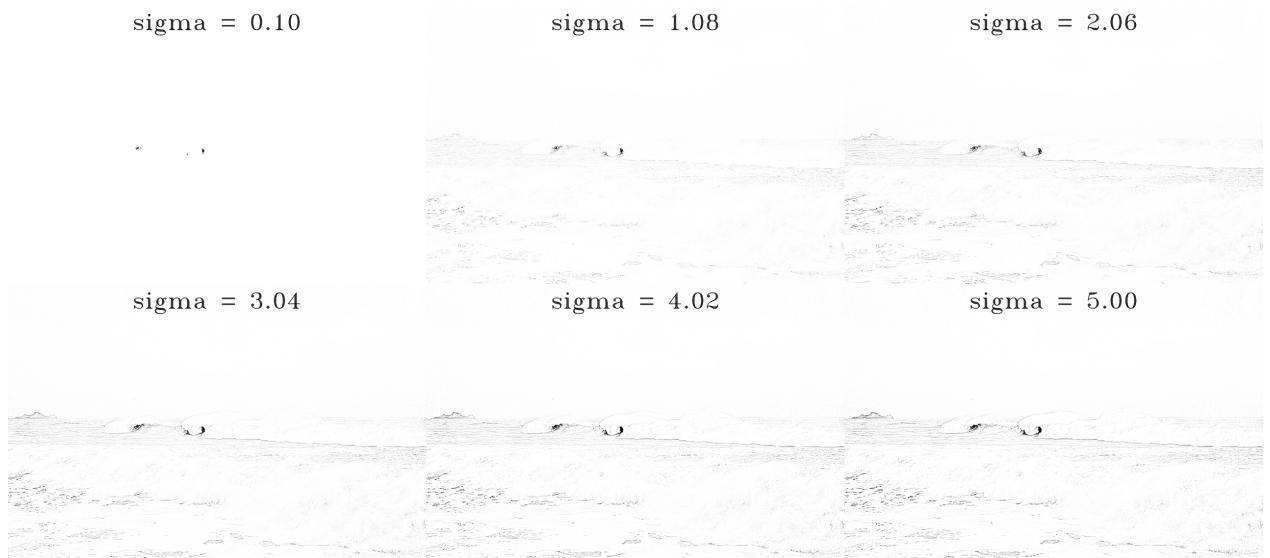


Figura 4 – Esboço de lápis gerado a partir de diferentes valores de σ

É evidente, pela visualização das Figuras 3 e 4, que o aumento dos parâmetros resulta no reforço dos contornos.

Contudo, também foi constatado que manter um dos parâmetros fixo resulta em retornos decrescentes. Claramente, dado um aumento do tamanho da máscara sem aumento proporcional de σ , os valores mais distantes do centro terão pesos desprezíveis devido à distribuição Gaussiana, tornando o aumento da máscara ineficaz. Por outro lado, aumentar σ sem ampliar a máscara faz com que os valores da máscara se tornem cada vez mais semelhantes entre si, reduzindo o efeito do filtro e tornando as diferenças insignificantes.

Após alguns testes utilizando $\sigma = \frac{\sqrt{n-1}}{2}$ com diferentes valores de n , onde n é a dimensão da máscara $n \times n$, observou-se um efeito característico do aumento desse parâmetro, conforme ilustrado na Figura 5. Com máscaras grandes, intensifica-se o efeito de sombra próximo às bordas, resultante da maior suavização da imagem.

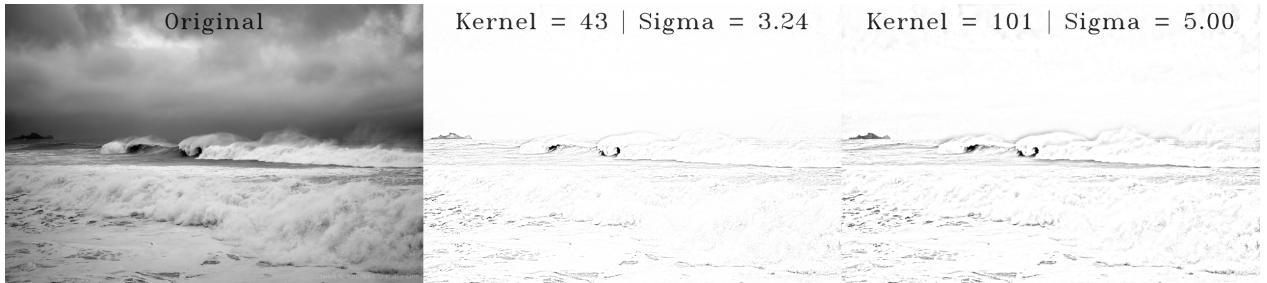


Figura 5 – Esboço de lápis gerado a partir de diferentes tamanhos de máscara e valores de σ

3.1.3 Discussão dos resultados e limitações

Embora a biblioteca OpenCV lide automaticamente com o cálculo de σ para gerar a imagem suavizada, o efeito de contorno de lápis gerado a partir desse método pode não ser tão evidente quanto desejado devido às características da imagem. Uma possibilidade não explorada aqui é o cálculo dinâmico desses parâmetros, o que poderia ser feito com base no histograma da imagem. Outra possibilidade é a equalização do histograma seguida da utilização de parâmetros pré-definidos.

Além disso, como é possível observar nas Figuras 3 e 4, os aumentos da dimensão da máscara e do desvio padrão resultam em contornos mais destacados. Isso ocorre por conta da maior redução de detalhes de alta frequência (o filtro Gaussiano se comporta como um filtro passa-baixas), enfatizando apenas as transições mais significativas entre regiões claras e escuras da imagem. Assim, a divisão nas regiões de alta frequência resulta em valores maiores.

Vale ressaltar que o código submetido utiliza uma máscara de tamanho 21×21 e $\sigma = 3,5$, uma vez que esses valores produziram o efeito desejado para todas as imagens submetidas, salvo alguns detalhes como no caso da imagem `waves.png`. Ademais, na função `cv.divide`, foi utilizada escala 255, pois, segundo a documentação da biblioteca OpenCV, o cálculo realizado é $\text{saturate}(\text{img1} * \text{scale}/\text{img2})$, i.e., o resultado da divisão é multiplicado por 255 mas fica limitado entre 0 e 255.

Por fim, como não há verificação para o caso em que as dimensões da imagem são menores que as da máscara, essa situação pode resultar em comportamento indesejado.

3.2 Ajuste de Brilho

Esse exercício foi realizado seguindo os passos do enunciado: primeiro, ocorre uma normalização das intensidades dos pixels do intervalo $[0, 255]$ para $[0, 1]$, seguida da exponenciação por γ e, por fim, normalização de volta para o intervalo $[0, 255]$.

Para realizar uma normalização, seja $A_n = \{x \in \mathbb{R} | 0 \leq x \leq n\}$. Dados A_M e A_N , temos que a conversão de $m \in A_M$ para $n \in A_N$ será obtida por meio de:

$$\frac{M - 0}{N - 0} = \frac{m - 0}{n - 0} \iff n = \frac{m * N}{M}$$

Portanto, a conversão entre os intervalos A_{255} e A_1 pode ser obtida por meio de multiplicações e divisões. Para converter de A_{255} para A_1 , basta dividir os valores de intensidade da imagem por 255, e, para converter de A_1 para A_{255} , basta multiplicar os valores por 255.

Além disso, sabe-se que $(\forall x \in A_1)(\forall n \in \mathbb{R}_+)x^n \in A_1$, i.e., o resultado da exponenciação de um número menor que 1 também é menor que 1. Portanto, para $\gamma \geq 0$, se os valores de intensidade da imagem $F(x, y)$ estiverem no intervalo $[0, 1]$, $F(x, y)^{\frac{1}{\gamma}}$, também estará.

Sabendo disso, a lógica aplicada para gerar a imagem com ajuste de brilho $B(x, y)$, a partir da imagem original $A(x, y)$, foi:

$$B(x, y) = \lfloor 255 \cdot \left(\frac{A(x, y)}{255} \right)^{\frac{1}{\gamma}} \rfloor \quad (1)$$

O resultado dessa operação pode ser visto na Figura 6.

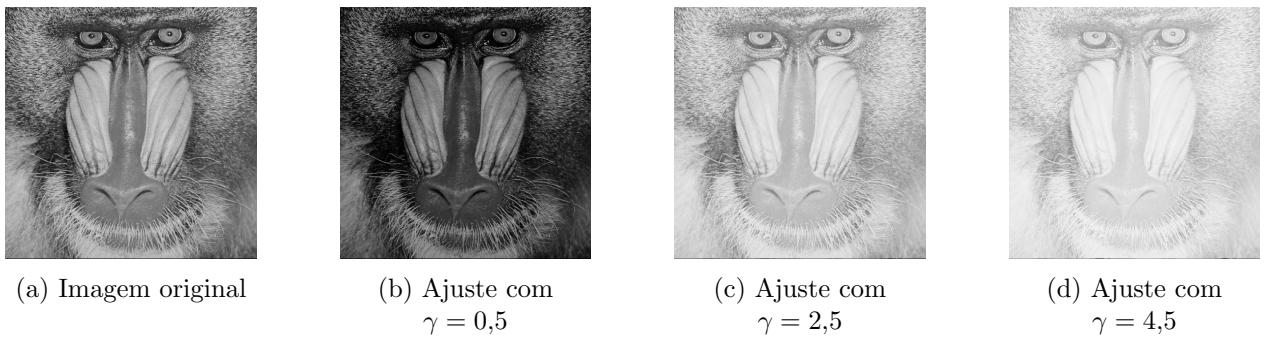


Figura 6 – Ajuste de brilho utilizando diferentes valores de γ na imagem `baboon_monocromatica.png`. A escolha de $\gamma < 1$ reduz o brilho da imagem, enquanto $\gamma > 1$ o aumenta.

Foi escolhida a função piso para transformar os valores de intensidade em inteiros na operação da equação 1; contudo, poderiam ter sido utilizadas as funções teto ou arredondamento. A depender da escolha, alguns pixels terão uma diferença absoluta de intensidade de 1 na escala de cinza. Como essa diferença é muito pequena, os resultados na imagem final não são visualmente perceptíveis, como pode ser observado na Figura 7.

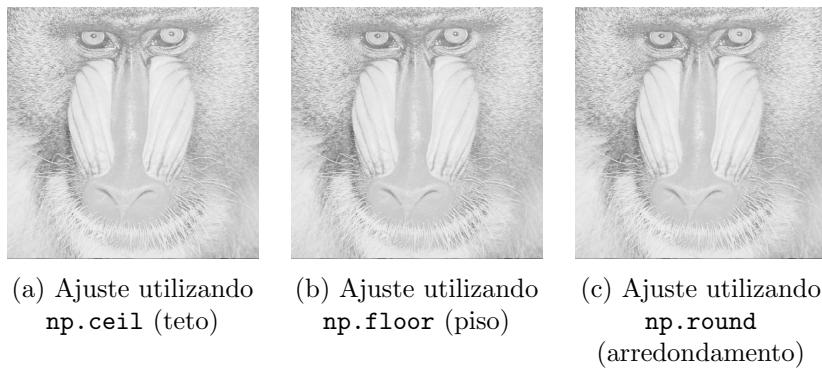


Figura 7 – Ajuste de brilho utilizando $\gamma = 3,5$ e diferentes estratégias de arredondamento na imagem `baboon_monocromatica.png`

3.3 Mosaico

3.3.1 Implementação e resultado

O método escolhido para representar as trocas de blocos em código foi a criação de um dicionário no qual os pares chave-valor equivalem a um mapeamento "bloco novo-bloco antigo":

```

1 mapping = {
2     0: 5,  1: 10,  2: 12, 3: 2,
3     4: 7,  5: 15,  6: 0,  7: 8,
4     8: 11, 9: 13, 10: 1, 11: 6,
5     12: 3, 13: 14, 14: 9, 15: 4,
6 }
```

Note que foi subtraído 1 da posição dos blocos nesse mapeamento, a fim de que seja possível calcular a posição da origem (pixel superior esquerdo) de cada bloco a partir das seguintes relações:¹

$$r = h_b * (i // 4), c = w_b * (i \% 4)$$

Onde i é o índice do bloco (iniciado em 0), h_b e w_b são a altura e a largura dos blocos, respectivamente, e $a = (r, c)$ é a posição da origem do bloco.

Dessa forma, a posição do ponto inferior direito do bloco pode ser definida como $b = (r + h_b, c + w_b)$. Conhecendo a e b , é possível obter um bloco rapidamente em código realizando um *slicing* na imagem. Assim, o resultado final é obtido a partir da iteração sobre os 16 pares do mapeamento, com *slicings* para posicionar os blocos em suas novas posições.

Vale ressaltar que a implementação não foi feita *inplace*: primeiro é inicializada uma imagem com zeros, a qual é alterada durante a iteração sobre o mapeamento, com base na imagem original. Caso contrário, um bloco i poderia receber o bloco j e, mais adiante, outro

¹Foram utilizados os símbolos $//$ para representar a divisão exata (quociente da divisão) e $\%$ para representar o módulo (resto da divisão).

bloco k poderia receber o bloco i . Desse modo, $i = k = j$, ou seja, haveria dois blocos iguais no resultado final.

O resultado obtido pode ser observado na Figura 8. As dimensões da imagem são 512x512.



Figura 8 – Mosaico da imagem `house.png`

3.3.2 Limitações

Essa implementação foi testada para imagens cujas dimensões não são múltiplos de 4, a fim de identificar se essa seria uma fonte de limitação. Foi observado que, nesses casos, alguns pixels das bordas direita e inferior são eliminados da imagem, uma vez que as dimensões dos blocos são definidas a partir de uma divisão exata daquelas da imagem original por 4 (por conta disso, no máximo 3 linhas e/ou colunas são removidas). Os pixels remanescentes são pretos, visto que o mosaico foi inicializado como um vetor de zeros, como pode ser observado na Figura 9.

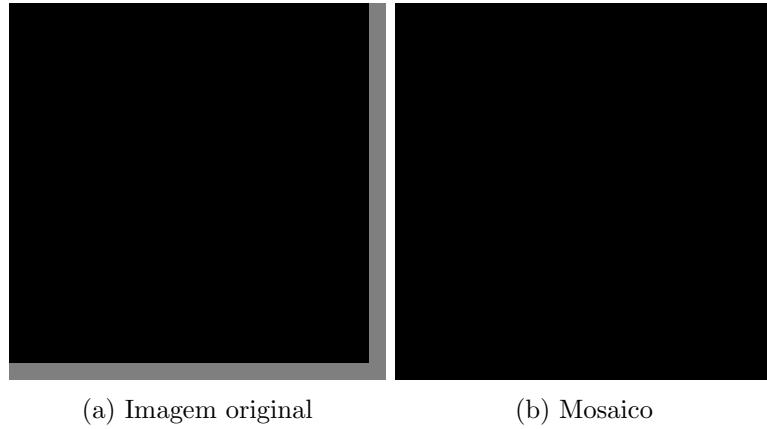


Figura 9 – Mosaico de uma imagem 67x67, na qual os pixels das 3 últimas linhas e colunas são cinzas, enquanto os outros são pretos.

3.4 Alteração de Cores

3.4.1 Explicação do conceito

A multiplicação dos pixels com canais RGB pela matriz de transformação gera canais RGB com novos valores de intensidade, segundo a fórmula:

$$[R \ G \ B] \cdot \begin{bmatrix} 0,393 & 0,769 & 0,189 \\ 0,349 & 0,686 & 0,168 \\ 0,272 & 0,534 & 0,131 \end{bmatrix}^T = \begin{bmatrix} 0,393R + 0,769G + 0,189B \\ 0,349R + 0,686G + 0,168B \\ 0,272R + 0,534G + 0,131B \end{bmatrix}^T = [R' \ G' \ B']$$

Note que a soma dos pesos ultrapassa 1 nas duas primeiras entradas do vetor resultante, o que pode causar *overflow* - principalmente nos casos que a intensidade de verde da imagem original for elevada, devido ao alto peso desse canal -; portanto, é necessário limitar os valores gerados ao intervalo de 0 a 255.

3.4.2 Implementação e resultado

Embora a operação aplicada seja simples, houveram dois pontos de dificuldade durante a implementação da transformação.

Em primeiro lugar, a multiplicação da imagem pela matriz de transformação foi inicialmente realizada sem a transposição dessa, o que gerou o resultado da Figura 10.b.

Em segundo lugar, a biblioteca OpenCV utiliza o formato BGR por padrão para salvar e mostrar imagens; portanto, se a imagem não for convertida para o formato correto, os canais estarão invertidos, como pode ser observado na Figura 10.c.

É importante mencionar que, para garantir que os valores dos pixels estivessem no intervalo [0, 255], foi realizada uma transformação linear dos pixels utilizando `cv.normalize`.

Após solucionar os pontos mencionados, a imagem obtida foi a da Figura 10.d.



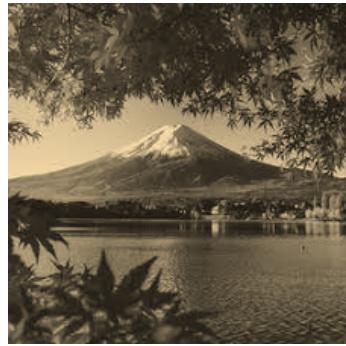
(a) Imagem original



(b) Sem transposição da matriz



(c) Sem conversão para BGR



(d) Resultado final

Figura 10 – Resultados da aplicação do filtro de alteração de cores na imagem `fuji.png` (escolha arbitrária). Note que as imagens geradas em (b) e (c) têm valores de intensidade maiores de verde e azul, respectivamente.

3.5 Transformação de Imagens Coloridas

3.5.1 Item (a)

A transformação requisitada nesse item é similar à do exercício 4, com a diferença de que é necessário limitar os valores dos canais em 255 (ou, pelo menos, assim foi interpretada). Portanto, no lugar de realizar uma normalização linear, deve-se aplicar a seguinte operação em código:

1 A[A > 255] = 255

O resultado, tal como a imagem produzida no item 4, podem ser observados na Figura 11. Visualmente, nota-se que a imagem gerada desta forma possui valores de intensidade menores, uma vez que o recorte de valores altera apenas aqueles que ultrapassam o valor de corte (255); em contrapartida, a normalização altera o valor de todos os pixels. Além disso, como os pixels das folhas apresentam elevadas intensidades de verde, são gerados valores maiores na intensidade de cinza (recorde que o canal verde é o que possui maior peso na matriz de transformação), os quais são limitados a 255. Isso leva a uma diminuição do contraste nas folhas.



(a) Resultado da transformação do exercício 4



(b) Resultado da transformação do exercício 5.a

Figura 11 – Resultados da aplicação dos filtros de alteração de cores na imagem `fiji.png`.

3.5.2 Item (b)

A média ponderada desejada pode ser calculada a partir de uma multiplicação dos pixels da imagem por uma matriz de transformação, analogamente ao processo do exercício 4. A matriz (vetor) utilizada é:

$$\begin{bmatrix} 0,2989 \\ 0,5870 \\ 0,1140 \end{bmatrix}$$

Note que os pesos da matriz são tais que $0,2989 + 0,5870 + 0,1140 = 0,9999 \leq 1$. Por conta disso, as intensidades da imagem resultante estarão no intervalo $[0, 255]$, não havendo assim necessidade de ajustá-las.

O resultado da transformação pode ser visualizado na Figura 12.



(a) Imagem original



(b) Resultado da transformação

Figura 12 – Transformação para uma banda de cor (níveis de cinza) da imagem `fiji.png`.

3.6 Planos de Bits

Um modo eficiente de obter apenas o i -ésimo bit de uma imagem representada com n bits é realizar a operação $\&$ (AND bit-a-bit) entre ela e uma máscara $mask$, cujo l -ésimo bit é definido por:

$$mask_l = \begin{cases} 1, & \text{se } l = i \\ 0, & \text{se } l \neq i \end{cases}, \forall 0 \leq l \leq n - 1 \quad (2)$$

A partir dessa operação, o valor do i -ésimo bit da imagem é mantido, enquanto todos os outros bits assumem o valor 0.

Para que os efeitos dessa transformação fossem mais facilmente visualizados, foi realizada uma conversão do resultado para o intervalo $[0, 255]$. Como há apenas dois valores presentes na imagem após a operação $\&$ (0 ou 2^i), essa conversão faz com que os pixels da imagem assumam ora o valor 0, ora o valor 255 de intensidade.

Os resultados da transformação podem ser observados nas Figuras 13 e 14. Um fenômeno interessante pode ser notado ao comparar essas duas imagens: no plano de bit 0 da imagem `baboon_monocromatica` não é possível observar algo que remeta à forma da imagem original, apenas uma configuração que se assemelha a um ruído; em contrapartida, no plano de bit 0 da `butterfly.png`, o formato da borboleta é nítido. Ambas as imagens possuem dimensões 512x512.

Vale ressaltar também que, para ambas as imagens, quanto mais significativo o bit, mais seu plano se assemelha da imagem original, uma vez que esses bits armazenam mais informação.

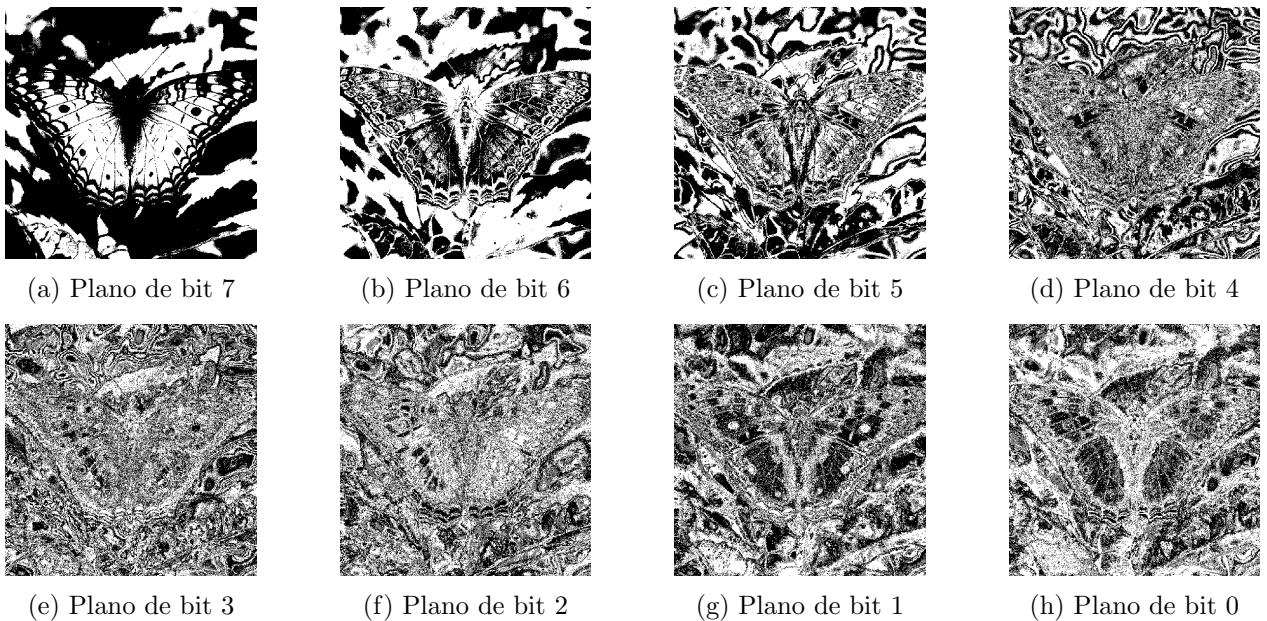


Figura 13 – Diferentes planos de bit (de 7 a 0) da imagem `butterfly.png`. Cada plano mostra a contribuição de um bit específico para a composição da imagem em tons de cinza.

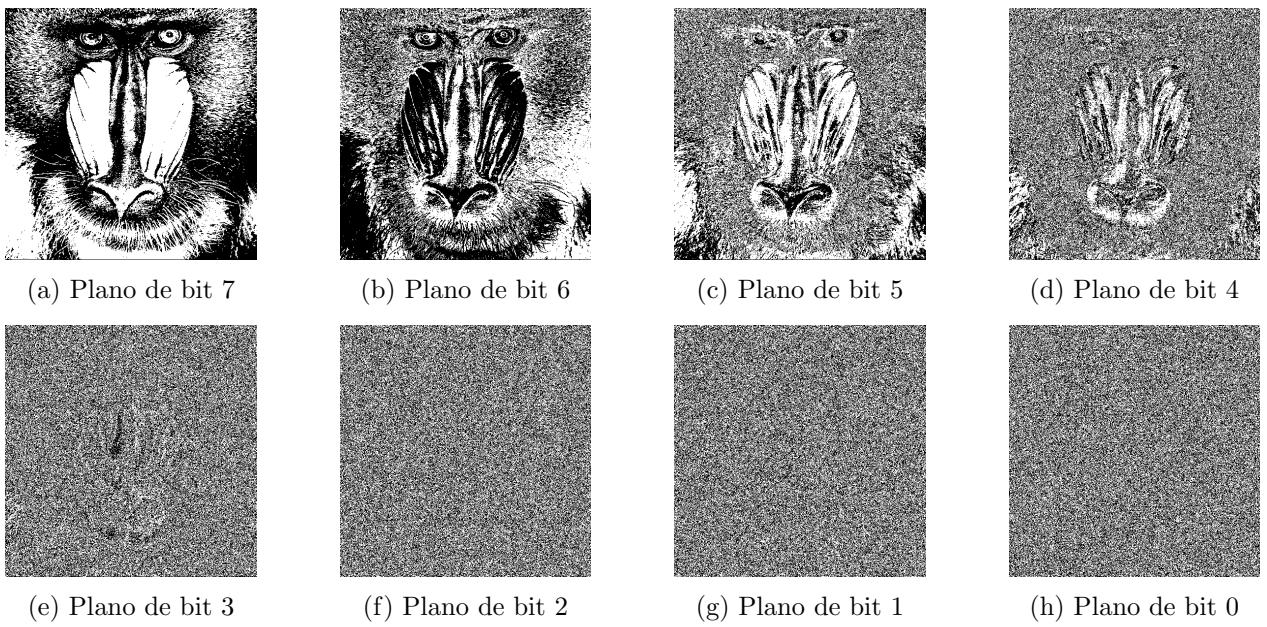


Figura 14 – Diferentes planos de bit (de 7 a 0) da imagem `baboon_monocromatica.png`.

3.7 Combinação de Imagens

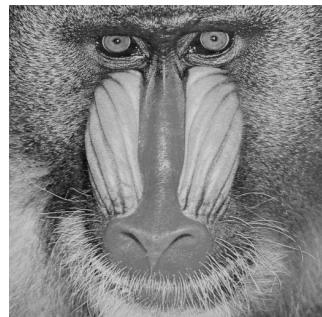
3.7.1 Implementação e resultado

Nesse exercício, o resultado esperado pode ser gerado, conforme indicado no enunciado, a partir da multiplicação de cada imagem pelo respectivo escalar, seguida de uma soma dos produtos. Isso foi realizado a partir do seguinte trecho de código, onde w1 e w2 são os pesos associados à primeira e à segunda imagem, respectivamente:

```
1 result = np.floor(w1*image_1 + w2*image_2)
```

Note que, se a soma dos pesos for menor ou igual a 1, o resultado necessariamente estará no intervalo [0, 255], uma vez que esse é o intervalo das imagens originais. Além disso, a função escolhida para arredondamento foi `np.floor`, o que, como citado no exercício 2, não gera diferença visual significativa quando comparada às outras possibilidades.

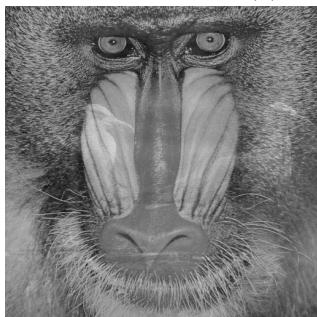
O resultado pode ser observado na figura 15. As imagens utilizadas foram `baboon_monocromatica.png` e `seagull.png`, uma vez que a primeira possui o foco no centro da imagem (nariz e olhos do babuíno), enquanto na segunda o foco maior está à esquerda e à direita do centro (onde estão as gaivotas), o que facilita o discernimento entre elas no resultado.



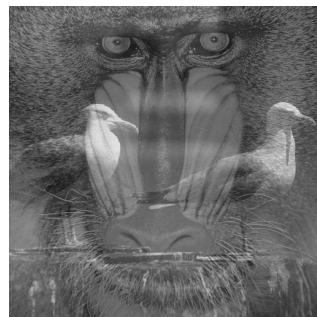
(a) Imagem A



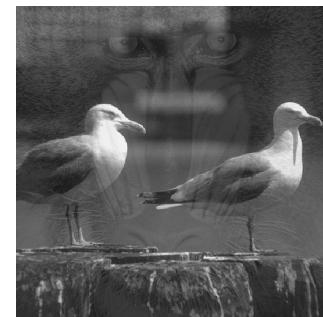
(b) Imagem B



(c) 0.8A + 0.2B



(d) 0.5A + 0.5B



(e) 0.2A + 0.8B

Figura 15 – Diferentes combinações das imagens `baboon_monocromatica.png` e `seagull.png`

3.7.2 Limitações

Foram identificadas duas situações nas quais o código mencionado pode gerar erros ou resultados indesejados.

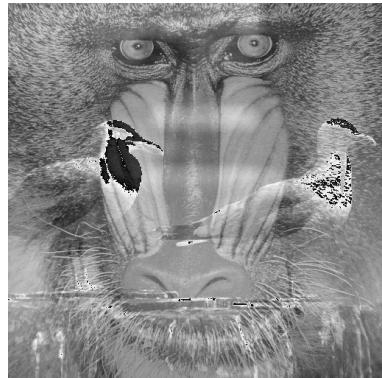


Figura 16 – Combinação de imagens com pesos $w_1 = w_2 = 0.7$. Algumas partes das gaivotas ficam escuras apesar dessa região ser clara em ambas as imagens, por conta de *overflow*.

Em primeiro lugar, a utilização de pesos cuja soma é maior que 1 pode gerar *overflow*, uma vez que a imagem foi representada com 8 bits, como pode ser observado na figura 16. Uma possibilidade de solução, visando evitar *overflow*, é a utilização de mais bits para representar o resultado, seguida de uma normalização para o intervalo $[0, 255]$ caso seja necessário utilizar apenas 8 bits.

Em segundo lugar, tentar combinar duas imagens de dimensões diferentes pode gerar erros ou, por conta do *broadcasting*, resultados indesejados. Isso poderia ser contornado realizando o redimensionamento ou *slicing* de alguma das imagens (por exemplo, fazer com que a maior delas passe a ter as mesmas dimensões da menor ou remover pixels das bordas).

3.8 Transformação de Intensidade

A explicação desse exercício será dividida para cada item, e os resultados serão exibidos ao final de cada item.

3.8.1 Item (b)

```
1 negativo = np.subtract(255, image)
```

O código acima gera o resultado esperado, uma vez que é feito um *broadcasting* do escalar 255 para as dimensões da imagem. Note que, dada a intensidade i de um pixel, ele terá intensidade $255 - i$ no negativo, i.e., o nível de cinza 0 é convertido para 255, 1 é convertido para 254 e assim por diante.

Vale citar que não foi realizada conversão para o intervalo $[0, 255]$; portanto, se a imagem original tiver intensidades no intervalo $[a, b]$, as intensidades do negativo estarão em $[255 - b, 255 - a]$ - pois, como $a \leq b$, então $-b \leq -a$.

3.8.2 Item (c)

A conversão do intervalo da imagem para $[100, 200]$ foi realizada a partir de um recorte de valores, i.e., valores maiores que 200 foram convertidos para 200 e valores menores que

100 foram convertidos para 100.

Contudo, o resultado, que pode ser observado na Figura 17.a, não foi o mesmo do enunciado. Para isso, foi necessário realizar uma transformação linear das intensidades para o intervalo [0, 255], o que gerou a imagem da Figura 17.b. Note que a segunda imagem possui intensidades mais escuras, além de maior contraste, que a primeira, pois o intervalo de intensidades é maior.

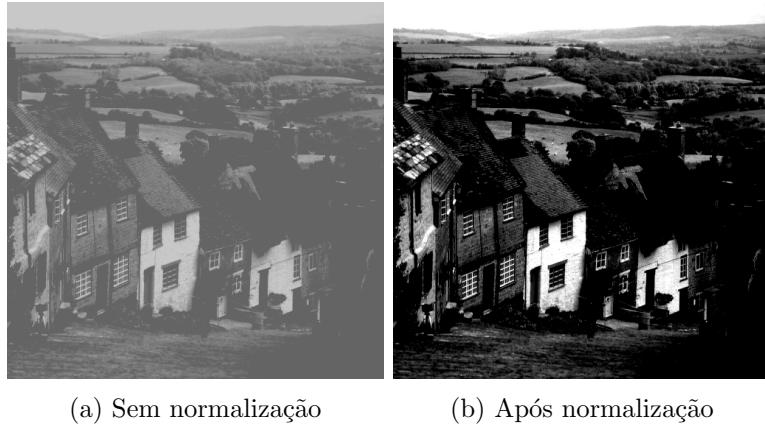


Figura 17 – Resultados obtidos convertendo as intensidades da imagem `city.png` para [100, 200].

3.8.3 Item (d)

Nesse item, basta inverter as linhas pares. Para isso, a implementação utilizou um *slicing* iniciando da primeira linha (índice 0) com passo 2 para obter as linhas pares. Em seguida, apenas foi necessário inverter os valores de intensidade no eixo das colunas.

3.8.4 Item (e)

O item pede que as linhas da metade superior da imagem sejam replicadas na metade inferior, invertidas. A implementação utilizou *slicing* para selecionar essas metades, invertendo a inferior no eixo das linhas.

Essa implementação, porém, não tinha verificação da paridade do número de linhas inicialmente; assim, quando uma imagem com número ímpar de linhas era utilizada, a metade inferior e superior tinham dimensões diferentes por 1 pixel, o que gerava um erro. A fim de contornar isso, a implementação submetida faz uma verificação e, se o número de linhas for ímpar, a linha no meio da imagem não é utilizada (i.e., ela se mantém inalterada).

3.8.5 Item (f)

Nesse item, basta inverter toda a imagem no eixo das linhas. Isso pode ser realizado trivialmente com um *slicing* ou, como no código submetido, utilizando a função de inversão “up-down” (`flipud`) da biblioteca *NumPy*.

3.8.6 Resultado final

Os resultados das transformações podem ser observados na Figura 18. A escolha da imagem `seagull.png` se deu pois os efeitos gerados são de fácil visualização.

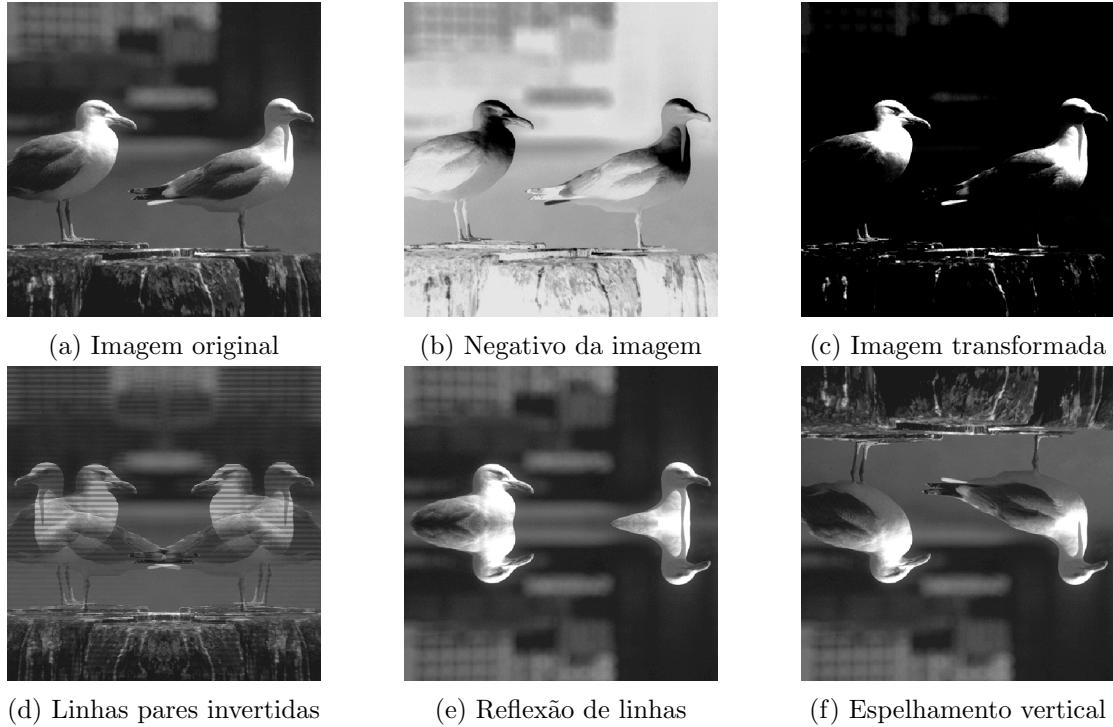


Figura 18 – Resultado de diferentes transformações sobre a imagem `seagull.png`.

3.9 Quantização de Imagens

3.9.1 Implementação

De acordo com [1], a quantização uniforme, uma das formas mais simples de quantização em escala de cinza, pode ser realizada a partir da seguinte operação:

$$g(x, y) = \lfloor \frac{f(x, y)}{\Delta} \rfloor \cdot \Delta + \frac{\Delta}{2}, \text{ onde:} \quad (3)$$

- $g(x, y)$ é o valor de intensidade quantizado (com arredondamento para o inteiro mais próximo).
- $f(x, y)$ é o valor de intensidade original.
- $\Delta = \frac{256}{L}$ é o tamanho de cada intervalo de quantização², dada a quantidade de níveis de intensidade desejada L .

²A referência [1] define $\Delta = \frac{255}{L-1}$; contudo, dado $f(x, y) = 255$, tem-se $g(x, y) = L \frac{255}{L-1} + \frac{\Delta}{2} > L \frac{255}{L-1} > 255$ (ocorre *overflow*). Além disso, esta fórmula implica na divisão em apenas $L - 1$ intervalos. O cálculo realizado, elaborado durante a confecção deste trabalho, foi $\Delta = \frac{256}{L}$, que divide corretamente a intensidade em L intervalos e mapeia $f(x, y) = 255$ para o último deles.

Essa quantização se baseia na divisão dos 256 níveis de cinza em L intervalos l_0, l_1, \dots, l_{L-1} , tal que $l_i = [i \cdot \Delta, (i + 1) \cdot \Delta], \forall 0 \leq i \leq L - 1$.

Dado o valor de intensidade de um pixel na imagem original, podemos calcular seu intervalo l_j a partir de $j = \lfloor \frac{f(x,y)}{\Delta} \rfloor$. Assim, $g(x, y) = j \cdot \Delta + \frac{\Delta}{2}$, i.e., todos os pixels dentro de um mesmo intervalo assumem o valor de seu ponto médio. Com isso, a imagem passa a ter apenas L valores distintos de intensidade.

A solução implementada realiza a conversão mencionada sem adicionar $\frac{\Delta}{2}$; dessa forma, as intensidades dos pixels na imagem quantizada são dadas pelo valor mínimo de seus respectivos intervalos. Isso é seguido de uma normalização para o intervalo $[0, 255]$, a qual, por ser uma transformação linear, não altera a quantidade de valores usados.

3.9.2 Resultado

O resultado obtido pode ser observado na Figura 19. O motivo para o uso da imagem `seagull.png` foi o alto contraste entre as gaivotas e o fundo, fazendo com que a intensidade de seus pixels tenham maior tendência a ser substituídas por valores diferentes entre si - note que, mesmo com 2 níveis de quantização, parte do corpo das gaivotas ainda tem baixa intensidade.

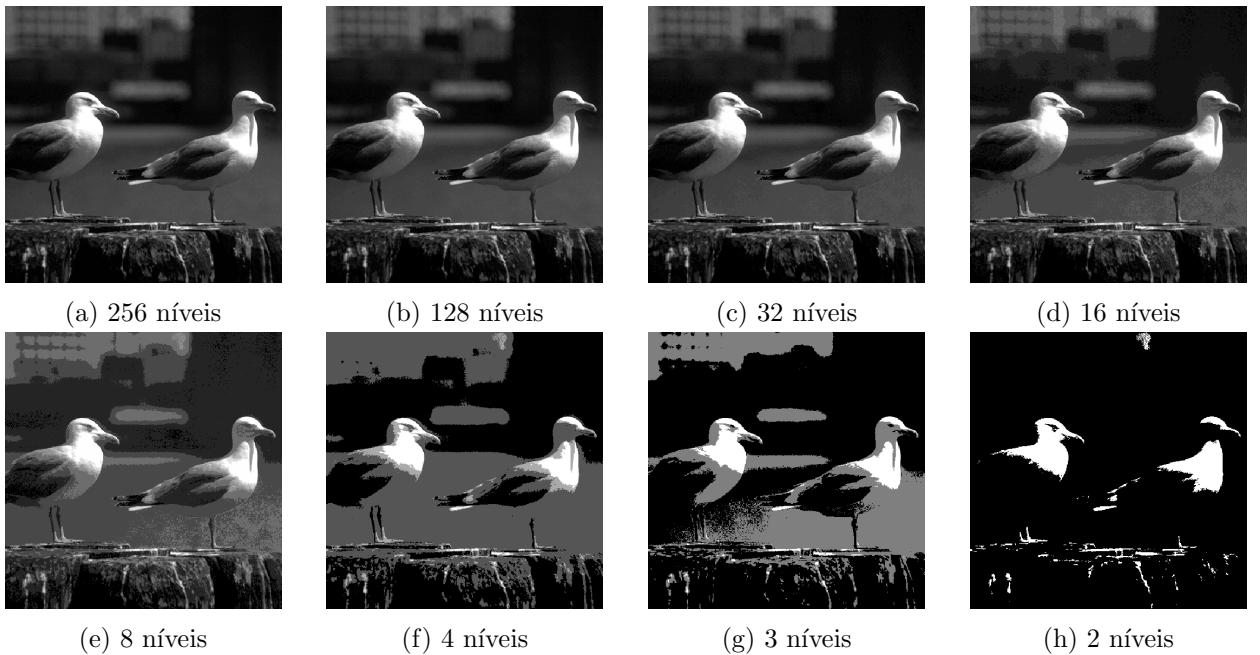


Figura 19 – Imagem `seagull.png` representada com diferentes níveis de quantização.

Vale ressaltar que a diferença visual entre níveis altos de quantização é, em geral, imperceptível a olho nu - as imagens com 256 e 32 níveis de cinza são visualmente similares -. Além disso, o espaço de memória ocupado pelas imagens tende a diminuir conforme o número de níveis de quantização é reduzido. Logo, mesmo que os valores de intensidade ainda estejam representados no mesmo intervalo $[0, 255]$, uma imagem com menos níveis distintos pode ser mais facilmente comprimida por formatos como PNG.

3.10 Filtragem de Imagens

3.10.1 Implementação

A aplicação dos filtros foi realizada utilizando a função `filter2D` da biblioteca OpenCV, a qual, segundo sua documentação, realiza a correlação de um filtro sobre a imagem. Portanto, para realizar uma convolução, foi necessário inverter os filtros vertical e horizontalmente.

Além disso, foi aplicado um *padding* de 4 pixels antes de realizar a convolução, uma vez que o maior filtro utilizado nesse exercício possui dimensões 9x9 (nesse caso, para calcular o valor de um pixel, são utilizados os vizinhos com distância menor ou igual a 4 em vizinhança-8). Para tal, foi utilizada estratégia de replicação de borda, a qual atribui aos pixels do *padding* a mesma intensidade do ponto da imagem mais próximo em vizinhança-4; dessa forma, os filtros passa-baixas tendem a manter as bordas inalteradas, ao passo que os passa-altas tendem a deixar esses pixels pretos, como será observado nos resultados.

Em suma, primeiramente é realizada a inversão do filtro, seguida da adição do *padding* à imagem original; então, é feita a correlação com o filtro invertido (o que equivale à convolução) e, por fim, ocorre a remoção do *padding*.

A imagem utilizada para explicar os efeitos dos filtros será a `waterfall_cinza.png` (Figura 20), uma vez que esta apresenta formas diferentes e traços bem definidos em diversas direções.

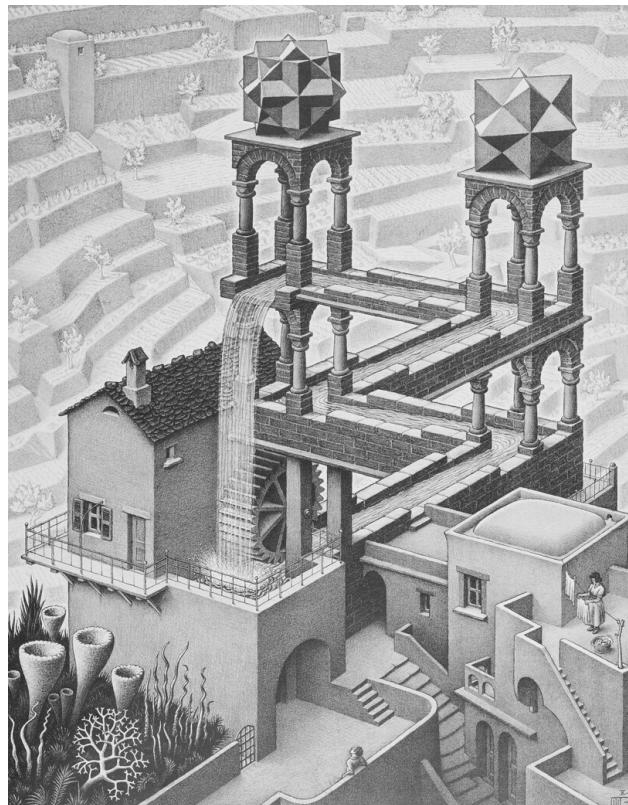


Figura 20 – Imagem `waterfall_cinza.png`, com dimensões 813x1038.

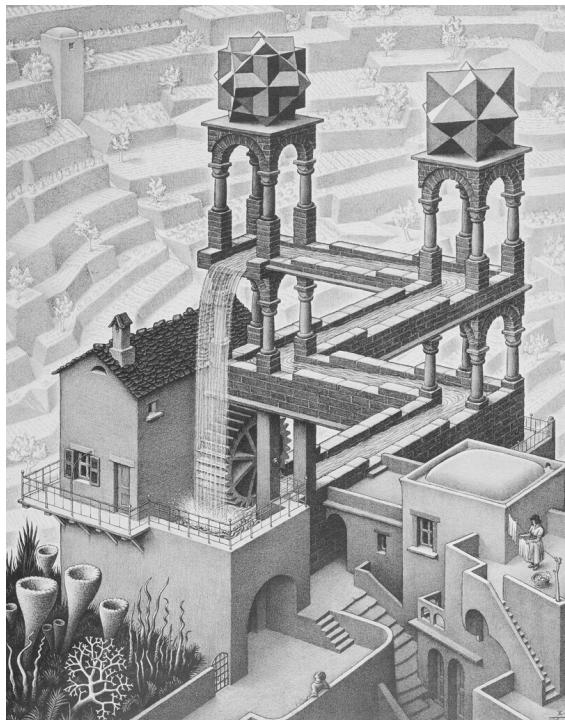
3.10.2 Resultados e discussão

- Filtro h_1 :

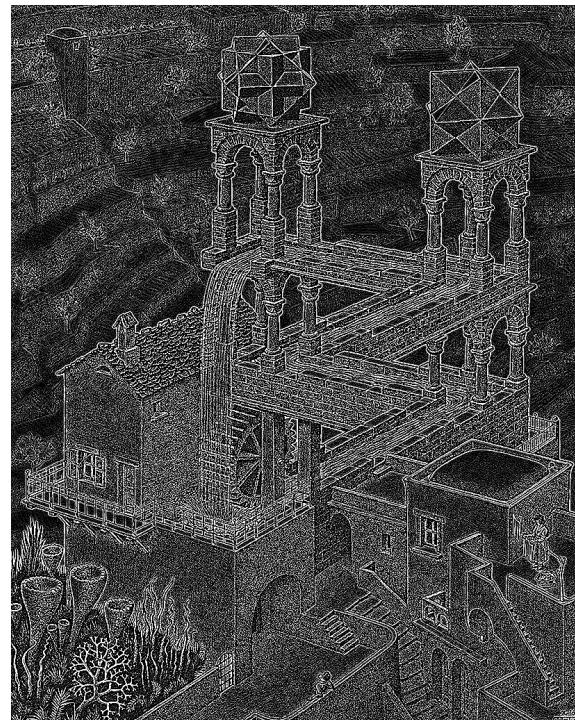
$$h_1 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Note que este filtro possui soma dos coeficientes igual a zero; portanto, se todos os pixels em uma região tiverem valores de intensidade próximos ou constantes, o resultado da convolução será um valor próximo de zero (escuro). Além disso, o ponto central é o único que possui peso positivo, enquanto os demais possuem pesos negativos distribuídos em torno dele, decrescendo conforme se afastam do centro (considerando vizinhança-4).

Sendo assim, a convolução por h_1 atribui valores altos a regiões com variação brusca de intensidade em relação à vizinhança — ou seja, realça componentes de alta frequência (bordas e linhas). Por isso, trata-se de um filtro passa-altas, como pode ser observado na Figura 21.



(a) Imagem original



(b) Imagem após filtragem

Figura 21 – Resultado da aplicação do filtro h_1

- Filtro h_2 :

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

Este filtro realiza uma média ponderada dos valores ao redor do pixel central, com atribuição de peso maior aos pixels mais próximos ao centro. Além disso, a soma dos pesos é igual a 1; portanto, trata-se de um filtro passa-baixas, que possui o efeito visual de "borrar" a imagem. Mais especificamente, trata-se de um filtro gaussiano 5x5, cuja aplicação introduz ruído impulso à imagem, como pode ser observado na Figura 22.

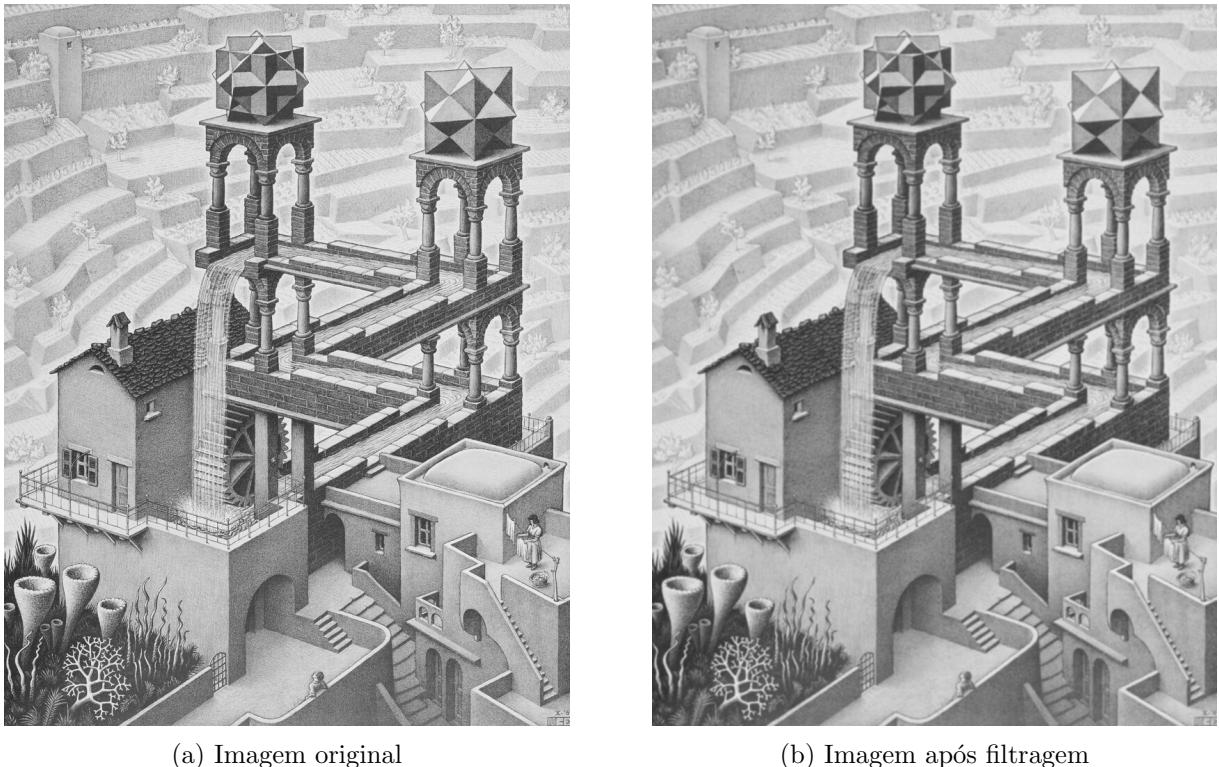


Figura 22 – Resultado da aplicação do filtro h_2

- Filtro h_3 :

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

Note que, em relação à coluna central, o filtro possui os mesmos valores, porém opostos. Portanto, se os valores dos pixels à direita e à esquerda do pixel central forem próximos ou iguais, o resultado da convolução será um valor próximo de zero (escuro). Por outro

lado, quando estes diferirem, o resultado será um valor alto (ou negativo, o que, devido à representação como inteiro sem sinal, também resulta em um valor alto).

Desta forma, é possível concluir que o filtro realça regiões de alta frequência no sentido horizontal, gerando linhas verticais. Como pode ser observado na Figura 23, as linhas da cachoeira e das bordas das paredes na vertical são ressaltadas após a convolução.

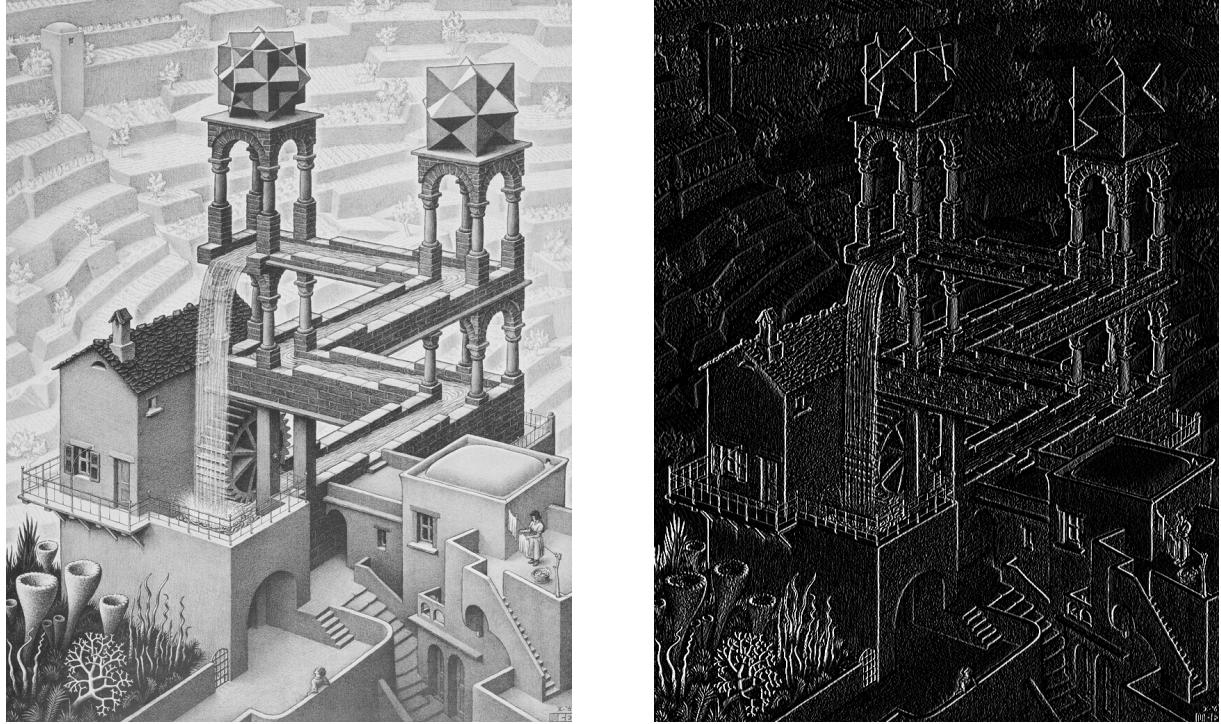


Figura 23 – Resultado da aplicação do filtro h_3

- Filtro h_4 :

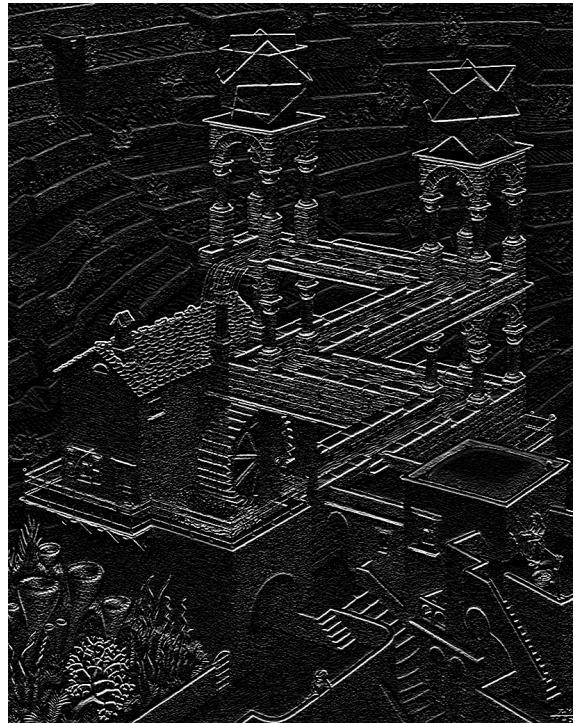
$$h_4 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Note que este filtro é dado por $h_4 = h_3^T$. Logo, seu comportamento é análogo ao de h_3 , mas na direção inversa - realça variações no sentido vertical, gerando linhas horizontais.

O resultado da filtragem por h_4 pode ser visualizado na Figura 24. Note que a cachoeira, destacada pelo filtro anterior, não aparece no resultado desta filtragem, pois ela é representada principalmente por traços na vertical.



(a) Imagem original



(b) Imagem após filtragem

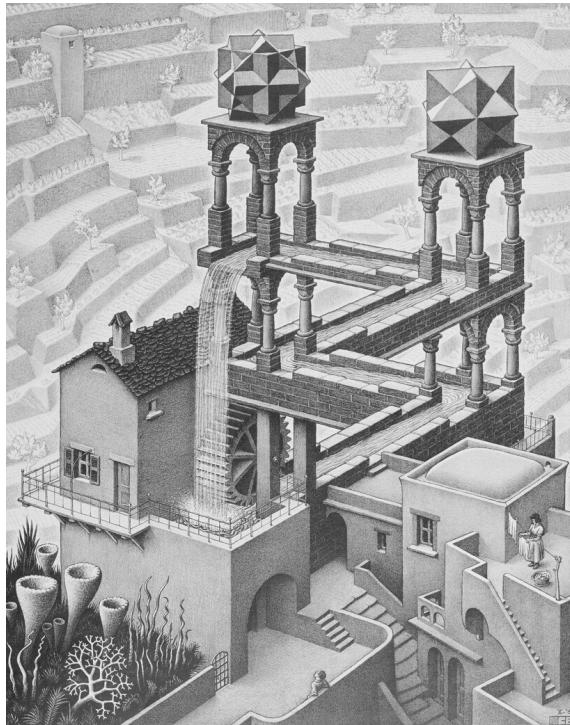
Figura 24 – Resultado da aplicação do filtro h_4

- Filtro h_5 :

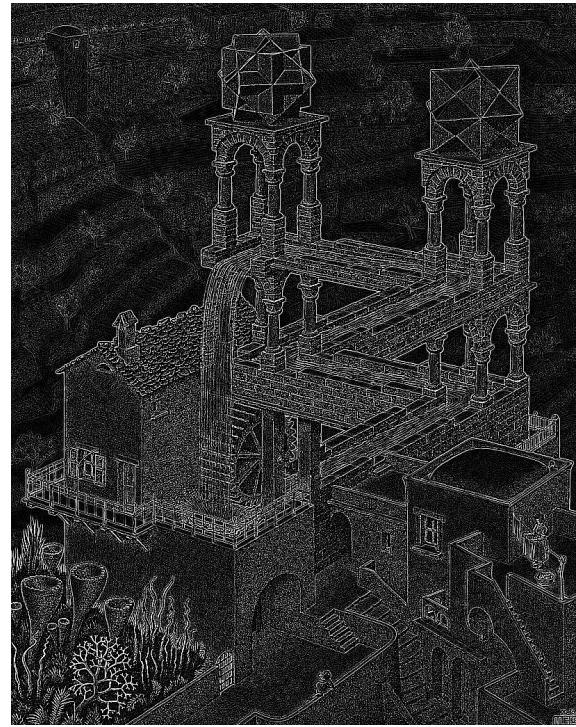
$$h_5 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Assim como o filtro h_1 , este também possui soma dos coeficientes igual a zero e atribui peso positivo apenas ao pixel central. No entanto, ao contrário do h_1 , a influência dos vizinhos é distribuída de forma simétrica em todas as direções com a mesma intensidade.

Por conta disso, esse filtro é do tipo passa-altas e realça mudanças bruscas de intensidade em qualquer direção, como bordas finas ou detalhes pontuais na imagem. Seu efeito pode ser visualizado na Figura 25. Observa-se que os pequenos pontos decorrentes do estilo de pintura são destacados (esse fenômeno é mais perceptível na parede da casa, à esquerda da imagem), assim como as bordas e linhas.



(a) Imagem original



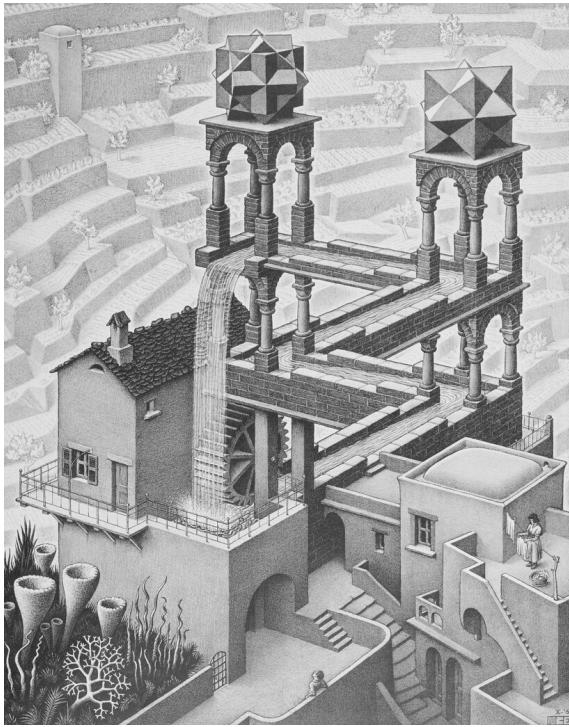
(b) Imagem após filtragem

Figura 25 – Resultado da aplicação do filtro h_5

- Filtro h_6 :

$$h_6 = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Esse filtro realiza uma simples média aritmética de todos os pontos adjacentes ao central em vizinhança-8, com soma dos pesos igual a 1. Portanto, caracteriza-se como um filtro passa-baixas, cujo efeito é introduzir ruído impulsivo à imagem. O resultado de sua aplicação pode ser visualizado na Figura 26, e é similar ao da aplicação do filtro h_2 , porém com menos efeito de borramento devido à dimensão inferior do filtro (3x3).



(a) Imagem original



(b) Imagem após filtragem

Figura 26 – Resultado da aplicação do filtro h_6

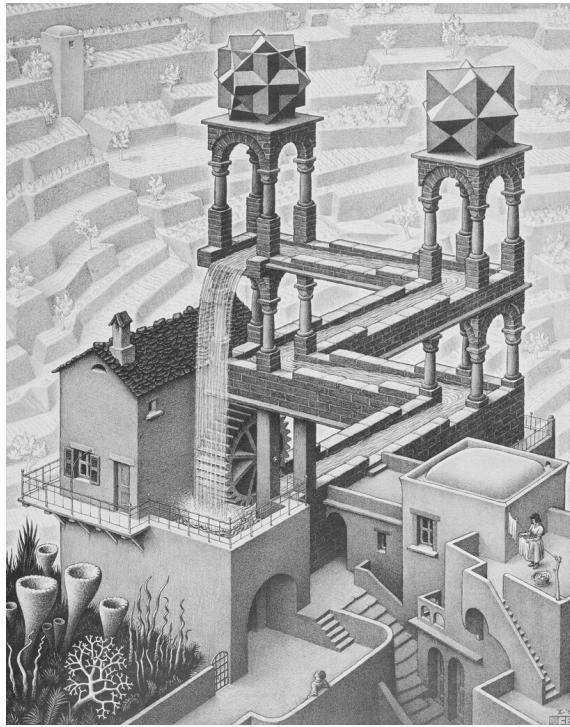
- Filtro h_7 :

$$h_7 = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

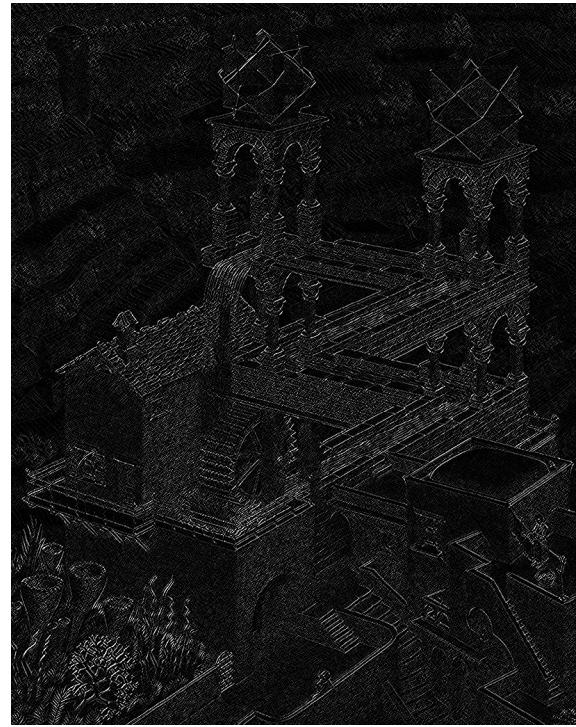
Esse filtro possui soma dos pesos igual a 0; logo, se trata de um filtro passa-altas. Mais especificamente, devido aos valores positivos na diagonal secundária, esse filtro realça transições abruptas no sentido da diagonal principal da imagem.

Além disso, devido à soma dos pesos igual a zero em todas as linhas e colunas de h_7 , bordas horizontais e verticais não são realçadas.

Ambos os efeitos citados podem ser observados na Figura 27: bordas diagonais foram destacadas, em detrimento das verticais e horizontais (observe que apenas a parte superior da cachoeira foi realçada).



(a) Imagem original



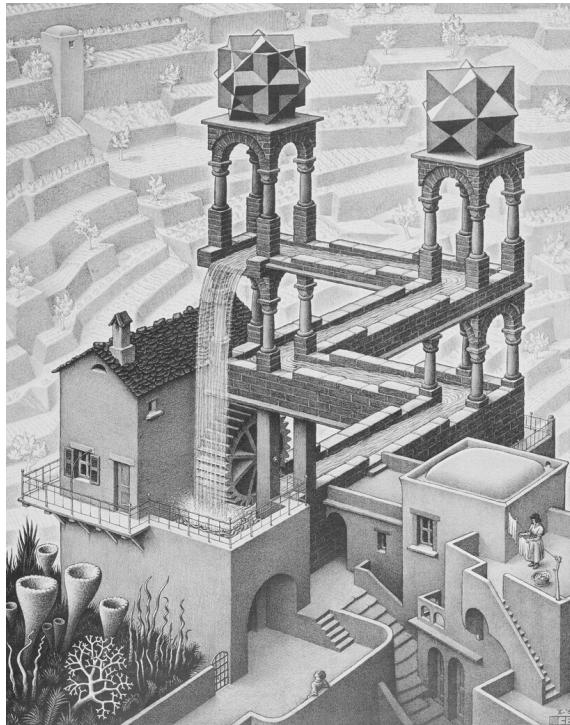
(b) Imagem após filtragem

Figura 27 – Resultado da aplicação do filtro h_7

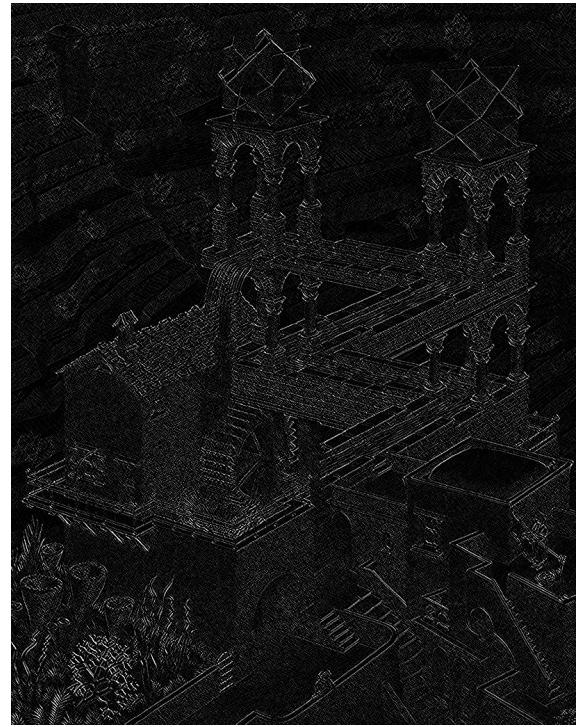
- Filtro h_8 :

$$h_8 = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

Note que h_8 pode ser obtido a partir de h_7 a partir de um espelhamento vertical, então o efeito esperado após aplicá-lo é que as bordas paralelas à diagonal principal sejam destacadas. Contudo, como se pode observar na Figura 28, a imagem gerada é visualmente parecida com aquela obtida aplicando h_7 .



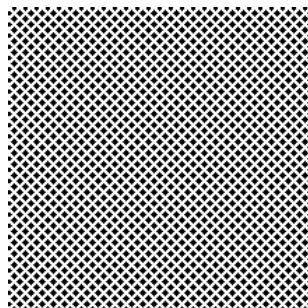
(a) Imagem original



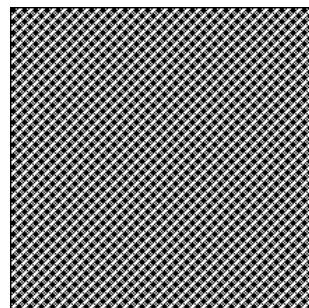
(b) Imagem após filtragem

Figura 28 – Resultado da aplicação do filtro h_8

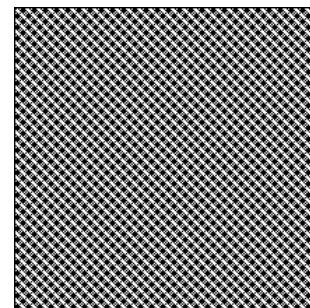
A fim de evidenciar a diferença entre esses filtros, foi gerada uma imagem com várias linhas nas diagonais. É possível observar, na Figura 29, que a aplicação de h_7 e h_8 nessa imagem realça as linhas em ambas as diagonais, mas com maior intensidade naquela em que o filtro possui valores positivos.



(a) Imagem original



(b) Filtragem com h_7



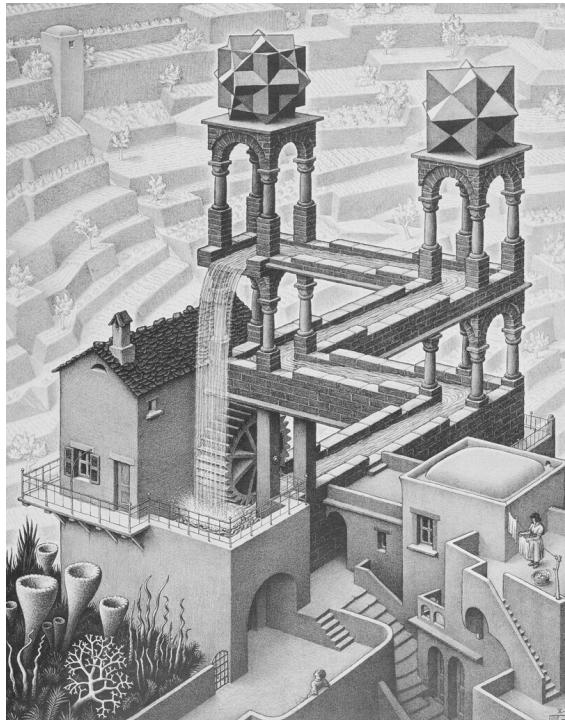
(c) Filtragem com h_8

Figura 29 – Aplicação de h_7 e h_8 na imagem `bordado.png`.

- Filtro h_9 :

$$h_9 = \frac{1}{9} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Novamente, devido à soma dos pesos igual a 1, esse é um filtro passa-baixas, o qual realiza a média aritmética dos pixels na diagonal principal em relação ao pixel central. Portanto, a aplicação de h_9 introduz um ruído na direção da diagonal principal, como pode ser observado na Figura 30.



(a) Imagem original



(b) Imagem após filtragem

Figura 30 – Resultado da aplicação do filtro h_9

- Filtro h_{10} :

$$h_{10} = \frac{1}{8} \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & 2 & 8 & 2 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Note que a soma dos coeficientes desse filtro é igual a 0,5; portanto, em regiões homogêneas, a intensidade do pixel é reduzida pela metade. Além disso, o filtro combina características de filtros passa-baixas, pois realiza uma média ponderada dos valores, e passa-altas, uma vez que possui pesos negativos.

O efeito de sua aplicação pode ser visto na Figura 31. Observa-se que o resultado possui maior saturação, além de realçar as bordas.

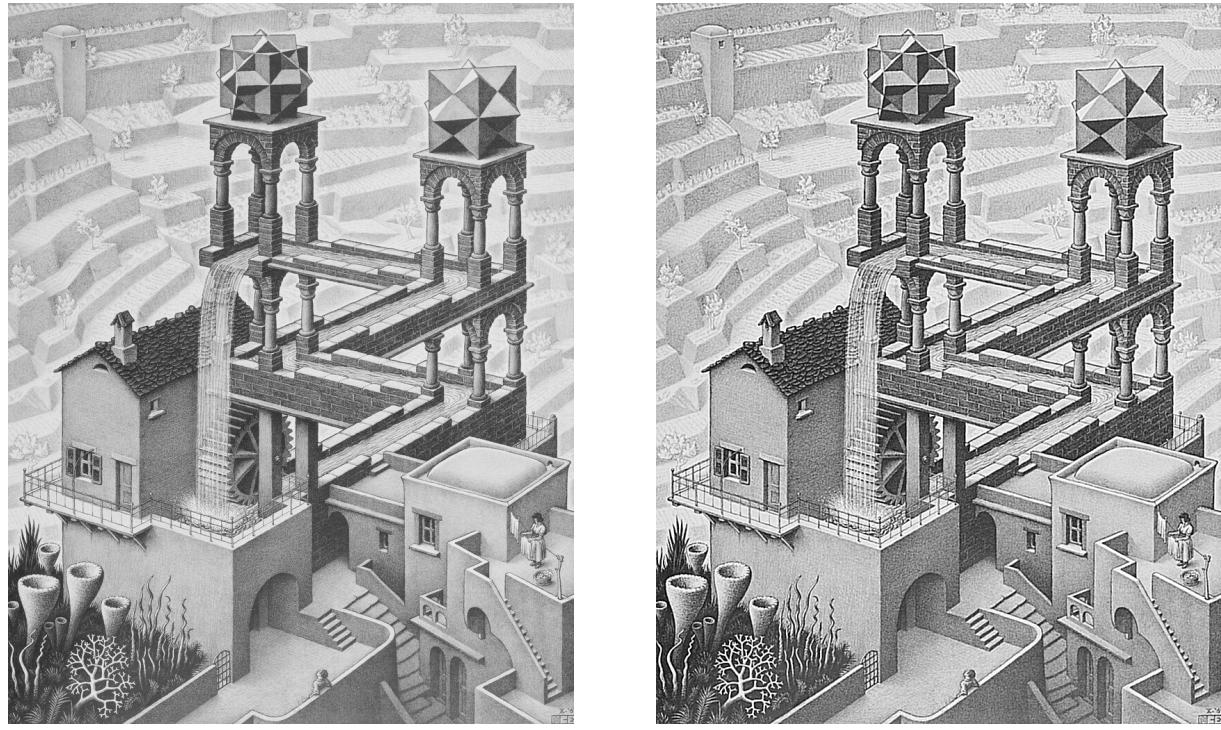


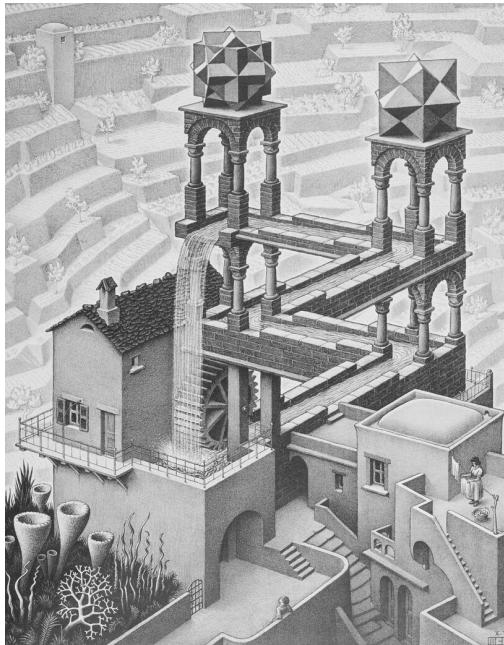
Figura 31 – Resultado da aplicação do filtro h_{10}

- Filtro h_{11} :

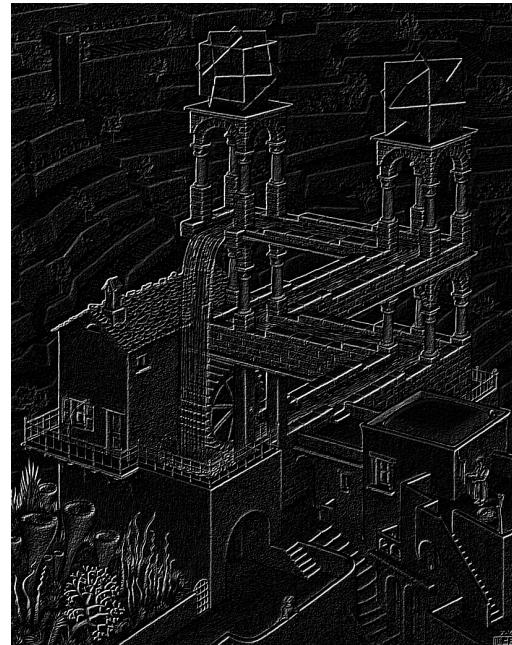
$$h_{11} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

O filtro h_{11} é similar ao h_4 e ao h_5 , porém possui valores nulos na diagonal secundária; por conta disso, o efeito gerado é o de realçar as bordas paralelas a ela, em detrimento das bordas paralelas à diagonal principal. Note, também, que a soma dos coeficientes nas colunas e nas linhas são -2, 0 e 2, respectivamente, fazendo com que haja também o realce de bordas horizontais e verticais.

Os efeitos mencionados são, de fato, os que foram obtidos, como pode ser visto na Figura 32.



(a) Imagem original



(b) Imagem após filtragem

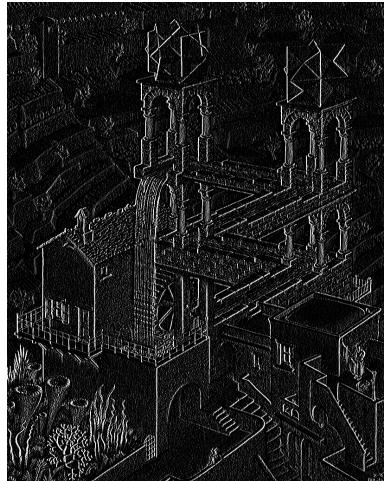
Figura 32 – Resultado da aplicação do filtro h_{11}

- Combinação de h_3 e h_4 :

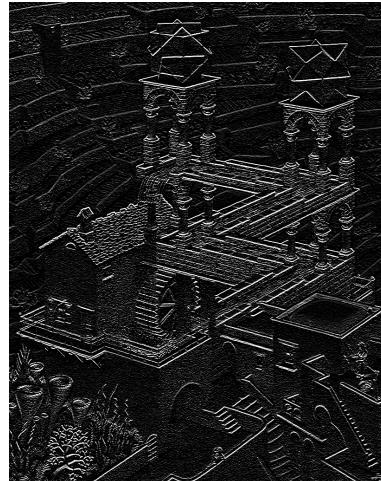
Como mencionado no enunciado, podemos definir um filtro H , dado por:

$$H = \sqrt{h_3^2 + h_4^2} \quad (4)$$

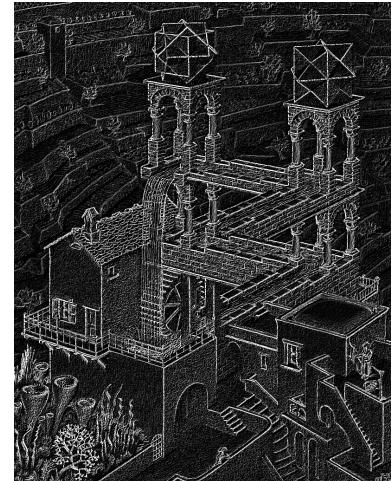
Uma vez que h_3 e h_4 realçam componentes de alta frequência nas direções horizontal e vertical, respectivamente, a aplicação de H gera uma imagem que realça detalhes em ambas essas direções, como pode ser observado na Figura 33.



(a) Efeito gerado por h_3



(b) Efeito gerado por h_4



(c) Efeito gerado por H

Figura 33 – Resultado da aplicação do filtro $H = \sqrt{h_3^2 + h_4^2}$

4 Conclusão

Neste trabalho, foi possível aplicar e analisar diversos conceitos fundamentais de processamento de imagens por meio da implementação prática com as bibliotecas *OpenCV* e *NumPy*. Ademais, o trabalho reforçou a importância da escolha correta de parâmetros e das formas de representar imagens.

5 Referências

- [1] Contribuidores da Wikipedia. *Quantization (image processing)*. Disponível em: [`https://en.wikipedia.org/wiki/Quantization_\(image_processing\)`](https://en.wikipedia.org/wiki/Quantization_(image_processing)). Acesso em: 4 de abril de 2025.