MINISTÉRIO DA EDUCAÇÃO
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
DEPARTAMENTO DE TELEMÁTICA

RELATÓRIO DE ATIVIDADE 2024.1

IMPLEMENTAÇÃO E ANÁLISE DE OPERAÇÕES BÁSICAS EM PROCESSAMENTO DE IMAGENS

AVALIAÇÃO DA EFICÁCIA E EFICIÊNCIA DOS FILTROS E OPERAÇÕES DESENVOLVIDAS

Relatório da Atividade para disciplina de Visão Computacional sob orientação do Professor Dr. Reboucas Filho Pedro Pedrosa.

Marcelo de Araújo
Campus Fortaleza
Março / 2024

RESUMO

Este trabalho apresenta a implementação de diversos filtros e técnicas de processamento de imagens utilizando as bibliotecas OpenCV, NumPy e Matplotlib em Python. Foram desenvolvidos filtros passa baixa, incluindo o filtro de média, que calcula a média dos valores dos pixels em uma janela definida pelo usuário, e o filtro gaussiano, que utiliza uma máscara gaussiana para suavização da imagem. Além disso, foram implementados filtros passa alta, como o filtro laplaciano e os operadores de detecção de bordas de Prewitt e Sobel, destinados ao realce e detecção de bordas na imagem, respectivamente. Adicionalmente, técnicas de análise de histograma foram aplicadas, incluindo o cálculo e apresentação do histograma em diferentes canais (RGB ou Grayscale) e a equalização do histograma para melhorar o contraste da imagem. A conversão de imagens coloridas para escala de cinza foi realizada através do cálculo da luminância. A limiarização foi empregada para segmentação de imagens em áreas claras e escuras, enquanto a multilimiarização permitiu a segmentação em múltiplos níveis.

Este relatório abordará os conceitos fundamentais dos filtros passa-baixa e passa-alta, bem como as técnicas de cálculo, apresentação e equalização de histogramas, limiarização e multilimiarização, explorando suas aplicações e importância no processamento de imagens.

1 INTRODUÇÃO

Conforme (SMITH; JONES; BROWN, 2005), filtros passa-baixa são amplamente utilizados em aplicações de suavização de imagens para eliminar ruídos e detalhes finos, enquanto filtros passa-alta são eficazes para realçar bordas e detalhes na imagem; esses algoritmos são essenciais no processamento de sinais e imagens, com o objetivo de modificar as características do sinal original para extrair informações desejadas ou suprimir componentes. No contexto do processamento de imagens, filtros passa-baixa e passa-alta são amplamente utilizados na suavização e realce de imagens, respectivamente. Os filtros passa-baixa têm como função atenuar as frequências mais altas de uma imagem, preservando as componentes de baixa frequência. Eles são comumente empregados em aplicações de suavização de imagens, eliminando ruídos e detalhes finos que podem não ser relevantes para a análise desejada. Por outro lado, os filtros passa-alta realçam as frequências mais altas, destacando bordas e detalhes na imagem.

O cálculo, apresentação e equalização de histogramas são técnicas no processamento de imagens para melhorar a visualização das informações contidas na imagem. A equalização de histogramas é um método de realce de contraste que redistribui os níveis de intensidade dos pixels, tornando a distribuição de intensidades uniforme. A limiarização e multilimiarização são técnicas utilizadas na segmentação de imagem em diferentes regiões com base nos níveis de intensidade dos pixels. A limiarização envolve a aplicação de um único limite para separar os pixels em duas classes distintas, enquanto a multilimiarização utiliza múltiplos limiares para segmentar a imagem em várias classes.

2 OBJETIVO GERAL

A finalidade deste trabalho é fornecer uma implementação em Python de diferentes técnicas de processamento de imagens, incluindo filtros de suavização (média, mediana, gaussiano), filtros, que são utilizados para reduzir o ruído e as imperfeições em imagens. Estes filtros operam através da substituição do valor de um pixel por alguma medida de tendência relacionada aos valores dos pixels vizinhos. Os filtros de realce de bordas (laplaciano, Prewitt, Sobel) são projetados para destacar as transições de intensidade na imagem, realçando as bordas e contornos. A operação destes filtros é realizada por meio de convolução da imagem original com máscaras ou kernels específicos, combinando dois conjuntos de dados para produzir um terceiro conjunto.

O histograma de uma imagem representa a distribuição de intensidades dos pixels. É uma ferramenta útil para análise e processamento de imagens, pois fornece informações sobre o contraste, brilho e distribuição das intensidades. A equalização de histograma é uma técnica utilizada para melhorar o contraste de uma imagem. O objetivo é redistribuir as intensidades dos pixels de forma que o histograma resultante seja aproximadamente uniforme. A limiarização é uma técnica de segmentação binária que divide uma imagem em duas regiões: objeto e fundo. Um valor de limiar é escolhido e todos os pixels com intensidade acima desse valor são atribuídos a uma classe, enquanto os pixels com intensidade abaixo desse valor são atribuídos à outra classe. A multilimiarização é uma extensão da limiarização simples e é utilizada para segmentar uma imagem em múltiplos níveis de intensidade com base em limiares definidos.

Essas funcionalidades são fundamentais no processamento de imagens para diversas aplicações, como melhoria de qualidade de imagem, detecção de bordas, segmentação e análise de informações visuais. O trabalho foi desenvolvido visando a facilidade de uso e compreensão, permitindo a aplicação direta das técnicas de processamento de imagens em diferentes tipos de imagens e cenários.

3 OBJETIVOS ESPECÍFICOS

A proposta deste trabalho é implementar uma série de filtros e técnicas de processamento de imagem utilizando as bibliotecas **OpenCV** para leitura e a conversão dos resultados em imagens; **Numpy** com finalidade realizar operacões matriciais e matemáticas através de estrutura de dados do tipo vetor; **Matplotlib** com o objetivo da plotagem de gráficos para representar alguma análise feita nas imagens. O trabalho oferece o desenvolvimento de algoritmos funcionais para aplicar filtros de **média, mediana, gaussiano, laplaciano, Prewitt e Sobel**. Além disso, também inclui funções para **calcular e apresentar histogramas** de imagens em diferentes formatos e realizar **equalização de histograma** e **limiarização** e **multilimiarização** de imagens.

Descrição das Funcionalidades:

- Filtro de Média: Aplica um filtro de média sobre a imagem para suavização.
- Filtro de Mediana: Utiliza o filtro de mediana para reduzir ruídos impulsivos na imagem.
- Filtro Gaussiano: Implementa o filtro gaussiano para suavização da imagem, utilizando um kernel gaussiano.
- Filtro Laplaciano: Aplica o filtro laplaciano para detecção de bordas.
- Filtro Prewitt: Implementa o operador Prewitt para detecção de bordas.
- Filtro Sobel: Utiliza o operador Sobel para detecção de bordas.
- Histograma: Calcula e apresenta o histograma da imagem em diferentes formatos: RGB, RGBA e Grayscale.
- Equalização de Histograma: Realiza a equalização do histograma da imagem para melhorar o contraste.
- Limiarização: Aplica a técnica de limiarização para binarizar a imagem.
- Multilimiarização: Realiza a multilimiarização da imagem de acordo com os limiares e níveis especificados.

Este trabalho oferece uma implementação prática e eficiente de técnicas fundamentais de processamento de imagem, contribuindo para a pesquisa e desenvolvimento em diversas áreas científicas e tecnológicas. A implementação de tais filtros clássicos e operadores são amplamente utilizados na literatura sendo frequentemente citados em artigos científicos e são fundamentais em diversas aplicações de processamento de imagem e na visão computacional.

4 METODOLOGIA

4.1 Filtros Passa-Baixa

Os filtros passa-baixa são amplamente utilizados em processamento de imagens para suavizar ou borar uma imagem, reduzindo assim o ruído e os detalhes finos. Estes filtros operam permitindo a passagem de frequências baixas (frequências relacionadas a variações lentas de intensidade) e atenuando as frequências altas (frequências relacionadas a variações rápidas de intensidade). O calculo genérico de um filtro passa-baixa é dado por:

$$H(u,v) = \begin{cases} 1 & \text{se } D(u,v) \leq D_0 \\ 0 & \text{caso contrário} \end{cases}$$

onde $H(u, v)$ é a função de transferência do filtro passa-baixa, $D(u, v)$ é a distância da frequência (u, v) ao centro da frequência da imagem, e D_0 é o raio do corte do filtro.

4.1.1 Filtro da Média

O filtro de média é um filtro linear simples que é comumente usado para suavizar uma imagem, reduzindo o ruído e detalhes finos. A ideia principal por trás do filtro de média é substituir o valor de cada pixel na imagem pela média dos valores dos pixels vizinhos, incluindo o próprio pixel. Dada uma imagem I de dimensões $m \times n$, o valor de um pixel (i, j) na imagem filtrada I_{media} usando um filtro de média de tamanho de janela $k \times k$ é dado por:

$$I_{\text{media}}(i, j) = \frac{1}{k^2} \sum_{l=0}^{k-1} \sum_{m=0}^{k-1} I(i - l, j - m) \tag{1}$$

onde $I(i, j)$ é o valor do pixel na posição (i, j) na imagem original e $I_{\text{media}}(i, j)$ é o valor do pixel na posição (i, j) na imagem filtrada. Para implementar o filtro de média em uma imagem digital, o seguinte algoritmo foi desenvolvido:

1. Inicialize uma nova imagem com as mesmas dimensões da imagem original I, preenchendo-a com zeros.
2. Para cada pixel (i, j) na imagem original I:
    a) Defina uma janela de tamanho $k \times k$ centrada no pixel (i, j).
    b) Calcule a média dos valores dos pixels dentro da janela usando a Equação 1.
    c) Atribua o valor médio calculado ao pixel (i, j) na nova imagem .
3. Converta os valores da imagem resultante para o tipo de dados original da imagem I e salve a imagem resultante em um arquivo.

4.1.2 Filtro da mediana

O filtro de mediana é um método de filtragem não linear que é frequentemente usado para suavizar uma imagem, preservando ao mesmo tempo as bordas. Em vez de substituir o valor de um pixel pela média dos valores dos pixels vizinhos, como no filtro de média, o filtro de mediana substitui o valor do pixel pelo valor mediano dos valores dos pixels vizinhos. Dada uma imagem I de dimensões $m \times n$, o valor de um pixel (i, j) na imagem filtrada I_{mediana} usando um filtro de mediana de tamanho de janela $k \times k$ é dado por:

$$I_{\text{mediana}}(i, j) = \text{mediana}\left(\{I(i - l, j - m) \mid 0 \leq l < k,\ 0 \leq m < k\}\right) \tag{2}$$

onde mediana é a função que retorna o valor mediano de um conjunto de valores. Para implementar o filtro de mediana em uma imagem digital, o seguinte algoritmo foi desenvolvido:

1. Inicializa uma nova imagem com as mesmas dimensões da imagem original I, preenchendo a nova imagem com zeros.

2. Para cada pixel (i, j) na imagem original I:

   a) Define uma janela de tamanho $k \times k$ centrada no pixel (i, j).

   b) Extrai os valores dos pixels dentro da janela.

   c) Ordena os valores extraídos em ordem crescente.

   d) Atribue o valor mediano calculado ao pixel (i, j) na nova imagem usando a Equação 2.

3. Converte os valores da imagem obtida para o tipo de dados original da imagem I e salve a imagem resultante em um arquivo.

4.1.3 Filtro Gaussiano

O filtro Gaussiano é um filtro linear utilizado para suavizar imagens, reduzindo o ruído e os detalhes finos. A operação do filtro Gaussiano é baseada na aplicação de um kernel Gaussiano à imagem original. O kernel Gaussiano é uma matriz bidimensional que contém valores calculados a partir da função Gaussiana. Dada uma imagem I de dimensões $m \times n$ e um kernel Gaussiano G de tamanho $k_m \times k_n$ e desvio padrão σ, a imagem filtrada $I_{\text{gaussiano}}$ é dada por:

$$G(i, j) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i - \text{centro}_x)^2 + (j - \text{centro}_y)^2}{2\sigma^2}\right) \tag{3}$$

$$I_{\text{gaussiano}}(x, y) = \sum_{i=0}^{k_m-1} \sum_{j=0}^{k_n-1} I(x + i, y + j) \times G(i, j)$$

Onde centro_x e centro_y são as coordenadas do centro do kernel Gaussiano, σ é o desvio padrão do kernel Gaussiano e $I(x, y)$ é o valor do pixel na posição (x, y) na imagem original. Para implementar o filtro Gaussiano em uma imagem digital, o seguinte algoritmo foi desenvolvido:

1. Determine o tamanho do kernel Gaussiano. Caso o tamanho da janelas seja par, $(k_m + 1) \times (k_n + 1)$, caso contrário $k_m \times k_n$ com base no tamanho da janela especificado e calcule o centro do kernel.

2. Calcule os valores do kernel Gaussiano usando a Equação 3.

3. Normalize o kernel para garantir que a soma de todos os seus elementos seja igual a 1.

4. Aplique o kernel Gaussiano à imagem original usando a Equação 3 para calcular o valor de cada pixel na imagem filtrada.

5. Converta os valores da imagem filtrada para o tipo de dados original da imagem I e salve a imagem resultante em um arquivo.

A implementação do filtro Gaussiano considera tanto janelas de tamanho par quanto ímpar. Se o tamanho da janela for par, um kernel de tamanho $(k_m + 1) \times (k_n + 1)$ é utilizado para garantir que o centro do kernel corresponda ao centro da janela. Se o tamanho da janela for ímpar, um kernel de tamanho $k_m \times k_n$ é utilizado.

4.2 Filtros Passa-Alta

Os filtros passa-alta são amplamente utilizados em processamento de imagens para realçar bordas e detalhes de uma imagem, enfatizando as frequências altas e atenuando as frequências baixas. Estes filtros operam permitindo a passagem de frequências altas (frequências relacionadas a variações rápidas de intensidade) e atenuando as frequências baixas (frequências relacionadas a variações lentas de intensidade). O cálculo genérico de um filtro passa-alta é dado por:

$$H(u, v) = \begin{cases} 1 & \text{se } D(u, v) > D_0 \\ 0 & \text{caso contrário} \end{cases}$$

onde $H(u, v)$ é a função de transferência do filtro passa-alta, $D(u, v)$ é a distância da frequência (u, v) ao centro da frequência da imagem, e D_0 é o raio do corte do filtro.

4.2.1 Filtro Laplaciano

O filtro Laplaciano é um filtro linear utilizado para realçar as bordas em uma imagem, realçando as mudanças abruptas de intensidade. A operação do filtro Laplaciano é baseada na convolução da imagem original com uma máscara Laplaciana. Dada uma imagem I de dimensões $m \times n$ e uma máscara Laplaciana L de tamanho $k \times k$, a imagem filtrada $I_{\text{laplaciana}}$ é dada por:

$$I_{\text{laplaciana}}(x, y) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} I(x + i, y + j) \times L(i, j) \tag{4}$$

onde $I_{\text{laplaciana}}(x, y)$ é o valor do pixel na posição (x, y) na imagem filtrada, $I(x, y)$ é o valor do pixel na posição (x, y) na imagem original e $L(i, j)$ é o valor do elemento na posição (i, j) na máscara Laplaciana. Para implementar o filtro Laplaciano em uma imagem digital, o seguinte algoritmo foi desenvolvido:

1. Verifique se o tamanho da máscara Laplaciana é válido para o tamanho da janela especificado. Se não for, lance um erro indicando um tamanho de máscara inválido.

2. Para cada pixel (i, j) na imagem original I, extraia a janela de tamanho $k \times k$ centrada em (i, j).

3. Calcule o valor Laplaciano para a janela usando a máscara Laplaciana.

4. Atribua o valor Laplaciano calculado ao pixel (i, j) na imagem filtrada $I_{\text{laplaciana}}$ usando a Equação 4.

5. Converta os valores da imagem filtrada para o tipo de dados original da imagem I e salve a imagem resultante em um arquivo.

O desenvolvimento da função `filtroLaplaciano` tem uma peculiaridade em relação ao algoritmo clássico. O algoritmo clássico utiliza uma máscara Laplaciana específica que é aplicada à imagem inteira, enquanto a implementação desenvolvida aceita uma máscara Laplaciana genérica fornecida pelo usuário permitindo uma maior flexibilidade na aplicação do filtro Laplaciano e a utilização de diferentes máscaras Laplacianas nas imagens.

4.2.2 Filtro de Prewitt

O filtro Prewitt é um filtro linear utilizado para detecção de bordas em uma imagem. Ele é composto por duas máscaras convolucionais, uma para detecção de bordas horizontais e outra para detecção de bordas verticais. Dada uma imagem I de dimensões $m \times n$, as máscaras convolucionais M_x e M_y para o filtro Prewitt são definidas como:

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

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

A imagem filtrada I_{PrewittX} e I_{PrewittY} são calculadas pelas equações:

$$I_{\text{PrewittX}}(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} I(x + i, y + j) \times M_x(i, j) \tag{5}$$

$$I_{\text{PrewittY}}(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} I(x + i, y + j) \times M_y(i, j) \tag{6}$$

A magnitude do gradiente da imagem é calculada pela seguinte equação:

$$I_{\text{Prewitt}}(x, y) = \sqrt{I_{\text{PrewittX}}(x, y)^2 + I_{\text{PrewittY}}(x, y)^2} \tag{7}$$

Onde:

- $I_{\text{PrewittX}}(x, y)$ é o valor do gradiente na direção horizontal no pixel (x, y).
- $I_{\text{PrewittY}}(x, y)$ é o valor do gradiente na direção vertical no pixel (x, y).
- $I_{\text{Prewitt}}(x, y)$ é a magnitude do gradiente no pixel (x, y).
- $I(x, y)$ é o valor do pixel na posição (x, y) na imagem original.

Para implementar o filtro Prewitt em uma imagem digital, o seguinte algoritmo foi desenvolvido:

1. Definir as máscaras convolucionais M_x e M_y.
2. Para cada pixel (i, j) na imagem original I, extraia a janela de tamanho 3×3 centrada em (i, j).
3. Calcule os valores I_{PrewittX} e I_{PrewittY} usando as Equações 5 e 6.
4. Calcule a magnitude do gradiente I_{Prewitt} usando a Equação 7.
5. Normalize os valores da imagem filtrada para o intervalo [0, 255].
6. Converta os valores da imagem filtrada para o tipo de dados original da imagem I e salve a imagem resultante em um arquivo.

A implementação da função `filtroPrewitt` segue o algoritmo clássico do filtro Prewitt, utilizando as máscaras convolucionais padrão para detecção de bordas horizontais e verticais. No entanto, o processo de normalização dos valores da imagem filtrada é realizado para garantir que os valores estejam no intervalo [0, 255].

4.2.3 Filtro de Sobel

O filtro Sobel é um filtro linear utilizado para detecção de bordas em uma imagem. Ele é composto por duas máscaras convolucionais, uma para detecção de bordas horizontais e outra para detecção de bordas verticais. Dada uma imagem I de dimensões $m \times n$, as máscaras convolucionais M_x e M_y para o filtro Sobel são definidas como:

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

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

A imagem filtrada I_{SobelX} e I_{SobelY} são calculadas pelas equações:

$$I_{\text{SobelX}}(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} I(x + i, y + j) \times M_x(i, j) \tag{8}$$

$$I_{\text{SobelY}}(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} I(x + i, y + j) \times M_y(i, j) \tag{9}$$

A magnitude do gradiente da imagem é calculada pela seguinte equação:

$$I_{\text{Sobel}}(x, y) = \sqrt{I_{\text{SobelX}}(x, y)^2 + I_{\text{SobelY}}(x, y)^2} \tag{10}$$

Onde:

- $I_{\text{SobelX}}(x, y)$ é o valor do gradiente na direção horizontal no pixel (x, y).
- $I_{\text{SobelY}}(x, y)$ é o valor do gradiente na direção vertical no pixel (x, y).
- $I_{\text{Sobel}}(x, y)$ é a magnitude do gradiente no pixel (x, y).
- $I(x, y)$ é o valor do pixel na posição (x, y) na imagem original.

Para implementar o filtro Sobel em uma imagem digital, o seguinte algoritmo foi desenvolvido:

1. Definir as máscaras convolucionais M_x e M_y.
2. Para cada pixel (i, j) na imagem original I, extraia a janela de tamanho 3×3 centrada em (i, j).
3. Calcule os valores I_{SobelX} e I_{SobelY} usando as Equações 8 e 9.
4. Calcule a magnitude do gradiente I_{Sobel} usando a Equação 10.
5. Normalize os valores da imagem filtrada para o intervalo [0, 255].

6. Converta os valores da imagem filtrada para o tipo de dados original da imagem I e salve a imagem resultante em um arquivo.

A implementação da função `filtroSobel` segue o algoritmo clássico do filtro Sobel, utilizando as máscaras convolucionais padrão para detecção de bordas horizontais e verticais. Assim como no filtro Prewitt, o processo de normalização dos valores da imagem filtrada é realizado para garantir que os valores estejam no intervalo [0, 255].

4.3 Operadores

Os operadores são algoritmos cruciais para a extração de informações em imagens. As etapas fundamentais desse processo englobam o cálculo e apresentação do histograma, a equalização do histograma, a limiarização e a multilimiarização. Estas etapas visam, respectivamente, proporcionar uma representação gráfica da distribuição de intensidades na imagem, melhorar o contraste da mesma e realizar a segmentação detalhada da imagem.

4.3.1 Cálculo e Apresentação de Histograma

O histograma de uma imagem é uma representação gráfica da distribuição de intensidades de pixels em uma imagem. Para uma imagem em escala de cinza, o histograma é uma função $h(i)$ que mapeia a intensidade i no intervalo [0, 255] para o número de pixels com essa intensidade. Para uma imagem colorida, o histograma é dividido em três canais: vermelho (R), verde (G) e azul (B). Para cada canal, o histograma é uma função $h(c, i)$, onde c representa o canal de cor (R, G ou B) e i representa a intensidade no intervalo [0, 255].

$$h(i) = \sum_{x=0}^{M} \sum_{y=0}^{N} \delta(I(x, y) - i)$$

$$h(c, i) = \sum_{x=0}^{M} \sum_{y=0}^{N} \delta(I(x, y, c) - i)$$

Onde:

- $I(x, y)$ é a intensidade do pixel na posição (x, y).
- $I(x, y, c)$ é a intensidade do canal c do pixel na posição (x, y).
- δ é a função delta de Dirac, que é 1 se a condição for verdadeira e 0 caso contrário.
- M e N são as dimensões da imagem.

Para determinar e verificar o formato da imagem (RGB, RGBA ou Grayscale), a função `verificarImagem` é utilizada. Esta função verifica o número de canais da imagem e retorna

uma string indicando o formato. A função `apresentarHistograma` é responsável por calcular e apresentar o histograma da imagem. Dependendo do formato da imagem, esta função calculará e apresentará o histograma de um canal de cor ou o histograma da imagem em escala de cinza.

- Calcula os histogramas individuais para o todos os canais da imagem utilizando a função `calcularHistograma`.
- Apresenta os histogramas dos canais em subplots separados.
- Salva o histograma resultante em um arquivo.

A função `calcularHistograma` é adaptada para suportar imagens tanto em escala de cinza quanto coloridas. Se a imagem for colorida, o histograma é calculado para cada canal de cor individualmente. Além disso, a função `apresentarHistograma` utiliza a biblioteca `matplotlib.pyplot` para visualizar e salvar os histogramas, proporcionando uma representação visual clara da distribuição de intensidades de pixels na imagem.

4.3.2 Equalização de Histograma

A equalização de histograma é uma técnica utilizada para melhorar o contraste em imagens. O objetivo é distribuir as intensidades de pixel de forma mais uniforme, resultando em uma imagem com melhor qualidade visual e detalhes mais claros. A equalização de histograma é definida pela seguinte equação:

$$\text{Probabilidade}(r_k) = \sum_{j=0}^{k} \frac{n_j}{n}$$

$$\text{Equalização}(s_k) = T(r_k) = \text{round}\left(\text{Probabilidade}(r_k) \times L\right)$$

Onde:

- r_k é o valor de intensidade de pixel na imagem original.
- s_k é o valor de intensidade de pixel na imagem equalizada.
- n_j é o número de pixels com intensidade j na imagem original.
- n é o número total de pixels na imagem.
- L é o número de níveis de intensidade (geralmente 256 para imagens de 8 bits).

Antes de aplicar a equalização de histograma, a imagem colorida é convertida para escala de cinza (luminância) utilizando a seguinte fórmula:

$$\text{Luminância} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Onde R, G e B são os valores dos canais de cor vermelho, verde e azul, respectivamente.

A função `equalizador` realiza a equalização de histograma da imagem. Primeiramente, a frequência de cada valor de intensidade de pixel é calculada. Em seguida, a probabilidade acumulada e o novo valor de intensidade equalizado são calculados para cada valor de intensidade. A imagem original é então atualizada com os novos valores de intensidade equalizados. A função `equalizarImagemHistograma` utiliza a função de conversão para luminância para transformar a imagem colorida em uma imagem em escala de cinza. Em seguida, a equalização de histograma é aplicada à imagem em escala de cinza e o histograma equalizado é plotado. Além disso, a imagem equalizada é exibida e salva.

4.3.3 Limiarização

A limiarização é uma técnica de processamento de imagem utilizada para segmentar uma imagem em duas regiões: uma região de interesse, que é geralmente de interesse para análise ou processamento posterior, e uma região de fundo. A limiarização é definida pela seguinte equação:

$$\text{Imagem Binária}(x, y) = \begin{cases} 255 & \text{se Luminância}(x, y) > \text{Limiar} \\ 0 & \text{caso contrário} \end{cases}$$

Onde:

- Luminância(x, y) é o valor de luminância do pixel na posição (x, y).
- Limiar é o valor de limiar especificado pelo usuário para a limiarização.

A função `limiarizacao` recebe uma imagem e um valor de limiar como entrada. Primeiramente, a função converte a imagem colorida em uma imagem em escala de cinza (luminância) utilizando a função `luminancia`. Em seguida, a função percorre cada pixel da imagem e compara o valor de luminância do pixel com o valor de limiar especificado. Se o valor de luminância for maior que o limiar, o pixel é definido como branco (255); caso contrário, o pixel é definido como preto (0). A imagem binarizada resultante é então salva em um arquivo.

A função `limiarizacao` utiliza a conversão para luminância da imagem antes de aplicar a limiarização. Esta abordagem é particularmente útil para imagens coloridas, onde a limiarização é aplicada na imagem em escala de cinza para garantir uma segmentação precisa. A implementação é direta e eficiente, realizando a limiarização em um único passo sem a necessidade de cálculos adicionais ou estruturas de dados complexas.

4.3.4 Multilimiarização

A multilimiarização é uma extensão da técnica de limiarização, onde a imagem é segmentada em múltiplas regiões usando vários limiares. A multilimiarização é definida pela seguinte equação:

$$\text{Imagem Multinível}(x, y) = \begin{cases} n_1 & \text{se Luminância}(x, y) < \text{Limiar}_1 \\ n_2 & \text{se Limiar}_1 \leq \text{Luminância}(x, y) < \text{Limiar}_2 \\ \vdots & \\ n_k & \text{se Limiar}_{k-1} \leq \text{Luminância}(x, y) < \text{Limiar}_k \\ n_{k+1} & \text{se Luminância}(x, y) \geq \text{Limiar}_k \end{cases}$$

Onde:

- Luminância(x, y) é o valor de luminância do pixel na posição (x, y).
- Limiar$_i$ é o i-ésimo valor de limiar especificado pelo usuário.
- n_i é o i-ésimo nível de intensidade para o pixel segmentado.

A função `niveisDeLimiarizacao` recebe uma imagem em escala de cinza e uma lista de limiares e níveis como entrada. A função inicializa uma imagem binária multilimiarizada com zeros, e então atribui a cada pixel um valor de nível com base nos limiares fornecidos. A função `multilimiarizacao` utiliza a função `niveisDeLimiarizacao` para gerar uma imagem multilimiarizada e a salva em um arquivo. A função `niveisDeLimiarizacao` é capaz de segmentar a imagem em múltiplos níveis de intensidade usando uma abordagem direta e eficiente. Ao contrário da limiarização tradicional que gera uma imagem binária, a multilimiarização permite uma segmentação mais granular da imagem, atribuindo diferentes níveis de intensidade a diferentes regiões da imagem com base nos limiares especificados.

5 RESULTADOS E DISCUSSÕES

Nesta seção, apresentamos os resultados e discussões da aplicação de diferentes filtros de processamento de imagens utilizando janelas de tamanhos variados. Os filtros utilizados incluem os filtros de média, mediana e gaussiano, que possuem janelas de tamanhos 2×2, 3×3 e 4×4. Além desses, foram utilizados os filtros laplaciano, prewitt e sobel, que possuem janelas de tamanhos fixos. A interface do trabalho foi implementada em Python, utilizando o módulo argparse para o processamento de argumentos da linha de comando. A imagem utilizada nos experimentos foi uma imagem colorida que é convertida em tons de cinza, carregada usando a biblioteca OpenCV. Para medir os tempos de computação de cada filtro e operação, foi utilizada a biblioteca time do Python. A imagem utilizada é a seguinte:

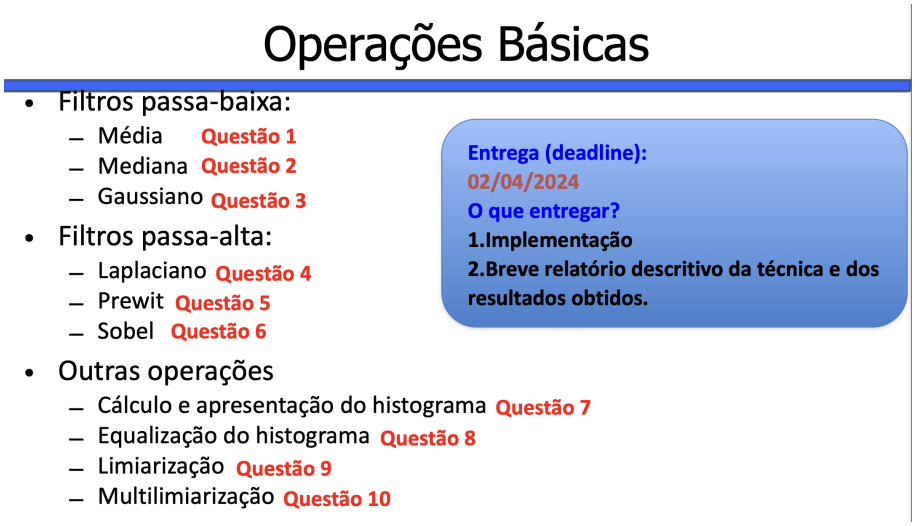

Figura 1 – Imagem usada.

5.1 Filtro da média

O tempo de execução para cada filtro da média é apresentado na Tabela 1.

Tamanho da Janela	Tempo de Execução (s)
2×2	5.823
3×3	8.645
4×4	12.350

Tabela 1 – Tempo de execução do filtro da média para diferentes tamanhos de janela.

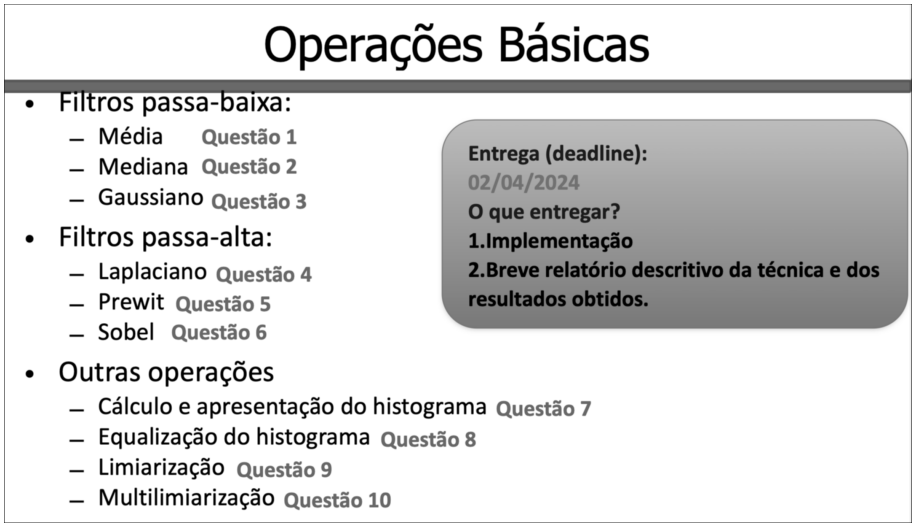

Figura 2 – Imagens resultantes da aplicação do filtro da média com janelas de 3×3.

A aplicação do filtro da média demonstrou eficácia na redução de ruídos nas imagens experimentais. Conforme esperado, janelas de tamanho maior produzem uma suavização mais acentuada da imagem, mas também podem levar à perda de detalhes. A escolha do tamanho da janela deve ser feita com base na natureza do ruído presente na imagem e no nível de detalhe desejado na imagem resultante.

5.2 Filtro da mediana

O tempo de execução para cada calculo do filtro da mediana nas janelas propostas é apresentado na Tabela 2.

Tamanho da Janela	Tempo de Execução (s)
2×2	3.115
3×3	5.608
4×4	8.288

Tabela 2 – Tempo de execução do filtro da mediana para diferentes tamanhos de janela.

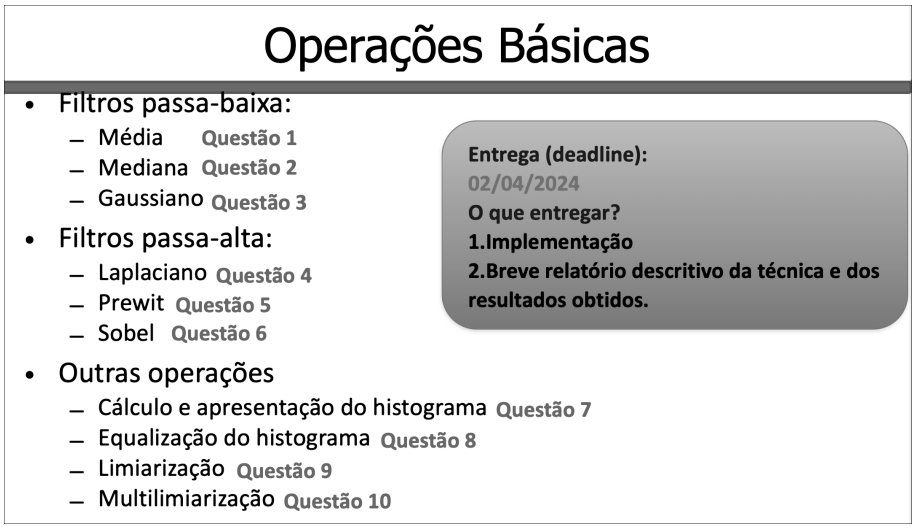

Figura 3 – Imagens resultantes da aplicação do filtro da mediana com janelas de 3×3.

A implementação da função `filtroMediana` segue a abordagem clássica de aplicar a janela em cada pixel da imagem e calcular a mediana dos valores dos pixels dentro da janela. O resultado é então armazenado na imagem resultante. Um ponto a ser observado é que a função implementada calcula a mediana usando a ordenação da janela, o que é um processo custoso. No entanto, essa abordagem garante a correta determinação da mediana. Em aplicações práticas, dependendo do tamanho da janela e do hardware utilizado, pode ser necessário otimizar a função para melhorar o desempenho.

5.3 Filtro Gaussiano

A função `filtroGaussiano` implementa o filtro Gaussiano em uma imagem. A seguir, apresentamos a função e sua discussão detalhada. O tempo de execução para cada calculo do filtro gaussiano nas janelas propostas é apresentado na Tabela 3. Nesse experimento utilizaremos o sigma com valor de 1 .

Nesse experimento buscamos o tempo de execução para cada calculo do filtro gaussiano com janelas fixa em 3x3, pórem com o sigma variando. Isso é apresentado na Tabela 4.

Tamanho da Janela	Tempo de Execução (s)	Sigma
2×2	8.132	1.0
3×3	6.632	1.0
4×4	8.147	1.0

Tabela 3 – Tempo de execução do filtro gaussiano para diferentes tamanhos de janela.

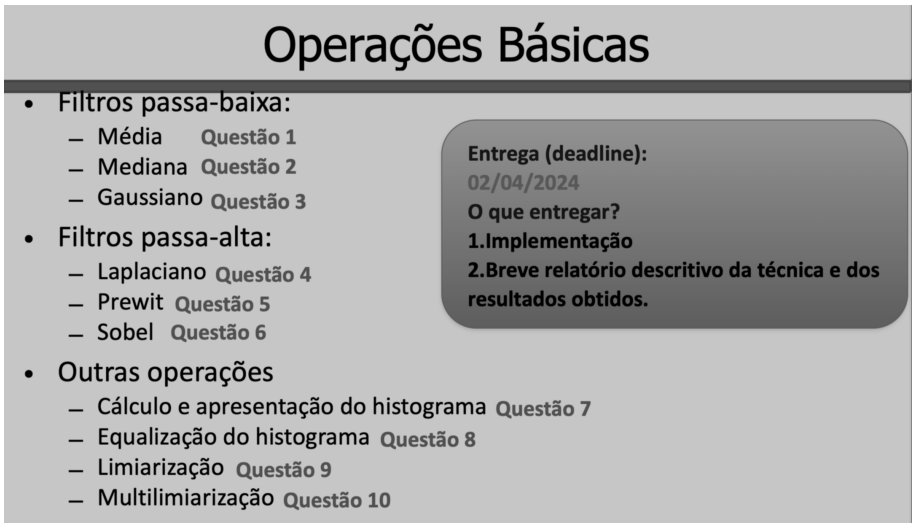

Figura 4 – Imagens resultantes da aplicação do filtro gaussiano com janelas de 3×3.

Tamanho da Janela	Tempo de Execução (s)	Sigma
3×3	8.132	0.5
3×3	6.632	1.5
3×3	8.147	2.0

Tabela 4 – Tempo de execução do filtro gaussiano para diferentes valores de sigma.

A implementação inclui a verificação do tamanho da janela para assegurar que o kernel seja gerado corretamente. O kernel é determinado utilizando a função Gaussiana. O filtro Gaussiano é um filtro linear frequentemente utilizado para suavizar imagens. A função `filtroGaussiano` realiza a suavização Gaussiana por meio da operação de convolução entre a imagem e o kernel Gaussiano. É crucial observar que, para janelas de tamanho par, o kernel é multiplicado por 2 após sua criação. Isso é necessário para garantir que a soma total dos valores no kernel seja aproximadamente igual a 1, preservando assim o brilho da imagem. A escolha do tamanho da janela e do valor de σ deve ser baseada no tipo de ruído presente na imagem e no grau de suavização desejado na imagem final. Janelas maiores e valores mais altos de σ proporcionam uma suavização mais intensa, porém podem resultar na perda de detalhes na imagem. (GONZALEZ; WOODS, 2008)

A implementação inclui a verificação do tamanho da janela para assegurar que o kernel seja gerado corretamente. O kernel é determinado utilizando a função Gaussiana:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

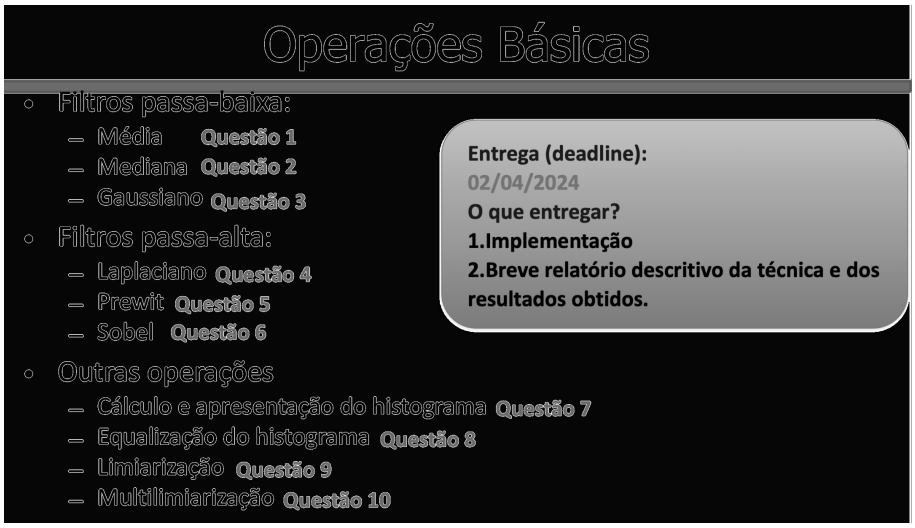

Figura 5 – Imagens resultantes da aplicação do filtro gaussiano com sigma de 0,5.

5.4 Filtro Laplaciano

O tempo de execução para o cálculo do filtro Laplaciano em cada uma das janelas propostas está apresentado na Tabela 5.

Nesse experimento utilizaremos máscaras e janelas de tamanho 2×2, 3×3 e 4×4 respectivamente. Importatente salientar que as máscaras devem ser matrizes quadradas. Os valores das máscaras estão descritos abaixo:

$$\text{Máscara } 2 \times 2 = \begin{bmatrix} 2 & -1 \\ -1 & 5 \end{bmatrix} \tag{11}$$

$$\text{Máscara } 3 \times 3 = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & 5 & 3 \end{bmatrix} \tag{12}$$

$$\text{Máscara } 4 \times 4 = \begin{bmatrix} 3 & -1 & 0 & 4 \\ -1 & 4 & -1 & 2 \\ 0 & -1 & 0 & 1 \\ 2 & 4 & 1 & 0 \end{bmatrix} \tag{13}$$

Tamanho da Janela	Tempo de Execução (s)	Máscara
2×2	6.499	11
3×3	6.566	12
4×4	6.515	13

Tabela 5 – Tempo de execução do filtro lapraciano para diferentes tamanhos de janela e máscaras

.

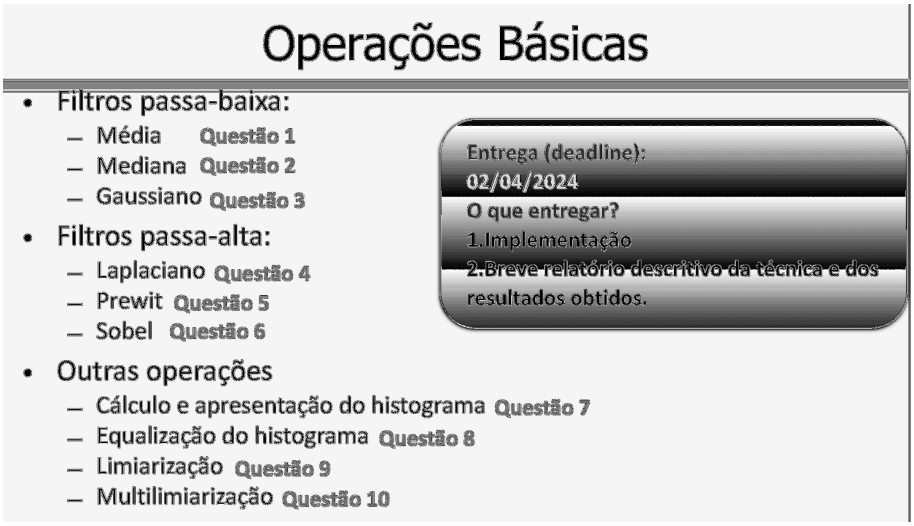

Figura 6 – Imagens resultantes da aplicação do filtro laplaciano com janela e máscara 3×3.

O filtro Laplaciano é um filtro usado principalmente para detecção de bordas em imagens. A função `filtroLaplaciano` implementa o filtro Laplaciano utilizando a operação de convolução entre a imagem e uma máscara laplaciana. A escolha da máscara do filtro Laplaciano e do tamanho da janela deve ser feita com base no tipo de bordas que se deseja detectar e na natureza do ruído presente na imagem. Note que a função termina a execução se o tamanho da máscara não for válido para o filtro Laplaciano, garantindo assim que a operação de convolução seja realizada corretamente.

5.5 Filtro Prewitt

O tempo médio de execução para o cálculo do filtro Prewitt em cada uma das janelas propostas está apresentado na Tabela 6.

A finalidade do filtro Prewitt é a detecção de bordas em uma imagem e após aplicar as máscaras de Prewitt horizontal e vertical, a imagem gradiente é calculada usando a magnitude das derivadas em ambas as direções.

O filtro Prewitt duas máscaras: uma para detecção de bordas horizontais e outra para detecção de bordas verticais. Como a janela é fixa o experimento será executado 100 vezes e calculado a média. As máscaras utilizadas são:

$$\text{Máscara horizontal: } \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{14}$$

$$\text{Máscara vertical: } \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{15}$$

Tamanho da Janela	Tempo de Execução (s)
3×3	12.625

Tabela 6 – Tempo médio de execução do filtro Prewitt para 100 requisições .

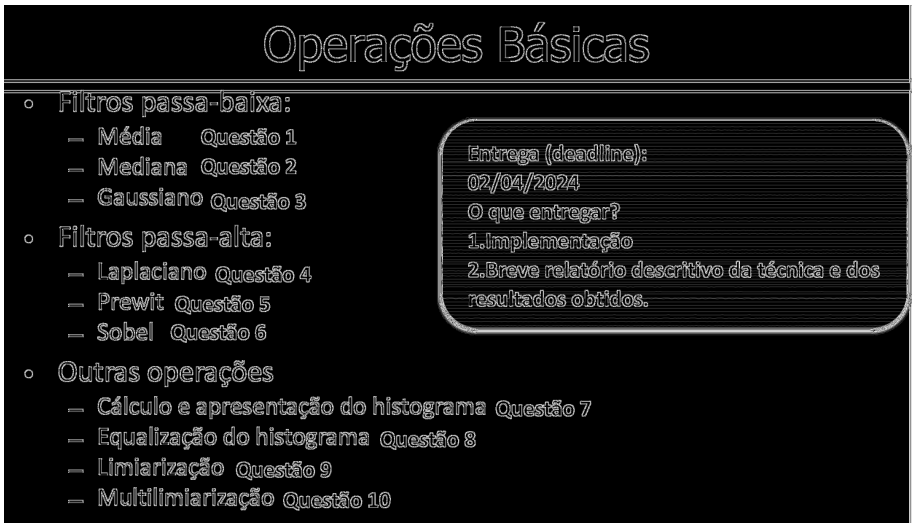

Figura 7 – Imagens resultantes da aplicação do filtro Prewitt

O filtro desenvolvido mostrou-se eficaz na detecção de bordas em imagens, fornecendo uma representação clara das transições de intensidade na imagem original. A implementação da função ‘`filtroPrewitt` demonstrou ser simples e direta, utilizando operações matriciais para processar a imagem de forma eficiente e a resultante é normalizada para mapear os valores para o intervalo [0, 255].

5.6 Filtro Sobel

A função `filtroSobel` foi implementada utilizando laços aninhados para percorrer a imagem original e aplicar as máscaras de Sobel.

Como a janela é fixa o experimento será executado 100 vezes e calculado a média.

O tempo médio de execução para o cálculo do filtro Sobel em cada uma das janelas propostas está apresentado na Tabela 7.As máscaras utilizadas são:

$$\text{Máscara das abscissas: } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{16}$$

$$\text{Máscara das ordenadas: } \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{17}$$

Tamanho da Janela	Tempo de Execução (s)
3×3	12.852

Tabela 7 – Tempo médio de execução do filtro Sobel para 100 requisições .

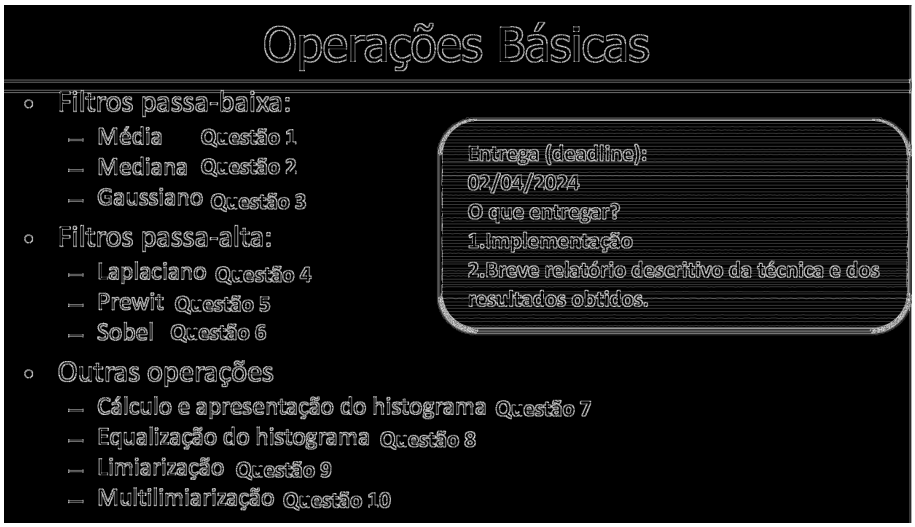

Figura 8 – Imagens resultantes da aplicação do filtro Sobel

O filtro de Sobel desenvolvido é eficaz na detecção de bordas em imagens, proporcionando uma representação clara das transições de intensidade na imagem original. A implementação do filtro demonstrou ser simples e direta, utilizando operações matriciais para processar a imagem de forma eficiente.

5.7 Calculo e Apresentação de Histograma

O histograma é uma ferramenta importante no processamento de imagens, sendo frequentemente utilizada para análise e manipulação de imagens.

Neste contexto, as funções calcularHistograma, verificarImagem e apresentarHistograma foram desenvolvidas para calcular e visualizar o histograma de imagens em diferentes formatos: RGB, RGBA e Grayscale. O estudo apresenta uma abordagem para a análise e interpretação de imagens digitais por meio da implementação de funções específicas que calculam e visualizam o histograma de imagens em diversos formatos. A utilização dessas funções é integrada à biblioteca Matplotlib para a representação gráfica dos histogramas.

O experimento calcula o tempo de execução das respectivas imagens propostas está apresentado na Tabela 8.

Imagem	Tempo de Execução (s)
RGB	1.452
RGBA	1.871
Grayscale	1.223

Tabela 8 – Tempo de execução do cálculo e apresentação de histograma para imagens.

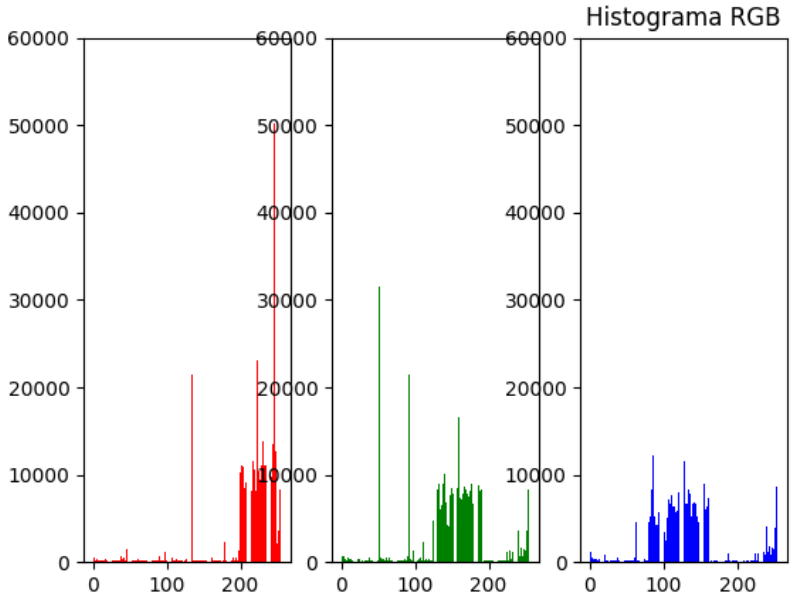

Figura 9 – Imagens resultantes do Calculo e Apresentação de Histograma

No contexto das imagens RGB (Red, Green, Blue) e RGBA (Red, Green, Blue, Alpha), o histograma de cada canal de cor (Vermelho, Verde e Azul) desempenha um papel crucial na compreensão da composição da imagem. Ele permite a identificação de áreas de alta ou baixa intensidade em cada canal, fornecendo informações valiosas sobre a distribuição espectral da imagem (SMITH, 2005).

Por outro lado, para imagens em escala de cinza (Grayscale), o histograma revela a distribuição de intensidades de cinza na imagem, permitindo uma análise detalhada das variações tonais presentes.

5.8 Equalização de Histograma

O objetivo da técnica implementada é redistribuir as intensidades dos pixels de uma imagem de forma que o histograma resultante seja aproximadamente uniforme. Neste contexto, as funções equalizador e equalizarImagemHistograma foram desenvolvidas para equalizar o histograma de uma imagem e visualizar o resultado da equalização.

O experimento calcula o tempo de execução das respectivas imagens propostas está apresentado na Tabela 9. Como a equalização é uma operação fixa o experimento será executado 100 vezes e calculado a média.

Imagem	Tempo de Execução (s)
Grayscale	1.218

Tabela 9 – Tempo médio de execução da equalização de histograma para imagens grayscale.

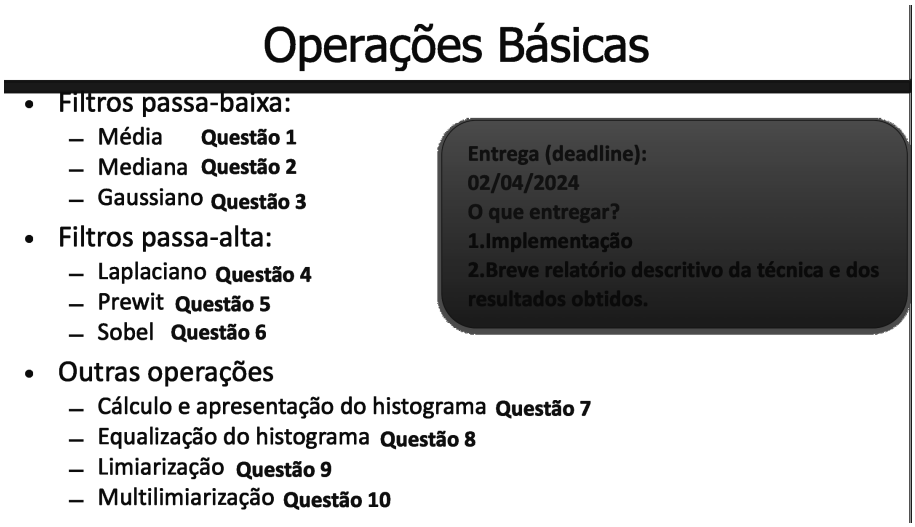

Figura 10 – Imagens resultantes da equalização de Histograma

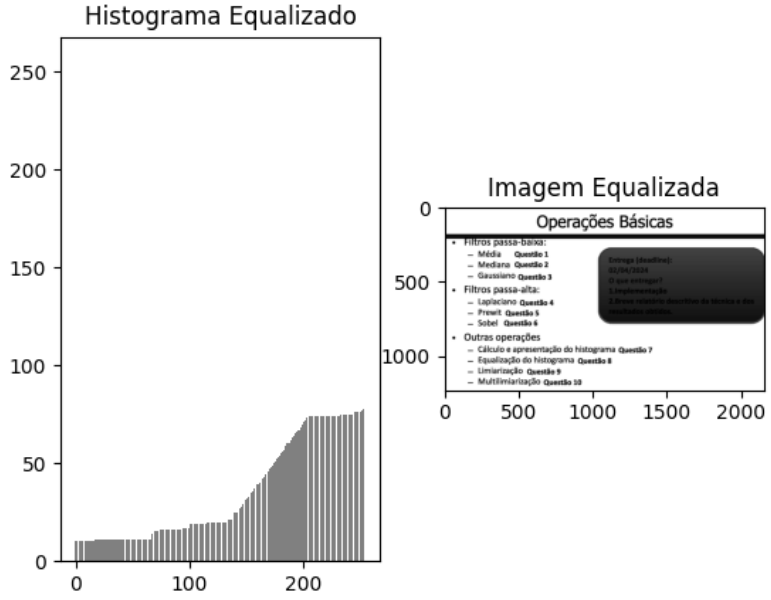

Figura 11 – Gráfico da equalização de Histograma

A implementação destas funções permite equalizar o histograma de uma imagem e visualizar o resultado da equalização, melhorando o contraste e a qualidade visual da imagem. No entanto, é importante observar que a equalização de histograma pode não ser adequada para todas as imagens, especialmente aquelas que já possuem um bom contraste.

5.9 Limiarização

As técnicas desenvolvidas para esse aspecto do trabalho são o cálculo da luminância e a limiarização. A luminância é uma medida da intensidade da luz percebida por um observador humano, enquanto a limiarização é o processo de transformar uma imagem em uma imagem binária com base em um valor de limiar. Neste contexto, as funções luminancia e limiarizacao

foram implementadas para calcular a luminância de uma imagem e realizar a limiarização da imagem.

O experimento calcula o tempo de execução dos seguintes limiares: 50, 127, 150 e 200. O resultado está apresentado na Tabela 10.

Limiar	Tempo de Execução (s)
50	2.044
127	2.024
150	2.088
200	2.031

Tabela 10 – Tempo de execução da limiarização para imagens grayscale.

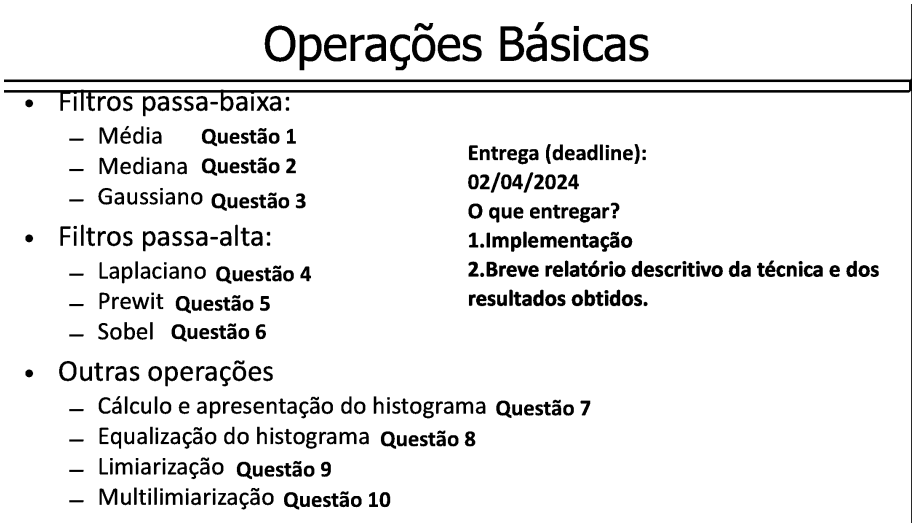

Figura 12 – Imagem resultante da limiarização

A combinação das funções de luminância e limiarização permite calcular a luminância de uma imagem colorida e posteriormente realizar a limiarização da imagem de luminância, transformando-a em uma imagem binária. Esta abordagem é essencial para diversas aplicações em processamento de imagens, facilitando a análise e interpretação de imagens digitais. Inicialmente, a função de luminância foi aplicada a uma imagem colorida de teste. Em seguida, a função de limiarização foi aplicada à imagem de luminância resultante, utilizando os valores propostos no experimento.

5.10 Multilimiarização

Em contraste com a limiarização simples, que divide a imagem em apenas duas regiões (por exemplo, objeto e fundo), a multilimiarização pode segmentar a imagem em múltiplos níveis de intensidade. Neste contexto, a função multilimiarizacao foi implementada para realizar a multilimiarização de uma imagem utilizando os valores de limiar e os níveis especificados.

O experimento investiga o tempo de execução ao utilizar 3 e 4 limiares e níveis diferentes. Os limiares adotados foram 50, 100, 150 e 200, enquanto os níveis variaram entre 0, 85, 170 e 255. Os resultados obtidos estão organizados na Tabela 11.

Niveis	Limiares	Tempo de Execução (s)
2	2	0.059
2	3	0.043
2	4	0.045
3	2	0.048
3	3	0.047
3	4	0.054
4	3	0.059
4	4	0.066

Tabela 11 – Tempo de execução da multilimiarização em diferentes niveis e limiares para imagens grayscale.

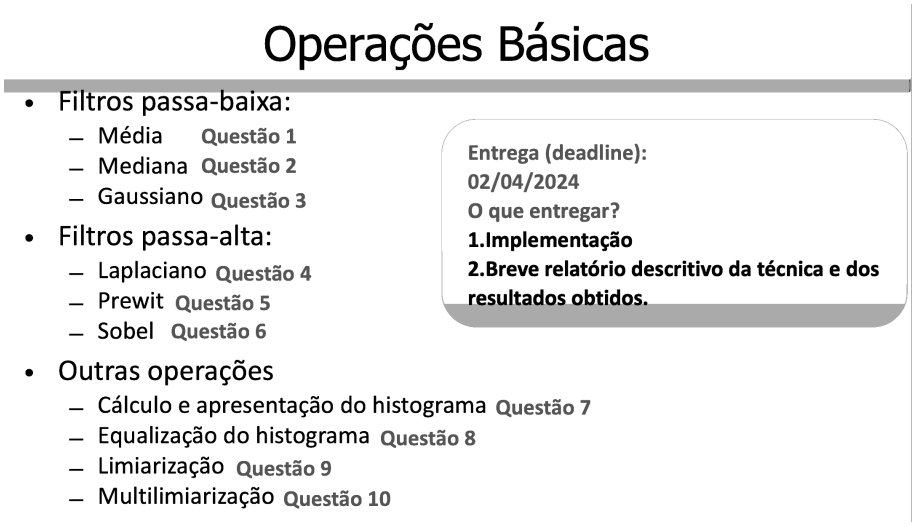

Figura 13 – Imagem resultante da multilimiarização

A combinação das funções luminancia e multilimiarizacao possibilita o cálculo da luminância de uma imagem colorida e a subsequente segmentação da imagem em diferentes níveis de intensidade por meio da multilimiarização. A função multilimiarizacao foi eficientemente implementada para executar a segmentação da imagem utilizando os valores de limiar e os níveis determinados. Além disso, a função niveisDeLimiarizacao foi empregada para aplicar a multilimiarização na imagem de luminância, segmentando-a em diversos níveis de intensidade com base nos valores de limiar e nos níveis definidos.

6 CONCLUSÕES

Neste trabalho, foi apresentada uma implementação completa de diversos filtros e técnicas de processamento de imagens utilizando Python e bibliotecas como NumPy, OpenCV,

Matplotlib e como desenvolvimento de interface a biblioteca Argparse. Destaque como os principais pontos e contribuições deste trabalho:

- **Filtros de Suavização**: Foi implementado três filtros de suavização - Filtro da Média, Filtro da Mediana e Filtro Gaussiano. Estes filtros são essenciais para a redução de ruído e suavização de imagens, tornando-as mais apropriadas para processamentos subsequentes.
- **Filtros de Detecção de Bordas**: Foram implementados os Filtros de Prewitt e Sobel. Estes filtros são amplamente utilizados na detecção de bordas em imagens, o que é crucial em muitas aplicações de processamento de imagens, como detecção de objetos e segmentação de imagens.
- **Filtro Laplaciano**: Implementou-se o Filtro Laplaciano, que é útil para realçar regiões de alta frequência em uma imagem e pode ser usado para detecção de bordas.
- **Técnicas de Equalização de Histograma**: Foi desenvolvido o equalizador de histograma e a equalização de imagem utilizando o histograma. Estas técnicas são importantes para melhorar o contraste e a qualidade visual das imagens.
- **Técnicas de Limiarização**: Desenvolvimento da tecnica de limiarização simples e a multilimiarização. Estas técnicas são utilizadas para segmentar imagens em diferentes regiões com base em um ou mais valores de limiar.
- **Visualização de Histogramas**: Implementação das funções para calcular e apresentar histogramas de imagens em diferentes formatos (RGB, RGBA e Grayscale), permitindo uma análise detalhada das distribuições de intensidade de pixel nas imagens.

Em resumo, este trabalho proporciona uma base sólida e abrangente para o processamento de imagens digitais, cobrindo uma ampla gama de técnicas e filtros essenciais. A implementação destes algoritmos e técnicas em Python facilita a compreensão e o estudo das operações de processamento de imagens, contribuindo para o desenvolvimento de aplicações mais avançadas e sofisticadas nesta área.

7 REFERÊNCIAS

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 3rd. ed. [S.l.]: Prentice Hall, 2008.

SMITH, J. Digital image processing techniques. **Journal of Image Analysis**, v. 12, n. 3, p. 45–58, 2005.

SMITH, J.; JONES, M.; BROWN, L. **Fundamentals of Image Processing**. [S.l.]: Academic Press, 2005.