Universidade do Estado
de Santa Catarina
UDESC

Transformações sobre pixel

Experimento 1: Estudo sobre contraste

Aluno
Alex Halatiki Vicente
Prof. Responsável
Gilmário

4 de junho de 2023

Introdução

Neste trabalho iremos realizar estudos sobre filtro de imagens, aplicando operadores de filtro sobre elas para analisar os resultados obtidos.

Fundamentação

Serão abordados temas como convolução, operador de media, operador gaussiano, operador laplaciano e operador gradiente, além das implementações dos mesmos.

A convolução é uma operação fundamental no processamento de sinais e imagens. Essa técnica é geralmente aplicada utilizando uma máscara ou filtro, também conhecido como kernel. O kernel é uma matriz de números M x N, sendo M e N números ímpares, para que a matriz possua centro, que define como a convolução será realizada. A operação de convolução envolve a multiplicação ponto a ponto dos elementos do kernel com a vizinhança correspondente na imagem, seguida pela soma dos produtos resultantes. Esse processo é repetido para cada pixel na imagem, resultando em uma nova imagem convoluída. A convolução permite que o kernel seja aplicado a cada pixel da imagem, ponderando a contribuição dos pixels vizinhos de acordo com os valores definidos no kernel. A convolução é utilizada na aplicação de todos os operadores que serão explicados a seguir.

O operador de média, também conhecido como filtro de média, é uma técnica comumente usada no processamento de imagens para suavizar a imagem e reduzir o ruído. O operador de média é um tipo de filtro linear que substitui o valor de cada pixel na imagem pela média dos valores dos pixels vizinhos. O kernel, no caso do operador de media, possui todos os seus valores iguais.

O operador gaussiano, também conhecido como filtro gaussiano, é um tipo de filtro utilizado no processamento de imagens para suavizar a imagem, reduzir o ruído e realçar características em uma forma mais natural. Ao contrário do operador de média, o operador gaussiano aplica um kernel que segue uma distribuição gaussiana, também conhecida como curva gaussiana, em vez de utilizar pesos iguais para todos os pixels na vizinhança.

O operador Laplaciano é um operador diferencial usado no processamento de imagens para detectar mudanças abruptas de intensidade. Ele é aplicado para realçar bordas e características em uma imagem. O operador calcula a segunda derivada espacial da imagem e destaca regiões de intensa curvatura. É comumente usado na detecção de bordas e em outras aplicações de análise de imagens.

O operador gradiente é uma técnica usada no processamento de imagens para calcular a magnitude e a direção das mudanças de intensidade em uma imagem. Ele é frequentemente aplicado para detecção de bordas e realce de características. Ele é usado para identificar áreas onde ocorrem mudanças significativas de intensidade, como bordas e contornos. A magnitude do gradiente representa a intensidade dessas mudanças, enquanto a direção do gradiente indica a orientação da mudança.

Etapa Experimental

Questão 1

a) -

```python
import cv2
import numpy as np

def operadorMedia(caminho, nome, formato, linhasMedia, colunasMedia, valorMedia):
    image = cv2.imread(caminho, cv2.IMREAD_GRAYSCALE)

    border_heigth = int((linhasMedia - 1) / 2)
    border_width = int((colunasMedia - 1) / 2)

    image_with_border = cv2.copyMakeBorder(image, border_heigth, border_heigth, border_width, border_width, cv2.BORDER_CONSTANT, value=0)

    image = image.astype(np.float32)

    rows, cols = image_with_border.shape

    for row in range(border_heigth, rows - border_heigth):
        for col in range(border_width, cols - border_width):
            soma = 0
            for i in range(int(row - ((linhasMedia - 1) / 2)), int(row + ((linhasMedia - 1) / 2) + 1)):
                for j in range(int(col - ((colunasMedia - 1) / 2)), int(col + ((colunasMedia - 1) / 2) + 1)):
                    soma += (image_with_border[i][j] * valorMedia)
            image[row - border_heigth][col - border_width] = np.round(soma)

    image = image - image.min()
    image = image * 255 / image.max()
    image = np.uint8(image)

    cv2.imwrite(f'./ex01/imagensMedia/{nome}_media.{formato}', image)

operadorMedia('./imagens/Lua1_gray.jpg', 'Lua1_gray_3x3', 'jpg', 3, 3, 1/9)
operadorMedia('./imagens/Lua1_gray.jpg', 'Lua1_gray_5x5', 'jpg', 5, 5, 1/9)
```

Com o código acima feito em python, aplicamos bordas de 0 na matriz da imagem original para realizar o processo de convolução com o operador de media em um kernel 3x3 e 5x5.

b) -

```python
import cv2
import numpy as np
import math as m

def operadorGaussiano(caminho, nome, formato, linhasMatriz, colunasMatriz, matrizGaussiana):
    image = cv2.imread(caminho, cv2.IMREAD_GRAYSCALE)

    border_heigth = int((linhasMatriz - 1) / 2)
    border_width = int((colunasMatriz - 1) / 2)

    image_with_border = cv2.copyMakeBorder(image, border_heigth, border_heigth, border_width, border_width, cv2.BORDER_CONSTANT, value=0)

    rows, cols = image_with_border.shape

    for row in range(border_heigth, rows - border_heigth):
        for col in range(border_width, cols - border_width):
            soma = 0
            linhaGaussiana = 0
            for i in range(int(row - ((linhasMatriz - 1) / 2)), int(row + ((linhasMatriz - 1) / 2) + 1)):
                colunaGaussiana = 0
                for j in range(int(col - ((colunasMatriz - 1) / 2)), int(col + ((colunasMatriz - 1) / 2) + 1)):
                    soma += (image_with_border[i][j] * matrizGaussiana[linhaGaussiana][colunaGaussiana])
                    colunaGaussiana += 1
                linhaGaussiana += 1
            if(soma > 255):
                print('teste')
            image[row - border_heigth][col - border_width] = np.round(soma)

    cv2.imwrite(f'./ex01/imagensGauss/{nome}_gauss.{formato}', image)


def matrizGaussiana(X, Y, o):
    matrizGaussiana = []

    for i in range(0, 3):
        aux = []
        for j in range(0, 3):
            aux.append(1 / (2*m.pi*m.pow(o, 2)) * m.exp(-(m.pow(X[i][j], 2) + m.pow(Y[i][j], 2)) / (2*m.pow(o, 2))))
        matrizGaussiana.append(aux)

    return matrizGaussiana


X = [
    [-1, 0, 1],
    [-1, 0, 1],
    [-1, 0, 1]
]

Y = [
    [-1, -1, -1],
    [0, 0, 0],
    [1, 1, 1]
]

operadorGaussiano('./imagens/Lua1_gray.jpg', 'Lua1_gray3x3_0.6', 'jpg', 3, 3, matrizGaussiana(X, Y, 0.6))
operadorGaussiano('./imagens/Lua1_gray.jpg', 'Lua1_gray3x3_1.0', 'jpg', 3, 3, matrizGaussiana(X, Y, 1.0))
```

Com o código acima feito em python, primeiramente calcula-se os valores do kernel 3x3 gaussiano a partir das matrizes X e Y, e do valor de σ que pode variar. Assim, adicionamos bordas de 0 na matriz da imagem original para realizar o processo de convolução com o operador gaussiano a partir do kernel gerado.

Questão 2

```python
import cv2
import numpy as np

def operadorLaplaciano(caminho, nome, formato, linhasMatriz, colunasMatriz, matrizLaplaciana):
    image = cv2.imread(caminho, cv2.IMREAD_GRAYSCALE)

    border_heigth = int((linhasMatriz - 1) / 2)
    border_width = int((colunasMatriz - 1) / 2)

    image_with_border = cv2.copyMakeBorder(image, border_heigth, border_heigth, border_width, border_width, cv2.BORDER_CONSTANT, value=0)

    image_float = image.astype(np.float32)

    rows, cols = image_with_border.shape

    for row in range(border_heigth, rows - border_heigth):
        for col in range(border_width, cols - border_width):
            soma = 0
            linhaLaplaciana = 0
            for i in range(int(row - ((linhasMatriz - 1) / 2)), int(row + ((linhasMatriz - 1) / 2) + 1)):
                colunaLaplaciana = 0
                for j in range(int(col - ((colunasMatriz - 1) / 2)), int(col + ((colunasMatriz - 1) / 2) + 1)):
                    soma += (image_with_border[i][j] * matrizLaplaciana[linhaLaplaciana][colunaLaplaciana])
                    colunaLaplaciana += 1
                linhaLaplaciana += 1
            image_float[row - border_heigth][col - border_width] += (-soma)

    image_float = image_float - image_float.min()
    image_float = image_float * 255 / image_float.max()
    image_float += image
    image_float = image_float - image_float.min()
    image_float = image_float * 255 / image_float.max()
    image_float = np.uint8(image_float)

    cv2.imwrite(f'./ex02/imagensLaplace/{nome}_laplace.{formato}', image_float)


A = [
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]
]

B = [
    [1, 1, 1],
    [1, -8, 1],
    [1, 1, 1]
]

operadorLaplaciano('./imagens/11_test.png', '11_test_A', 'png', 3, 3, A)
operadorLaplaciano('./imagens/11_test.png', '11_test_B', 'png', 3, 3, B)
```

Com o código acima feito em python, adicionamos bordas de 0 na matriz da imagem original, para assim, aplicarmos o processo de convolução com o operador laplaciano com o kernel 3x3 A, e com o kernel 3x3 B.

Questão 3

Primeiramente definimos as matrizes necessárias para a questão como mostra a imagem abaixo:

```
prewittX = [
    [-1, 0, 1],
    [-1, 0, 1],
    [-1, 0, 1]
]

prewittY = [
    [-1, -1, -1],
    [0, 0, 0],
    [1, 1, 1]
]

sobelX = [
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
]

sobelY = [
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]
]

scharrX = [
    [-3, 0, 3],
    [-10, 0, 10],
    [-3, 0, 3]
]

scharrY = [
    [-3, -10, -3],
    [0, 0, 0],
    [3, 10, 3]
]

operadorGradiente('./imagens/Lua1_gray.jpg', 'Lua1_gray_Prewitt', 'jpg', 3, 3, prewittX, prewittY)
operadorGradiente('./imagens/Lua1_gray.jpg', 'Lua1_gray_Sobel', 'jpg', 3, 3, sobelX, sobelY)
operadorGradiente('./imagens/Lua1_gray.jpg', 'Lua1_gray_Scharr', 'jpg', 3, 3, scharrX, scharrY)
```

Assim, podemos aplicar o processo explicado na descrição da tarefa, realizando a convolução com o operador gradiente, implementado no código abaixo. Vale ressaltar que foi utilizado o operador de media descrito na questão 1 com a finalidade de suavizar os ruídos da imagem.

```python
def escolhaVizinhos(i, j, valor):
    if(22.5 < valor <= 67.5):
        return i-1, j+1, i+1, j-1
    elif(67.5 < valor <= 112.5):
        return i-1, j, i+1, j
    elif(112.5 < valor <= 157.5):
        return i-1, j-1, i+1, j+1
    elif(157.5 < valor <= 180):
        return i, j-1, i, j+1
    elif(-22.5 < valor <= 22.5):
        return i, j+1, i, j-1
    elif(-67.5 < valor <= -22.5):
        return i+1, j+1, i-1, j-1
    elif(-112.5 < valor <= -67.5):
        return i+1, j, i-1, j
    elif(-157.5 < valor <= -112.5):
        return i+1, j-1, i-1, j+1
    elif(-180 < valor <= -157.5):
        return i, j-1, i, j+1

def operadorGradiente(caminho, nome, formato, linhasMatriz, colunasMatriz, X, Y):
    image = cv2.imread(caminho, cv2.IMREAD_GRAYSCALE)

    image = operadorMedia(image, 3, 3, 1/9)

    border_heigth = int((linhasMatriz - 1) / 2)
    border_width = int((colunasMatriz - 1) / 2)

    image_with_border = cv2.copyMakeBorder(image, border_heigth, border_heigth, border_width, border_width, cv2.BORDER_CONSTANT, value=0)
    image = image.astype(np.float32)

    image_float = image_with_border.astype(np.float32)
    direction = []

    rows, cols = image_with_border.shape

    for row in range(border_heigth, rows - border_heigth):
        aux = []
        for col in range(border_width, cols - border_width):
            somaX = 0
            somaY = 0
            linhaAux = 0
            for i in range(int(row - ((linhasMatriz - 1) / 2)), int(row + ((linhasMatriz - 1) / 2) + 1)):
                colunaAux = 0
                for j in range(int(col - ((colunasMatriz - 1) / 2)), int(col + ((colunasMatriz - 1) / 2) + 1)):
                    somaX += (image_with_border[i][j] * X[linhaAux][colunaAux])
                    somaY += (image_with_border[i][j] * Y[linhaAux][colunaAux])
                    colunaAux += 1
                linhaAux += 1
            image_float[row][col] = (m.sqrt(m.pow(somaX, 2) + m.pow(somaY, 2)))
            aux.append(m.degrees(m.atan2(somaY, (somaX + m.pow(10, -8)))))
        direction.append(aux)

    for row in range(border_heigth, rows - border_heigth):
        for col in range(border_width, cols - border_width):
            i1, j1, i2, j2 = escolhaVizinhos(row, col, direction[row - border_heigth][col - border_width])
            if(image_float[row][col] > image_float[i1][j1] and image_float[row][col] > image_float[i2][j2]):
                image[row - border_heigth][col - border_width] = image_float[row][col]

    image = image - image.min()
    image = image * 255 / image.max()
    image = np.uint8(image)

    cv2.imwrite(f'./ex03/imagensGradiente/{nome}_gradiente.{formato}', image)
```

Resultados

Questão 1

a) -

Lua1_gray.jpg após aplicação do operador de media 3x3:

Lua1_gray.jpg após a aplicação do operador de media 5x5:

Percebe-se que quanto maior o kernel de media aplicado, mais suavizados os detalhes da imagem ficam, o que nos leva a perda de detalhes finos e bordas nas imagens, devido a natureza de suavização do filtro.

b) -

Lua1_gray.jpg após aplicação do operador gaussiano 3x3 com σ = 0.6:

Lua1_gray.jpg após aplicação do operador gaussiano 3x3 com σ = 1.0:

Percebe-se que quanto maior o valor de σ, mais suaves os detalhes da imagem ficam. Além disso, nota-se que o operador gaussiano é capaz de suavizar imagens de forma mais natural, preservando melhor os detalhes e as bordas em comparação ao operador de media.

Questão 2

11_test.png após aplicação do operador laplaciano com a matriz A:

11_test.png após aplicação do operador laplaciano com a matriz B:

Percebe-se que ambas as matrizes geraram resultados bons e bem semelhantes. Assim, nota-se uma melhora significante na nitidez da imagem, em que ouve um realce nas bordas e nos detalhes originais.

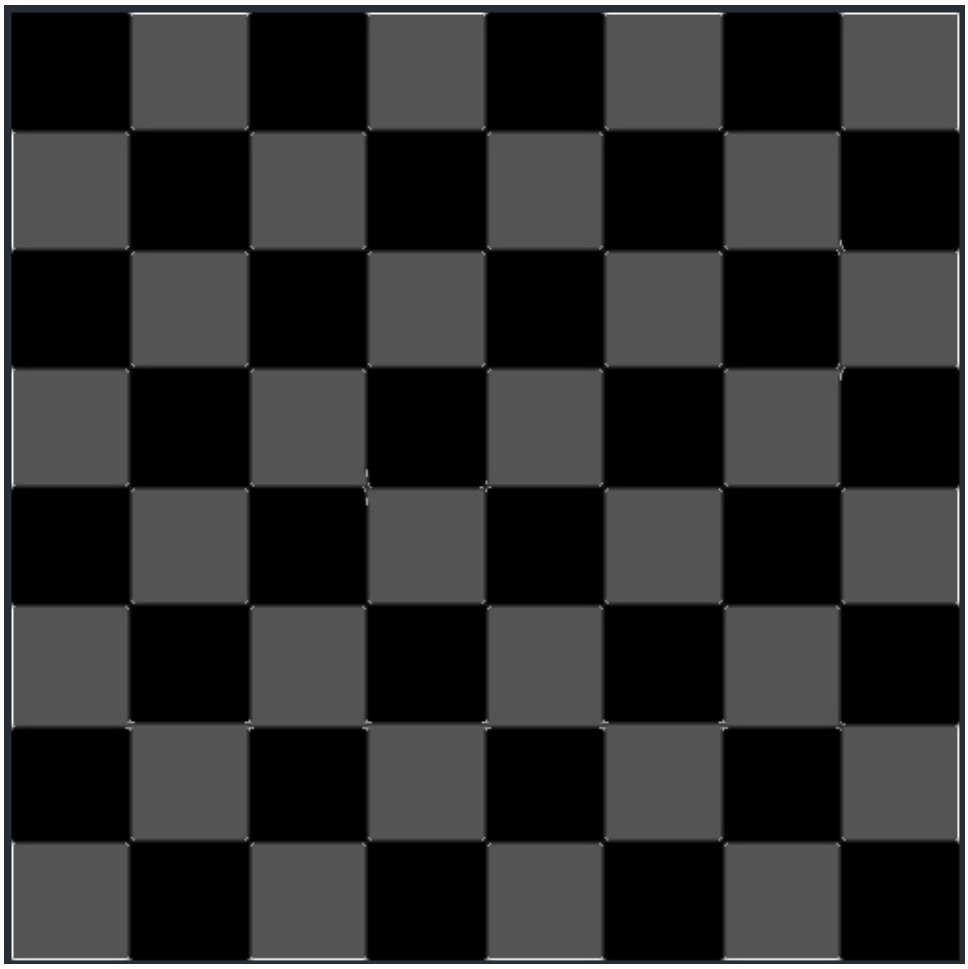
Questão 3

Lua1_gray.jpg apos a aplicação do gradiente de Prewitt:

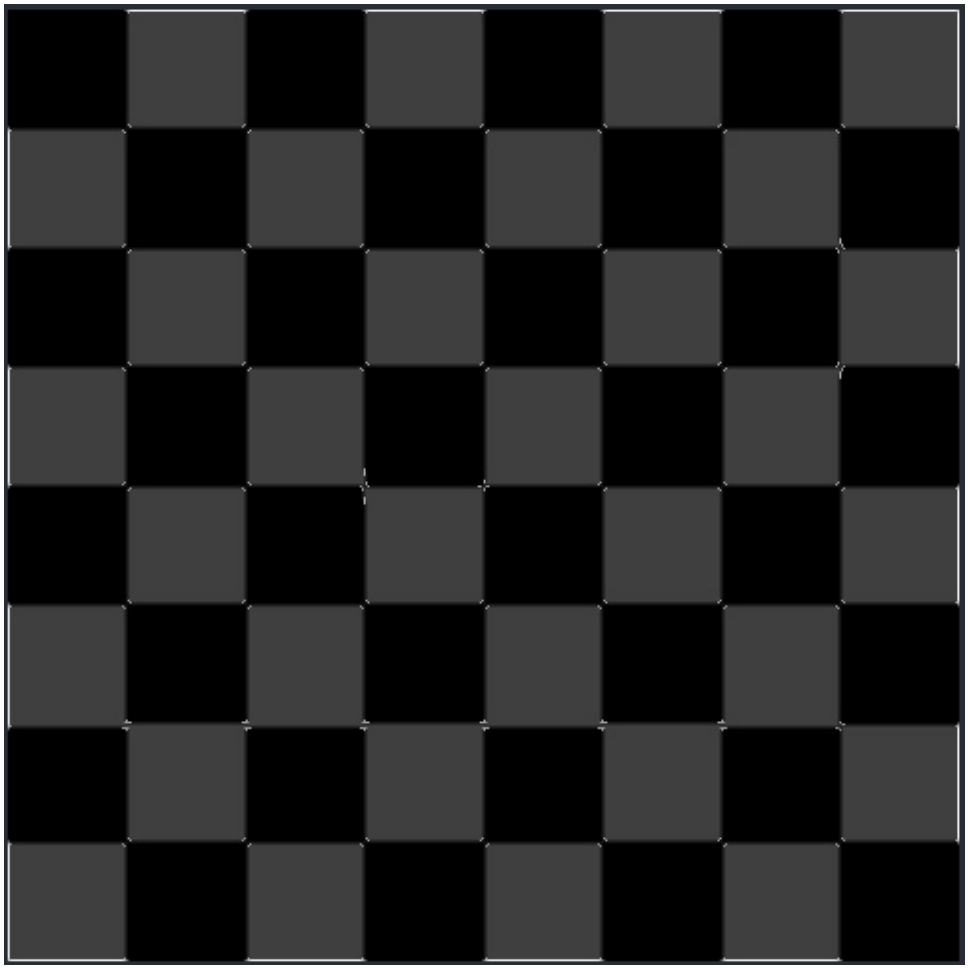
Lua1_gray.jpg apos a aplicação do gradiente de Sobel:

Lua1_gray.jpg apos a aplicação do gradiente de Scharr:

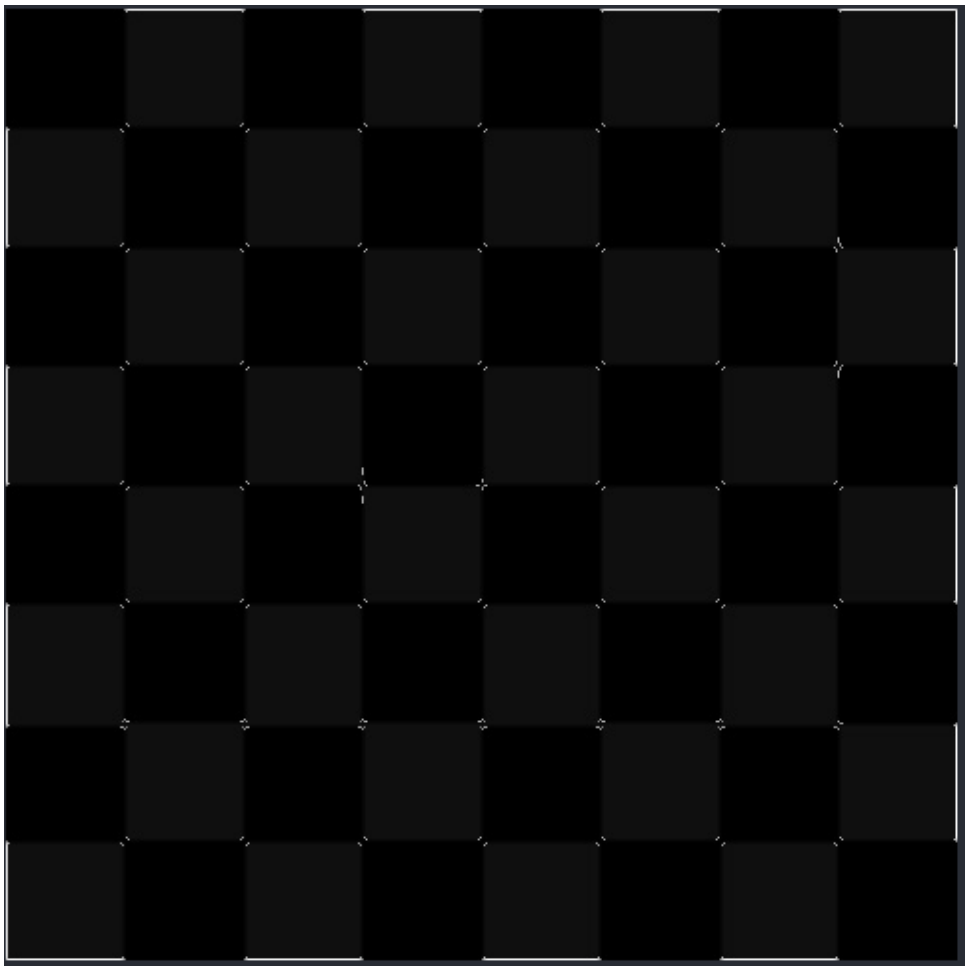
chessboard_inv.png apos a aplicação do gradiente de Prewitt:



chessboard_inv.png apos a aplicação do gradiente de Sobel:



chessboard_inv.png apos a aplicação do gradiente de Scharr:



Percebe-se que a aplicação do operador gradiente na imagem chessboard_inv.png resultou em um resultado superior em comparação com a aplicação na imagem Lua1_gray.jpg. Isso ocorre devido ao fato de que a imagem Lua1_gray.jpg apresenta uma quantidade significativa de ruído que não foi devidamente suavizada antes da aplicação do operador. Utilizar um filtro de mediana em vez de um filtro de média para a remoção do ruído poderia resultar em um resultado melhor para ambas as imagens.

Conclusões

Em conclusão, neste trabalho abordamos quatro importantes operadores utilizados no processamento de imagens: o filtro de média, o operador gaussiano, o operador Laplaciano e o operador gradiente.

O filtro de média é uma técnica básica para suavização de imagens, redução de ruído e realce de características. O operador gaussiano vai além, proporcionando uma suavização mais natural ao aplicar um kernel gaussiano ponderado. O operador Laplaciano destaca mudanças abruptas de intensidade, sendo especialmente útil na detecção de bordas e realce de características importantes.

O operador gradiente, por sua vez, permite calcular a magnitude e a direção das mudanças de intensidade em uma imagem. Ele é usado para identificar áreas de variação intensa, como bordas e contornos, fornecendo informações valiosas sobre a estrutura e os padrões presentes na imagem.

A combinação desses operadores oferece uma gama ampla de recursos para processamento de imagens. O filtro de média e o operador gaussiano são eficazes para suavização e redução de ruído, enquanto o operador Laplaciano destaca bordas e características salientes. O operador gradiente complementa essas técnicas ao fornecer informações sobre a intensidade e a direção das mudanças na imagem.

Ao compreender e utilizar de maneira adequada esses operadores, é possível aprimorar a qualidade visual das imagens, remover ruídos indesejados, destacar detalhes relevantes e extrair informações valiosas para uma variedade de aplicações, como visão computacional, análise de imagens médicas, reconhecimento de padrões e muito mais.