UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CURSO DE CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

HUMBERTO NUNES DE LIRA
JÚLIA MORAES DA SILVA
LUIZ EDUARDO DE ALMEIDA SIQUEIRA SILVA
PEDRO LUCAS SIMÕES CABRAL
PEDRO MIGUEL CECATO VALOES

RELATÓRIO

JOÃO PESSOA
2024

LISTA DE IMAGENS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E APRESENTAÇÃO DO TEMA

O processamento digital de imagens (PDI) desempenha um papel fundamental em diversas áreas da ciência e da tecnologia,e envolve a manipulação e análise de imagens digitais. Através de operações matemáticas sobre os pixels de uma imagem, é possível realçar informações, remover ruídos e extrair informações relevantes.

Este projeto buscou desenvolver um sistema em Python para abrir, exibir, manipular e salvar imagens RGB de 24 bits/pixel, implementando operações clássicas de correlação bidimensional e técnicas de equalização e expansão de histograma.

1.2 FUNDAMENTAÇÃO TEÓRICA

Uma imagem digital em cores pode ser representada no modelo RGB, no qual cada pixel é formado por três componentes (vermelho, verde e azul), cada uma codificada em 8 bits, com valores variando de 0 a 255.

A correlação bidimensional consiste em aplicar uma máscara sobre uma vizinhança m×n de cada pixel da imagem. A máscara desliza sobre a imagem de entrada e, em cada ponto (i,j), calcula-se o produto interno de Frobenius entre h e v(i,j), sendo v para representar a vizinhança, atribuindo-se o resultado ao ponto nas mesmas coordenadas na imagem resultante. Entre os filtros utilizados neste projeto temos:

- Gaussiano: envolve a correlação da imagem original com uma matriz de pesos, conhecida como máscara ou kernel Gaussiano. Usado para suavizar imagens e reduzir ruídos.
- Filtro Box: realiza uma média entre os elementos da vizinhança da imagem.Usado para suavização.
- Filtros Sobel (horizontal e vertical): funciona aproximando a derivada da imagem nas direções horizontal e vertical. Usado para detecção de bordas.

A função de ativação identidade é uma função simples utilizada em redes neurais, que não altera o valor de entrada de forma alguma. Ela retorna o mesmo valor da entrada, ou seja, f(x) = x.

A função de ativação ReLU (Rectified Linear Unit) também é muito usada em redes neurais porque é simples e ajuda a evitar problemas que funções como o sigmoide e a tangente hiperbólica apresentam, como o desvanecimento do gradiente (gradiente muito pequeno), permitindo que a rede aprenda de forma mais eficiente em camadas profundas. Sua definição matemática é que ReLU(x) = max (0,x) , ou seja, ,se x > 0, então ReLU(x) = x , e se x ≤ 0 temos ReLU(x) = 0.

O histograma de uma imagem é a representação gráfica de n_l x l, sendo n_l o número de vezes que o nível de cinza ocorre em uma imagem com n pixels. A expansão de histograma é um método de normalização de imagens, que pode produzir uma imagem visualmente mais rica. Consiste em redistribuir os níveis de cinza de forma que o nível mínimo fique igual a 0 e máximo igual a L-1, sendo L o número total de possíveis tons de cinza na imagem. Quando a imagem tem pixels de valor 0 e L-1 (ou próximos a esses extremos) a expansão de histograma é ineficaz, daí podemos utilizar da equalização de histograma que consiste em gerar imagem com distribuição de níveis de cinza uniforme.

1.3 OBJETIVOS

- Desenvolver um sistema para abrir, exibir, manipular e salvar imagens.
- Implementar correlação bidimensional com filtros Gaussiano 5x5, Box 1x10, Box 10x1, Box 10x10, Sobel horizontal e Sobel vertical.
- Realizar equalização e expansão de histograma local.
- Analisar os resultados após aplicar os filtros nas imagens.

2 MATERIAIS E MÉTODOS

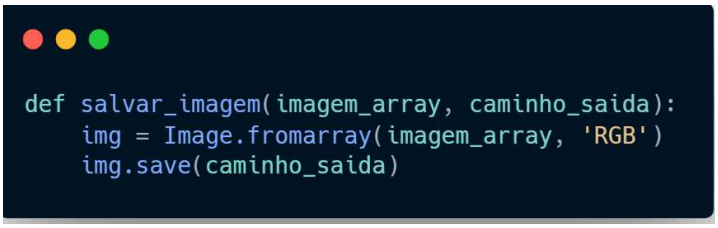
2.1 DESCRIÇÃO DAS ATIVIDADES DESENVOLVIDAS

```python
def extrair_pixels_rgb(caminho_imagem):
    img = Image.open(caminho_imagem)
    img = img.convert('RGB')
    pixels = np.array(img, dtype=np.uint8)
    return pixels
```

Figura 1 - Função extrair pixels

```python
def exibir_imagem(imagem, titulo="Imagem"):
    plt.figure(figsize=(10, 8))
    plt.imshow(imagem)
    plt.title(titulo)
    plt.axis('off')
    plt.tight_layout()
    plt.show()
```

Figura 2 - Função exibir imagem



```python
def salvar_imagem(imagem_array, caminho_saida):
    img = Image.fromarray(imagem_array, 'RGB')
    img.save(caminho_saida)
```

Figura 3 - Função para salvar imagem

Para manipulação básica das imagens foram criadas no início do arquivo as funções de extrair os pixels da imagem, exibir a imagem e salvar a imagem.

```python
def ler_filtro(arquivo):
    with open(arquivo, 'r') as f:
        linhas = f.readlines()
    mascara = []
    bias = 0
    ativacao = 'identidade'
    for linha in linhas:
        if linha.startswith('mascara:'):
            continue
        elif linha.startswith('bias:'):
            bias = int(linha.split(':')[1].strip())
        elif linha.startswith('ativacao:'):
            ativacao = linha.split(':')[1].strip().lower()
        elif linha.strip():
            mascara.append([float(x) for x in linha.strip().split()])
    print(mascara)
    return np.array(mascara), bias, ativacao

def ativacao_relu(x):
    return np.maximum(0, x)

def ativacao_identidade(x):
    return x
```

Figura 4 - Função para ler filtros

A função para ler os filtros recebe como parâmetro uma arquivo.txt com a estrutura semelhante a da Figura 5.

```
mascara:
0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
bias: 10
ativacao: identidade
```

Figura 5 - Máscara 1x10

Na primeira linha do arquivo.txt temos escrito “máscara”, indicando que a partir dali serão apresentados os valores que compõem a máscara usada na operação de correlação. Na Figura 5 temos a estrutura de uma máscara box 1x10.

Após o fim da máscara, temos escrito “bias” no início da linha, seguido por dois pontos e o valor correspondente, que neste caso é 10. Esse valor será somado ao resultado da correlação para ajustar a saída.

Em seguida, temos escrito “ativação”, também seguido por dois pontos e pelo nome da função de ativação a ser utilizada. No exemplo, foi escolhida a função

identidade, que significa que o valor de saída será exatamente o valor obtido após a correlação mais o bias, sem nenhuma transformação adicional.

Na implementação, a função **`ler_filtro`** abre o arquivo para leitura e percorre todas as suas linhas. Inicialmente, são definidas variáveis padrão: uma lista vazia para armazenar a máscara, o bias igual a 0, e a função de ativação configurada como identidade. Durante a leitura, o laço `for` identifica cada seção do arquivo e atualiza as variáveis de acordo com as informações encontradas. Ao final, a função retorna a máscara, o bias e a ativação lidos do arquivo.

Por fim, também estão definidas as funções de ativação identidade e ReLU.

```python
def correlacao_manual(imagem, mascara, bias, ativacao, clip=False):
    linha, coluna , canais = imagem.shape
    m, n = mascara.shape
    resultado = np.zeros_like(imagem, dtype=np.int32)

    for canal in range(canais):
        for i in range(linha - m + 1):
            for j in range(coluna - n + 1):
                regiao = imagem[i:i+m, j:j+n, canal]
                valor = np.sum(regiao * mascara) + bias
                if ativacao == 'relu':
                    resultado[i, j, canal] = ativacao_relu(valor)
                else:
                    resultado[i, j, canal] = ativacao_identidade(valor)
```

Figura 6 - Função de correlação manual

A função **`correlacao_manual`** recebe como parâmetros uma imagem, a máscara, o bias, a função de ativação e o parâmetro opcional clip, que pode ser definido como verdadeiro ou falso.

Primeiramente, são extraídas as dimensões da imagem (linhas, colunas e canais) e da máscara (m e n). Em seguida, é inicializada a matriz resultado, com o mesmo tamanho da imagem de entrada, para armazenar os valores calculados.

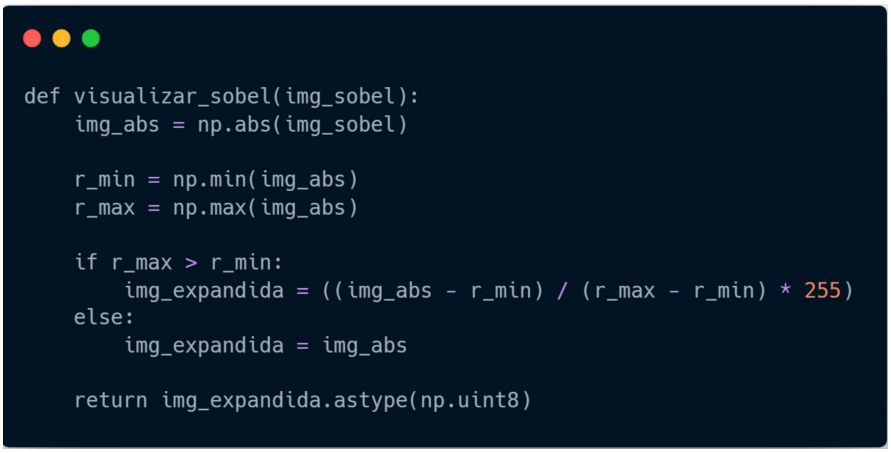
A operação principal é realizada em três laços `for`:

1. O laço externo percorre os canais da imagem (vermelho, verde e azul, no caso de uma imagem RGB).

2. O segundo laço percorre as linhas da imagem até o limite em que a máscara ainda pode ser aplicada.
3. O terceiro laço percorre as colunas da mesma forma, garantindo que a máscara se ajuste corretamente à região da imagem.

Dentro desses laços, é extraída uma região da imagem correspondente ao tamanho da máscara. Em seguida, calcula-se o valor resultante do produto interno entre a região da imagem e a máscara, somado ao bias.Esse valor é então passado para a função de ativação escolhida.

Por fim, caso o parâmetro clip seja definido como verdadeiro, todos os valores da matriz resultante são limitados ao intervalo [0, 255], garantindo compatibilidade com a representação padrão de imagens de 8 bits por canal. A função então retorna a imagem resultante da operação de correlação.



```
def visualizar_sobel(img_sobel):
    img_abs = np.abs(img_sobel)

    r_min = np.min(img_abs)
    r_max = np.max(img_abs)

    if r_max > r_min:
        img_expandida = ((img_abs - r_min) / (r_max - r_min) * 255)
    else:
        img_expandida = img_abs

    return img_expandida.astype(np.uint8)
```

Figura 7 - Função de visualizar sobel

A função **`visualizar_sobel`** recebe como parâmetro uma imagem que já tenha passado por um filtro, como o operador de Sobel.

Primeiro, são calculados os valores absolutos da imagem, garantindo que todos os pixels sejam não negativos (já que o Sobel pode gerar valores positivos e negativos).

Em seguida, são obtidos o valor mínimo e o valor máximo da matriz resultante. Com base nesses valores, aplica-se uma expansão de histograma, que

normaliza os dados para o intervalo [0, 255], de acordo com a fórmula apresentada em sala de aula.

Esse procedimento garante que os contrastes da imagem fiquem mais visíveis. Caso todos os valores da imagem sejam iguais, isto é valor máximo igual ao valor mínimo, a expansão não pode ser aplicada; nesse caso, a imagem é retornada sem modificação.

Por fim, a matriz resultante é convertida para o tipo **uint8** (8 bits sem sinal), compatível com a representação padrão de imagens, e retornada para visualização.

```python
def equalizacao_local(imagem, m=50, n=50):
    L = 256
    linhas, colunas, canais = imagem.shape
    saida = np.zeros_like(imagem, dtype=np.uint8)

    half_m = m // 2
    half_n = n // 2

    for canal in range(canais):
        for i in range(linhas):
            for j in range(colunas):
                i_ini = max(0, i - half_m)
                i_fim = min(linhas, i + half_m + 1)
                j_ini = max(0, j - half_n)
                j_fim = min(colunas, j + half_n + 1)

                janela = imagem[i_ini:i_fim, j_ini:j_fim, canal]

                hist = np.bincount(janela.flatten(), minlength=L)

                cdf = np.cumsum(hist)
                cdf_norm = np.round((L - 1) * cdf / cdf[-1]).astype(np.uint8)

                saida[i, j, canal] = cdf_norm[imagem[i, j, canal]]

    return saida
```

Figura 8 - Função equalização local

A função **`equalizacao_local`** recebe como parâmetros os valores m e n, que representam as dimensões da janela utilizada no processo de equalização local. Define-se também L = 256, correspondente ao número total de níveis de intensidade possíveis em imagens de 8 bits por canal.

Em seguida, são obtidas as dimensões da imagem (linhas, colunas e canais), e a matriz saída é inicializada com zeros, mantendo o mesmo tamanho da imagem original. Para cada pixel, a função calcula a vizinhança correspondente à janela m × n centrada nele, ajustando os limites da janela quando o pixel está próximo das bordas da imagem.

A partir dessa vizinhança, é calculado o histograma local, seguido pela CDF (Cumulative Distribution Function). A CDF é normalizada de modo que seus valores variem no intervalo [0, L−1]. Essa transformação é então aplicada apenas ao pixel central da janela, substituindo sua intensidade pelo valor correspondente da CDF normalizada.

Esse processo é repetido para todos os pixels e canais da imagem, resultando em uma nova imagem onde cada pixel foi equalizado de acordo com sua vizinhança local. O efeito obtido é um realce adaptativo de contraste, permitindo destacar detalhes que não seriam evidenciados pela equalização global.

Ao final, a função retorna a imagem equalizada localmente.

2.2 DESCRIÇÃO DAS FERRAMENTAS UTILIZADAS

Neste projeto foi usado a linguagem de programação Python 3.13.2, por uma questão de afinidade da equipe e por possuir bibliotecas otimizadas para trabalhar com processamento digital de imagens. Do Python foram usadas as bibliotecas *numpy*, amplamente utilizada para computação numérica, a biblioteca *PIL (Python Imaging Library/ Pillow),* usada para processamento de imagens e a biblioteca *Matplotib* usada para visualização de dados.

O projeto foi desenvolvido no ambiente do Jupyter Notebook devido à sua capacidade de integrar código, visualizações e documentação em um único fluxo.

As imagens selecionadas para os testes foram escolhidas considerando suas características visuais e como elas se comportariam à demonstração dos efeitos de cada filtro aplicado.

Figura 9 - Cidade a noite

Figura 10 - Beco a noite

Figura 11 - Lena

Figura 12 - Formas

2.3 DESCRIÇÃO DOS CONHECIMENTOS UTILIZADOS

O desenvolvimento deste projeto envolveu a aplicação de conhecimentos de Processamento Digital de Imagens, Álgebra Linear e Estatística Foram utilizados os conceitos de convolução bidimensional, permitindo a aplicação de filtros sobre imagens para suavização, detecção de bordas e realce de detalhes.

A implementação do pós-processamento envolveu a utilização da função de ativação, tanto a ReLU, que garante a eliminação de valores negativos, como a Identidade que não altera em nada Foram usadas também as técnicas de equalização e expansão de histograma local, com o objetivo de melhorar o contraste das imagens.

Além disso, foi aplicada a representação das imagens como arrays numéricos, possibilitando operações matemáticas vetorizadas com eficiência. Para análise e interpretação dos resultados, foram empregados conceitos de visualização de dados, facilitando a compreensão dos efeitos de cada filtro e transformação aplicada.

3 RESULTADOS

Figura 13 - Formas com box 10x10

O filtro do tipo “box” 10×10 proporcionou um efeito de borramento na imagem. Na Figura 14, observa-se a aplicação do filtro gaussiano 5×5, que produz um borramento mais suave e uniforme.

Figura 14 - Formas com gaussiana 5x5

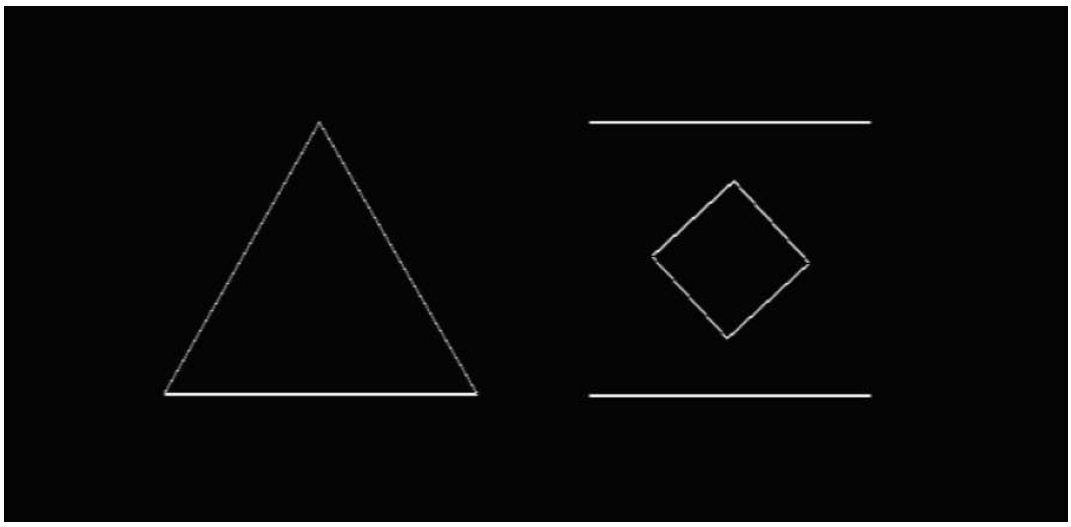


Figura 15 - Formas com sobel horizontal

Figura 16 - Formas com sobel vertical

Nas Figuras 15 e 16 vemos o efeito do sobel na detecção de bordas tanto na vertical quanto na horizontal.

Figura 17 - Beco a noite equalizado

Vemos na Figura 17 o efeito de equalização, ao aumentar o contraste da imagem.

4 DISCUSSÃO

4.1 PROBLEMAS E DIFICULDADES ENCONTRADAS

Uma das principais dificuldades encontradas durante o desenvolvimento do projeto foi implementar a lógica necessária para que a máscara percorresse apenas as regiões válidas da imagem, não utilizando a expansão por zero nas bordas durante a execução da correlação manual.

Além disso, tivemos dificuldade na implementação da equalização local, pois as imagens geradas apresentavam um aspecto de “blocos” visíveis. Descobrimos que isso ocorria porque a equalização estava sendo aplicada de forma independente em cada bloco, sem suavização entre as regiões vizinhas.

Figura 18 - Beco a noite equalizado

4.2 COMENTÁRIOS CRÍTICOS

O filtro gaussiano apresentou melhor desempenho na redução de ruídos, preservando de forma mais eficiente as características da imagem original em comparação ao filtro do tipo “box”. Observou-se que quanto maior a região do filtro box, mais intenso é o efeito de borramento na imagem.

Já os filtros de Sobel, tanto na orientação horizontal quanto na vertical, destacaram-se na detecção de bordas, sendo ferramentas essenciais para a extração de contornos e detalhes estruturais da imagem.

Ao realizar a equalização foi possível visualizar melhor os detalhes da imagem devido ao contraste.

5 CONCLUSÃO

O desenvolvimento deste projeto permitiu aplicar na prática conceitos fundamentais de Processamento Digital de Imagens (PDI), integrando teoria e implementação em Python. As funções desenvolvidas possibilitaram abrir, exibir, manipular e salvar imagens RGB de 24 bits/pixel, além de aplicar operações

clássicas de correlação bidimensional e técnicas de equalização e expansão de histograma.

Os resultados obtidos demonstraram que diferentes filtros possuem comportamentos específicos conforme o objetivo da aplicação.

No pós-processamento, foi possível observar que a equalização e expansão de histograma melhoraram significativamente o contraste, realçando detalhes.

Apesar das dificuldades encontradas, como a implementação da correlação manual sem expansão por zero e a perda de contraste quando a expansão de histograma não é aplicada, a experiência foi essencial para consolidar o entendimento sobre a manipulação de imagens em baixo nível, sem depender exclusivamente de bibliotecas de alto nível.

Conclui-se, portanto, que o sistema desenvolvido atendeu aos objetivos propostos, permitindo explorar de forma prática os principais conceitos de PDI e evidenciando a importância da escolha adequada de filtros e técnicas de realce conforme a aplicação desejada.

REFERÊNCIAS

GONZALEZ, Rafael C.; WOODS, Richard E. Digital Image Processing. 4. ed. Global ed. Harlow: Pearson Education, 2018. ISBN 978-1-292-22304-9.

MENIMATO. Filtragem Gaussiana – Entenda a Técnica. Disponível em: https://menimato.github.io/IMGedu.jl/filtragem.html. Acesso em: 22 ago. 2025.

VIDAL, Leonardo. *Processamento Digital de Imagens – PDI 2025.1*. João Pessoa, 2025. Disponível em: https://sig-arq.ufpb.br/arquivos/2025202061fe477682574d627a53c57c7/PDI2025.1.pdf. Acesso em: 22 ago. 2025.

DEEPSEEK. DeepSeek Chat. Versão 2025, DeepSeek, 2025. Disponível em: https://www.deepseek.com/. Acesso em: 22 ago. 2025.

GITHUB. *GitHub Copilot* [software]. Disponível em: https://github.com/features/copilot. Acesso em: 22 ago. 2025.