

UNIVERSIDADE ESTADUAL DE CAMPINAS



MC920 - INTRODUÇÃO AO PROCESSAMENTO DE IMAGEM DIGITAL

PROF. HÉLIO PEDRINI

Trabalho 1

Ranter Soares DO CARMO R.A.: 186289

8 de abril de 2025

Conteúdo

1	Introdução	3
2	Materiais e métodos	3
2.1	Requerimentos	3
2.2	Organização dos arquivos	3
3	Códigos	4
3.1	ajusteDeBrilho.py	4
3.1.1	Passo a passo	4
3.1.2	Exemplo	5
3.2	alteracaoDeCores.py	5
3.2.1	Passo a passo	6
3.2.2	Exemplo	6
3.3	combinacaoDeImagens.py	6
3.3.1	Passo a passo	7
3.3.2	Exemplo	7
3.4	esbocoALapis.py	8
3.4.1	Passo a passo	8
3.4.2	Interface de Linha de Comando	8
3.4.3	Exemplo	9
3.5	filtragemDeImagens.py	9
3.5.1	Princípio Matemático	9
3.5.2	Catálogo de Filtros Implementados	9
3.5.3	Fluxo de Processamento	10
3.5.4	Interface de Linha de Comando	10
3.5.5	Exemplo	10
3.5.6	Características Técnicas	11
3.6	mosaico.py	12
3.6.1	Princípio de Funcionamento	12
3.6.2	Passo a Passo	12
3.6.3	Interface de Linha de Comando	13
3.6.4	Exemplo	13
3.7	planoDeBits.py	13
3.7.1	Princípio Matemático	13
3.7.2	Passo a Passo	14
3.7.3	Interface de Linha de Comando	14
3.7.4	Exemplo	14
3.8	quantizacaoDeImagens.py	15
3.8.1	Princípio Matemático	15
3.8.2	Passo a Passo	15
3.8.3	Interface de Linha de Comando	16
3.8.4	Aplicações	16
3.8.5	Exemplo	17
3.9	transformacaoDeImagensColoridas.py	18

3.9.1	Transformação Sépia	18
3.9.2	Transformação Monocromática	18
3.9.3	Passo a Passo	18
3.9.4	Interface de Linha de Comando	18
3.9.5	Exemplo	19
3.10	transformacaoDeIntensidade.py	19
3.10.1	Transformações Implementadas	19
3.10.2	Passo a Passo	19
3.10.3	Interface de Linha de Comando	20
3.10.4	Exemplo	21
4	Conclusões	23

1 Introdução

Este relatório descreve a implementação de dez técnicas de processamento digital de imagens, conforme especificado no Trabalho 1 da disciplina. As operações incluem filtragem espacial, transformações de intensidade, quantização, combinação de imagens e manipulação de cores, utilizando Python e bibliotecas como OpenCV, NumPy e Matplotlib.

O objetivo principal foi aplicar algoritmos clássicos de processamento de imagens, garantindo eficiência através de vetorização de operações e tratamento adequado de bordas. Os resultados foram validados visualmente em comparação com os exemplos fornecidos e com o que se era esperado do processamento.

2 Materiais e métodos

2.1 Requerimentos

Para a implementação dos algoritmos foi utilizada a linguagem Python (versão 3.9.13) e os seguintes pacotes.

Pacote	Versão
OpenCV	4.11.0
NumPy	2.2.4
Matplotlib	3.10.1
Argparse	3.2

Tabela 1: Pacotes utilizados e suas versões

2.2 Organização dos arquivos

Para melhor proveito e facilitar entendimento o arquivo foi organizado da seguinte forma:

- **Entradas** Pasta para imagens de entrada
- **Saidas** Pasta para imagens de saída
- **ajusteDeBrilho.py** Implementa correção gamma para ajuste de brilho em imagens monocromáticas.
- **alteracaoDeCores.py** Implementa transformações de cores (sépia e conversão para tons de cinza).
- **combinacaoDeImagens.py** implementa a combinação de duas imagens usando média ponderada $(\alpha A + (1 - \alpha)B)$.
- **esbocoALapis.py** Implementa o efeito de esboço (filtro Gaussiano 21×21 + divisão).
- **filtragemDeImagens.py** Implementa 11 operações de convolução (Sobel, Laplaciano, etc.)
- **mosaico.py** Implementa a reorganização de uma imagem em blocos 4×4 com permutação específica.
- **planoDeBits.py** Implementa a extração e exibe planos de bits individuais (0 a 7).
- **quantizacaoDeImagens.py** implementa a redução de níveis de cinza (2^n onde $n \in \{1, \dots, 8\}$).
- **transformacaoDeImagens.py** Implementa operações geométricas (espelhamento, rotação).

- **transformacaoDeIntensidade.py** Implementa a modificação de histograma (negativo, intervalo [100,200])
- **relatorio.pdf** Relatório descritivo do trabalho 1.

3 Códigos

3.1 ajusteDeBrilho.py

Esse arquivo implementa um modificador de brilho de uma imagem com uma técnica chamada de correção gamma. Essa técnica realiza a modificação de forma não linear, modificando a relação entre os valores dos pixels e a intensidade de luz exibida. A correção gamma é aplicada usando a seguinte fórmula:

A correção gamma segue a equação:

$$I_{\text{saída}} = I_{\text{entrada}}^{1/\gamma} \quad (1)$$

onde:

- I_{entrada} é o valor do pixel normalizado (0 a 1)
- $I_{\text{saída}}$ é o valor corrigido
- γ é o fator de correção

3.1.1 Passo a passo

- **Transformar em escala de cinza:** Primeiramente, certifica-se que a imagem está em escala de cinza.
- **Normalização:** Os valores dos pixels (originalmente entre 0 a 255) são convertidos para [0,1]
- **Aplicação do gamma:** O valor convertido é elevado a $1/\gamma$. Se $\frac{1}{\gamma} < 1$, há uma expansão dos tons escuros e a imagem fica mais clara. Se $\frac{1}{\gamma} > 1$, há uma compressão dos tons claros e a imagem fica mais escura.
- **Volta para o intervalo [0,255]:** Multiplica-se por 255 e converte para inteiro (uint8)

Usando ajusteDeBrilho.py: Para realizar a conversão basta usar o terminal, dando como argumento uma das imagens da pasta "Entradas"

Terminal

```

1 python ajusteDeBrilho.py foto.jpg --saida resultado
2
3 ou apenas
4
5 python ajusteDeBrilho.py foto.jpg

```

Saída: Serão realizadas três correções com os gamas: 1.5, 2.5 e 3.5 e as três imagens resultantes serão salvas na pasta "Saidas" com nome no modelo: foto_gamma[valor do gama]

3.1.2 Exemplo

```
1 python ajusteDeBrilho.py ladygaga.png
```



Figura 1: Imagem original



(a) $\gamma = 1.5$



(b) $\gamma = 2.5$



(c) $\gamma = 3.5$

Figura 2: Resultados da correção gamma com diferentes valores

3.2 alteracaoDeCores.py

Esse arquivo implementa uma transformação de cores em imagens utilizando uma matriz de conversão para criar um efeito sépia. A técnica aplica uma transformação linear nos canais de cores da imagem.

A transformação de cores é aplicada usando a seguinte operação matricial:

$$\text{RGB}_{\text{saída}} = \text{RGB}_{\text{entrada}} \times M^T \quad (2)$$

onde:

- $\text{RGB}_{\text{entrada}}$ é a imagem original no espaço RGB
- $\text{RGB}_{\text{saída}}$ é a imagem transformada
- M é a matriz de transformação sépia:

$$M = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix}$$

3.2.1 Passo a passo

- **Carregamento da imagem:** A imagem é carregada e convertida para float32 no intervalo [0,1]
- **Conversão de cores:** A imagem é convertida de BGR para RGB
- **Aplicação da matriz:** A matriz de transformação sépia é aplicada multiplicando cada pixel
- **Normalização:** Os valores são clipados para [0,1] e convertidos para uint8 [0,255]
- **Conversão final:** A imagem é convertida de volta para BGR para salvamento
- **Salvamento:** A imagem resultante é salva na pasta de saídas

Usando alteracaoDeCores.py: Para realizar a conversão basta usar o terminal, dando como argumento uma das imagens da pasta "Entradas"

Terminal

```
1 python alteracaoDeCores.py foto.jpg --saida resultado  
2  
3 ou apenas  
4  
5 python alteracaoDeCores.py foto.jpg
```

Saída: A imagem transformada será salva na pasta "Saidas" com o nome no modelo: transformada_[nome original].png ou com o nome especificado pelo usuário

3.2.2 Exemplo

```
1 python alteracaoDeCores.py lanadelrey.png
```



Figura 3: Imagem original



Figura 4: Imagem em sépia

3.3 combinacaoDeImagens.py

Esse arquivo implementa uma combinação ponderada de duas imagens monocromáticas usando média ponderada. A técnica realiza uma interpolação linear entre os pixels correspondentes das imagens de entrada.

A combinação de imagens segue a equação:

$$I_{\text{saída}} = \alpha \cdot I_{\text{entrada}_A} + (1 - \alpha) \cdot I_{\text{entrada}_B} \quad (3)$$

onde:

- I_{entrada_A} é a primeira imagem normalizada (0 a 1)
- I_{entrada_B} é a segunda imagem normalizada (0 a 1)
- α é o peso da imagem A (entre 0 e 1)
- $I_{\text{saída}}$ é a imagem resultante

3.3.1 Passo a passo

- **Carregamento das imagens:** As imagens são carregadas em tons de cinza e normalizadas para float32 no intervalo [0,1]
- **Verificação de tamanho:** Confirma que ambas imagens têm as mesmas dimensões
- **Combinação ponderada:** Aplica a média ponderada pixel a pixel
- **Normalização:** Os valores são clipados para [0,1] e convertidos para uint8 [0,255]
- **Salvamento:** A imagem resultante é salva na pasta de saídas

Usando combinacaoDeImagens.py: Para realizar a combinação use o terminal com os seguintes argumentos:

Terminal

```

1 python combinacaoDeImagens.py foto1.png foto2.png --peso_a 0.7 --saida
      resultado

2

3 apenas com os pesos:
4 python combinacaoDeImagens.py foto1.jpg foto2.png --peso_a 0.7

5

6 ou com valores padrão(50,50):
7

8 python combinacaoDeImagens.py foto1.jpg foto2.png

```

Saída: A imagem combinada será salva na pasta "Saidas" com nome no formato:
combinada_[peso]A_[nome1]_[nome2].png (ex: **combinada_0_7A_foto1_foto2.png**)

3.3.2 Exemplo

Para obter uma imagem com 30% da primeira e 70% da segunda:

```

1 python combinacaoDeImagens.py katyperry.png nickminaj.png --peso_a 0.3

```



Figura 5: Imagem original 1



Figura 6: Imagem original 2



Figura 7: Imagens combinadas

3.4 esbocoALapis.py

Este arquivo implementa uma transformação de imagem em esboço a lápis. O método combina desfoco gaussiano e divisão de imagens para simular o efeito de desenho artístico.

A transformação segue o princípio:

$$\text{Esboço} = \frac{\text{Imagen_Cinza}}{\text{Imagen_Desfocada} + \epsilon} \quad (4)$$

onde:

- `Imagen_Cinza` é a versão em tons de cinza da original
- `Imagen_Desfocada` é a versão com filtro gaussiano aplicado
- ϵ (1e-6) é uma constante para evitar divisão por zero

3.4.1 Passo a passo

1. **Conversão para tons de cinza:** Transformação do espaço de cores RGB para escala de cinza
2. **Aplicação de desfoco gaussiano:** Filtro com kernel 21×21 para suavização
3. **Divisão de imagens:** Operação pixel a pixel entre a imagem original e a desfocada
4. **Normalização:** Ajuste dos valores para o intervalo $[0,255]$ e conversão para uint8
5. **Salvamento:** Armazenamento do resultado em formato PNG

3.4.2 Interface de Linha de Comando

Modo de Uso

```
1 python esbocoALapis.py entrada.jpg --saida meu_esboco
2
3 ou, simplismente:
4
5 python esbocoALapis.py entrada.png
```

3.4.3 Exemplo

```
1 python esbocoALapis.py marinasesena.png
```



Figura 8: Imagem original

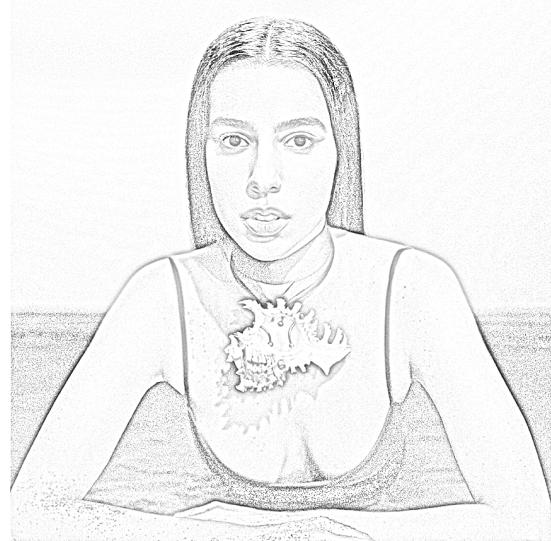


Figura 9: Imagem com efeito de Esboço a lápis

3.5 filtragemDeImagens.py

Este arquivo implementa uma coleção de 11 operadores de convolução para processamento de imagens em tons de cinza, incluindo filtros para detecção de bordas, suavização, realce e efeitos especiais.

3.5.1 Princípio Matemático

A operação básica é a convolução 2D:

$$I_{\text{saída}}(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k I_{\text{entrada}}(x + i, y + j) \cdot K(i, j) \quad (5)$$

onde:

- K é o kernel do filtro (matriz de coeficientes)
- k determina o tamanho da vizinhança ($3 \times 3, 5 \times 5$)
- I_{entrada} e $I_{\text{saída}}$ são os valores dos pixels antes e após o filtro

3.5.2 Catálogo de Filtros Implementados

- **Laplacianos** (h1, h5): Realce de bordas
- **Gaussiano** (h2): Suavização avançada
- **Sobel/Prewitt** (h3,h4,h7,h8): Detecção de bordas direcionais
- **Média** (h6,h9): Redução de ruído

- **Aguçamento** (h10): Melhoria de detalhes
- **Embossing** (h11): Efeito 3D de relevo

3.5.3 Fluxo de Processamento

1. Pré-processamento:

- Conversão para tons de cinza
- Normalização para float32 [0,1]

2. Núcleo de Convolução:

- Seleção do kernel conforme filtro especificado
- Aplicação via operação `cv2.filter2D`
- Normalização pós-filtro (para filtros diferenciais)

3. Casos Especiais:

- Combinação de Sobel (magnitude do gradiente)
- Tratamento de bordas (padding implícito)

3.5.4 Interface de Linha de Comando

Modo de Uso

```
1 python filtragemDeImagens.py entrada.png --filtro h3 --saida bordas_vert
```

Parâmetros principais:

- **-filtro**: Identificador do filtro (h1-h11) ou **all** para processar todos
- **-saída**: Nome personalizado para o arquivo de saída

3.5.5 Exemplo

Para aplicar todos os filtros em uma imagem:

```
1 python filtragemDeImagens.py nickminaj.png --filtro all
```



Figura 10: Imagem original



(a) h1



(b) h2



(c) h3



(d) h4



(e) h5



(f) h6



(g) h7



(h) h8



(i) h9



(j) h10



(k) h11



(l) Sobel combinado

Figura 11: Resultados dos operadores de convolução

Saída gerada (na pasta **Saidas**):

- Relatório automático com a descrição de cada filtro aplicado

3.5.6 Características Técnicas

- Suporte a kernels 3×3 e 5×5

- Normalização automática para filtros diferenciais
- Tratamento especial para combinação de operadores Sobel
- Gerenciamento automático de diretórios de saída

3.6 mosaico.py

Este arquivo implementa uma transformação que divide uma imagem em 16 blocos e os reorganiza em uma ordem específica para criar um efeito de mosaico. A técnica preserva todos os pixels originais, apenas alterando sua disposição espacial.

3.6.1 Princípio de Funcionamento

O algoritmo opera através de:

$$\text{Mosaico} = \mathcal{R} \left(\bigcup_{i=1}^{16} B_{\sigma(i)} \right) \quad (6)$$

onde:

- B_i representa o i-ésimo bloco da divisão 4×4
- $\sigma(i)$ é a permutação que define a nova ordem dos blocos
- \mathcal{R} é a operação de reconstrução da imagem

3.6.2 Passo a Passo

1. Divisão da Imagem:

- Conversão para tons de cinza
- Segmentação em 16 blocos retangulares de igual tamanho (4×4)

2. Reordenação:

- Aplicação da permutação fixa: [5,10,12,2,7,15,0,8,11,13,1,9,3,14,6,4]
- Os índices correspondem às posições originais dos blocos

3. Reconstrução:

- Composição dos blocos na nova ordem
- Combinação usando operações de empilhamento vertical (`np.vstack`)
- e horizontal (`np.hstack`)

3.6.3 Interface de Linha de Comando

Modo de Uso

```
1 python mosaico.py entrada.png --saida meu_mosaico
2
3 ou simplesmente:
4
5 python mosaico.py entrada.png
```

3.6.4 Exemplo

```
1 python mosaico.py katyperry.png
```



Figura 12: Imagem original



Figura 13: Efeito Mosaico

3.7 planoDeBits.py

Este arquivo implementa uma técnica de processamento de imagens que permite isolar planos de bits individuais de uma imagem monocromática. Cada pixel em uma imagem de 8 bits pode ser decomposto em 8 planos binários independentes.

3.7.1 Princípio Matemático

A extração do plano de bit segue a operação:

$$P_{\text{bit}}(x, y) = \left\lfloor \frac{I(x, y) \& 2^n}{2^n} \right\rfloor \times 255 \quad (7)$$

onde:

- $I(x, y)$ é o valor do pixel na posição (x, y)
- n é o número do plano de bit (0 a 7)
- $\&$ representa a operação AND bit a bit
- $P_{\text{bit}}(x, y)$ é o valor do pixel no plano extraído

3.7.2 Passo a Passo

1. Pré-processamento:

- Carregamento da imagem em tons de cinza
- Normalização para float32 [0,1]
- Conversão para valores de 8 bits [0,255]

2. Extração do Plano:

- Aplicação de máscara bit a bit (operação AND)
- Deslocamento de bits para isolar o plano desejado
- Normalização para 0 ou 255

3. Interpretação:

- Plano 7 (bit mais significativo): Contém a informação estrutural principal
- Plano 0 (bit menos significativo): Contém principalmente ruído e detalhes sutis

3.7.3 Interface de Linha de Comando

Modo de Uso

```
1 python extrairPlanosBits.py entrada.png --plano 3 --saída plano3_imagem
```

Parâmetros:

- **-plano**: Número do plano a extrair (0 a 7, obrigatório)
- **-saída**: Nome opcional para o arquivo de saída

3.7.4 Exemplo

```
1 python planoDeBits.py radiografia.png --plano 0
2 python planoDeBits.py radiografia.png --plano 1
3 python planoDeBits.py radiografia.png --plano 2
4 python planoDeBits.py radiografia.png --plano 3
5 python planoDeBits.py radiografia.png --plano 4
6 python planoDeBits.py radiografia.png --plano 5
7 python planoDeBits.py radiografia.png --plano 6
8 python planoDeBits.py radiografia.png --plano 7
```

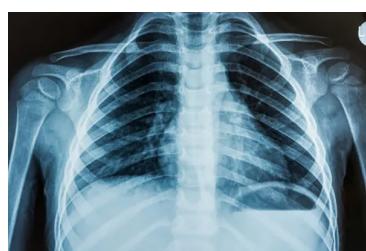


Figura 14: Imagem original

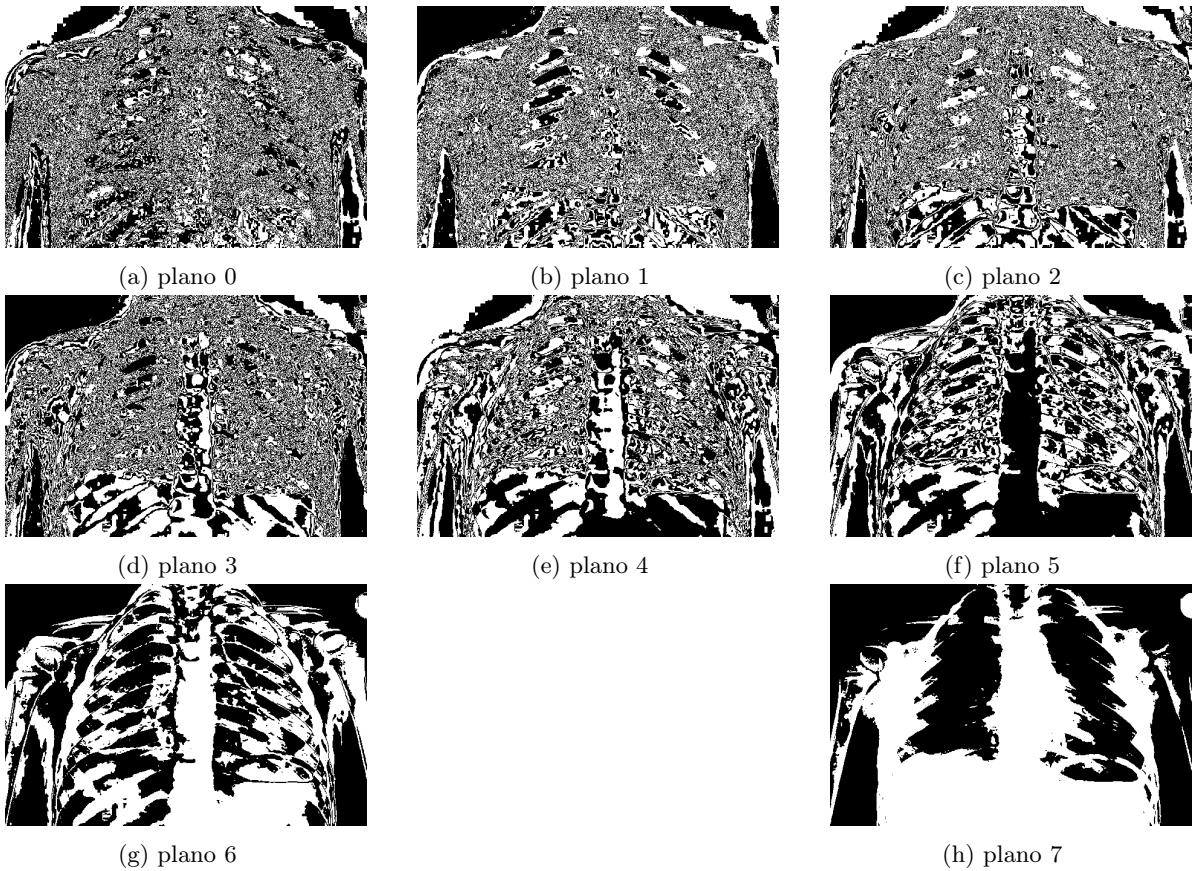


Figura 15: Comparação dos diferentes planos de bits de uma imagem (do bit 7 ao bit 0)

3.8 quantizacaoDeImagens.py

Este arquivo implementa uma técnica de redução de níveis de cinza em imagens monocromáticas através de quantização uniforme. O processo mapeia os valores originais de pixels para um conjunto discreto de níveis pré-definidos.

3.8.1 Princípio Matemático

A quantização segue a fórmula:

$$Q(x, y) = \left\lfloor \frac{I(x, y)}{k} \right\rfloor \times k \quad (8)$$

onde:

- $I(x, y)$ é o valor do pixel original [0,255]
- k é o tamanho do intervalo de quantização ($k = 256/\text{níveis}$)
- $Q(x, y)$ é o valor quantizado do pixel

3.8.2 Passo a Passo

1. Pré-processamento:

- Carregamento da imagem em tons de cinza
- Normalização para float32 [0,1]
- Conversão para valores de 8 bits [0,255]

2. Quantização:

- Cálculo do intervalo de quantização (k)
- Discretização dos valores dos pixels
- Caso especial para 2 níveis (binarização em 0 e 255)

3. Características:

- Suporta potências de 2 (2 a 256 níveis)
- Preserva a faixa dinâmica original [0,255]
- Distribuição uniforme dos níveis de cinza

3.8.3 Interface de Linha de Comando

Modo de Uso

```

1 python quantizacaoDeImagens.py entrada.jpg --niveis 8 --saida imagem_8niveis
2
3 ou, simplismente:
4
5 python quantizacaoDeImagens.py entrada.jpg --niveis 8

```

Parâmetros:

- **-niveis**: Número de níveis (2,4,8,16,32,64,256 - obrigatório)
- **-saída**: Nome opcional para o arquivo de saída

3.8.4 Aplicações

- Simulação de sistemas com paleta limitada
- Pré-processamento para compressão de imagens
- Efeitos artísticos e estilização
- Redução de complexidade em processamento de imagens

3.8.5 Exemplo

```
1 python quantizacaoDeImagens.py ladygaga.png --niveis 2
2 python quantizacaoDeImagens.py ladygaga.png --niveis 4
3 python quantizacaoDeImagens.py ladygaga.png --niveis 8
4 python quantizacaoDeImagens.py ladygaga.png --niveis 16
5 python quantizacaoDeImagens.py ladygaga.png --niveis 32
6 python quantizacaoDeImagens.py ladygaga.png --niveis 64
7 python quantizacaoDeImagens.py ladygaga.png --niveis 256
```



Figura 16: Imagem original



(a) 2



(b) 4



(c) 16



(d) 32



(e) 64



(f) 256

Figura 17: Comparação dos diferentes níveis de quantização

3.9 transformacaoDeImagensColoridas.py

Este arquivo implementa duas transformações de cores em imagens RGB: efeito sépia e conversão monocromática. Cada transformação utiliza operações matriciais para modificar os canais de cor de forma sistemática.

3.9.1 Transformação Sépia

Aplica um filtro vintage através da multiplicação matricial:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (9)$$

3.9.2 Transformação Monocromática

Converte para tons de cinza usando coeficientes de luminância:

$$I = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B \quad (10)$$

3.9.3 Passo a Passo

1. Pré-processamento:

- Carregamento da imagem colorida
- Normalização para float32 [0,1]
- Conversão de BGR para RGB (OpenCV para convenção padrão)

2. Núcleo de Transformação:

- **Sépia:** Multiplicação matricial com kernel de sépia
- **Monocromática:** Média ponderada dos canais RGB
- Clipagem de valores para [0,1]

3. Pós-processamento:

- Conversão para uint8 [0,255]
- Conversão de RGB para BGR (para salvamento)
- Salvamento em formato PNG

3.9.4 Interface de Linha de Comando

Modo de Uso

```
1 python transformacaoDeImagensColoridas.py entrada.png --transformacao sepia --
2           saida foto_sepia
```

```

3 ou, simplismente:
4
5 python transformacaoDeImagensColoridas.py entrada.png --transformacao sephia

```

Parâmetros:

- **-transformacao:** Tipo de transformação (**sephia** ou **monocromatica**)
- **-saida:** Nome opcional para o arquivo de saída

3.9.5 Exemplo

```

1 python transformacaoDeImagensColoridas.py lanadelrey.png --transformacao sephia
2 python transformacaoDeImagensColoridas.py lanadelrey.png --transformacao
   monocromatica

```



Figura 18: Imagem original



Figura 19: Sepia



Figura 20: Monocromatica

3.10 transformacaoDeIntensidade.py

Este arquivo) implementa cinco transformações distintas em imagens monocromáticas, operando sobre a distribuição de intensidade dos pixels ou sua disposição espacial.

3.10.1 Transformações Implementadas

- **Negativo:** $I_{\text{saída}} = 255 - I_{\text{entrada}}$
- **Intervalo:** $I_{\text{saída}} = 100 + \left(\frac{I_{\text{entrada}} - \min(I)}{\max(I) - \min(I)} \times 100 \right)$
- **Inversão de linhas pares:** $I_{\text{saída}}(2k, :) = I_{\text{entrada}}(2k, :: -1)$
- **Reflexão de linhas:** $I_{\text{metade inferior}} = I_{\text{metade superior espelhada}}$
- **Espelhamento vertical:** $I_{\text{saída}} = \text{flip}(I_{\text{entrada}}, 0)$

3.10.2 Passo a Passo

1. Pré-processamento:

- Carregamento em tons de cinza

- Normalização para float32 [0,1]
- Conversão para uint8 [0,255]

2. Núcleo de Transformação:

- Operações matemáticas ponto a ponto (negativo, intervalo)
- Manipulação de arrays NumPy (inversão, espelhamento)
- Preservação dos dados originais (cópias seguras)

3. Características:

- Transformações reversíveis (negativo, espelhamento)
- Transformações irreversíveis (intervalo, reflexão)
- Operações locais e globais

3.10.3 Interface de Linha de Comando

Modo de Uso

```
1 python transformacaoDeIntensidade.py entrada.png --transformacao negativo --
    saida foto_negativo
```

Parâmetros:

- **-transformacao**: Tipo de transformação (obrigatório)
 - **Negativo (negativo)**
 - * Fórmula: $I_{saída} = 255 - I_{entrada}$
 - * Efeito: Inversão tonal (preto ↔ branco)
 - * Exemplo: `python transformacaoDeIntensidade.py entrada.png -t negativo`
 - **Ajuste de Intervalo (intervalo)**
 - * Fórmula: normalização linear para [100, 200]
 - * Efeito: Compactação dinâmica com corte
 - * Exemplo: `python transformacaoDeIntensidade.py entrada.png -t intervalo`
 - **Inversão de Linhas Pares (inverter_pares)**
 - * Operação: Inversão horizontal das linhas com índice par
 - * Efeito: Padrão de "zebra espelhada"
 - * Exemplo: `python transformacaoDeIntensidade.py entrada.png -t inverter_pares`
 - **Reflexão de Linhas (reflexao_linhas)**
 - * Operação: Espelhamento da metade superior na inferior

- * Efeito: Simetria vertical artificial
- * Exemplo: `python transformacaoDeIntensidade.py entrada.png -t reflexao_linhas`
- **Espelhamento Vertical (espelhamento_vertical)**
 - * Operação: Flip vertical ($0 \leftrightarrow$ altura)
 - * Efeito: Imagem de cabeça para baixo
 - * Exemplo: `python transformacaoDeIntensidade.py entrada.png -t espelhamento_vertical`
- **-saída:** Nome opcional para o arquivo de saída

3.10.4 Exemplo

```

1 python transformacaoDeIntensidade.py marinasea.png -t negativo
2 python transformacaoDeIntensidade.py entrada.png -t intervalo
3 python transformacaoDeIntensidade.py marinasea.png -t inverter_pares
4 python transformacaoDeIntensidade.py marinasea.png -t reflexao_linhas
5 python transformacaoDeIntensidade.py marinasea.png -t espelhamento_vertical

```



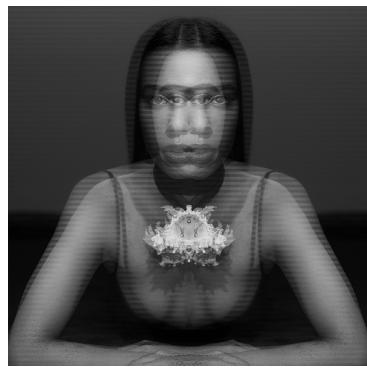
Figura 21: Imagem original



(a) negativo



(b) intervalo



(c) inverter_pares



(d) reflexao_linhas



(e) espelhamento_vertical

Figura 22: Resultados das transformações de intensidade

4 Conclusões

O trabalho apresentado demonstra a implementação prática de diversas técnicas fundamentais de processamento digital de imagens, abrangendo desde transformações de intensidade e cores até filtragem espacial, quantização e manipulação geométrica. Utilizando Python e bibliotecas como OpenCV e NumPy, foram desenvolvidos algoritmos eficientes que aplicam conceitos teóricos de forma vetorizada, garantindo desempenho e precisão.

Principais Contribuições

- **Transformações de Intensidade:** Implementação de operações como negativo, ajuste de intervalo e reflexão de linhas, que modificam a distribuição tonal da imagem para realce ou efeitos visuais.
- **Filtragem Espacial:** Aplicação de kernels de convolução para detecção de bordas (Sobel, Laplaciano), suavização (Gaussiano) e efeitos artísticos (embossing).
- **Manipulação de Cores:** Conversão para tons de cinza e efeito sépia via operações matriciais, destacando a importância do espaço de cores RGB.
- **Técnicas Avançadas:** Quantização uniforme para redução de níveis de cinza, extração de planos de bits e combinação ponderada de imagens, ilustrando aplicações em compressão e análise de imagens.
- **Estrutura Modular:** Organização clara dos scripts em uma pasta de projetos, com entradas e saídas padronizadas, facilitando a reprodução e extensão dos métodos.

Validação e Resultados

Os resultados foram validados visualmente, comparando-os com exemplos esperados e demonstrando a eficácia das técnicas. Por exemplo:

- A correção gamma (`ajusteDeBrilho.py`) ajustou dinamicamente o brilho conforme o parâmetro γ .
- O efeito sépia (`transformacaoDeImagensColoridas.py`) preservou a estrutura da imagem enquanto aplicava um filtro vintage.
- A filtragem com operadores de Sobel (`filtragemDeImagens.py`) destacou bordas verticais e horizontais de forma eficiente.

Desafios e Soluções

- **Tratamento de Bordas:** Em operações de convolução, foi utilizado padding implícito para evitar artefatos.
- **Eficiência Computacional:** As operações foram vetorizadas com NumPy para evitar loops explícitos.
- **Flexibilidade:** A interface de linha de comando (`argparse`) permitiu a personalização de parâmetros.

Aplicações Práticas

- **Medicina:** Análise de planos de bits em radiografias (`planoDeBits.py`).
- **Arte Digital:** Efeitos sépia e esboço a lápis (`esbocoALapis.py`).
- **Processamento de Sinais:** Redução de ruído via filtro Gaussiano.

Melhorias Futuras

- Extensão para imagens coloridas em todas as técnicas.
- Implementação de interfaces gráficas para facilitar o uso.

Em suma, o trabalho consolidou conhecimentos teóricos e práticos em processamento de imagens, fornecendo uma base sólida para projetos mais complexos. A abordagem modular e documentada facilita a reutilização do código em futuras aplicações acadêmicas ou industriais.