INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA** **Campus Birigui** **Bacharelado em Engenharia de Computação**

Disciplina: Processamento Digital de Imagem	**Atividade**
Professor: Prof. Dr. Murilo Varges da Silva	**Data:**
Nome do Aluno: Henrique Akira Hiraga	**Prontuário:** BI300838X

TRANSFORMAÇÕES E HISTOGRAMA

Transformação logarítimica

Implementação:

1. Importação de bibliotecas:

- cv2: Biblioteca OpenCV para processamento de imagem.
- numpy (como np): Usado para operações matriciais e de array.
- matplotlib.pyplot (como plt): Utilizado para exibir imagens.

2. Leitura da imagem:

- A imagem é lida usando a função cv2.imread(), com o nome de arquivo 'fractured_spine.jpg'.

3. Aplicação da transformação logarítmica:

- A transformação logarítmica é aplicada à imagem original com a seguinte fórmula: imagem_log = 50 * (np.log(imagem + 1)). Isso envolve a adição de 1 a cada pixel antes de calcular o logaritmo para evitar valores negativos, pois o logaritmo de 0 não está definido.
- O resultado é armazenado na variável imagem_log.

4. Conversão para inteiros:

- Para que os valores resultantes sejam convertidos para inteiros (uma vez que as imagens geralmente são representadas como inteiros de 0 a 255), a matriz resultante é convertida para o tipo de dados np.uint8.

5. Exibição das imagens:

- As duas imagens, a original (imagem) e a transformada (imagem_log), são exibidas usando plt.imshow() e plt.show().

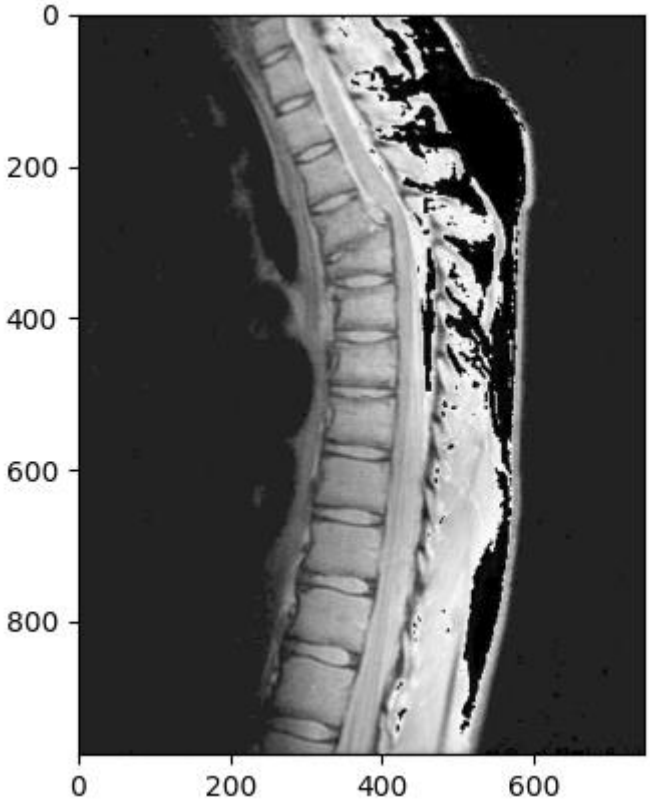
```
# Importar as bibliotecas necessárias
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Ler uma imagem
imagem = cv2.imread('fractured_spine.jpg')

# Aplicar o método de transformação logarítmica
imagem_log = 50 * (np.log(imagem + 1))

# Especificar o tipo de dados para que
# os valores float sejam convertidos para inteiros
imagem_log = np.array(imagem_log, dtype=np.uint8)

# Exibir ambas as imagens
plt.imshow(imagem)
plt.show()
plt.imshow(imagem_log)
plt.show()
```

Efeito na imagem:

A transformação logarítmica é frequentemente usada em processamento de imagem para realçar detalhes em áreas escuras de uma imagem. A aplicação do logaritmo a cada pixel da imagem original resulta em um aumento do contraste nas áreas de baixa intensidade, tornando os detalhes mais visíveis. A multiplicação por 50 é um fator de escala que controla o grau de realce.

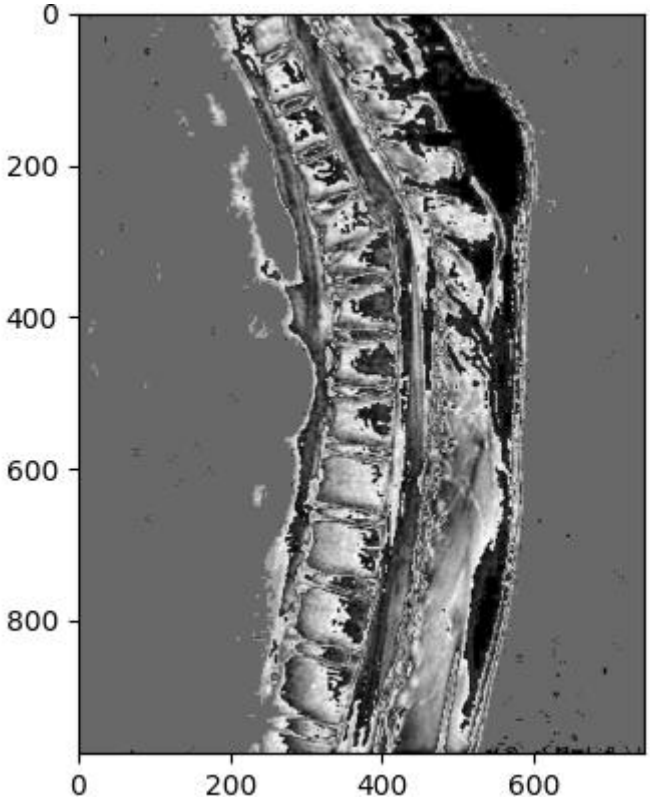
Portanto, a imagem exibida após a aplicação da transformação logarítmica (imagem_log) deve mostrar uma imagem com maior contraste, onde os detalhes nas áreas escuras podem ser mais visíveis em comparação com a imagem original (imagem). Esta técnica é especialmente útil em casos em que os detalhes nas sombras são importantes e precisam ser realçados para análise ou visualização.

Resultado com c=50:



Resultado com c=100:

Resultado com c=150:



Transformação de potência

Implementação:

1. Importação de bibliotecas:

- cv2: Biblioteca OpenCV para processamento de imagem.
- numpy (como np): Usado para operações matriciais e de array.

2. Abrir a imagem:

- A imagem é lida usando a função cv2.imread(), com o nome de arquivo 'fractured_spine.jpg'.

3. Aplicar correção gama:

- O código realiza um loop for para iterar através de quatro valores diferentes de gama (0.1, 0.5, 1.2 e 2.2).
- Para cada valor de gama, a correção gama é aplicada à imagem usando a fórmula gama_corrigido = np.array(255*(img / 255) ** gamma, dtype='uint8'). Isso eleva cada pixel da imagem à potência do valor de gama e ajusta a escala para o intervalo

de 0 a 255, resultando na imagem corrigida.

4. Salvar imagens editadas:

- As imagens corrigidas para cada valor de gama são salvas em arquivos diferentes, com os nomes 'gamma_transformed0.1.jpg', 'gamma_transformed0.5.jpg', 'gamma_transformed1.2.jpg' e 'gamma_transformed2.2.jpg'.

```
# Importar as bibliotecas necessárias
import cv2
import numpy as np

# Abrir a imagem.
img = cv2.imread('fractured_spine.jpg')

# Tentar 4 valores de gama diferentes.
for gamma in [0.1, 0.5, 1.2, 2.2]:

    # Aplicar correção gama.
    gama_corrigido = np.array(255*(img / 255) ** gamma,
dtype='uint8')

    # Salvar imagens editadas.
    cv2.imwrite('gamma_transformed' + str(gamma) + '.jpg',
gama_corrigido)
```

Efeito na imagem:

A correção gama é uma técnica utilizada para ajustar o brilho e o contraste de uma imagem. Ela é frequentemente usada para melhorar a visualização de detalhes em áreas escuras ou claras de uma imagem.

- Quando o valor de gama é menor que 1 (como 0.1 e 0.5), a correção gama "escurece" a imagem, tornando as regiões escuras mais escuras e realçando os detalhes nessas áreas.
- Quando o valor de gama é maior que 1 (como 1.2 e 2.2), a correção gama "clareia" a imagem, tornando as regiões claras mais claras e realçando os detalhes nessas áreas.

Portanto, o efeito dessa implementação é criar quatro imagens corrigidas com diferentes níveis de contraste e brilho, dependendo do valor de gama escolhido. Cada imagem

resultante terá uma aparência ligeiramente diferente em termos de realce de detalhes e ajuste de brilho, em comparação com a imagem original 'fractured_spine.jpg'. As imagens corrigidas são salvas em arquivos separados para análise ou comparação posterior.

Resultado com gamma = 0.1:

Resultado com gamma = 0.5:

Resultado com gamma = 1.2:

Resultado com gamma = 1.1:

Plano de bits

Implementação:

1. Importação de bibliotecas:

- cv2: Biblioteca OpenCV para processamento de imagem.
- matplotlib.pyplot (como plt): Utilizado para exibir imagens.
- numpy (como np): Usado para operações matriciais e de array.
- math: Importado, mas não é utilizado no código.

2. Ler a imagem em tons de cinza:

- A imagem é lida em tons de cinza usando a função cv2.imread() com o argumento 0, o que carrega a imagem em escala de cinza.

3. Iterar sobre cada pixel e converter para binário:

- O código itera sobre cada pixel da imagem e converte o valor do pixel para uma representação binária usando np.binary_repr(). O resultado é armazenado em uma lista chamada lst.

4. Extrair planos de bits:

- Os planos de bits são extraídos da lista lst e convertidos de binário para valores inteiros. Cada plano de bits (de 1 a 8) é armazenado em uma matriz separada, onde o valor de cada pixel no plano de bits é multiplicado por uma potência de 2, dependendo do plano.

5. Concatenar imagens para exibição:

- As imagens resultantes dos planos de bits são concatenadas horizontalmente para que possam ser exibidas em uma única imagem usando a função cv2.hconcat().

6. Exibir as imagens dos planos de bits:

- As imagens dos planos de bits (de 1 a 8) são exibidas em uma grade usando o Matplotlib.

7. Combinar 4 planos de bits:

- Os planos de bits 5, 6, 7 e 8 são combinados para formar uma nova imagem. Isso é feito somando os valores dos pixels nos quatro planos de bits.

8. Exibir a imagem combinada:

- A imagem resultante da combinação dos planos de bits 5, 6, 7 e 8 é exibida.

Efeito na imagem:

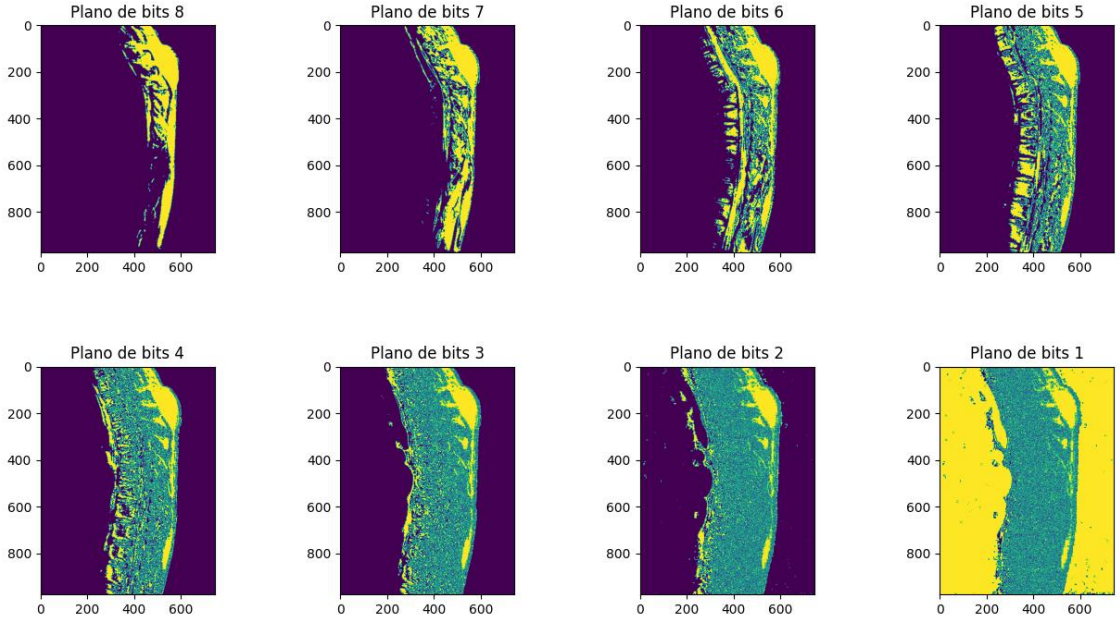
O código divide a imagem original em seus oito planos de bits individuais, cada um representando uma parte diferente da intensidade de cinza da imagem. Isso permite uma

análise detalhada das informações contidas em cada plano de bits.

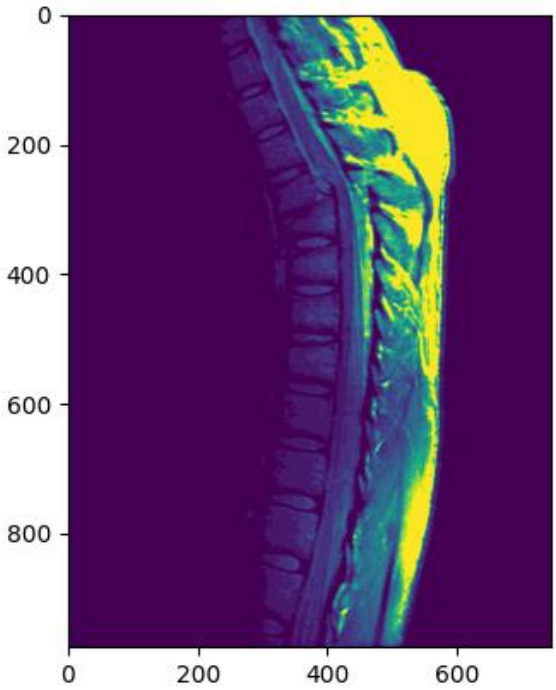
- Planos de bits mais significativos (7 e 8) geralmente representam os detalhes mais visíveis na imagem.
- Planos de bits intermediários (4, 5 e 6) contêm informações intermediárias de intensidade de cinza.
- Planos de bits menos significativos (1, 2 e 3) representam os detalhes mais finos e sutis da imagem.

Ao combinar os planos de bits 5, 6, 7 e 8, você está enfatizando as informações mais significativas e detalhadas da imagem, enquanto ignora os detalhes mais sutis e o ruído nas partes menos significativas dos planos de bits. O resultado é uma imagem que destaca os detalhes importantes e é menos sensível ao ruído nas partes menos significativas. Isso pode ser útil em várias aplicações de processamento de imagem, como realce de detalhes importantes.

Resultado do plano de bits:



Resultado da combinação do plano de bits:



Equalização do histograma

Implementação:

1. Importação de bibliotecas:

- cv2: Biblioteca OpenCV para processamento de imagem.
- numpy (como np): Usado para operações matriciais e de array.

2. Ler a imagem:

- A imagem é lida usando a função cv2.imread() e é armazenada na variável img.

3. Converter a imagem para o espaço de cores YUV:

- A imagem é convertida do espaço de cores BGR (padrão do OpenCV) para o espaço de cores YUV usando cv2.cvtColor(). Isso separa a imagem em três canais: Y (luminância), U (crominância azul) e V (crominância vermelha).

4. Realizar equalização de histograma no canal Y (luminância):

- A equalização de histograma é aplicada apenas ao canal Y (luminância) da imagem YUV. Isso é feito ajustando o histograma do canal Y para tornar a

distribuição de intensidades mais uniforme.

5. Converter a imagem resultante de volta para o espaço de cores BGR:

- Após a equalização de histograma no canal Y, a imagem é convertida novamente para o espaço de cores BGR usando cv2.cvtColor(). Isso recombina os canais YUV em uma única imagem BGR.

6. Salvar a imagem resultante da equalização de histograma:

- A imagem resultante, após a equalização de histograma, é salva em um arquivo chamado 'resultado.jpg' usando cv2.imwrite().

```
# Importar as bibliotecas necessárias
import cv2
import numpy as np

# Ler a imagem
img = cv2.imread('fractured_spine.jpg')

# Converter a imagem para o espaço de cores YUV
img_para_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

# Realizar equalização de histograma no canal Y (luminância)
img_para_yuv[:, :, 0] = cv2.equalizeHist(img_para_yuv[:, :,
0])

# Converter a imagem resultante de volta para o espaço de
cores BGR
resultado_equalizacao_hist = cv2.cvtColor(img_para_yuv,
cv2.COLOR_YUV2BGR)

# Salvar a imagem resultante da equalização de histograma
cv2.imwrite('resultado.jpg', resultado_equalizacao_hist)
```

Efeito na imagem:

A equalização de histograma é uma técnica de processamento de imagem usada para melhorar o contraste e a qualidade visual de uma imagem. Ela redistribui as intensidades de pixel de modo que a gama completa de valores seja utilizada, o que pode tornar detalhes mais visíveis e a imagem mais nítida.

No contexto deste código:

- A conversão para o espaço de cores YUV é realizada para separar a luminância (Y) dos componentes de cor (U e V).
- A equalização de histograma é aplicada apenas ao canal Y (luminância), o que significa que o contraste e a distribuição de intensidades de luminância na imagem são melhorados.
- A conversão de volta para o espaço de cores BGR recombina o canal Y equalizado com os canais de cor originais (U e V).

O resultado final é uma imagem 'resultado.jpg' com maior contraste e melhorias visuais devido à equalização de histograma no canal Y. Esta técnica é frequentemente utilizada para melhorar a qualidade de imagens médicas, como radiografias, onde o realce de detalhes é crucial para diagnósticos precisos.

Resultado da equalização do histograma: