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 imagens	**Atividade 01**
Professor: Prof. Dr. Murilo Varges da Silva	**Data:** 21/08/2023
Nome do Aluno: Henrique Akira Hiraga	**Prontuário:** BI300838X

1. PROPOSTA

Os exercícios tem como proposta a implementação dos fundamentos teóricos aprendidos até o dia 21 de Agosto de 2020. Deverá ser implementado as seguintes técnicas:

- Operação ponto a ponto:
  - Calcular o negativo das imagens;
  - Diminuir pela metade a intensidade dos pixels;
  - Incluir 4 quadrados brancos 10x10 pixels em cada canto das imagens;
  - Incluir 1 quadrado preto 15X15 no centro das imagens.
- Operação por vizinhança:
  - Calcular o filtro da média;
  - Calcular o filtro da mediana.
- Transformações geométricas:
  - Escala: Redução em 1.5x e aumentar em 2.5x;
  - Rotação em 45º, 90º e 100º;
  - Translação utilizar os parâmetros que quiser nas coordenadas x e y;
  - Translação em 35 pixel no eixo X, 45 eixo Y;

2. DESENVOLVIMENTO

Para o seguinte trabalho foi utilizado a linguagem python para realizar a programação e implementação de técnicas aprendidas. Foram utilizadas as seguintes bibliotecas:

- Numpy;
- Matplotlib;
- Pillow;
- OpenCV;
- Scipy.

2.1. Operação Ponto a Ponto

Este tópico apresentará o que foi desenvolvido para as requisições.

2.1.1. Calcular o negativo das imagens

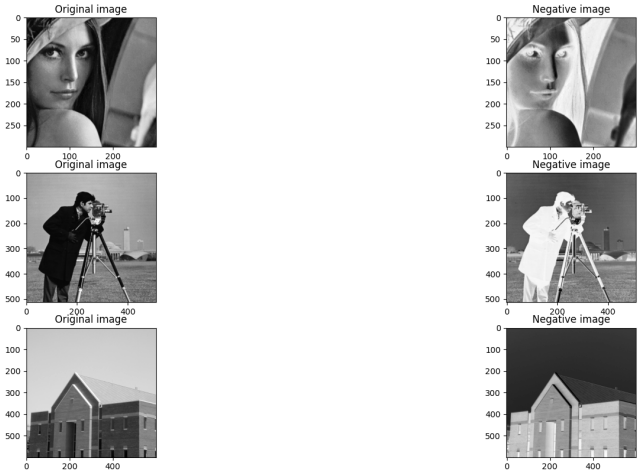
Para o cálculo do negativo das imagens foi utilizado o seguinte código:

```
def negative():
    # Abrir as images
    imageLena = Image.open('images/lena.jpg')
    imageCameraman = Image.open('images/cameraman.tif')
    imageHouse = Image.open('images/house.tif')

    # Converter as imagens para numpy array
    # e tranforma-las em negativa
    # Lena
    npImageLena = np.array(imageLena)
    negativeImageLena = 255 - npImageLena

    # Cameraman
    npImageCameraman = np.array(imageCameraman)
    negativeImageCameraman = 255 - npImageCameraman

    # House
    npImageHouse = np.array(imageHouse)
    negativeImageHouse = 255 - npImageHouse

    # Plotar imagens
    fig = plt.figure()
    ax1 = plt.subplot(3,2,1)
    ax2 = plt.subplot(3,2,2)
    ax3 = plt.subplot(3,2,3)
    ax4 = plt.subplot(3,2,4)
    ax5 = plt.subplot(3,2,5)
    ax6 = plt.subplot(3,2,6)
    ax1.title.set_text('Original image')
    ax2.title.set_text('Negative image')
    ax3.title.set_text('Original image')
    ax4.title.set_text('Negative image')
    ax5.title.set_text('Original image')
    ax6.title.set_text('Negative image')

    im1 = ax1.imshow(npImageLena, cmap='gray')
    im2 = ax2.imshow(negativeImageLena, cmap='gray')
```

```
    im3 = ax3.imshow(npImageCameraman, cmap='gray')
    im4 = ax4.imshow(negativeImageCameraman, cmap='gray')
    im5 = ax5.imshow(npImageHouse, cmap='gray')
    im6 = ax6.imshow(negativeImageHouse, cmap='gray')
    plt.show()
```

Ele permite a visualização do resultado da implementação através do matplotlib. A seguir será apresentado os resultados das imagens:

Figura 1: Aplicação do negativo nas imagens propostas



Fonte: Elaborado pelo autor.

2.1.2. Diminuir pela metade a intensidade dos pixels

Para esta atividade foi utilizado o seguinte código:

```
def lowerRes():
    # Abrir as images
    imageLena = Image.open('images/lena.jpg')
    imageCameraman = Image.open('images/cameraman.tif')
    imageHouse = Image.open('images/house.tif')

    # Converter as imagens para numpy array
    # e tranforma-las em negativa
    # Lena
    npImageLena = np.array(imageLena)
    npImageLenaLowPixel =  (npImageLena / 2).astype(int)

    # Cameraman
```

```
    npImageCameraman = np.array(imageCameraman)
    npImageCameramanLowPixel =  (npImageCameraman /
2).astype(int)

    # House
    npImageHouse = np.array(imageHouse)
    npImageHouseLowPixel =  (npImageHouse / 2).astype(int)

    # Plotar imagens
    fig2 = plt.figure()
    ax1 = plt.subplot(3,2,1)
    ax2 = plt.subplot(3,2,2)
    ax3 = plt.subplot(3,2,3)
    ax4 = plt.subplot(3,2,4)
    ax5 = plt.subplot(3,2,5)
    ax6 = plt.subplot(3,2,6)
    ax1.title.set_text('Original image')
    ax2.title.set_text('Negative image')
    ax3.title.set_text('Original image')
    ax4.title.set_text('Negative image')
    ax5.title.set_text('Original image')
    ax6.title.set_text('Negative image')

    im1 = ax1.imshow(npImageLena, cmap='gray')
    im2 = ax2.imshow(npImageLenaLowPixel, cmap='gray')
    im3 = ax3.imshow(npImageCameraman, cmap='gray')
    im4 = ax4.imshow(npImageCameramanLowPixel, cmap='gray')
    im5 = ax5.imshow(npImageHouse, cmap='gray')
    im6 = ax6.imshow(npImageHouseLowPixel, cmap='gray')
    plt.show()
```

O resultado será apresentado na Figura 2.

Figura 2: Aplicação da redução de resolução

Fonte: Elaborado pelo autor.

Podemos concluir que a redução da resolução só se torna perceptível ao darmos zoom na imagem. Para obter uma redução mais notória sem que haja a necessidade de dar zoom, é necessário reduzirmos ainda mais a resolução.

2.1.3. Incluir 4 quadrados brancos 10x10 pixels em cada canto das imagens

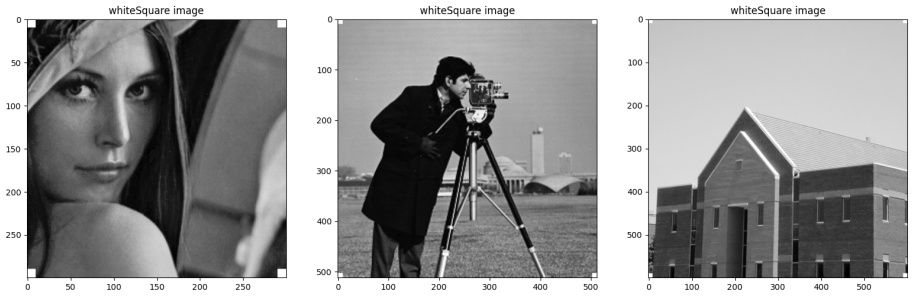
O seguinte código foi utilizado:

```
def whiteSquare():
    # Abrir as images
    imageLena = Image.open('images/lena.jpg')
    imageCameraman = Image.open('images/cameraman.tif')
    imageHouse = Image.open('images/house.tif')

    # Converter as imagens para numpy array
    # e tranforma-las em negativa
    # Lena
    npImageLena = np.array(imageLena)
    npImageLena[0:10,0:10] = 255
    npImageLena[0:10,290:301] = 255
    npImageLena[290:301,0:10] = 255
    npImageLena[290:301,290:301] = 255

    # Cameraman
```

```
npImageCameraman = np.array(imageCameraman)
npImageCameraman[0:10,0:10] = 255
npImageCameraman[0:10,503:513] = 255
npImageCameraman[503:513,0:10] = 255
npImageCameraman[503:513,503:513] = 255

# House
npImageHouse = np.array(imageHouse)
npImageHouse[0:10,0:10] = 255
npImageHouse[0:10,590:601] = 255
npImageHouse[590:601,0:10] = 255
npImageHouse[590:601,590:601] = 255

# Plotar imagens
fig3 = plt.figure()
ax1 = plt.subplot(1,3,1)
ax3 = plt.subplot(1,3,2)
ax5 = plt.subplot(1,3,3)

ax1.title.set_text('whiteSquare image')
ax3.title.set_text('whiteSquare image')
ax5.title.set_text('whiteSquare image')

im1 = ax1.imshow(npImageLena, cmap='gray')
im3 = ax3.imshow(npImageCameraman, cmap='gray')
im5 = ax5.imshow(npImageHouse, cmap='gray')
plt.show()
```

A Figura 3 demonstra o resultado obtido.

Figura 3: Quadrados brancos nas extremidades das imagens.



Fonte: Elaborado pelo autor.

2.1.4. Incluir 1 quadrado preto 15X15 no centro das imagens

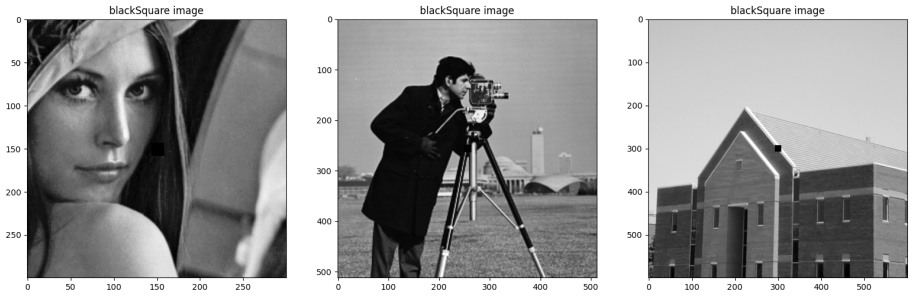
A seguir será apresentado o código utilizado para a implementação:

```
def blackSquare():
    # Abrir as images
    imageLena = Image.open('images/lena.jpg')
    imageCameraman = Image.open('images/cameraman.tif')
    imageHouse = Image.open('images/house.tif')

    # Converter as imagens para numpy array
    # e tranforma-las em negativa
    # Lena
    npImageLena = np.array(imageLena)
    npImageLena[144:159,144:159] = 0

    # Cameraman
    npImageCameraman = np.array(imageCameraman)
    npImageCameraman[249:264,249:264] = 0

    # House
    npImageHouse = np.array(imageHouse)
    npImageHouse[293:308,293:308] = 0
```

```
# Plotar imagens
fig4 = plt.figure()
ax1 = plt.subplot(1,3,1)
ax3 = plt.subplot(1,3,2)
ax5 = plt.subplot(1,3,3)

ax1.title.set_text('blackSquare image')
ax3.title.set_text('blackSquare image')
ax5.title.set_text('blackSquare image')

im1 = ax1.imshow(npImageLena, cmap='gray')
im3 = ax3.imshow(npImageCameraman, cmap='gray')
im5 = ax5.imshow(npImageHouse, cmap='gray')
plt.show()
```

O resultado será apresentado através da Figura 4.

Figura 4: Quadrados pretos no centro das imagens.



Fonte: Elaborado pelo autor.

Vale ressaltar que há quadrados pretos na imagem da Lena e do Cameraman, porém pelo fato de serem imagens com bastantes tons mais escuros, o quadrado está praticamente invisível.

2.2. Operação por vizinhança

Os exercícios envolvendo operações por vizinhança foram elaborados com 4 bibliotecas do python, sendo elas: Numpy, Pillow, OpenCV e Scipy. Para efeitos de

comparação, foi utilizado apenas a imagem da Lena.

2.2.1. Calcular o filtro da média

O filtro utilizando a média foi implementado da seguinte forma:

Numpy:

```
def numPyFilter():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Definir o filtro de média
    kernel = np.array([[1, 1, 1],
                       [1, 1, 1],
                       [1, 1, 1],]) / 9

    #Aplicar o filtro usando convolução 2D do numpy
    filteredImageNumpy = signal.convolve2d(npImage, kernel,
boundary='symm', mode='valid')

    # Converter imagem de volta para o formato PIL e salvar
    filteredImagePIL =
Image.fromarray(filteredImageNumpy.astype(np.uint8))
    filteredImagePIL.save("filteredImageNumpy.jpg")
```

Pillow:

```
def pillowFilter():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Aplicar o filtro de média usando o método filter do
pillow
    filteredImagePillow =
image.filter(ImageFilter.BoxBlur(3))

    #Salvar a imagem filtrada
    filteredImagePillow.save("filteredImagePillow.jpg")
```

OpenCV:

```
def opencvFilter():

    # Carregar a imagem
    image = cv2.imread("images/lena.jpg")

    # Aplicar o filtro de média usando a função filter2D do
opencv
    kernel = np.ones((3, 3), np.float32) / 9
    filteredImageOpencv = cv2.filter2D(image, -1, kernel)

    # Salvar a imagem filtrada
    cv2.imwrite("filteredImageOpencv.jpg",
filteredImageOpencv)
```

Scipy:

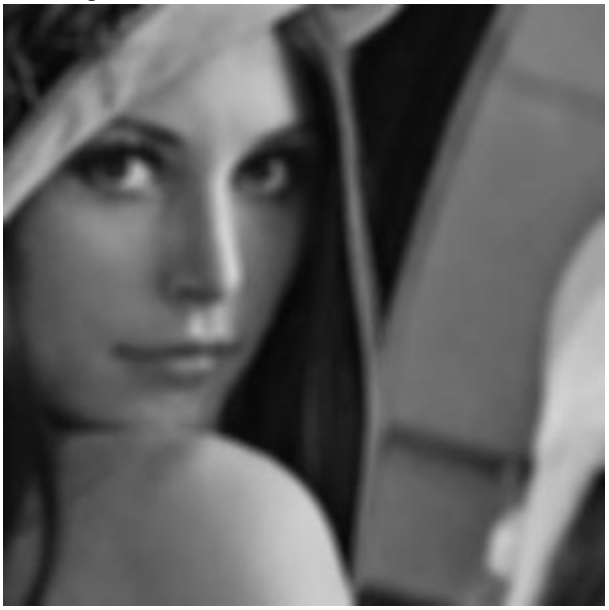
```
def scipyFilter():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Definir o filtro de média
    kernel = np.array([[1, 1, 1],
                       [1, 1, 1],
                       [1, 1, 1],]) / 9

    # Aplicar o filtro usando convolve2d do scipy
    filteredImageScipy = signal.convolve2d(npImage, kernel,
mode='valid', boundary='wrap')

    # Converter a imagem de volta para o formato PIL e
salvar
    filteredImagePIL =
Image.fromarray(filteredImageScipy.astype(np.uint8))
    filteredImagePIL.save("filteredImageScipy.jpg")
```

Os resultados dos seguintes códigos utilizando Numpy, Pillow, OpenCV e Scipy consecutivamente foram:

Figura 5: Filtro utilizando a média no Numpy.

Fonte: Elaborado pelo autor.

Figura 6: Filtro utilizando a média no Pillow.



Fonte: Elaborado pelo autor.

Figura 7: Filtro utilizando a média no OpenCV.

Fonte: Elaborado pelo autor.

Figura 8: Filtro utilizando a média no Scipy.

Fonte: Elaborado pelo autor.

Das imagens com filtro geradas pelos códigos, a que mais percebemos perda de qualidade foi a utilizando Pillow, para as demais foi possível manter um bom nível de qualidade.

2.2.2. Calcular o filtro da mediana

Assim como na média, para a mediana foi implementado nas 4 bibliotecas apresentadas. A seguir será possível ver os códigos:

Numpy:

```
def numpyMedianFilter():
```

```
    # Carregar a imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Definir o filtro de mediana
    kernel = 3

    #Aplicar o filtro usando convolução 2D do numpy
    filteredImageNumpy = signal.medfilt2d(npImage, kernel)

    # Converter imagem de volta para o formato PIL e salvar
    filteredImagePIL =
Image.fromarray(filteredImageNumpy.astype(np.uint8))
    filteredImagePIL.save("medianFilteredImageNumpy.jpg")
```

Pillow:

```
def pillowMedianFilter():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Aplicar o filtro de mediana usando o método filter do
pillow
    filteredImagePillow =
image.filter(ImageFilter.MedianFilter(3))

    #Salvar a imagem filtrada

filteredImagePillow.save("medianFilteredImagePillow.jpg")
```

OpenCV:

```
def opencvMedianFilter():

    # Carregar a imagem
    image = cv2.imread("images/lena.jpg")

    # Aplicar o filtro de mediana usando a função filter2D
do opencv
```

```
    filteredImageOpencv = cv2.medianBlur(image, 3)

    # Salvar a imagem filtrada
    cv2.imwrite("medianFilteredImageOpencv.jpg",
filteredImageOpencv)
```

Scipy:

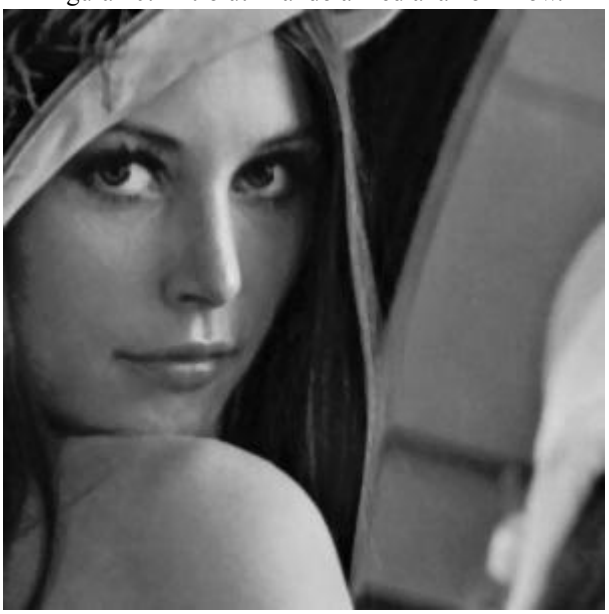
```
def scipyMedianFilter():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Definir o filtro de mediana
    kernel = 3

    # Aplicar o filtro usando convolve2d do scipy
    filteredImageScipy = ndimage.median_filter(npImage,
kernel)

    # Converter a imagem de volta para o formato PIL e
salvar
    filteredImagePIL =
Image.fromarray(filteredImageScipy.astype(np.uint8))
    filteredImagePIL.save("medianFilteredImageScipy.jpg")
```

Resultando nas seguintes imagens:

Figura 9: Filtro utilizando a mediana no Numpy.

Fonte: Elaborado pelo autor.

Figura 10: Filtro utilizando a mediana no Pillow.

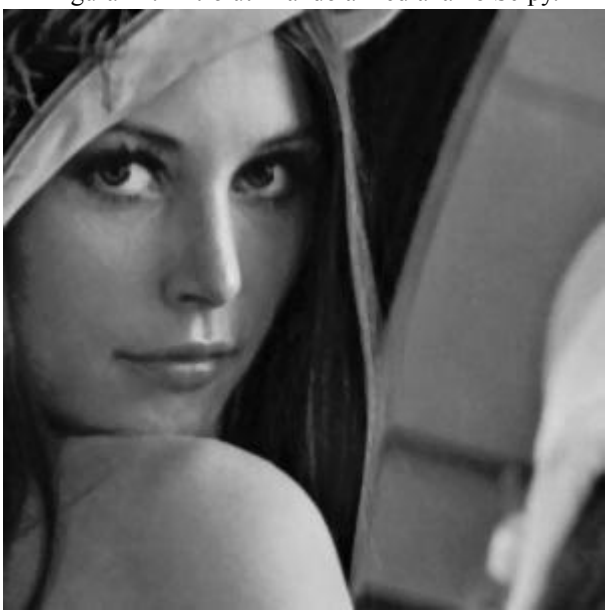


Fonte: Elaborado pelo autor.

Figura 11: Filtro utilizando a mediana no OpenCV.

Fonte: Elaborado pelo autor.

Figura 12: Filtro utilizando a mediana no Scipy.



Fonte: Elaborado pelo autor.

Utilizando a mediana foi possível notar que em todas as bibliotecas conseguiram manter um nível bom de qualidade da imagem.

2.3. Transformações geométricas

Os exercícios a seguir foram realizados utilizando as bibliotecas: Numpy, Pillow, OpenCV e Scipy.

2.3.1. Escala: Redução em 1.5x e aumentar em 2.5x

A seguir será apresentado os códigos para cada biblioteca:
Numpy:

```
def escalaNumpy():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Redução da escala em 1.5x
    reducedNp =
np.array(image.resize((int(image.width/1.5),
int(image.height/1.5))))

    # Aumento da escala em 2.5x
    enlargedNp =
np.array(image.resize((int(image.width*2.5),
int(image.height*2.5))))

    # Salvar o resultado das imagens
    Image.fromarray(reducedNp).save("reducedLenaNp.jpg")
    Image.fromarray(enlargedNp).save("enlargedLenaNp.jpg")
```

Pillow:

```
def escalaPillow():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Redução da escala em 1.5x
    reducedPillow = image.resize((int(image.width/1.5),
int(image.height/1.5)))

    # Aumento da escala em 2.5x
    enlargedPillow = image.resize((int(image.width*2.5),
int(image.height*2.5)))

    # Salvar o resultado das imagens
    reducedPillow.save("reducedLenaPillow.jpg")
    enlargedPillow.save("enlargedLenaPillow.jpg")
```

OpenCV:

```
def escalaOpencv():

    # Carregar imagem com opencv
    image = cv2.imread("images/lena.jpg")

    # Redução da escala em 1.5x
    reducedCv = cv2.resize(image, (int(image.shape[1]/1.5),
int(image.shape[0]/1.5)))

    # Aumento da escala em 2.5x
    enlargedCv = cv2.resize(image,
(int(image.shape[1]*2.5), int(image.shape[0]*2.5)))

    # Salvar o resultado das imagens
    cv2.imwrite("reducedLenaOpencv.jpg", reducedCv)
    cv2.imwrite("enlargedLenaOpencv.jpg", enlargedCv)
```

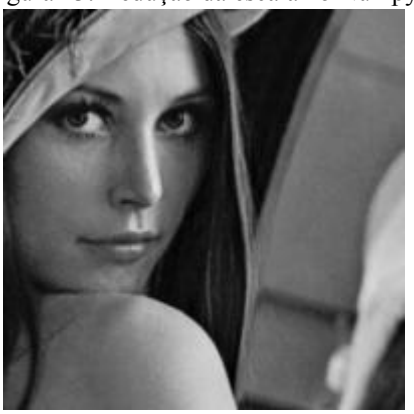
Scipy:

```
def escalaSpicy():

    # Carregar imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Redução da escala em 1.5x
    reducedSpicy = ndimage.zoom(npImage, (1/1.5, 1/1.5),
order=3)

    # Aumento da escala em 2.5x
    enlargedSpicy = ndimage.zoom(npImage, (2.5, 2.5),
order=3)

    # Salvar o resultado das imagens

Image.fromarray(reducedSpicy.astype(np.uint8)).save("reduce
dLenaSpicy.jpg")

Image.fromarray(enlargedSpicy.astype(np.uint8)).save("enlar
```

```
gedLenaSpicy.jpg")
```

Foram obtidos os seguintes resultados:

Figura 13: Redução da escala no Numpy.

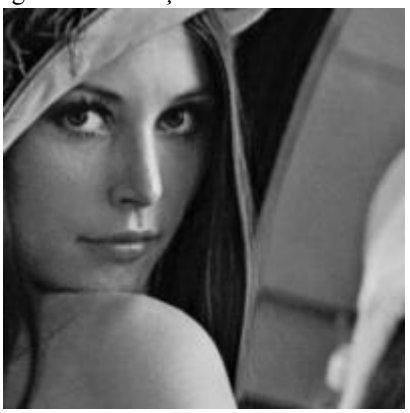


Fonte: Elaborado pelo autor.

Figura 14: Ampliação da escala no Numpy.

Fonte: Elaborado pelo autor.

Figura 15: Redução da escala no Pillow.

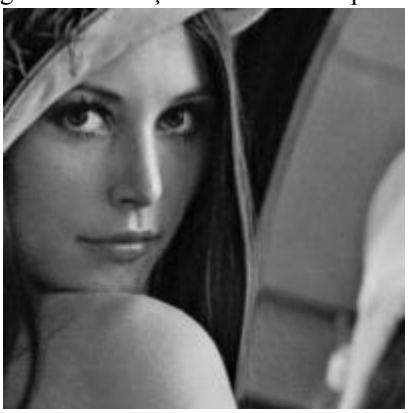


Fonte: Elaborado pelo autor.

Figura 16: Ampliação da escala no Pillow.

Fonte: Elaborado pelo autor.

Figura 17: Redução da escala no OpenCV.



Fonte: Elaborado pelo autor.

Figura 18:Ampliação da escala no OpenCV.

Fonte: Elaborado pelo autor.

Figura 19: Redução da escala no Scipy.

Fonte: Elaborado pelo autor.

Figura 20: Ampliação da escala no Scipy.

Fonte: Elaborado pelo autor.

2.3.2. Rotação em 45º, 90º e 100º

Os seguintes códigos foram utilizados para a elaboração do exercício:
Numpy:

```
def rotacaoNumpy():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Converter a imagem para um array numpy
    npImage = np.array(image)

    # Rotação em 45°
```

```
    rotated_45_np = np.rot90(npImage, k=1, axes=(0, 1))

    # Rotação em 90°
    rotated_90_np = np.rot90(npImage, k=2, axes=(0, 1))

    # Rotação em 100° (aproximação de 90°)
    rotated_100_np = np.rot90(npImage, k=1, axes=(0, 1))

    # Salvar as imagens resultantes

Image.fromarray(rotated_45_np).save("rotated_45_np.jpg")

Image.fromarray(rotated_90_np).save("rotated_90_np.jpg")

Image.fromarray(rotated_100_np).save("rotated_100_np.jpg")
```

Pillow:

```
def rotacaoPillow():

    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Rotação em 45° usando Pillow
    rotated_45_pillow = image.rotate(45, expand=True)

    # Rotação em 90° usando Pillow
    rotated_90_pillow = image.rotate(90, expand=True)

    # Rotação em 100° usando Pillow
    rotated_100_pillow = image.rotate(100, expand=True)

    # Salvar as imagens resultantes
    rotated_45_pillow.save("rotated_45_pillow.jpg")
    rotated_90_pillow.save("rotated_90_pillow.jpg")
    rotated_100_pillow.save("rotated_100_pillow.jpg")
```

OpenCV:

```
def rotacaoOpencv():
```

```
    # Carregar a imagem com OpenCV
    image = cv2.imread("images/lena.jpg")

    # Obter a altura e largura da imagem
    height, width = image.shape[:2]

    # Definir a matriz de rotação para 45°
    rotation_matrix_45 = cv2.getRotationMatrix2D((width/2,
height/2), 45, 1)

    # Definir a matriz de rotação para 90°
    rotation_matrix_90 = cv2.getRotationMatrix2D((width/2,
height/2), 90, 1)

    # Definir a matriz de rotação para 100°
    rotation_matrix_100 = cv2.getRotationMatrix2D((width/2,
height/2), 100, 1)

    # Rotação em 45° usando OpenCV
    rotated_45_cv2 = cv2.warpAffine(image,
rotation_matrix_45, (width, height))

    # Rotação em 90° usando OpenCV
    rotated_90_cv2 = cv2.warpAffine(image,
rotation_matrix_90, (width, height))

    # Rotação em 100° usando OpenCV
    rotated_100_cv2 = cv2.warpAffine(image,
rotation_matrix_100, (width, height))

    # Salvar as imagens resultantes com OpenCV
    cv2.imwrite("rotated_45_cv2.jpg", rotated_45_cv2)
    cv2.imwrite("rotated_90_cv2.jpg", rotated_90_cv2)
    cv2.imwrite("rotated_100_cv2.jpg", rotated_100_cv2)
```

Scipy:

```
def rotacaoScipy():
```

```
    # Carregar a imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Rotação em 45° usando Scipy
    rotated_45_scipy = ndimage.rotate(npImage, 45,
reshape=True)

    # Rotação em 90° usando Scipy
    rotated_90_scipy = ndimage.rotate(npImage, 90,
reshape=True)

    # Rotação em 100° usando Scipy
    rotated_100_scipy = ndimage.rotate(npImage, 100,
reshape=True)

    # Salvar as imagens resultantes

Image.fromarray(rotated_45_scipy.astype(np.uint8)).save("ro
tated_45_scipy.jpg")

Image.fromarray(rotated_90_scipy.astype(np.uint8)).save("ro
tated_90_scipy.jpg")

Image.fromarray(rotated_100_scipy.astype(np.uint8)).save("r
otated_100_scipy.jpg")
```

As figuras a seguir mostram os resultados obtidos através dos códigos implementados:

Figura 21: Rotação de 45 graus no Numpy.

Fonte: Elaborado pelo autor.

Figura 22: Rotação de 90 graus no Numpy.

Fonte: Elaborado pelo autor.

Figura 23: Rotação de 100 graus no Numpy.

Fonte: Elaborado pelo autor.

Figura 24: Rotação de 45 graus no Pillow.

Fonte: Elaborado pelo autor.

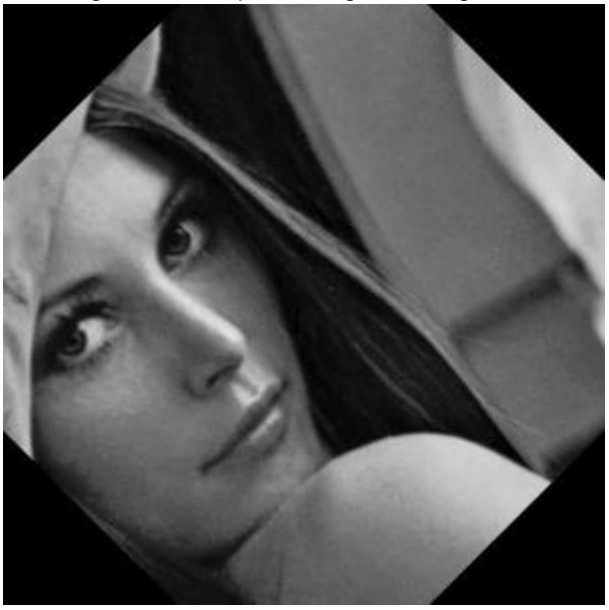
Figura 25: Rotação de 90 graus no Pillow.

Fonte: Elaborado pelo autor.

Figura 26: Rotação de 100 graus no Pillow.

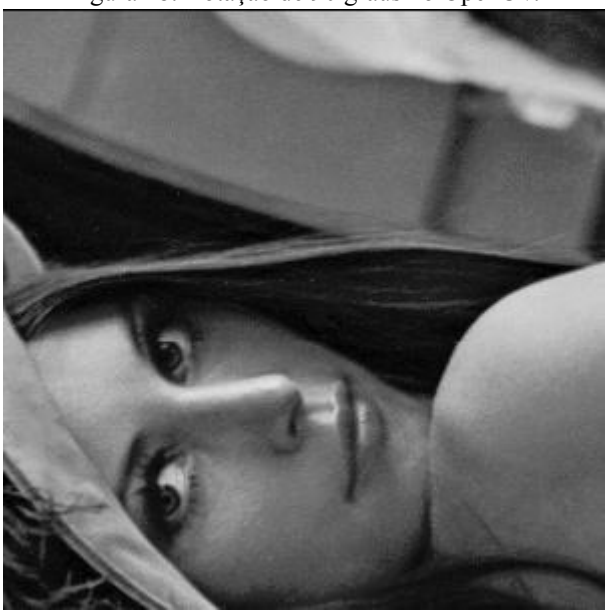
Fonte: Elaborado pelo autor.

Figura 27: Rotação de 45 graus no OpenCV.



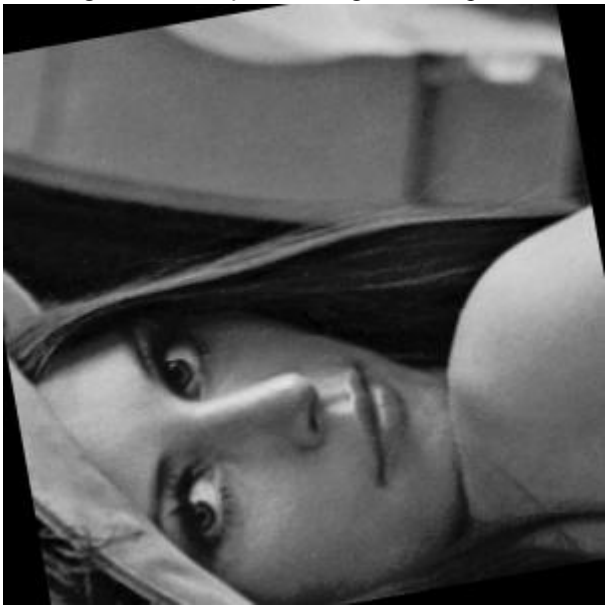
Fonte: Elaborado pelo autor.

Figura 28: Rotação de 90 graus no OpenCV.



Fonte: Elaborado pelo autor.

Figura 29: Rotação de 100 graus no OpenCV.



Fonte: Elaborado pelo autor.

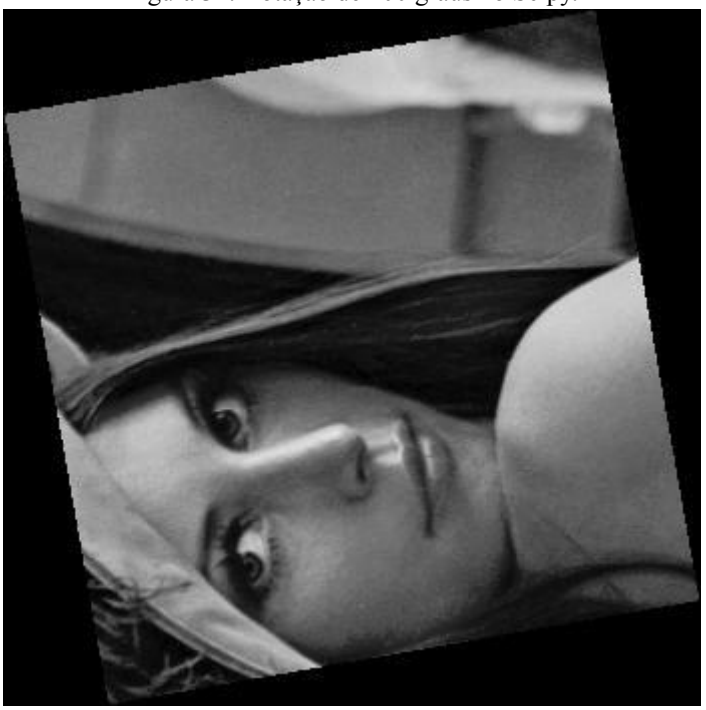
Figura 30: Rotação de 45 graus no Scipy.

Fonte: Elaborado pelo autor.

Figura 31: Rotação de 90 graus no Scipy.

Fonte: Elaborado pelo autor.

Figura 32: Rotação de 100 graus no Scipy.



Fonte: Elaborado pelo autor.

2.3.3. Translação utilizar os parâmetros que quiser nas coordenadas x e y

Considerando os dois últimos exercícios, ambos sendo para a translação da imagem, foi realizado em um só código.

Numpy:

```
def translacaoNumpy():
    # Carregar a imagem
```

```
    image = Image.open("images/lena.jpg")

    # Converter a imagem para um array numpy
    npImage = np.array(image)

    # Parâmetros de translação
    translation_x = 35
    translation_y = 45

    # Translação usando Numpy
        translated_np  =  np.roll(npImage,  (translation_y,
translation_x), axis=(0, 1))

    # Salvar a imagem resultante

Image.fromarray(translated_np).save("translated_np.jpg")
```

Pillow:

```
def translacaoPillow():
    # Carregar a imagem
    image = Image.open("images/lena.jpg")

    # Parâmetros de translação
    translation_x = 35
    translation_y = 45

    # Translação usando Pillow
         translated_pillow  =  image.transform(image.size,
Image.AFFINE, (1, 0, translation_x, 0, 1, translation_y))

    # Salvar a imagem resultante
    translated_pillow.save("translated_pillow.jpg")
```

OpenCV:

```
def translacaoOpencv():
    # Carregar a imagem com OpenCV
    image = cv2.imread("images/lena.jpg")
```

```
    # Parâmetros de translação
    translation_x = 35
    translation_y = 45

    # Definir a matriz de translação
    translation_matrix = np.float32([[1, 0, translation_x],
[0, 1, translation_y]])

    # Translação usando OpenCV
                translated_cv2    =    cv2.warpAffine(image,
translation_matrix, (image.shape[1], image.shape[0]))

    # Salvar a imagem resultante com OpenCV
    cv2.imwrite("translated_cv2.jpg", translated_cv2)
```

Scipy:

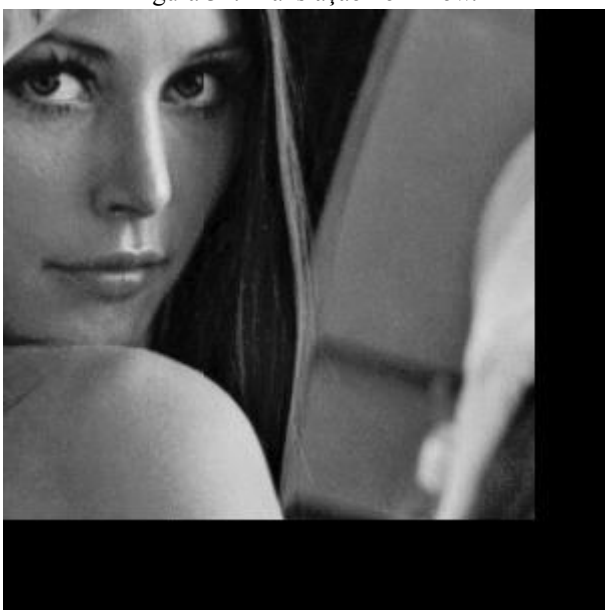
```
def translacaoScipy():
    # Carregar a imagem
    image = Image.open("images/lena.jpg")
    npImage = np.array(image)

    # Parâmetros de translação
    translation_x = 35
    translation_y = 45

    # Translação usando Scipy
              translated_scipy    =    ndimage.shift(npImage,
(translation_y, translation_x))

    # Salvar a imagem resultante

Image.fromarray(translated_scipy.astype(np.uint8)).save("tr
anslated_scipy.jpg")
```

Os resultados obtidos serão apresentados nas imagens a seguir:

Figura 33: Translação no Numpy.

Fonte: Elaborado pelo autor.

Figura 34: Translação no Pillow.



Fonte: Elaborado pelo autor.

Figura 35: Translação no OpenCV.

Fonte: Elaborado pelo autor.

Figura 35: Translação no Scipy.

Fonte: Elaborado pelo autor.