COM 3105

Processamento Digital de Imagens

LN03-02 - 2023

Frameworks e Bibliotecas para o Processamento de Imagens

Numpy

Prof. Eng. Marcos Cordeiro d'Ornellas, Ph.D.

1. Numpy

O NumPy é uma poderosa biblioteca Python que é usada principalmente para realizar cálculos em Arrays Multidimensionais. O NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos. Esses tipos de cálculos numéricos são amplamente utilizados em tarefas como o processamento de imagens e Computação Gráfica.

Imagens no computador são representadas como Arrays Multidimensionais de números. NumPy torna-se a escolha mais natural para o mesmo. O NumPy fornece algumas excelentes funções de biblioteca para **rápida manipulação de imagens**. Alguns exemplos são o espelhamento de uma imagem, a rotação de uma imagem por um determinado ângulo, entre outras funcionalidades.

NumPy é o pacote fundamental para computação científica em Python. É uma biblioteca Python que fornece um objeto de matriz multidimensional, vários objetos derivados (como matrizes e matrizes mascaradas) e uma variedade de rotinas para operações rápidas em matrizes, incluindo matemática, lógica, manipulação de forma, classificação, seleção, E/S , transformadas discretas de Fourier, álgebra linear básica, operações estatísticas básicas, simulação aleatória e muito mais.

No núcleo da biblioteca NumPy, está o objeto *ndarray*. Ele encapsula matrizes n-dimensionais de tipos de dados homogêneos, com muitas operações sendo realizadas em código compilado para desempenho.

1.1 Obtendo e Instalando o Numpy

O único pré-requisito para instalar o NumPy é o próprio Python. Se você ainda não tem o Python e deseja a maneira mais simples de começar, recomendamos que use a Distribuição Anaconda - ela inclui Python, NumPy e muitos outros pacotes comumente usados para computação científica e ciência de dados.

O NumPy pode ser instalado com conda, com pip, com um gerenciador de pacotes no macOS e Linux ou a partir da fonte. Para obter instruções mais detalhadas, consulte nosso guia de instalação do Python e NumPy abaixo.

Para obter mais ajuda para instalar o Pillow manualmente, consulte: https://numpy.org/install/ (https://numpy.org/install/)

In [1]:

```
try:
    import numpy
    print(numpy.__version__)
except ImportError:
    print(numpy is not installed)
```

executed in 485ms, finished 18:30:30 2023-03-26

```
1.23.0
```

1.3 Operações Básicas

1.3.1 Carregando as Bibliotecas e Funções

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

executed in 955ms, finished 18:30:31 2023-03-26

1.3.2 Abrindo e Visualizando uma Imagem

Matplotlib é uma biblioteca em Python e é uma extensão numérica - matemática da biblioteca NumPy. Pyplot é uma interface baseada em estado para um módulo Matplotlib que fornece uma interface semelhante ao MATLAB.

Syntax: matplotlib.pyplot.imread(fname, format=None)
Parameters:

- fname or file-like: The image file to read: a filename, a URL or a file-like object opened in read-binary mode.
- format, optional: The image file format assumed for reading the data. If not given, the format is deduced from the filename. If nothing can be deduced, PNG is tried.

Returns: image data

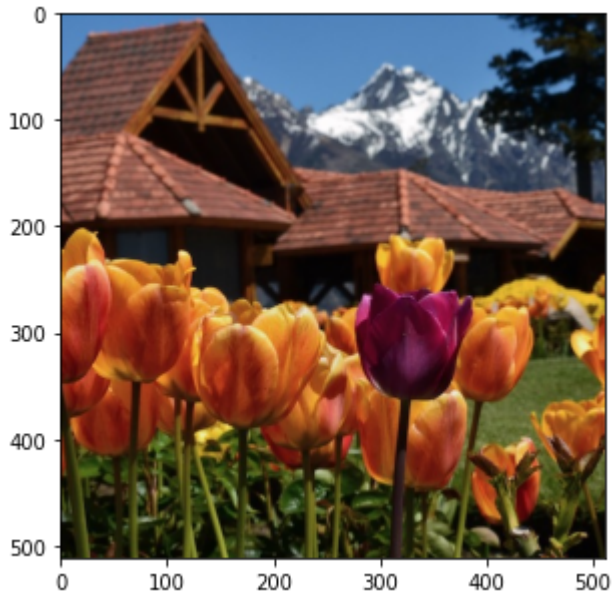
In [3]:

```
img = mpimg.imread('image_folder/bariloche512.jpg')

# show the image
# The show() function will display the image using your operating systems de
# img.show()

# imshow() from matplotlib
plt.figure(figsize=(5,5))
plt.imshow(img)
```

executed in 343ms, finished 18:30:32 2023-03-26

Out[3]:

```
<matplotlib.image.AxesImage at 0x7fa4c1c93a30>
```



1.3.3 Atributos da Imagem

A função *numpy.asarray()* é usada quando queremos converter a entrada em um array. A entrada pode ser listas, listas de tuplas, tuplas, tuplas de tuplas, tuplas de listas e ndarrays.

Syntax : numpy.asarray(arr, dtype=None, order=None)
Parameters:

- arr: [array_like] Input data, in any form that can be converted to an array. This includes lists, lists of tuples, tuples, tuples of tuples, tuples of lists and ndarrays.
- dtype: [data-type, optional] By default, the data-type is inferred from the input data. order : Whether to use row-major (C-style) or column-major (Fortran-style) memory representation. Defaults to ‘C’.

Return: [ndarray] Array interpretation of arr. No copy is performed if the input is already ndarray with matching dtype and order. If arr is a subclass of ndarray, a base class ndarray is returned.

In [4]:

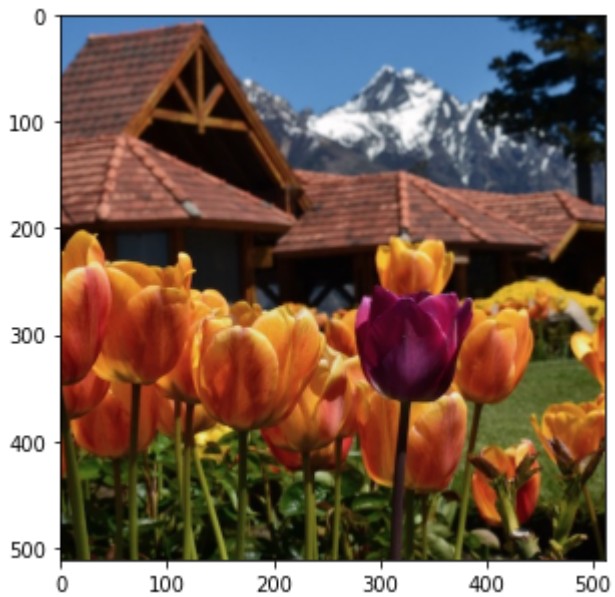
```
from numpy import asarray

img_array = asarray(img)

print('# of dims: ',img_array.ndim)      # dimension of an image
print('Img shape: ',img_array.shape)     # shape of an image
print('Dtype: ',img_array.dtype)
print(img_array[20, 20])                 # pixel value at [R, G, B]
print(img_array[:, :, 2].min())          # min pixel value at channel B

# imshow() from matplotlib
plt.figure(figsize=(5,5))
plt.imshow(img_array)
```

executed in 166ms, finished 18:30:32 2023-03-26

```
# of dims:  3
Img shape:  (512, 512, 3)
Dtype:  uint8
[ 81 137 196]
0
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x7fa490937190>
```



1.3.4 Salvando uma Imagem

A função *imsave()* é semelhante a *imshow()*, mas em vez de exibir uma imagem, ela a salva em um arquivo. No arquivo, cada elemento da matriz numpy descreve um pixel. A função *imsave()* não possui o argumento de interpolação.

Syntax: matplotlib.pyplot.imsave(*args, **kwargs)¶
Parameters:

- fname: Path string to a filename, or a Python file-like object. If format is None and fname is a string, the output format is deduced from the extension of the filename.

- arr: An MxN (luminance), MxNx3 (RGB) or MxNx4 (RGBA) array.
- vmin, vmax: vmin and vmax set the color scaling for the image by fixing the values that map to the colormap color limits. If either vmin or vmax is None, that limit is determined from the arr min/max value.
- cmap: For example, cm.viridis. If None, defaults to the image.cmap rcParam.
- format: One of the file extensions supported by the active backend. Most backends support png, pdf, ps, eps and svg.
- origin: Indicates whether the (0, 0) index of the array is in the upper left or lower left corner of the axes. Defaults to the image.origin rcParam.
- dpi: The DPI to store in the metadata of the file. This does not affect the resolution of the output image.

Returns: None

In [5]:

```
# plt.imsave('image_folder/saved_plt.jpg', image_array, cmap = 'inferno')
```

executed in 2ms, finished 18:30:32 2023-03-26

1.3.5 Convertendo uma Imagem

Para converter imagens coloridas RGB ou BGR em imagens em níveis de cinza, usamos frequentemente as seguintes fórmulas de conversão:

- Luminance (OpenGL) = 0.3086 * Red + 0.6094 * Green + 0.0820 * Blue
- Luminance (PAL/NTSC) = 0.299 * Red + 0.587 * Green + 0.114 * Blue
- Luminance (HDTV) = 0.2126 * Red + 0.7152 * G + 0.0722 * Blue

Observe que a intensidade da luminância é uma soma dos pesos diferentes de cada componente de cor. Se usarmos o mesmo peso, por exemplo, (R + G + B)/3, então vermelho puro, verde puro e azul puro resultam no mesmo nível de escala de cinza.

> Outra razão para usar pesos diferentes é que o olho humano é mais sensível aos componentes verdes e vermelhos do que ao canal azul

Utilizamos aqui o produto escalar entre duas matrizes definido na numpy com a função *numpy.dot()*.

Para maiores informações sobre quantização e dithering usando numpy acesse
https://sites.google.com/view/ananyamukherjeehome/image-processing/quantization-dithering
(https://sites.google.com/view/ananyamukherjeehome/image-processing/quantization-dithering)

Syntax: numpy.dot(a, b, out=None)
Parameters:

- a: First argument.
- b: Second argument.

out: This must have the exact kind that would be returned if it was not used. In particular, it must have the right type, must be C-contiguous, and its dtype must be the dtype that would be returned for dot(a,b). This is a performance feature. Therefore, if these conditions are not met, an exception is raised, instead of attempting to be flexible.

Returns: ndarray

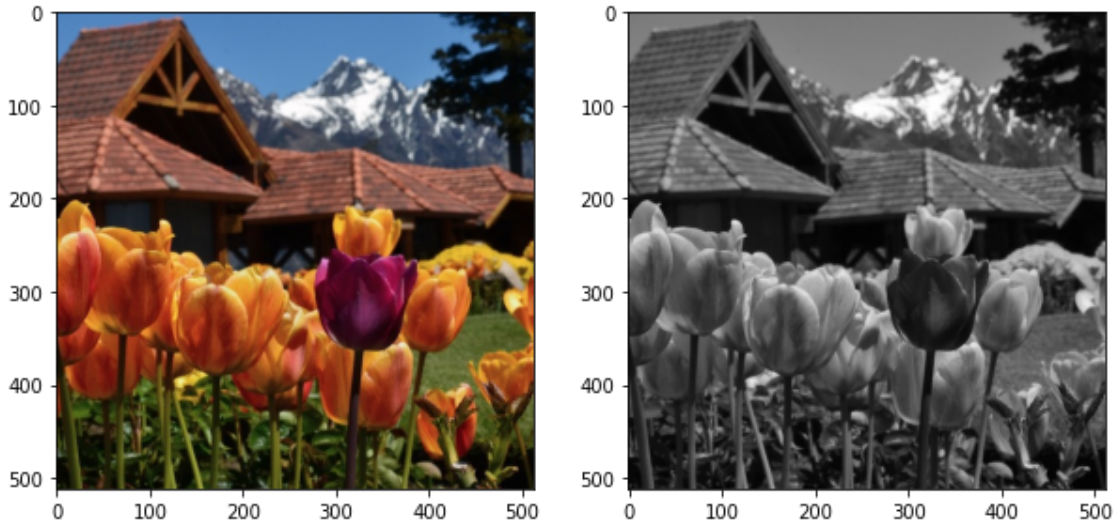
In [6]:

```
# Luminance  = 0.299 R + 0.587 G + 0.114 B
lum = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])
lum = lum(img_array)

plt.figure(figsize=(10,10))

plt.subplot(121)
plt.imshow(img_array)

plt.subplot(122)
plt.imshow(lum,cmap='gray', vmin = 0, vmax = 255)

```

executed in 315ms, finished 18:30:32 2023-03-26

Out[6]:

```
<matplotlib.image.AxesImage at 0x7fa49099ddc0>
```



O pontilhamento (dithering) no processamento de imagem é uma técnica usada para simular cores ou sombreamento.

O conceito básico por trás do pontilhamento é adicionar ruído, ou pixels adicionais, a um arquivo digital. Em gráficos, o pontilhamento adiciona padrões aleatórios de pixels para melhorar a qualidade da imagem, evitando formação de faixas.

> Em seus primeiros usos em jornais, revistas em quadrinhos e outras mídias impressas, o pontilhamento seria aplicado a imagens para criar níveis de tons de cinza simulados por posicionamento estratégico de pontos pretos. Usar o processo de pontilhamento forneceria uma imagem suave com tons de cinza, embora as impressoras suportassem apenas tinta preta.

O algoritmo de pontilhamento Floyd-Steinberg foi publicado por Robert Floyd e Louis Steinberg em 1976. O pontilhamento é um método de difusão de pixels para evitar bordas ásperas ou faixas onde as cores em uma imagem contrastam umas com as outras. Seu uso óbvio é na conversão de imagens de alta profundidade em uma paleta de cores limitada (256 ou menos). Existem muitos algoritmos de dithering e Floyd-Steinberg é um dos mais conhecidos.

Apresentamos o código a seguir que implementa o dithering (pontilhamento) usando numpy.

https://pythonawesome.com/an-implementation-of-ordered-dithering-algorithm-in-python/
(https://pythonawesome.com/an-implementation-of-ordered-dithering-algorithm-in-python/)

In [7]:

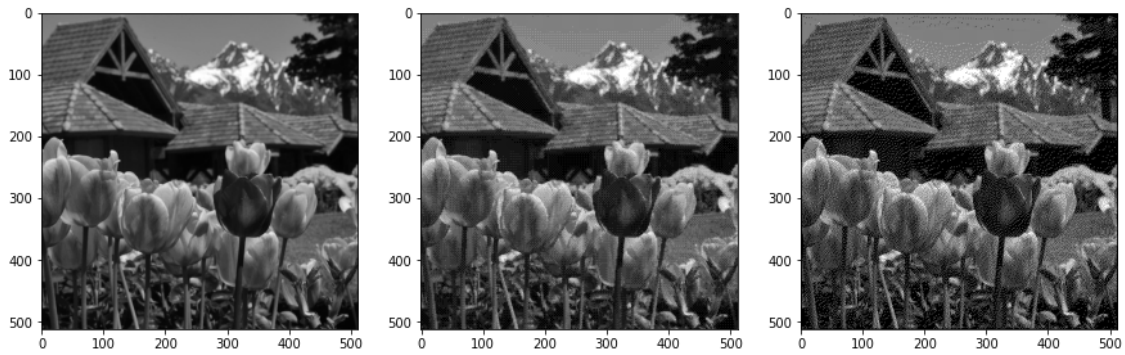
```
import numpy as np

def ordered_dithering(I: np.ndarray, N):
    def _order_matrix(n):
        if n == 2:
            return np.array([[0, 2],[3, 1]])/4
        else:
            M0 = _order_matrix(n//2)*(2*n//2)*4
            M2 = np.zeros( (n,n) )
            M2[:n//2] = np.column_stack([M0, M0+2])
            M2[n//2:] = np.column_stack([M0+3, M0+1])
            return M2/(n**2)
    tile_ = np.ceil(np.array(I.shape[:2])/N ).astype(np.int32)
    D = np.tile(_order_matrix(N), tile_ )
    D = D[:I.shape[0],:I.shape[1]]
    e = I - I.astype(np.uint8)
    P = np.copy(I)
    P[np.where(e > D)] = np.ceil(I[np.where(e > D)])
    P[np.where(e <= D)] = np.floor(I[np.where(e <= D)])
    return np.round(P).astype(np.uint8)

def floyd_steinberg_dithering(I: np.ndarray):
    P = np.copy(I)
    for i in range(P.shape[0]-1):
        for j in range(P.shape[1]-1):
            e = np.remainder(P[i,j], 1)
            P[i,j+1] += e * 7/16.0
            P[i+1,j+1] += e * 1/16.0
            P[i+1,j] += e * 5/16.0
            P[i+1,j-1] += e * 3/16.0
    return np.floor(P).astype(np.uint8)

def ordered_bits_compress(img: np.ndarray, bits):
    I = img*( (2**bits-1)/255)
    I_new = ordered_dithering(I, 2**bits)
    color_palette = np.linspace(0, 255, 2**bits, dtype=np.uint8)
    I_new = color_palette[I_new]
    return I_new

def floyd_bits_compress(img: np.ndarray, bits):
    I = img*( (2**bits-1)/255)
    I_new = floyd_steinberg_dithering(I)
    color_palette = np.linspace(0, 255, 2**bits, dtype=np.uint8)
    I_new = color_palette[I_new]
    return I_new

I = lum
I1 = ordered_bits_compress(I, 2)
I2 = floyd_bits_compress(I, 1)

fig = plt.figure()
fig.tight_layout()

plt.figure(figsize=(15,15))

plt.subplot(131)
plt.imshow(I,cmap='gray', vmin = 0, vmax = 255)

plt.subplot(132)
```

```
60 plt.imshow(I1,cmap='gray', vmin = 0, vmax = 255)
61
62 plt.subplot(133)
63 plt.imshow(I2,cmap='gray', vmin = 0, vmax = 255)
```

Out[7]:

executed in 1.28s, finished 18:30:33 2023-03-26

```
<matplotlib.image.AxesImage at 0x7fa4a0c78280>

<Figure size 432x288 with 0 Axes>
```



Numpy suporta a álgebra de imagens de tal forma que as imagens são identificadas como objetos, os quais podem ser operados através de equações.

In [8]:

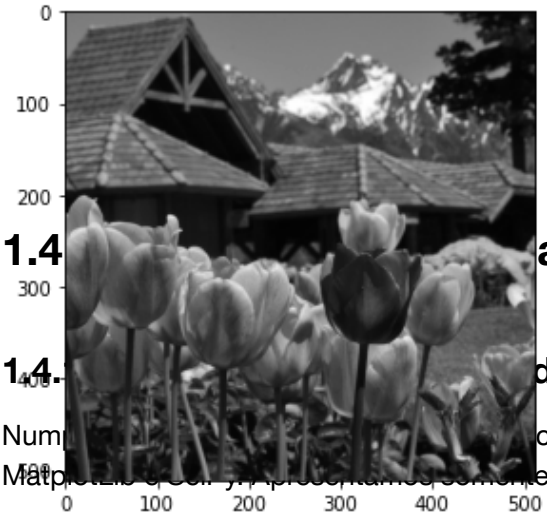
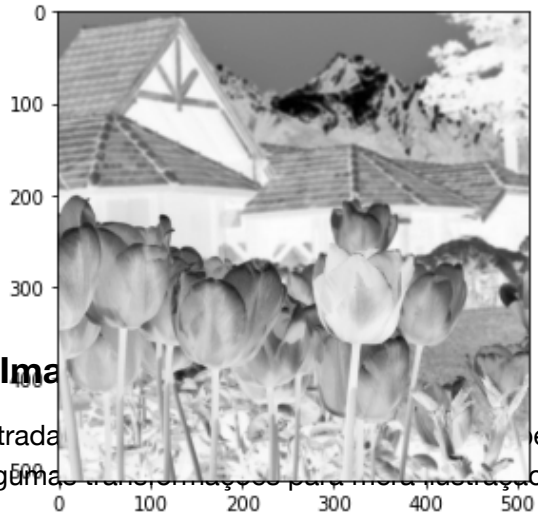
```
img1 = lum

img2 = 255 - img1 #invert image

img3 = (100.0/255) * img1 + 100 #clamp to interval 100...200

img4 = 255.0 * (img1/255.0)**2 #squared

fig = plt.figure()
fig.tight_layout()

plt.figure(figsize=(10,10))

plt.subplot(221)
plt.imshow(img1,cmap='gray', vmin = 0, vmax = 255)

plt.subplot(222)
plt.imshow(img2,cmap='gray', vmin = 0, vmax = 255)

plt.subplot(223)
plt.imshow(img3,cmap='gray', vmin = 0, vmax = 255)

plt.subplot(224)
plt.imshow(img4,cmap='gray', vmin = 0, vmax = 255)
```

executed in 470ms, finished 18:30:34 2023-03-26

Out[8]:

```
<matplotlib.image.AxesImage at 0x7fa4b0e0eca0>
```

```
<Figure size 432x288 with 0 Axes>
```

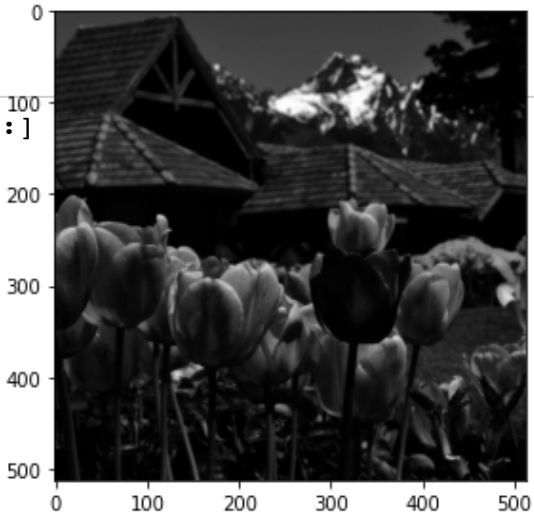



1.4 as

1.4. do Ima

Num contrada enCV,
Matplotlib e Scipy. Aprenderemos somente algumas transformações para melhor ilustração.

Para recortar uma imagem basta definir os limitantes:

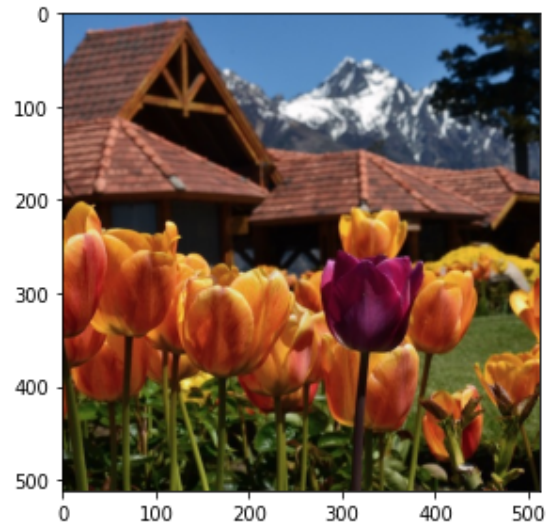
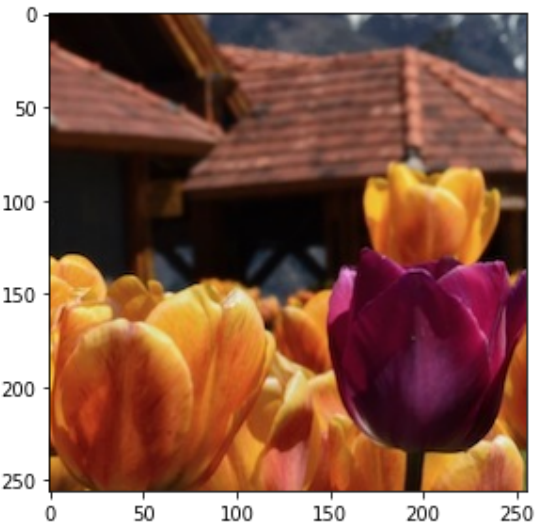
In [ ]:

```
crop = img[128:-128, 128:-128, :]

fig = plt.figure()
fig.tight_layout()

plt.figure(figsize=(10,10))

plt.subplot(121)
plt.imshow(img)

plt.subplot(122)
plt.imshow(crop)
```

executed in 301ms, finished 18:30:34 2023-03-26

Out[9]:

```
<matplotlib.image.AxesImage at 0x7fa4c2119ee0>

<Figure size 432x288 with 0 Axes>
```

Para rotacionar uma imagem em um dado ângulo, podemos usar o código a seguir:

In [10]:

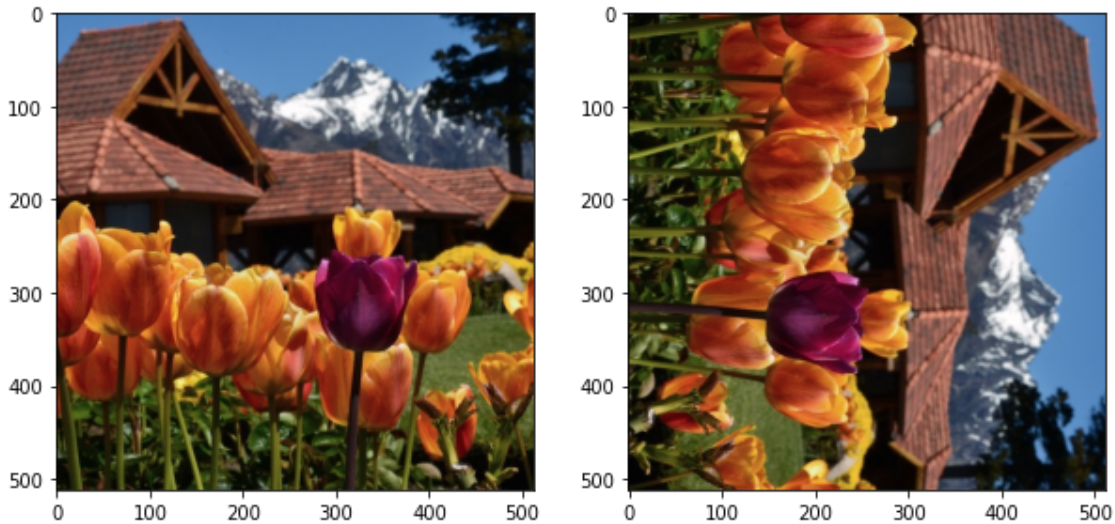
```
degrees = 90
# degrees = 45

img0 = img.copy()
for _ in range(degrees // 90):
    img0 = img0.transpose(1, 0, 2)
    for j in range(0, img0.shape[1] // 2):
        c = img0[:, j, :].copy()
        img0[:, j, :] = img0[: , img0.shape[1]-j-1, :]
        img0[: , img0.shape[1]-j-1, :] = c

fig = plt.figure()
fig.tight_layout()

plt.figure(figsize=(10,10))

plt.subplot(121)
plt.imshow(img)

plt.subplot(122)
plt.imshow(img0)
```

executed in 289ms, finished 18:30:34 2023-03-26

Out[10]:

```
<matplotlib.image.AxesImage at 0x7fa4908e1d30>
```

```
<Figure size 432x288 with 0 Axes>
```



1.4.2 Mudando o tamanho da Imagem

Para mudar o tamanho de uma imagem e realizar uma mudança de escala podemos utilizar a função scale definida localmente:

In [11]:

```
def scale(im, nR, nC):
    number_rows = len(im)      # source number of rows
    number_columns = len(im[0])  # source number of columns
    return [[ im[int(number_rows * r / nR)][int(number_columns * c / nC)]
                  for c in range(nC)] for r in range(nR)]

scaled = scale(img,30,30)

fig = plt.figure()
fig.tight_layout()

plt.figure(figsize=(10,10))

plt.subplot(121)
plt.imshow(img)

plt.subplot(122)
plt.imshow(scaled)
```

executed in 248ms, finished 18:30:35 2023-03-26

Out[11]:

```
<matplotlib.image.AxesImage at 0x7fa4b0ed58e0>

<Figure size 432x288 with 0 Axes>
```