iscte
TECNOLOGIAS
E ARQUITETURA

APVC

Introdução ao OpenCV

Espaços de cores

Binarização

iscte
TECNOLOGIAS
E ARQUITETURA

Introdução ao OpenCV

OpenCV – Em que consiste?

- *Open Source Computer Vision Library*
- Originalmente lançada pela Intel Research em 2000 (versão alfa)
  - Versão 1.0 em 2006, Versão 2 em 2009, Versão 3 em 2015, Versão 4 em 2018
- Atualmente inclui muitos algoritmos de processamento e análise de imagens
  - Reconhecimento facial
  - Deteção de objetos
  - Visão stereo
  - *Motion Tracking*
  - *…*

  … e muitos já usam IA

OpenCV – Principais características

- Escrita e compilada em C++, mas inclui também *bindings* para outras linguagens, como o Python
- As imagens são carregadas e processadas como matrizes
  - No caso do opencv-python, as matrizes são *Numpy arrays*
- De uma forma geral, os algoritmos implementados permitem realizar tarefas sofisticadas em tempo real
  - As funções do OpenCV estão otimizadas para operar sobre matrizes
  - Deve-se por isso evitar o processamento pixel-a-pixel
- Por defeito, o OpenCV usa representação BGR para imagens a cores
  - E por vezes é importante não nos esquecermos desse pormenor!

OpenCV – Ler e guardar imagens

- Ler uma imagem do disco

```
# lê a imagem flower.jpg, do diretório images, e carrega-a na matriz img
img = cv2.imread("images/flower.jpg")

# lê a imagem, converte-a para tons de cinzento, e carrega-a na matriz imgGray
imgGray = cv2.imread("images/flower.jpg", cv2.IMREAD_GRAYSCALE)
```

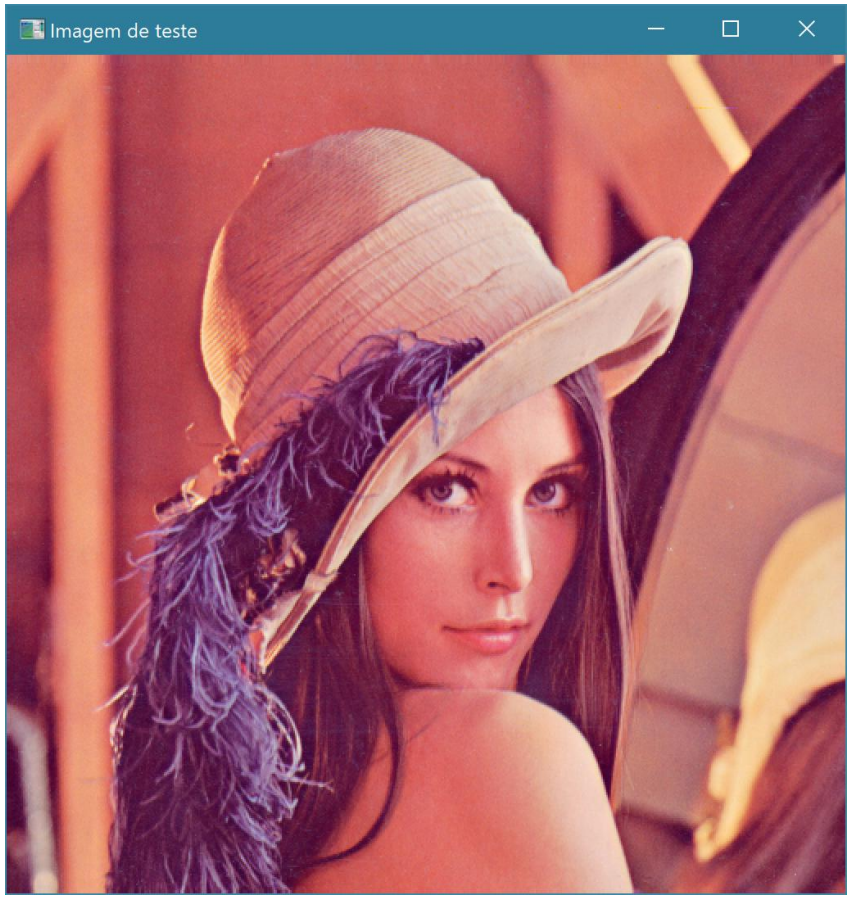
- Escrever uma imagem no disco

```
# grava a imagem img no ficheiro flower.png (usa a extensão para saber o formato da imagem a escrever)
cv2.imwrite("images/flower.png",img)
```

OpenCV – Mostrar uma imagem

Para mostrar uma imagem usa-se `imshow(…)`

```
img = cv2.imread("images/lenna.png")

# O primeiro argumento é o título da janela a criar
cv2.imshow("Imagem de teste", img)

# Espera que se carregue numa tecla antes de sair
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Nota – caso não se use `waitKey(…)`, a janela aparece e desaparece quase imediatamente, porque o programa entretanto chega ao fim…

OpenCV – Criar imagens (ou melhor, matrizes)

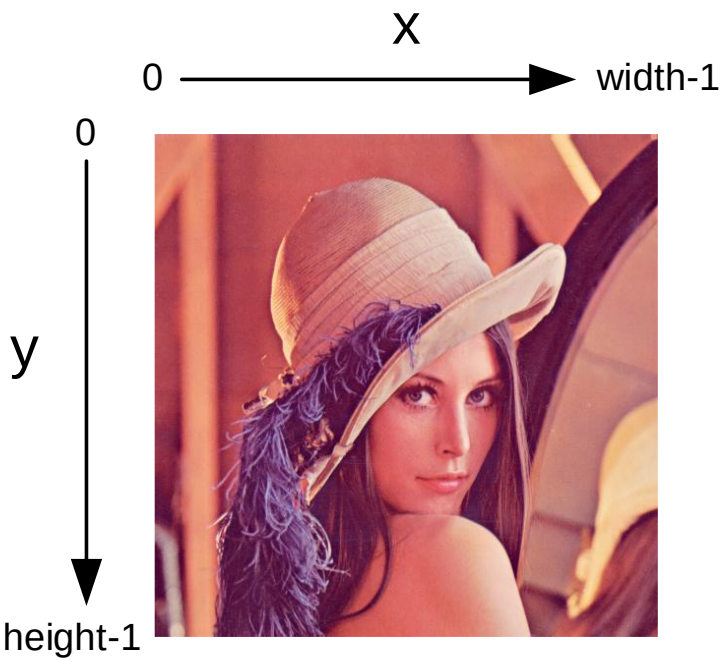
Para criar imagens novas podem-se simplesmente criar matrizes com as funções `zeros(…)` e `ones(…)` do Numpy (ou similares)

```
# cria uma imagem img1 de 256x256, com os pixels inicializados com o valor 255 (branco)
img1 = np.ones( (256, 256), np.uint8 ) * 255

# cria uma imagem img2 de 1080x720, com 3 canais de cor, com os pixels inicializados com o valor 255
img2 = np.ones( (720, 1080, 3), np.uint8 ) * 255

# cria uma imagem img3 de width x height pixels, com 3 canais de cor, com os pixels inicializados a 0
img3 = np.zeros( (height, width, 3), np.uint8 )
```

OpenCV – Aceder aos pixels

O acesso aos pixels é igual a uma matriz convencional

```
# obter o(s) valor(es) do pixel na posição y=100, x=50
px = img[100, 50]

# escrever 0's (preto) nas três componentes do pixel na linha 60, coluna 25
img[60, 25] = (0, 0, 0)
```



Particularidades do sistema de coordenadas

- a coordenada (0, 0) corresponde ao canto superior esquerdo da imagem
- ao aceder, a coordenada y (linha) é especificada antes da x (coluna)
  (isto por vezes baralha quem ainda não está habituado…)

OpenCV – Atributos das matrizes *Numpy*

As matrizes *Numpy* possuem atributos que por vezes dá jeito consultar

- `shape` – dimensões da matriz
- `dtype` – tipo de dados dos elementos (em imagens é usual ser uint8 – valores de 0 a 255)
- `ndim` – número de dimensões da matriz
- `size` – número total de elementos
- `itemsize` – dimensão (em bytes) de cada elemento

```
img = cv2.imread("images/flower.jpg")
print(f"ndim: {img.ndim}")
print(f"shape: {img.shape}")
print(f"size: {img.size}")
print(f"dtype: {img.dtype}")
print(f"itemsize: {img.itemsize}")
```

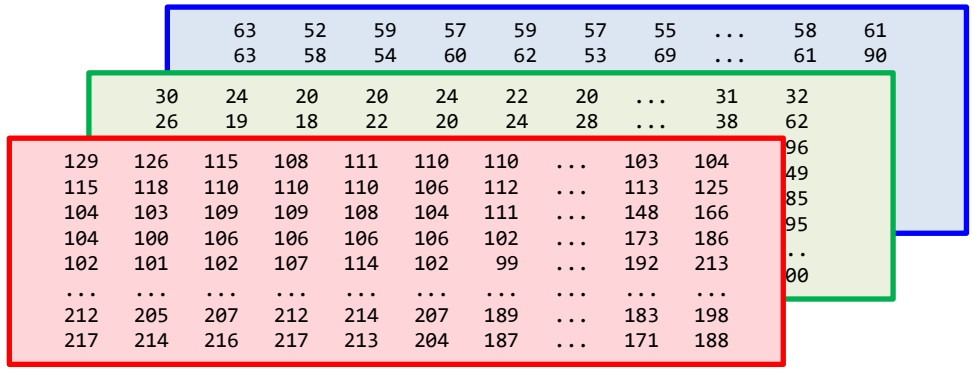
```
ndim: 3
shape: (612, 612, 3)
size: 1123632
dtype: uint8
itemsize: 1
```

iscte
TECNOLOGIAS
E ARQUITETURA

Espaços de cor

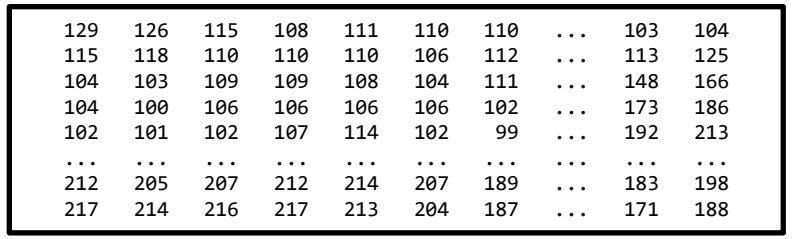
Representação de imagens

Imagem a cores (*policromática*)



3 canais de cores (tipicamente RGB)

Imagem em níveis de cinzento (*monocromática*)



Um único canal (*luminância*)

RGB

R

G

B

- RGB – espaço de cores mais usual (computadores, fotografia digital, etc.)
- A cor em cada pixel (*i.e*, em cada ponto da imagem) é composta por 3 valores entre 0 e 255 – R, G e B
- *Red*, *Green* e *Blue* são as 3 cores primárias num sistema de cores aditivo (i.e., sistema de cores onde a soma das 3 cores primárias é o branco)

YCbCr

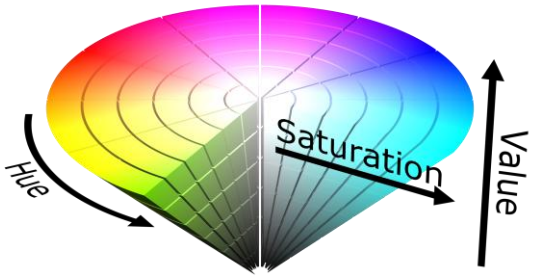
Y Cb Cr

- Espaço de cores habitualmente usado em transmissão de TV/vídeo
- 3 valores por pixel: luminância (Y) e crominâncias azul e vermelha (Cb e Cr)
- A luminância transporta a maior parte da informação estrutural da imagem – corresponde à imagem em tons de cinzento (como nas TVs ou fotos antigas)

HSV

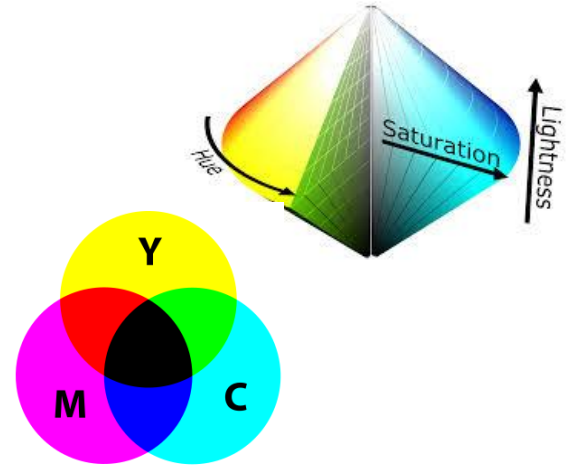
Hue (H) *Saturation (S)* *Value (V)*

- Espaço de cores popular em aplicações de visão por computador
- *Hue* indica a cor propriamente dita (amarelo, vermelho, azul, etc.)
- *Saturation* indica se é uma cor mais viva ou mais “esbatida”
- *Value* indica se é um tom claro ou escuro



Mais sobre espaços de cores

- Existem muito mais espaços de cor
  - HSL – *Hue, Saturation and Lightness* (semelhante ao HSV)
  - YUV – semelhante ao YCbCr
  - CMY – *Cyan, Magenta, Yellow*
    (cores primárias no modelo subtrativo, onde a soma das 3 cores dá preto)
  - …



- Os valores de cada componente da cor também podem assumir diferentes gamas de representação, *e.g.*:
  - Inteiros de 8 bits sem sinal, entre 0 e 255 (mais habitual)
  - Inteiros de 8 bits com sinal, entre -128 e 127 (é o caso de Cb e Cr nos sistemas de TV/vídeo)
  - Valores reais entre 0.0 e 1.0 ou de -1.0 a 1.0 – usual em bibliotecas de aprendizagem automática
  - No OpenCV, a componente *Hue* do espaço HSV assume valores entre 0 e 180

OpenCV – Separar e juntar canais de cor

- Separar canais – `split(…)`

```
# separa os canais BGR da imagem img – cada um dos canais será uma matriz bi-dimensional
(b, g, r) = cv2.split(img)
```

- Juntar canais – `merge(…)`

```
# junta os canais HSV numa nova imagem imgHsv – esta será uma matriz tri-dimensional
imgHsv = cv2.merge([h, s, v])
```

OpenCV – Conversões entre espaços de cor

Converter para outro espaço de cor – `cvtColor(…)`

```
# Conversão BGR -> HSV
imgHsv = cv2.cvtColor(imgBgr, cv2.COLOR_BGR2HSV)

# Conversão HSV -> BGR
imgBgr = cv2.cvtColor(imgHsv, cv2.COLOR_HSV2BGR)

# Conversão BGR -> YCrCb
imgYCrCb = cv2.cvtColor(imgBgr, cv2.COLOR_BGR2YCrCb)
```

Existem dezenas de possibilidades – o OpenCV suporta uma grande variedade de espaços de cor

iscte
TECNOLOGIAS
E ARQUITETURA

Binarização e máscaras (*thresholding*)

Binarização – Conceito

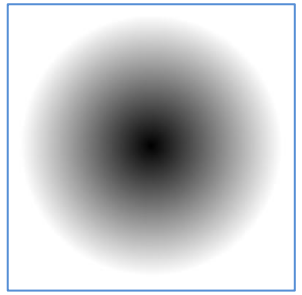
- Consiste em atribuir valores binários (0/1 ou 0/255) aos pixels da imagem
  - Gera uma imagem onde só há duas cores: branco ou preto – a *máscara*
  - Normalmente aplica-se individualmente a uma ou mais das componentes da cor
  - As condições para atribuição dos valores binários dependem da finalidade a que se destina
- Exemplos de aplicação
  - localizar objetos específicos
  - separar *foreground* do *background*
  - preparar imagens de texto para OCR
  - …

Binarização – *Thresholding*

- Uma das formas mais simples de realizar binarização de imagens é através de comparações com um valor de limiar (*thresholding*)
- Basicamente o resultado será 0 ou 1 consoante o valor do pixel esteja abaixo ou acima do valor de limiar
- A operação de *thresholding* pode ser realizada no opencv usando:
  - `threshold(…)` – gera uma máscara por comparação com um valor de limiar – tipicamente usada para processar um único canal; tem várias opções quanto ao output
  - `inRange(…)` – gera uma máscara com base em dois limiares (min e max); aos valores de pixel entre min e max corresponderá o valor lógico 1, aos restantes 0
- Nas máscaras geradas pelo opencv:
  - valor lógico 1 → pixel da máscara a 255 (ou outro valor especificado);
  - valor lógico 0 → pixel da máscara a 0;

Binarização – Exemplos no OpenCV

```
img = cv2.imread("images/circle.png", cv2.IMREAD_GRAYSCALE)

mask1 = cv2.inRange(img, 100, 200)

(t, mask2) = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)

(t, mask3) = cv2.threshold(img, 200, 255, cv2.THRESH_BINARY_INV)
```

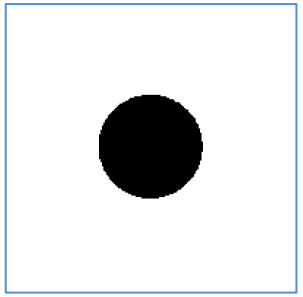


img
(original)

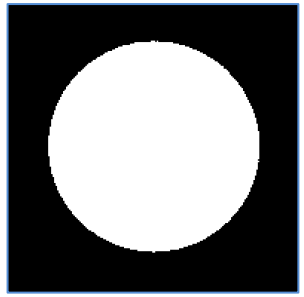
Notas:

- O terceiro argumento numérico da função `threshold()` é o valor a atribuir à máscara quando a operação dá valor lógico 1
- O "t" devolvido por `threshold(…)` é para ser ignorado a não ser que se use a opção `cv2.THRESH_OTSU` (método adaptativo)

mask1
(`inRange`)



mask2
(`THRESH_BINARY`)



mask3
(`THRESH_BINARY_INV`)

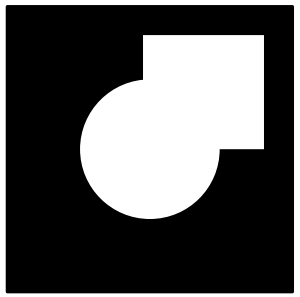
Máscaras – Combinações

Muitas vezes é útil combinar máscaras resultantes de condições ou canais diferentes

Para combinar as máscaras podem-se usar operações lógicas:

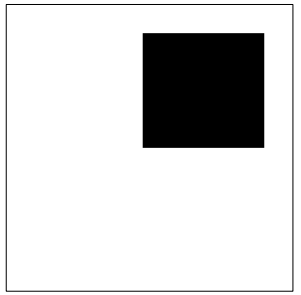
mask1 mask2

```
mask = cv2.bitwise_or(mask1, mask2)
```



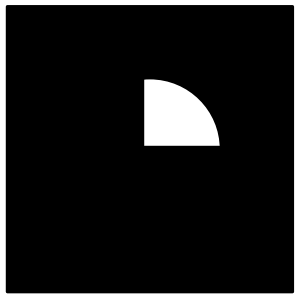
mask1 *or* mask2
(união)

```
mask = cv2.bitwise_not(mask1)
```



not mask1
(complemento)

```
mask = cv2.bitwise_and(mask1, mask2)
```

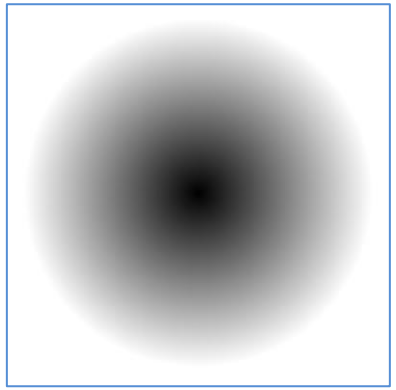


mask1 *and* mask2
(interseção)

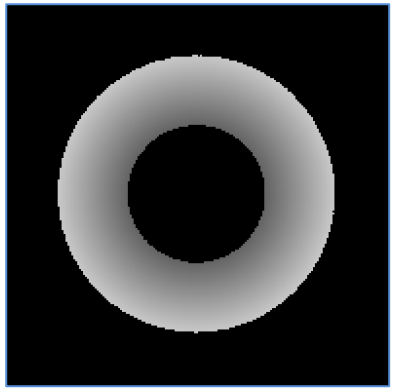
Máscaras – Outras coisas que dão jeito

- Combinar uma máscara com uma imagem

```
img = cv2.imread("images/circle.png", cv2.IMREAD_GRAYSCALE)

mask1 = cv2.inRange(img, 100, 200)

maskedImage = cv2.bitwise_and(img, img, mask=mask1)
```



img

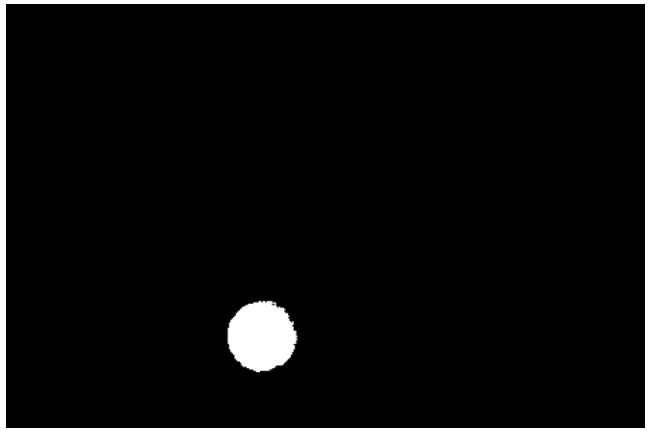
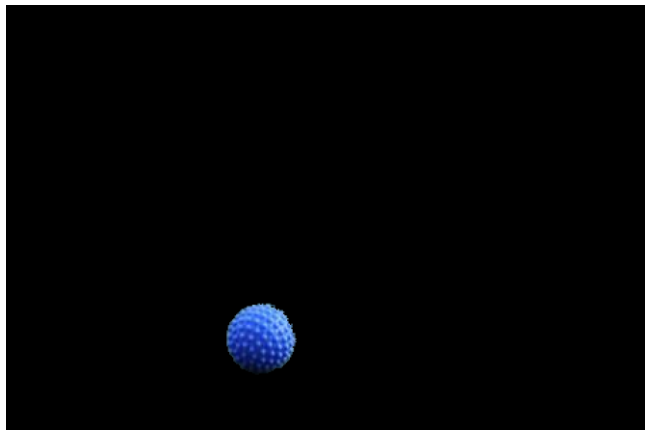
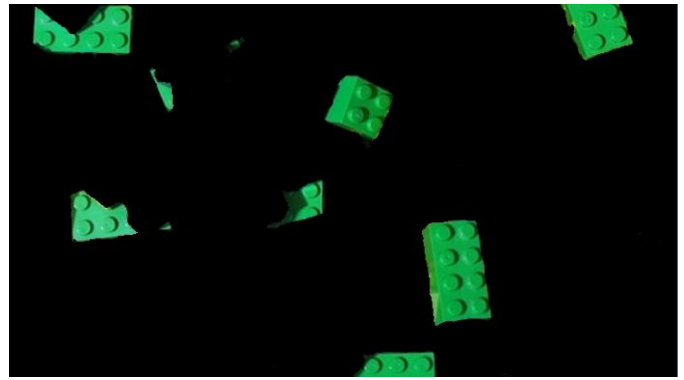


maskedImage

- Contar quantos pixels estão a “branco” numa máscara

```
nPix = cv2.countNonZero(mask1)
```

Máscaras – Exemplos

Imagem original	Máscara binária	Aplicação da máscara sobre a imagem original
		
		

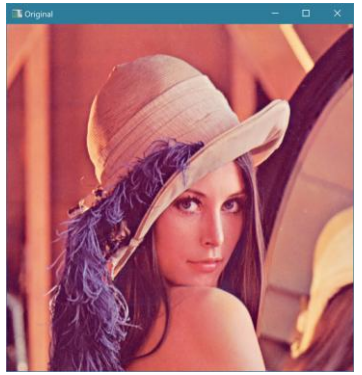
iscte
TECNOLOGIAS
E ARQUITETURA

Extras

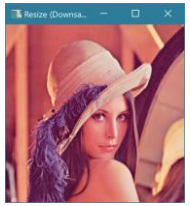
OpenCV – Redimensionar imagens

Para redimensionar imagens utiliza-se o método `resize(…)`

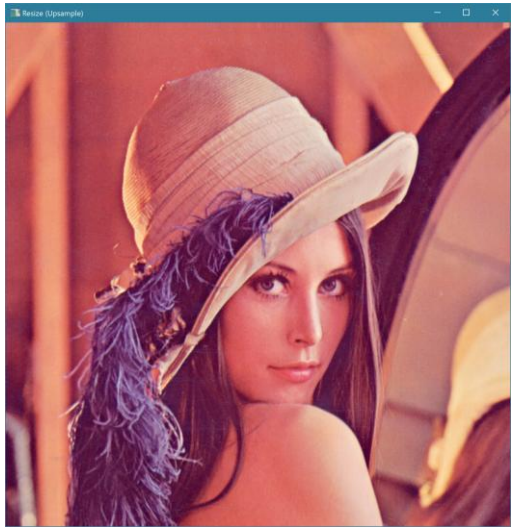
```
imgDown = cv2.resize(img, (256,256))

imgUp = cv2.resize(img, (800,800), interpolation=cv2.INTER_LANCZOS4)
```



img



imgDown



imgUp

Possibilidades para o algoritmo de interpolação

- INTER_NEAREST
- INTER_AREA
- INTER_LINEAR (default)
- INTER_CUBIC
- INTER_LANCZOS4 (normalmente com melhores resultados no *upsampling*)

OpenCV – Crop

"Crop" (ou "recorte") – operação onde se obtém uma subimagem que é uma parte da imagem original

Para realizar um "crop" manipula-se diretamente a matriz da imagem (não necessita de uma função específica)

```
img = cv2.imread("images/lenna.png")

imgCrop = img[150:450, 150:450]
```

img

imgCrop

Recursos

- Color and color spaces in Computer Vision, https://kharshit.github.io/blog/2020/01/17/color-and-color-spaces-in-computer-vision
- Imagem do cão e bola azul, https://www.countryliving.com
- Imagens “Lenna” e espaços de cor, https://en.wikipedia.org
- Imagem “flower”, https://www.freeimages.com
- Imagem “legos”, https://observador.pt/2017/09/02/lego-a-brincar-a-brincar-os-tijolos-fazem-85-anos/
- Tutoriais opencv python
  - GeekforGeeks - https://www.geeksforgeeks.org/opencv-python-tutorial/
  - LearnOpenCV - https://learnopencv.com/getting-started-with-opencv/
  - Pyimagesearch – https://www.pyimagesearch.com/
- Kenneth Dawson-Howe, *A Practical Introduction to Computer Vision with OpenCV*, Wiley, 2014