VISÃO COMPUTACIONAL

AULA 4

Prof. Leonardo Gomes

CONVERSA INICIAL

Nesta aula, vamos debater, em detalhes, dois conceitos muito importantes na visão computacional: as operações morfológicas, que consistem em modificações da imagem para uma melhor análise; e a segmentação de objetos, que, como o nome sugere, consiste em separar a região da imagem que contém o objeto de interesse. Novamente, teremos diversos exemplos de códigos utilizando Python e OpenCV.

Ao final desta aula, esperamos atingir os seguintes objetivos:

1. dominar técnicas de operações morfológicas como a erosão e a dilatação;
2. dominar técnicas de segmentação de objetos por cor, bordas etc.;
3. desenvolver códigos em OpenCV, que façam uso de operações morfológicas e de segmentação de objetos.

TEMA 1 – DILATAÇÃO E EROSÃO

Chamamos de *transformações morfológicas* os algoritmos que modificam o formato dos objetos que são representados numa imagem. Geralmente, na visão computacional utilizamos imagens binárias para representar os objetos de nosso interesse ou a área que eles ocupam. E, com frequência, o processo de binarização da nossa região de interesse gera resultados com algum nível de ruído, e as operações morfológicas podem ajudar a corrigir esses problemas. Neste tema, vamos discutir duas das operações morfológicas mais importantes, a **erosão** e a **dilatação**, que servem de base para outras operações mais complexas. Elas são utilizadas para, respectivamente, reduzir ou ampliar a área de borda da região branca. As duas operações empregam um conceito básico chamado de *elemento estruturante*.

1.1 Elemento estruturante

Os elementos estruturantes são semelhantes às máscaras dos filtros: trata-se de pequenas matrizes binárias que são aplicadas nas operações de dilatação e erosão para lhes dar um aspecto mais arredondado ou retilíneo, dentre outros, conforme a necessidade da aplicação. Na Figura 1 encontram-se alguns dos elementos estruturantes mais comuns, que podem, inclusive, ser

gerados por funções predefinidas no OpenCV, antecedidos do código que gera esses elementos estruturantes, respectivamente: quadrado, elipse e cruz.

Figura 1 – Código dos elementos estruturantes e representação de três elementos estruturantes básicos, na ordem: quadrado, elipse e cruz

```
cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
```

1	1	1	1	1	0	0	1	0	0	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	0	1	0	0
1	1	1	1	1	0	0	1	0	0	0	0	1	0	0

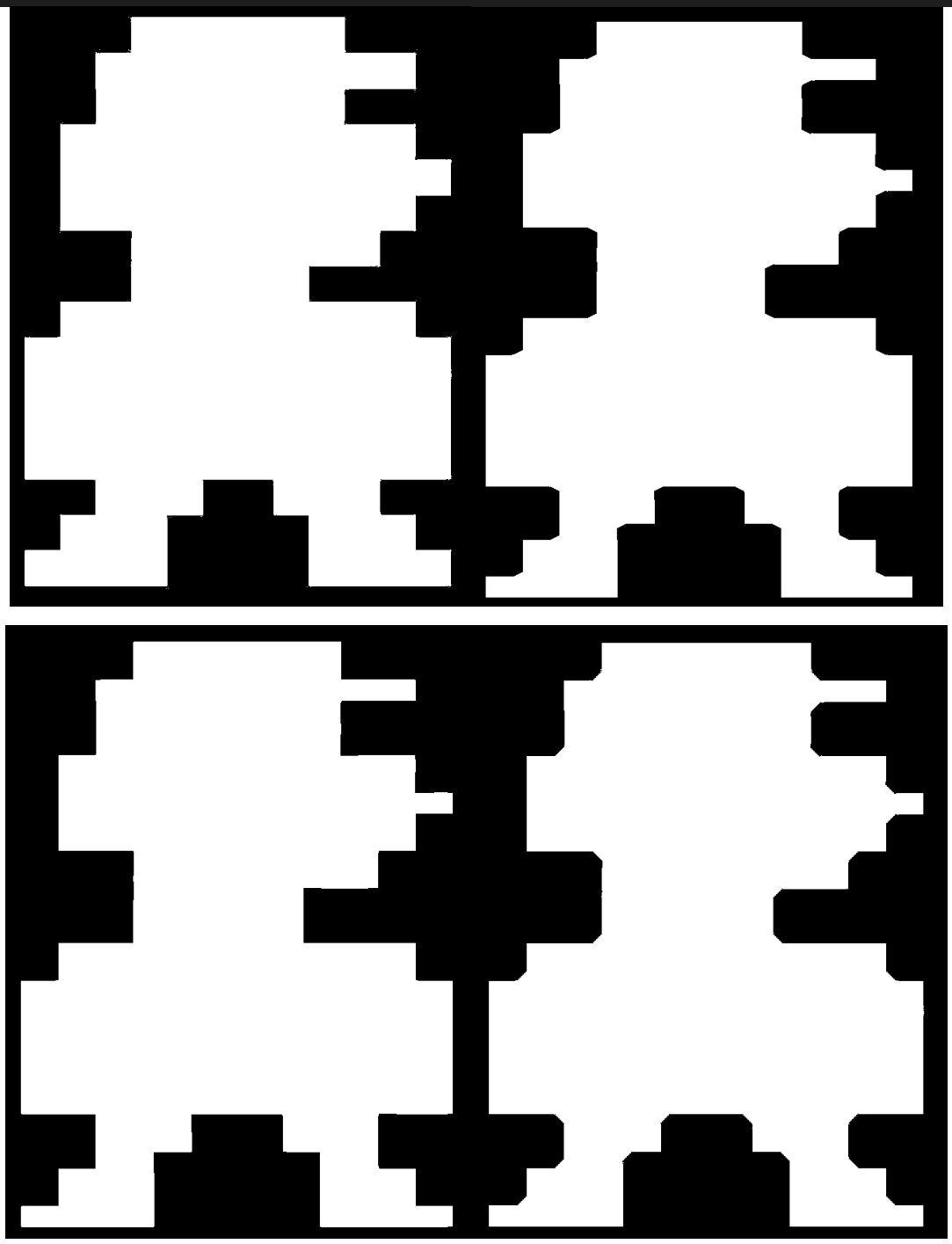
1.2 Erosão

A operação de erosão aplica o elemento estruturante na imagem, com a finalidade de reduzir a área do objeto, consumindo suas bordas. A ideia aqui é que o centro do elemento estruturante passa por todos os *pixels* da imagem, de modo semelhante a uma máscara. E, se o *pixel* analisado e a sua vizinhança coincidem exatamente com os valores do *pixel* do elemento estruturante (ignorando os seus zeros), o *pixel* se torna parte da imagem; caso contrário, o *pixel* se torna fundo da imagem. No OpenCV, a erosão é feita pela função *erode*, que recebe por parâmetros: a imagem; o elemento estruturante; e o número de aplicações que se deseja. Confira, na Figura 2, o código utilizado para isso e o seu respectivo resultado.

Figura 2 – Código da operação de erosão e seu resultado, na seguinte ordem: imagem original; erosão por cruz; erosão por retângulo; erosão por elipse

```
import cv2
imgOrig = cv2.imread("mario.jpg",0)
elemento = cv2.getStructuringElement(
cv2.MORPH_CROSS, (5,5))
```

```
resultado = cv2.erode(
imgOrig, elemento, iterations = 6)
cv2.imshow("Original", imgOrig)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



Na Figura 2, vemos que todos os diferentes elementos estruturantes resultantes da operação de erosão foram capazes de remover pequenos ruídos, pontos brancos isolados da imagem original; no entanto, cada um apresenta um

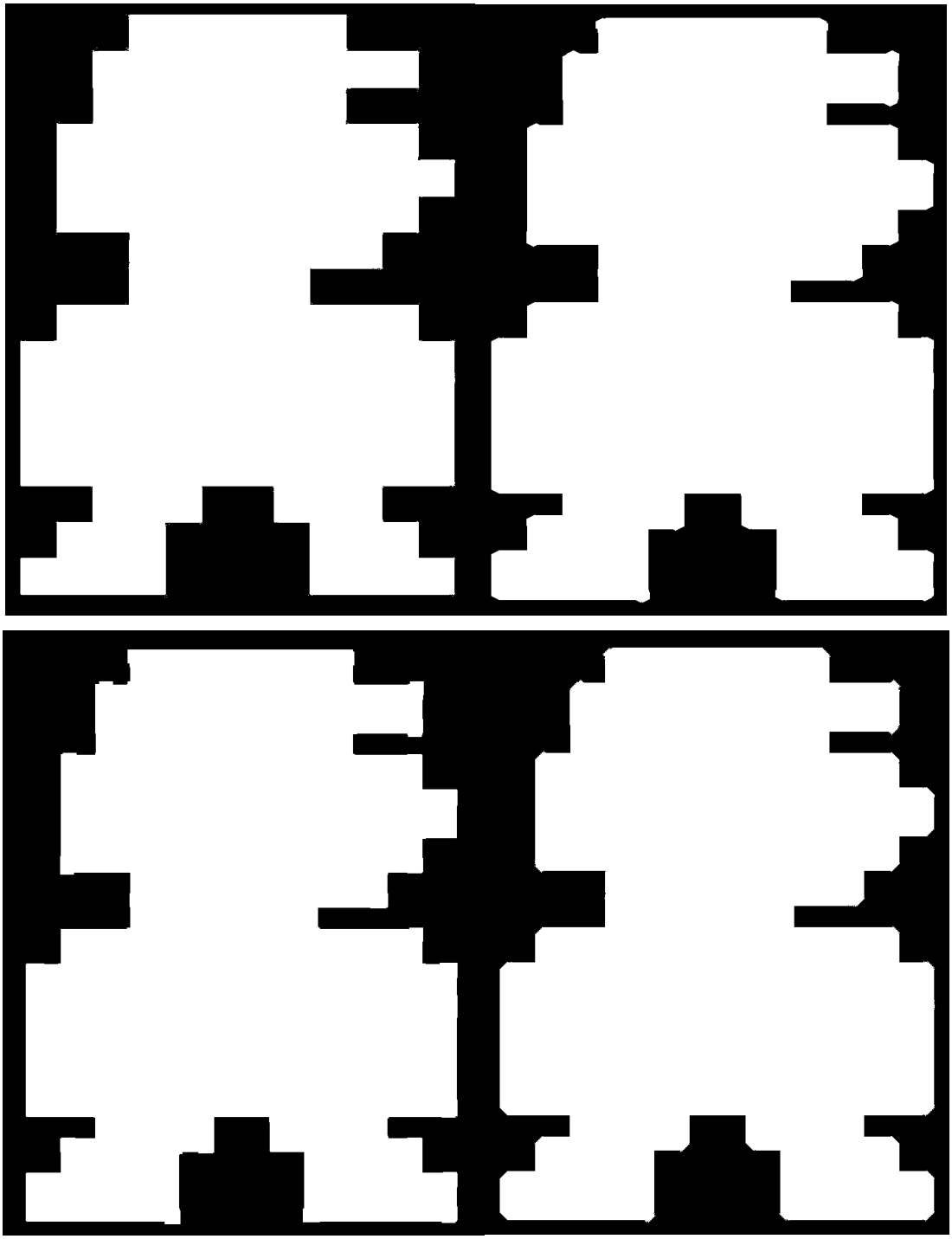
efeito levemente diferente: o elemento retângulo deixa bordas retas, enquanto os outros operadores criaram quinas arredondadas ou de aspecto diagonal.

1.3 Dilatação

A operação de dilatação aplica o elemento estruturante na imagem, com a finalidade de aumentar a área do objeto nela, ampliando suas bordas. A dilatação funciona de forma muito semelhante à erosão. O elemento estruturante é aplicado na imagem da mesma forma; no entanto, caso o *pixel* analisado e sua vizinhança coincidam com qualquer um dos *pixels* de valor 1 do elemento estruturante, *o pixel* se torna parte da imagem e, caso contrário, o *pixel* se torna fundo da imagem, ao passo que na erosão precisavam ser todos. No OpenCV, a dilatação é feita pela função *dilate*, que recebe por parâmetros: a imagem; o elemento estruturante; e o número de aplicações que se deseja. Confira, na Figura 3, o código para isso e o seu respectivo resultado.

Figura 3 – Código da operação de dilatação e seu resultado, na seguinte ordem: imagem original; dilatação por elipse; dilatação por retângulo; dilatação por cruz

```
import cv2
imgOrig = cv2.imread("mario.jpg",0)
elemento = cv2.getStructuringElement(
cv2.MORPH_CROSS, (5,5))
resultado = cv2.dilate(
imgOrig, elemento, iterations = 5)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



Na Figura 3, vemos o resultado da dilatação com diferentes elementos estruturantes. Ao contrário da erosão, ruídos compostos por *pixels* brancos são ampliados pela dilatação e ruídos por *pixels* pretos são removidos. Observamos também que o elemento retângulo deixa bordas retas, enquanto a elipse gera quinas arredondadas e a cruz gera diagonais.

TEMA 2 – ABERTURA, FECHAMENTO E GRADIENTE MORFOLÓGICO

Neste tema, vamos debater algumas operações morfológicas compostas por erosão e dilatação, como abertura, fechamento e gradiente morfológico.

2.1 Abertura

A abertura é realizada para tratar ruídos brancos, regiões de falso positivo que não fazem parte do objeto, mas são marcadas como tal. Sozinha, a operação de erosão tem esse efeito; no entanto, ela degrada a borda do objeto de interesse e, por isso, a operação de dilatação é executada, na sequência, para restaurar a borda perdida. Essa técnica funciona muito bem para ruídos pequenos como de sal e pimenta; ruídos maiores podem demandar elementos estruturantes também maiores, ao custo de uma maior chance de perda de informação, pela deformação do objeto estruturante.

Em resumo, a abertura é uma erosão seguida de uma dilatação. No OpenCV, a função *morphologyEx* pode ser executada com essa finalidade. Ela possui três parâmetros: a imagem; a operação que será executada, no caso, *MORPH_OPEN*; e, por fim, o elemento estruturante. Confira, na Figura 4, o código para isso e o seu respectivo resultado.

Figura 4 – Código da operação de abertura e seu resultado, na seguinte ordem: imagem original com ruído; imagem após a operação de abertura

```
import cv2
imgOrig = cv2.imread("RuidoB.png",0)
elemento = cv2.getStructuringElement(
cv2.MORPH_ELLIPSE, (5,5))
resultado = cv2.morphologyEx(
imgOrig, cv2.MORPH_OPEN, elemento)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```





No resultado da Figura 4, é possível notar que o ruído foi removido sem distorção significativa da imagem.

2.2 Fechamento

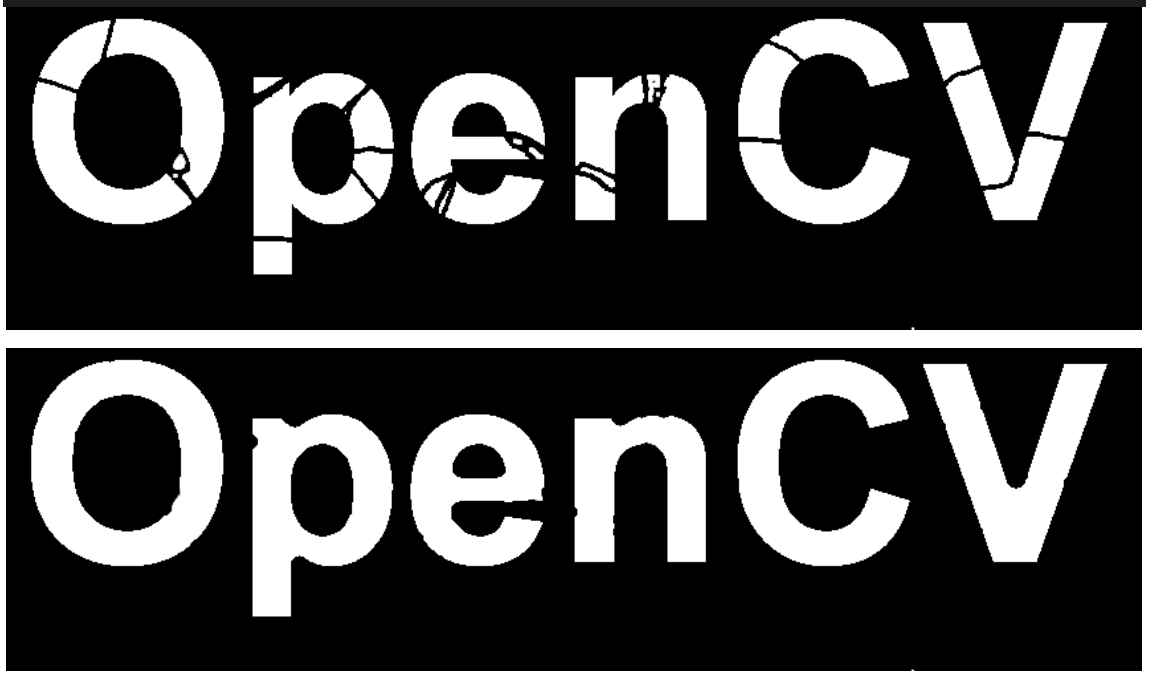
O fechamento é realizado para tratar ruídos pretos, regiões de falso negativo que não fazem parte do objeto, mas são marcadas como tal, pequenas obstruções, sombras, entre outras questões que, na aquisição e/ou binarização da imagem, podem gerar essas regiões e dividir o objeto muitas vezes. Sozinha, a operação de dilatação tem esse efeito; no entanto, ela aumenta a borda do objeto de interesse, além do desejado, muitas vezes. Por isso, a operação de erosão é executada, na sequência, para restaurar a borda de forma que fique próxima ao original. Semelhantemente à abertura, ruídos maiores podem demandar elementos estruturantes também maiores, ao custo de uma maior chance de perda de informação, pela deformação do objeto estruturante.

Em resumo, a abertura é uma erosão seguida de uma dilatação. No OpenCV, a função *morphologyEx* pode ser executada com essa finalidade. Ela possui três parâmetros: a imagem; a operação que será executada, no caso, *MORPH_CLOSE*; e, por fim, o elemento estruturante. Confira, na Figura 5, o código usado para isso e o seu respectivo resultado.

Figura 5 – Código da operação de fechamento e seu resultado, na seguinte ordem: imagem original com ruído, na forma de buracos nas letras; imagem após a operação morfológica de fechamento

```
import cv2
imgOrig = cv2.imread("RuidoP.png",0)
elemento = cv2.getStructuringElement(
cv2.MORPH_ELLIPSE, (13,13))
resultado = cv2.morphologyEx(
```

```
imgOrig, cv2.MORPH_CLOSE, elemento)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



No resultado da Figura 5, é possível notar que o ruído foi atenuado.

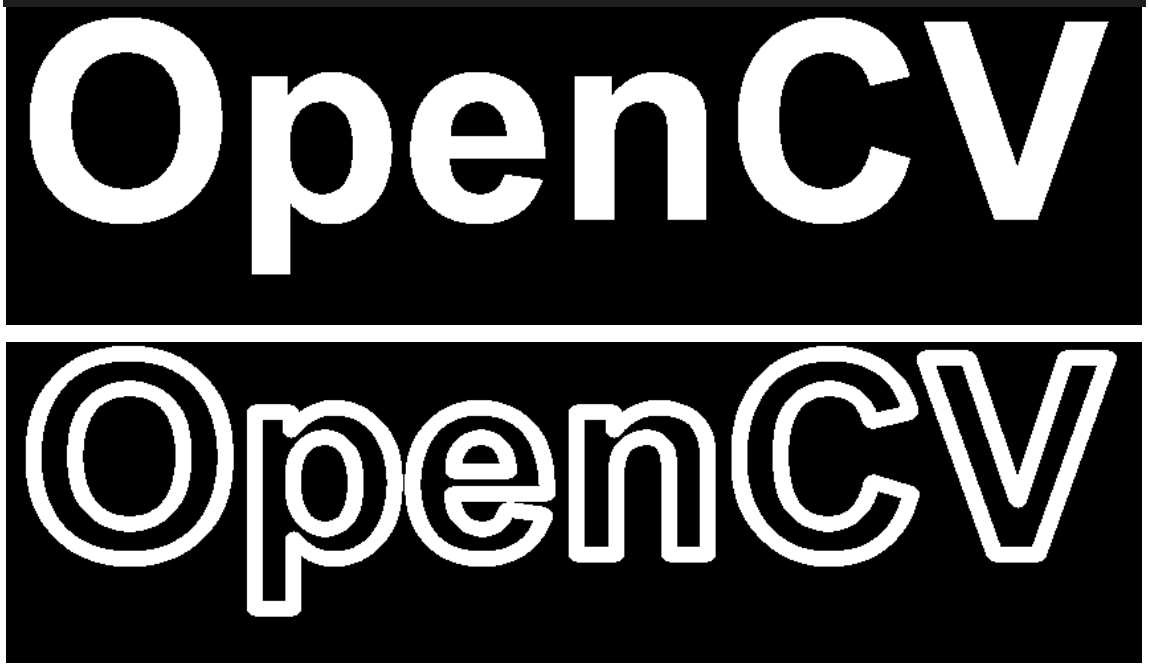
2.3 Gradiente morfológico

O gradiente morfológico tem por finalidade descobrir a borda do objeto de interesse. E isso é feito por meio da diferença entre a imagem dilatada e a imagem da erosão. No OpenCV, a função *morphologyEx* pode ser executada com essa finalidade. Ela possui três parâmetros: a imagem; a operação que será executada, no caso, *MORPH_GRADIENT*; e, por fim, o elemento estruturante. Confira, na Figura 6, um código de exemplo e o seu respectivo resultado.

Figura 6 – Código da operação de gradiente morfológico e seu resultado, na seguinte ordem: imagem original; imagem após a operação de gradiente morfológico

```
import cv2
imgOrig = cv2.imread("OpenCV.png",0)
elemento = cv2.getStructuringElement(
cv2.MORPH_ELLIPSE, (13,13))
resultado = cv2.morphologyEx(
```

```
imgOrig, cv2.MORPH_GRADIENT, elemento)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



As bordas são destacadas com bastante precisão nas imagens binárias e podem ter uma espessura maior ou menor, dependendo das dimensões do elemento morfológico adotado.

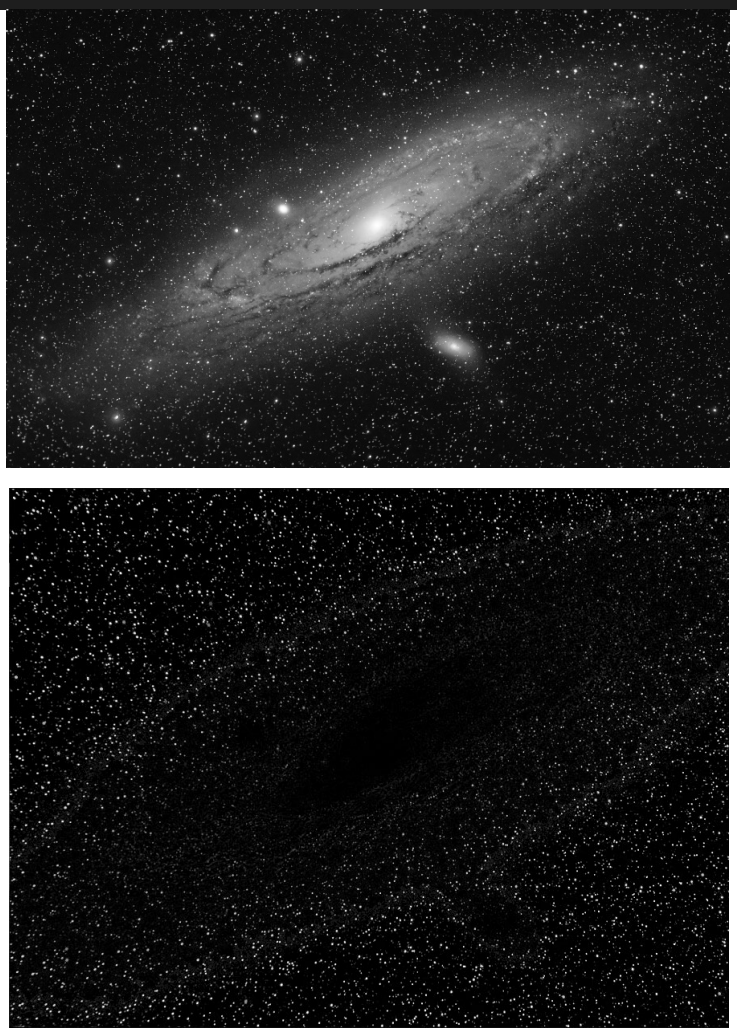
TEMA 3 – *TOP HAT*

A operação de *top hat* é utilizada principalmente para destacar objetos claros num fundo escuro. E isso é feito por meio da diferença entre a imagem original e a imagem que recebeu a operação de abertura. A operação de *top hat* destaca da imagem original elementos ou objetos que sejam menores do que o elemento estruturante e que tenham a característica de serem mais luminosos do que seus arredores. No OpenCV, a função *morphologyEx* pode ser executada com essa finalidade. Ela possui três parâmetros: a imagem; a operação que será executada, no caso, *MORPH_TOPHAT*; e, por fim, o elemento estruturante. Confira, na Figura 7, um código de exemplo e o seu respectivo resultado.

Figura 7 – Código para operação de *top hat* e seu resultado, na seguinte ordem: imagem original; imagem após a operação de *top hat*

```
import cv2
imgOrig = cv2.imread("galaxia.png",0)
```

```
elemento = cv2.getStructuringElement(
cv2.MORPH_ELLIPSE, (5,5))
resultado = cv2.morphologyEx(
imgOrig, cv2.MORPH_TOPHAT, elemento)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



Crédito: Antares_StarExplorer/Shutterstock.

Na Figura 7, temos o resultado da operação de *top hat*, em que as estrelas que são menores do que o elemento estruturante foram capturadas, ignorando-se os elementos maiores, como a representação da galáxia da imagem.

Existe outro operador similar ao *top hat*, chamado de *black hat*, que segue o mesmo princípio só que invertido: ele destaca regiões mais escuras que a vizinhança da imagem. Esse operador é composto pela diferença entre a imagem original e a imagem que recebeu a operação de fechamento. No

OpenCV, a função *morphologyEX* também é usada, porém com o operador *MORPH_BLACKHAT*.

TEMA 4 – SEGMENTAÇÃO POR BINARIZAÇÃO

A segmentação dos objetos é uma etapa muito importante na visão computacional, pois é por meio dela que identificamos as regiões de interesse que serão avaliadas. Sem identificar tais regiões, não temos como extrair suas características e prosseguir com o *pipeline*. Neste e no próximo temas vamos apresentar algumas das principais estratégias, nessa etapa.

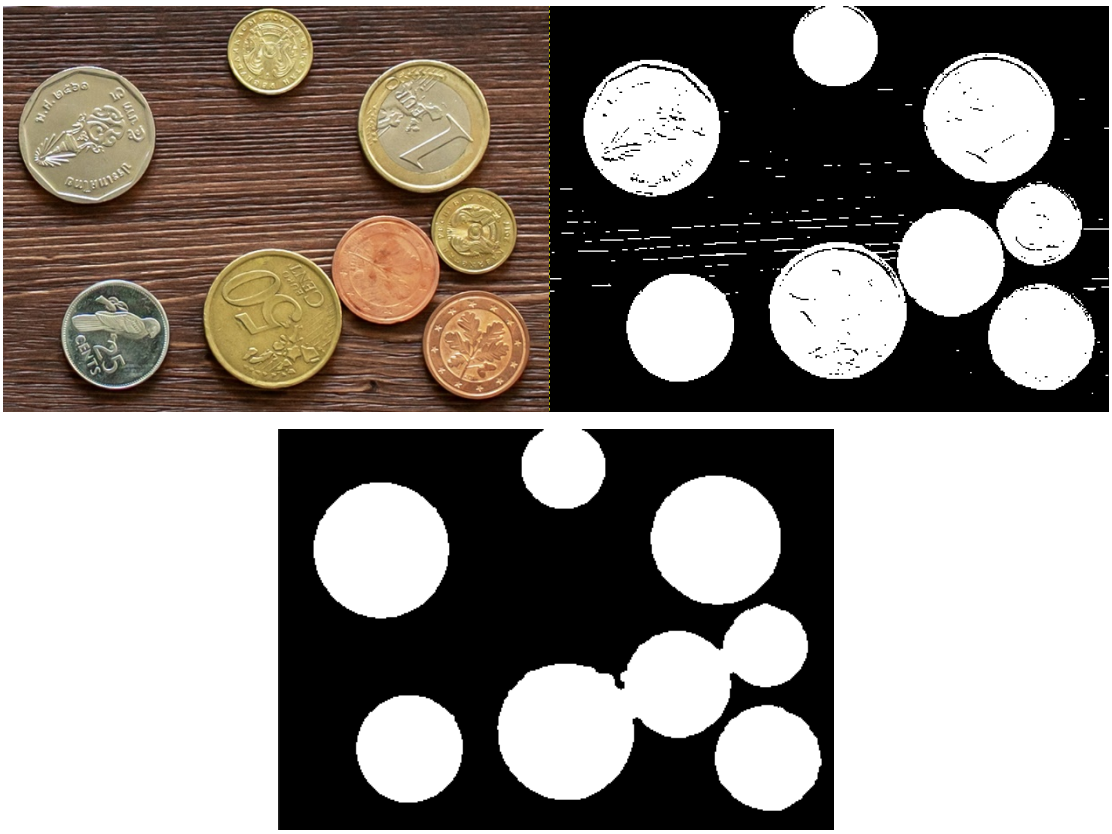
4.1 Limiarização

A binarização geralmente considera a intensidade luminosa dos *pixels*: *pixels* que estejam acima de um dado **limiar** são coloridos de branco e se tornam a região de interesse; e os que estiverem abaixo são coloridos de preto e considerados como o fundo da imagem (dependendo da necessidade, essa lógica pode ser, também, invertida).

No OpenCV, a função *threshold* é utilizada com essa finalidade. Ela possui quatro parâmetros: a imagem com tons de cinza; o segundo valor é o limiar; o terceiro parâmetro é o valor que os *pixels* acima do limiar receberão (255 para branco, geralmente); e, por fim, se a operação será inversa ou não, passando-se respectivamente as constantes *THRESH_BINARY* ou *THRESH_BINARY_INV*. Confira, na Figura 8, um código de exemplo e o seu respectivo resultado.

Figura 8 – Código para operação de limiarização e seu resultado, na seguinte ordem: imagem original; imagem após a operação de limiarização; imagem após a operação de abertura e fechamento por *top hat*

```
import cv2
imgOrig = cv2.imread("moedas.png",0)
rr,resultado = cv2.threshold(imgOrig,125,255,
cv2.THRESH_BINARY)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



Crédito: Vera Larina/Shutterstock.

Pequenos ruídos na imagem podem ser corrigidos com os operadores morfológicos de fechamento, para *pixels* pretos soltos; e de abertura, para *pixels* brancos soltos, conforme a Figura 8.

4.2 Binarização adaptativa

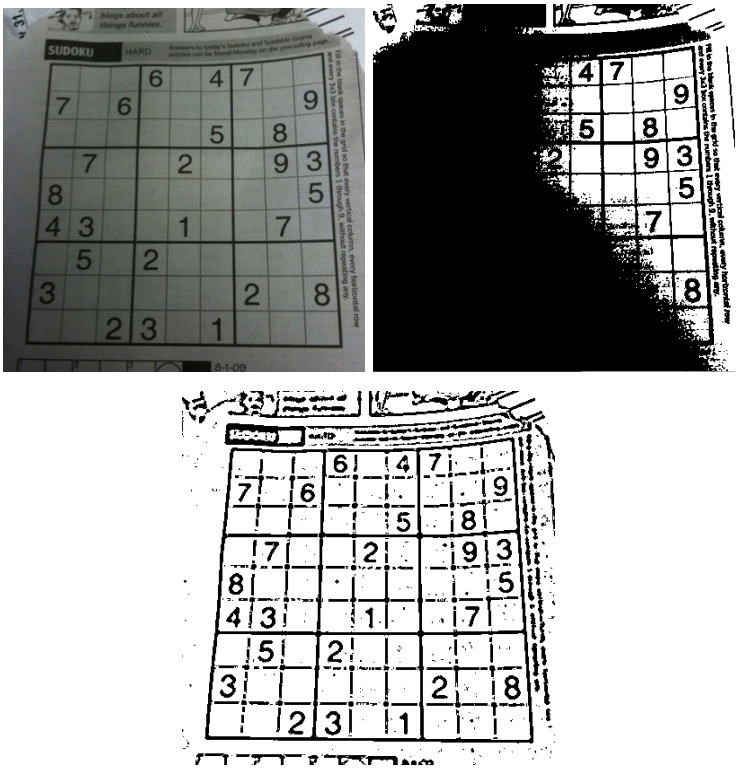
A estratégia de binarização apresenta algumas limitações, em especial quando trabalhamos com imagens cuja iluminação não é constante. Nesses casos, seria interessante que regiões mais claras pudessem ter um valor de limiar maior que regiões mais escuras. Nessas situações em que a iluminação não é uniforme, temos o algoritmo de binarização adaptativa, que calcula diferentes valores de limiar ao longo da imagem.

No OpenCV, a função *adaptiveThreshold* é utilizada com essa finalidade. Ela possui seis parâmetros: o primeiro tem por base a imagem com tons de cinza; o segundo parâmetro é a intensidade máxima do pixel; o terceiro, qual dos algoritmos de binarização será adotado, se *ADAPTIVE_THRESH_MEAN_C* ou *ADAPTIVE_THRESH_GAUSSIAN_C* (é interessante testar ambos para decidir o melhor para cada situação); o quarto parâmetro é se a binarização será regular ou inversa, respectivamente *THRESH_BINARY* ou *THRESH_BINARY_INV*; o

quinto é o tamanho da máscara utilizada no algoritmo que define a intensidade luminosa da vizinhança; e, por fim, o sexto parâmetro é uma constante inteira, geralmente positiva, que será subtraída da média utilizada no algoritmo escolhido no quarto parâmetro, e que ajuda a refinar o resultado para favorecer que haja valores claros ou escuros, dependendo do sinal do valor passado. Confira, na Figura 9, um código de exemplo e o seu respectivo resultado.

Figura 9 – Código para operação de binarização adaptativa e seu resultado, na seguinte ordem: imagem original; imagem após a operação de limiarização simples; imagem após a operação de limiarização adaptativa

```
import cv2
imgOrig = cv2.imread("sudoku.jpg",0)
imgMediana = cv2.medianBlur(imgOrig, 5)
resultado = cv2.adaptiveThreshold(
imgMediana, 255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 2)
cv2.imshow("Resultado", resultado)
cv2.waitKey(0)
```



Fonte: Elaborado com base em Image, [S.d.].

Na Figura 9, vemos que, como a imagem possui iluminação não uniforme, a binarização por limiar global não dá um resultado satisfatório para todas as regiões da imagem simultaneamente, ao passo que a limiarização adaptativa apresenta um resultado melhor.

TEMA 5 – SEGMENTAÇÃO POR COR

A segmentação por cor é indicada quando a região de interesse ou o fundo podem ser facilmente identificados por uma tonalidade de cor específica. Quando se trata de identificar cores pela sua tonalidade, a melhor forma é trabalharmos com o espaço de cores matiz/saturação/valor (HSV) ao invés de com o esquema vermelho/verde/azul (RGB), por aquele representar o matiz em um único canal. No OpenCV, podemos utilizar o método *inRange* para a segmentação por cor. Ela necessita de três parâmetros: o primeiro é a imagem; o segundo é o limite inferior, o valor mínimo que um determinado *pixel* precisa assumir para ser considerado no intervalo; e o terceiro parâmetro é o limite superior, o valor máximo que o *pixel* pode assumir.

Conforme visto anteriormente, o espaço de cor HSV trabalha com três canais: o primeiro é o matiz, que se refere ao tom da cor; o segundo é a saturação; que é o quão vibrante ou acinzentada a cor está representada; e o terceiro parâmetro é o valor, o quão iluminado ou escuro o *pixel* está representado. A seguir, a Tabela 1 ilustra os valores equivalentes a cada um dos espaços de cor.

Tabela 1 – Cores representadas em diferentes espaços de cor: HSV e RGB

Cor	Nome	(H, S, V)	(R, G, B)
	Preto	(0, 0, 0)	(0, 0, 0)
	Branco	(0, 0, 255)	(255, 255, 255)
	Vermelho	(180, 255, 255)	(255, 0, 0)
	Verde-lima	(60, 255, 255)	(0, 255, 0)
	Azul	(120, 255, 255)	(0, 0, 255)

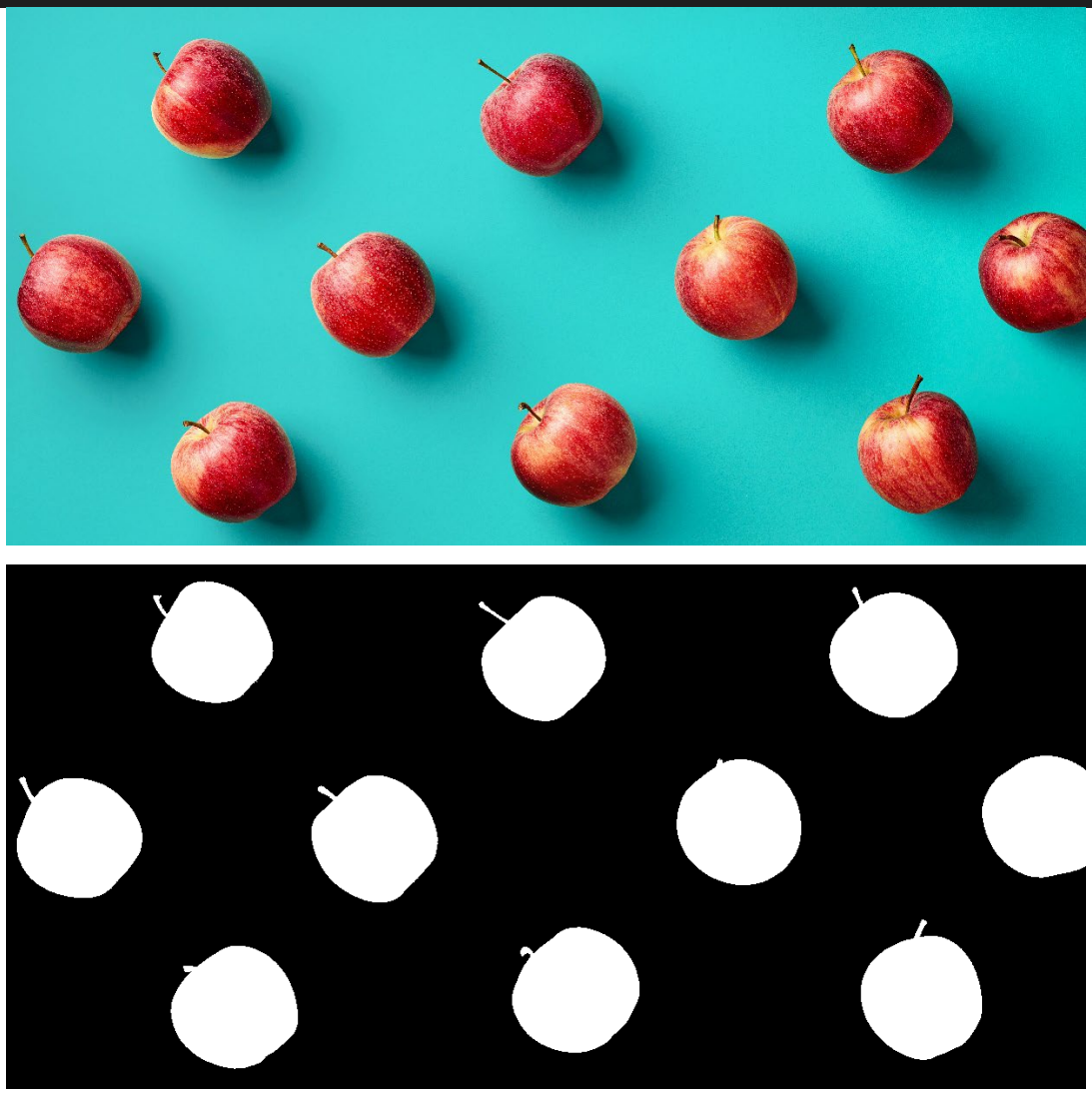
	Amarelo	(30, 255, 255)	(255, 255, 0)
	Ciano	(90, 255, 255)	(0, 255, 255)
	Magenta	(150, 255, 255)	(255, 0, 255)
	Cinza	(0, 0, 128)	(128,128,128)
	Marrom	(180, 255, 128)	(128, 0, 0)
	Verde	(60, 255, 128)	(0, 128, 0)
	Roxo	(150, 255, 128)	(128, 0, 128)
	Azul-marinho	(120, 255, 128)	(0, 0, 128)

Para os limites inferiores e superiores, é necessário passar as três informações. Se queremos, por exemplo, segmentar objetos com tons de vermelho, é interessante considerar o matiz vermelho, que, segundo a Tabela 1, equivale a 180. Como estamos buscando um intervalo, podemos ser mais ou menos restritivos. Mas, de forma geral, optar por 20 para mais ou para menos é interessante, nos levando ao intervalo do **matiz** de 100 a aproximadamente 140. Quanto a **saturação** e **valor**, naturalmente também vão depender da aplicação. Mas, se desejamos obter tons avermelhados bem distintos, ficar entre 100 a aproximadamente 255 é interessante, pois valores abaixo de 100, nesses atributos, tornam objeto muito acinzentado (no caso da saturação) ou escuro (no caso do valor). Para exemplificar, apresentamos na Figura 10 o código para isso e a imagem de teste a ele relacionada.

Figura 10 – Código para operação de segmentação por cor e seu resultado, na seguinte ordem: imagem original; imagem após a operação de segmentação por cor

```
import cv2
import numpy as np
imgOrig = cv2.imread("maca.jpeg")
imgHSV = cv2.cvtColor(imgOrig, cv2.COLOR_BGR2HSV)
minimo = np.array([80, 150, 30])
```

```
maximo = np.array([120, 255, 255])
imgFinal= (255 - cv2.inRange(imgHSV,minimo,maximo))
cv2.imshow("Resultado", imgFinal)
cv2.waitKey(0)
```



Créditos: Baibaz/Shutterstock.

No exemplo da Figura 10, como as maçãs combinam diferentes tons – vermelho, amarelado, marrom –, é mais interessante segmentar o fundo, pois ele é constantemente azul. Note que o fundo possui a sombra das maçãs, o que poderia ser um problema para uma segmentação que utilizasse RGB. No entanto, no espaço de cor HSV, é possível estabelecer um largo intervalo para o atributo *valor* (entre 30 a aproximadamente 255). Assim, tanto azul com maior quanto azul com menor intensidade de luz serão ao mesmo tempo selecionados, no intervalo especificado. No entanto, como o interesse são as maçãs e não o fundo, basta inverter o resultado com uma simples operação de: **255 - imagem**.

FINALIZANDO

Nesta aula, discutimos dois tópicos de grande importância: as operações morfológicas, que consistem em operações, na imagem, que modificam a área de interesse, entre outras finalidades; e a correção de ruídos que auxiliam na segmentação, que foi o segundo tópico principal desta aula, e consiste em identificar a região de interesse na imagem por meio de alguma característica saliente, como a cor.

Na próxima aula, vamos discutir como extrair características desses objetos segmentados, para finalmente podermos obter informações relevantes para as mais diversas aplicações.

REFERÊNCIAS

IMAGE thresholding. **OpenCV**, [S.d.]. Disponível em: <https://docs.opencv.org/4.5.1/d7/d4d/tutorial_py_thresholding.html>. Acesso em: 20 dez. 2021.