iscte
TECNOLOGIAS
E ARQUITETURA

APVC

Operações morfológicas
Convolução e filtros

Sumário

- Operações morfológicas
  - Dilatação e erosão
  - Abertura e fecho
  - Exemplos
- Convolução
  - Conceitos introdutórios
  - Convolução em imagens, passo-a-passo
  - Gradiente e deteção de contornos
  - Redução de ruído
  - Aumento da nitidez
  - Exemplos

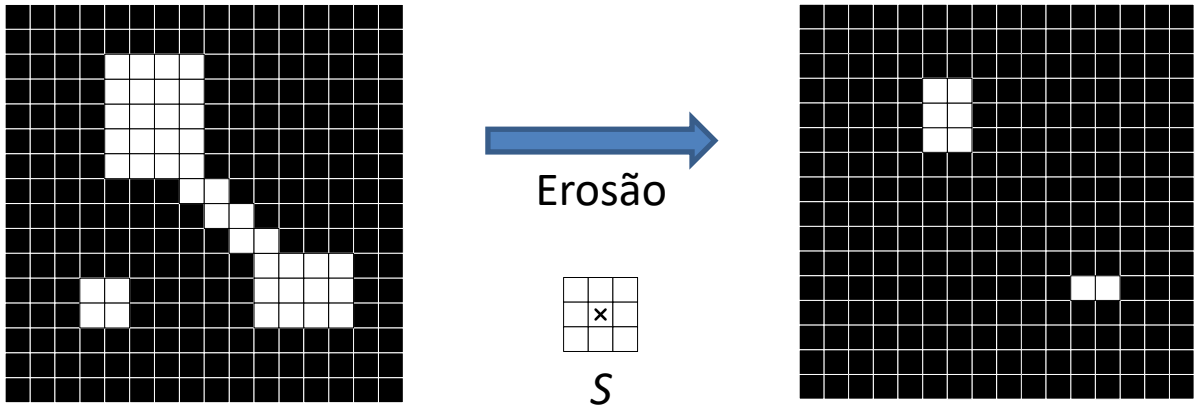
iscte
TECNOLOGIAS
E ARQUITETURA

Operações morfológicas

Operações morfológicas

- Operações não lineares que podem ser aplicadas tanto a imagens binárias (preto/branco) como a imagens em níveis de cinzento
- Quando aplicadas a imagens binárias:
  - remover ruído (e.g., remover ruído em máscaras "ruidosas")
  - preenchimento de "buracos" em regiões de pixéis brancos (componentes conexos)
  - …
- As operações morfológicas fazem uso de um “template”, designado por ***elemento estruturante***, cujas dimensões são normalmente muito menores que as da imagem.
- As operações morfológicas elementares são a ***erosão*** e a ***dilatação***.

Erosão

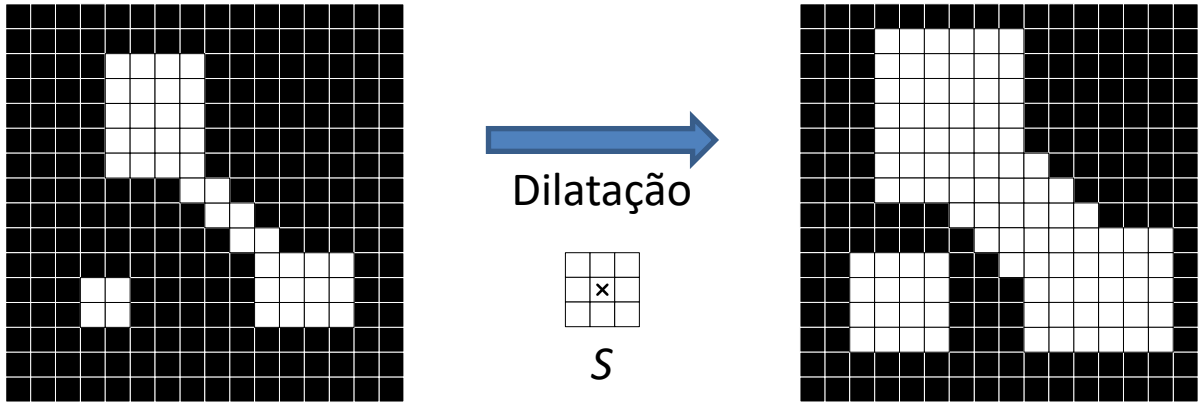


$$(I \ominus S)(x,y) = \begin{cases} 1 & se\ S \subseteq I \\ 0 & caso\ contrário \end{cases}$$

Resultado quando o elemento estruturante *S* se encontra centrado no pixel (*x*,*y*) da imagem:

- 1 – caso todos os pixéis da imagem estejam a '1' na zona de sobreposição com *S* (*i.e.*, o elemento estruturante "cabe" na zona de imagem que está a '1')
- 0 – quando há pelo menos um '0' da imagem sobreposto com *S*

Dilatação



$$(I \oplus S)(x,y) = \begin{cases} 1 & se\ I \cap S \neq \emptyset \\ 0 & caso\ contrário \end{cases}$$

Resultado quando o elemento estruturante *S* se encontra centrado no pixel (*x*,*y*) da imagem:

- 1 – quando existe pelo menos um pixel da imagem a ‘1’ na sobreposição com *S*
- 0 – quando todos os pixels da imagem sobrepostos com *S* estão a ‘0’

Dilatação e erosão – efeitos

- Erosão
  - "encolhe" os objetos
  - remove objetos pequenos
  - pode separar objetos

Imagem original

Binarização

- Dilatação
  - "alarga" os objetos
  - preenche "buracos" no interior dos objetos
  - objetos próximos podem ficar ligados

Dilatação

Erosão

Dilatação e erosão no OpenCV

- Definir um *elemento estruturante* – `getStructuringElement(…)`

```
# Forma circular/elíptica
strel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7,7))

# Forma quadrangular/rectangular
strel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))

# Forma de cruz
strel = cv2.getStructuringElement(cv2.MORPH_CROSS, (7,7))
```

Dimensões do elemento estruturante (largura, altura)

- Aplicar dilatação ou erosão – `dilate(…) e erode(…)`

```
# Dilatação
dilatedImg = cv2.dilate(img, strel)
```

```
# Erosão
erodedImg = cv2.erode(img, strel)
```

Abertura e Fecho

- Operações morfológicas compostas
  - Abertura – Erosão seguida de dilatação
    - Remove ruído fora dos objetos e remove objetos pequenos
    - Pode também separar objetos (quando estão “pegados” através de ligações “finas”)
    - $I \circ S = (I \ominus S) \oplus S$
  - Fecho – Dilatação seguida de erosão
    - Remove ruído dentro dos objetos e preenche buracos no interior dos objetos
    - Pode juntar objetos que estejam próximos
    - $I \bullet S = (I \oplus S) \ominus S$

Abertura e Fecho no OpenCv

- Abertura

```
# Abertura usando o elemento estruturante strel
imgOpened = cv2.morphologyEx(img, cv2.MORPH_OPEN, strel)
```

- Fecho

```
# Fecho usando o elemento estruturante strel
imgClosed = cv2.morphologyEx(imgBin, cv2.MORPH_CLOSE, strel)
```

Imagem binária

Abertura

Fecho

Outras operações morfológicas

- Fronteiras (Gradiente morfológico)

Dilatação

Imagem binária

Gradiente morfológico

- Operações morfológicas sobre níveis de cinzento

Original

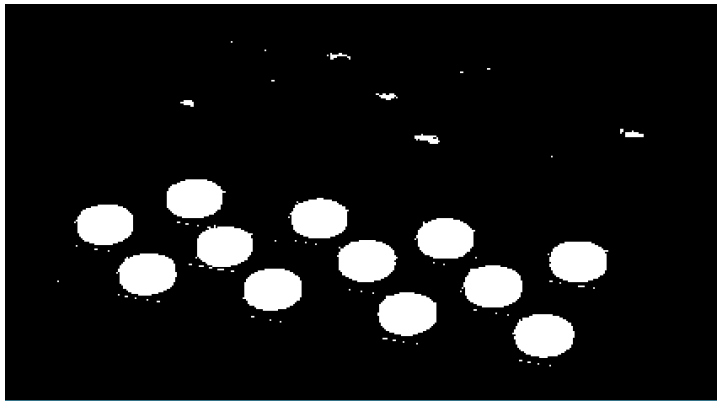
Dilatação

Erosão

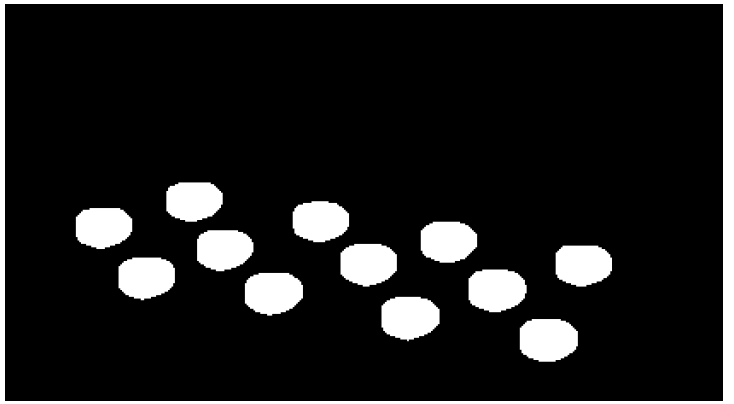
Exemplo de utilização

Binarização
(com base na cor)

Máscara (ruidosa)

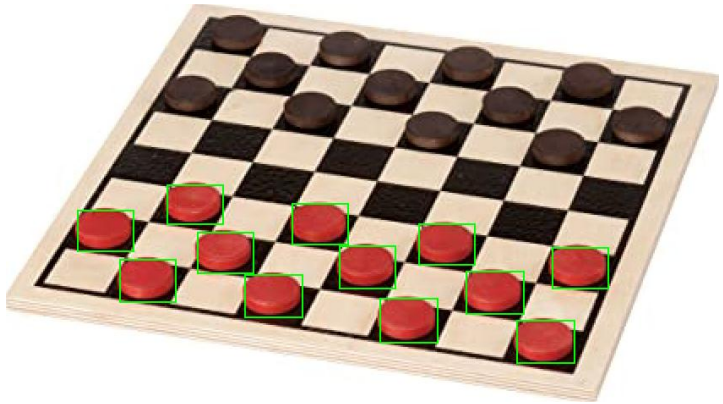


Abertura



Máscara (ruído removido)

Localização
(componentes conexos)



iscte
TECNOLOGIAS
E ARQUITETURA

Convolução

Convolução – Introdução

- Operação linear, muito importante em processamento de sinal
  - incluindo processamento de imagens!
- Em imagem, a convolução serve para diversas tarefas, entre as quais:
  - filtragem de ruído
  - estimação do gradiente (e, consequentemente, deteção dos contornos)
  - aumentar a nitidez / contraste
  - …
- Mais à frente, vamos ver também que a convolução tem um papel muito importante na aprendizagem profunda (*deep learning*)
  - é uma operação fundamental nas redes neuronais convolucionais (CNN's)

Convolução – Definição matemática

- Caso geral – convolução (*) entre um sinal s e um filtro (*kernel*) f
  - Contínuo $$(s * f)(x) = \int_{-\infty}^{+\infty} s(\tau) f(x - \tau) d\tau$$
  - Discreto $$(s * f)(x) = \sum_{m=-\infty}^{\infty} s(m) f(x - m)$$
- Caso específico das imagens
  - Para simplificar assume-se que queremos realizar a convolução de uma imagem I com um filtro f (*kernel*) de dimensões *M* x *N (*com *M* e *N* impares*)*

$$(I * f)(x, y) = \sum_{m=-a}^{a} \sum_{n=-b}^{b} I(x + m, y + n) f(m + a, n + b) \qquad a = \left\lfloor \frac{M}{2} \right\rfloor, b = \left\lfloor \frac{N}{2} \right\rfloor$$

Convolução passo-a-passo

-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	-1	-1	-1
-1	-1	1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1

$\odot$

1	-1	-1
-1	1	-1
-1	-1	1

-1x1 + -1x-1 + -1x-1 +
-1x-1 + 1x1 + -1x-1 +
-1x-1 + -1x-1 + 1x1 = **7**

7				

Para cada posição (x,y), centra-se o filtro nesse ponto; multiplicam-se ponto-a-ponto os coeficientes do filtro pelos valores de imagem correspondentes; o resultado da convolução será a soma dessas multiplicações

Convolução passo-a-passo

-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	-1	-1	-1
-1	-1	1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1

$\odot$

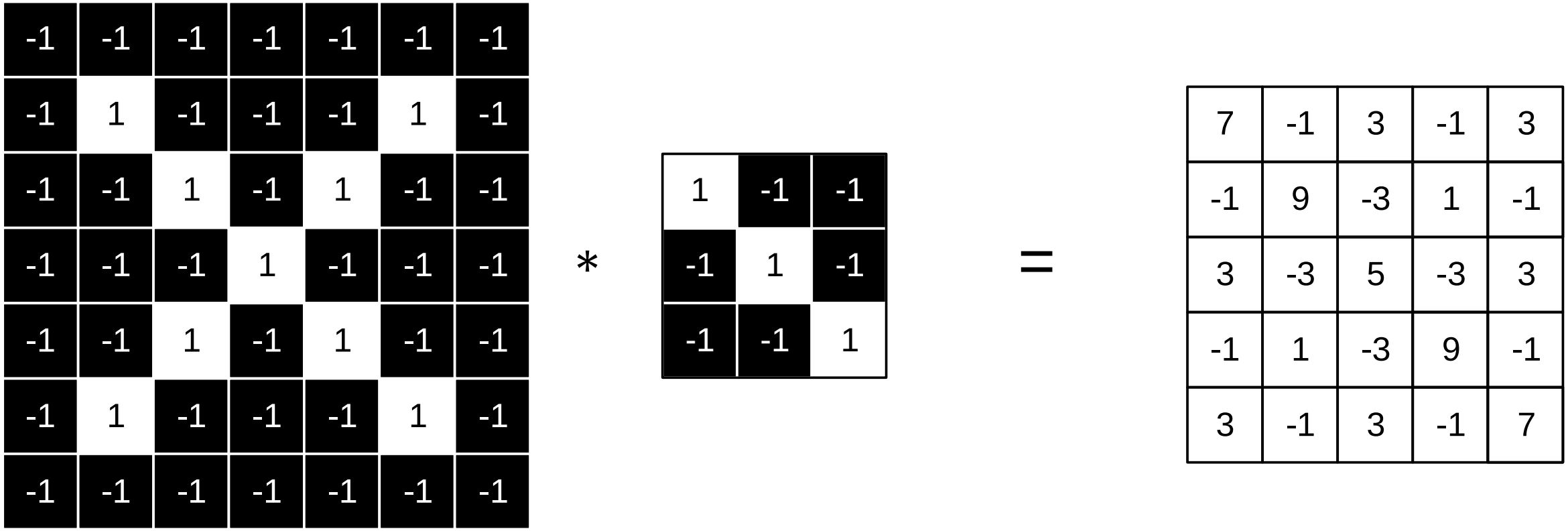
1	-1	-1
-1	1	-1
-1	-1	1

-1x1 + -1x-1 + -1x-1 +
1x-1 + -1x1 + -1x-1 +
-1x-1 + 1x-1 + -1x1 = **-1**

7	-1			

Vai-se deslizando o filtro por todas as posições possíveis e aplicando as multiplicações / soma.

Convolução passo-a-passo



O resultado final da convolução obtém-se deslizando o filtro por todas as posições possíveis na imagem. A não ser que se utilize *padding* (i.e., acrescentar artificialmente linhas/colunas à imagem), as dimensões da matriz do resultado são inferiores às da imagem, dado que não se consegue encaixar o filtro nos pontos situados nas bordas da imagem.

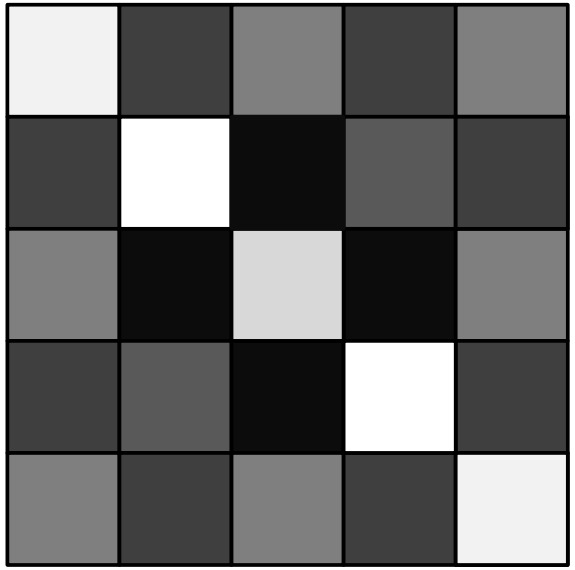
Convolução passo-a-passo

-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	-1	-1	-1
-1	-1	1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1

*

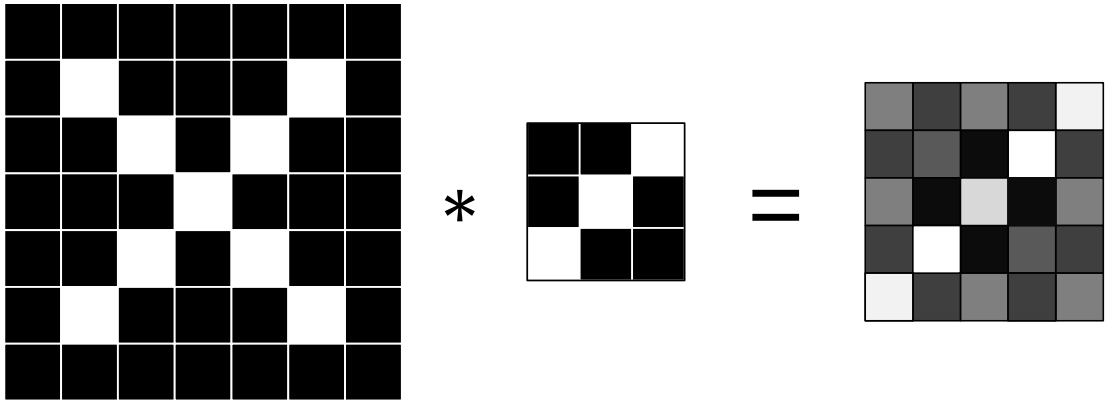
1	-1	-1
-1	1	-1
-1	-1	1

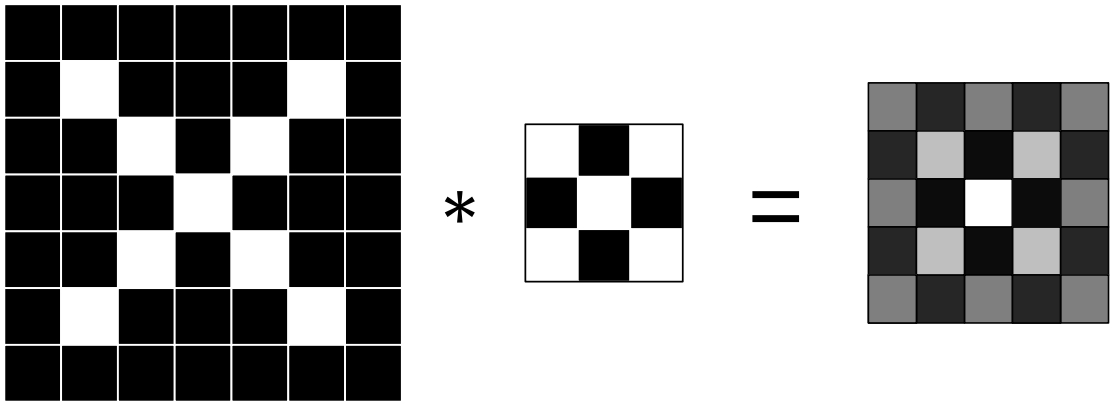
=



O resultado da convolução pode ser visualizado como uma imagem, normalizando os valores para a gama de 0…255

Resultados da convolução com outros kernels





Convolução no OpenCV

- Embora existam mais possibilidades, a forma mais genérica de realizar a convolução é usando a função `filter2D(…)`

```
# definição de um filtro (neste caso é um filtro de média)
meanKernel = np.array((
   [1, 1, 1],
   [1, 1, 1],
   [1, 1, 1]), dtype=np.float32) / 9

# aplicar a convolução com o filtro definido à imagem img
filteredImg = cv2.filter2D(img, None, meanKernel)

# Nota: o 2º parâmetro da função pode ser usado para indicar o formato numérico do output
# usar '-1' ou 'None' significa que o formato numérico do output será o mesmo que o da imagem de input
```

Algumas aplicações da convolução

Gradiente e deteção de contornos

-1	0	1
-2	0	2
-1	0	1

Convolução com *kernel de Sobel* segundo direção horizontal -> componente *x* do gradiente (G_x)

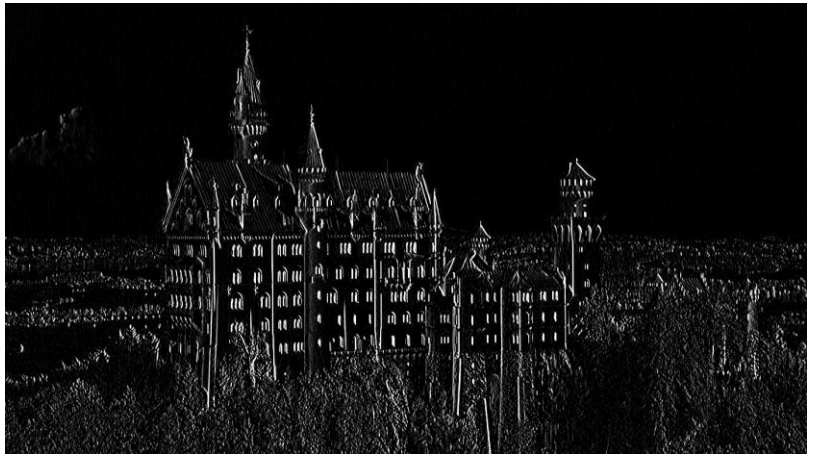


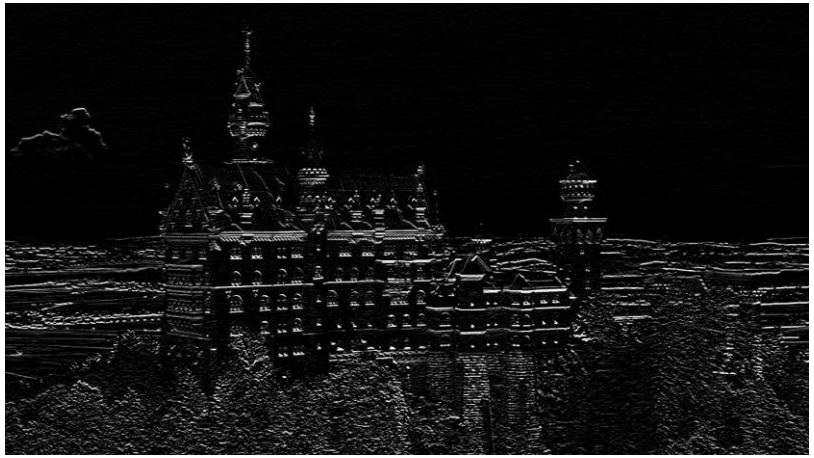
Imagem original

Gradiente de imagem (∇I): Derivadas ao longo das direções horizontal e vertical

$$\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} = \begin{bmatrix} G_x \\ G_y \end{bmatrix}$$

-1	-2	-1
0	0	0
1	2	1

Convolução com *kernel de Sobel* segundo direção vertical -> componente *y* do gradiente (G_y)



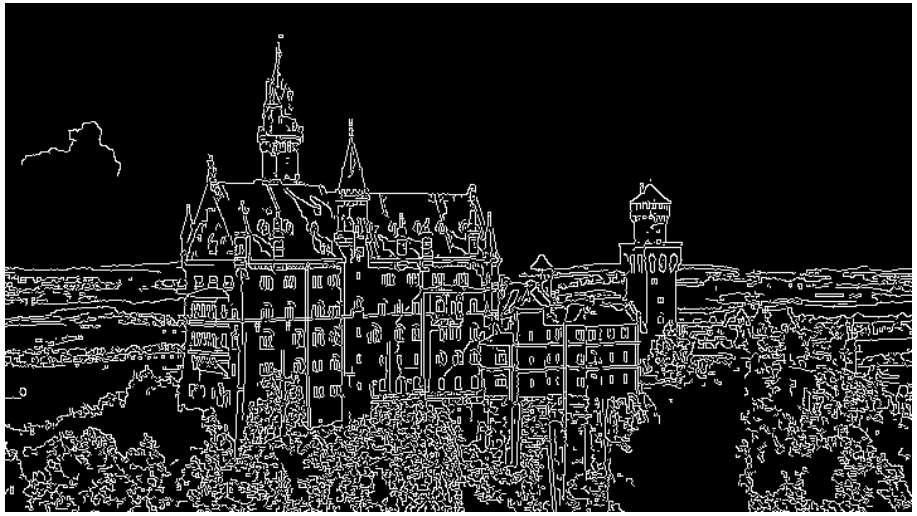
Gradiente e deteção de contornos

Magnitude do gradiente

$$G_{mag} = \sqrt{G_x^2 + G_y^2}$$

Binarização (Otsu)

Algoritmo de *Canny*



Aumentar a nitidez de imagens (*Sharpen*)

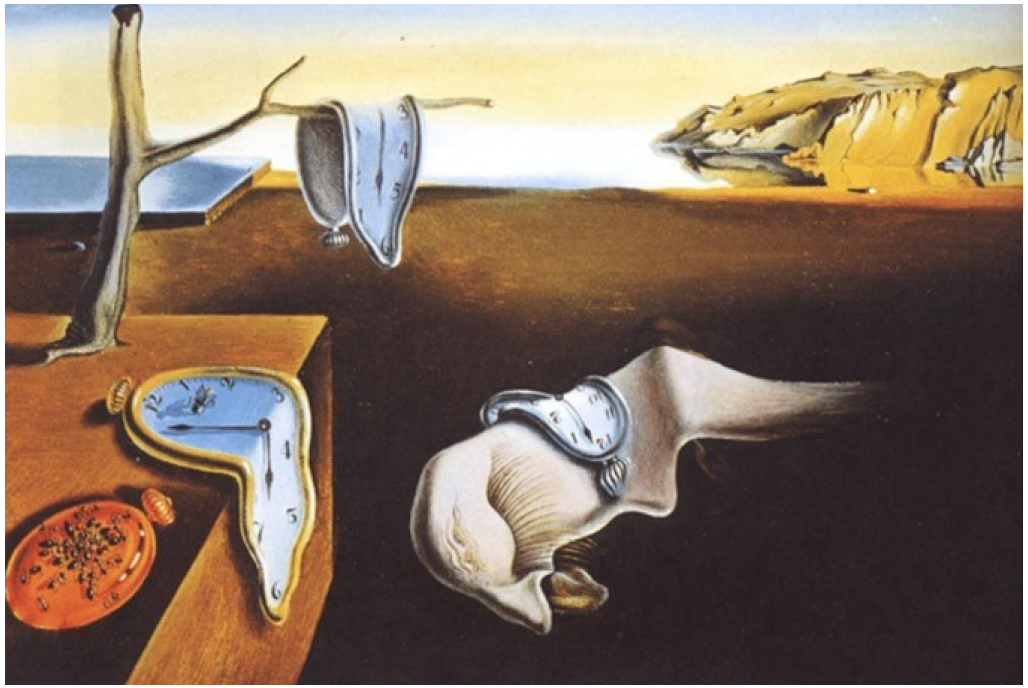
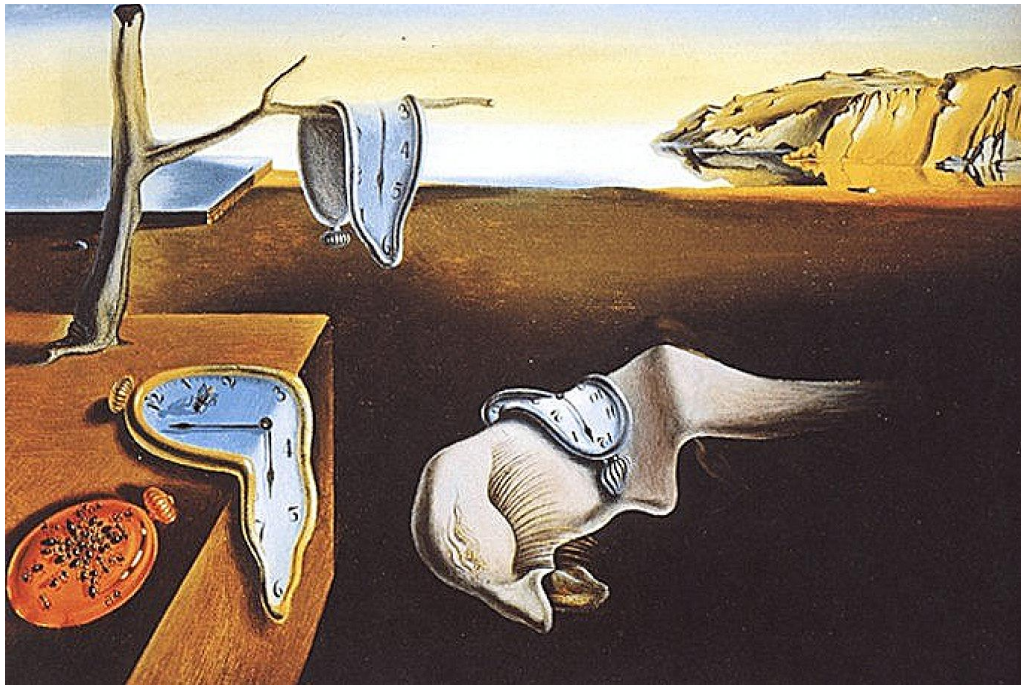


Imagem original



Convolução com o seguinte *kernel*:

0	-1	0
-1	5	-1
0	-1	0

Redução de ruído

Imagem com ruído

Convolução com *kernel* gaussiano 3x3

1	2	1
2	4	2
1	2	1

$$\times \frac{1}{16}$$

Redução de ruído

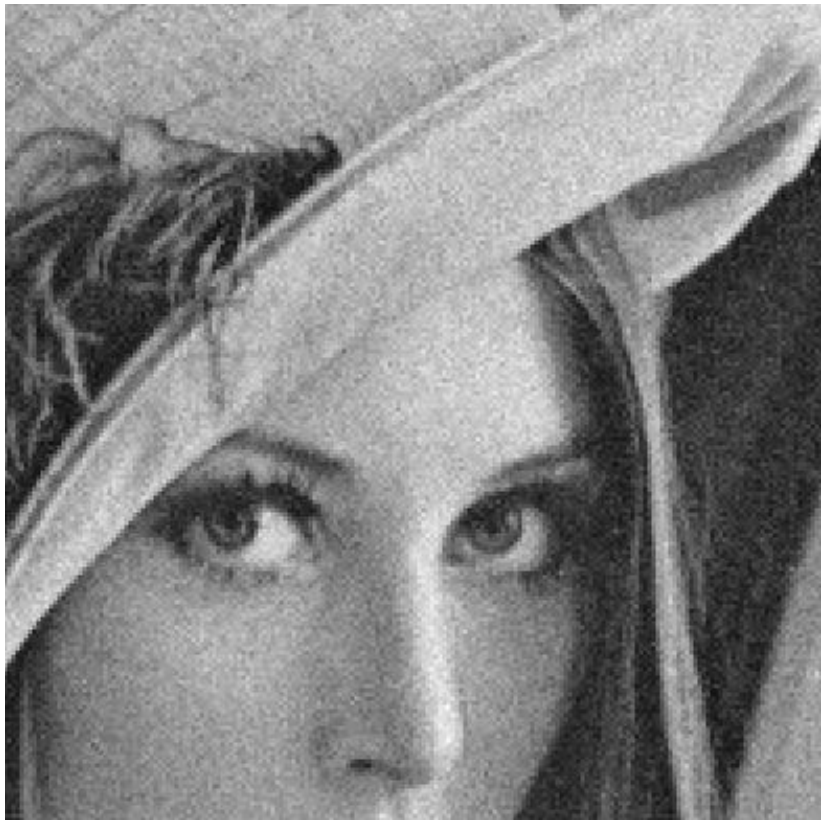


Imagem com ruído

Convolução com *kernel* gaussiano 3x3

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16}$$

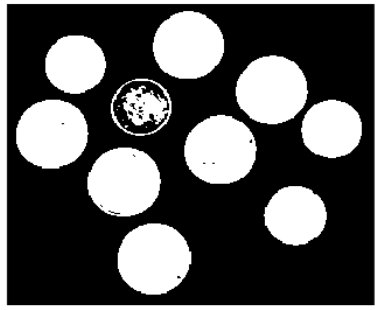
iscte
TECNOLOGIAS
E ARQUITETURA

Extras

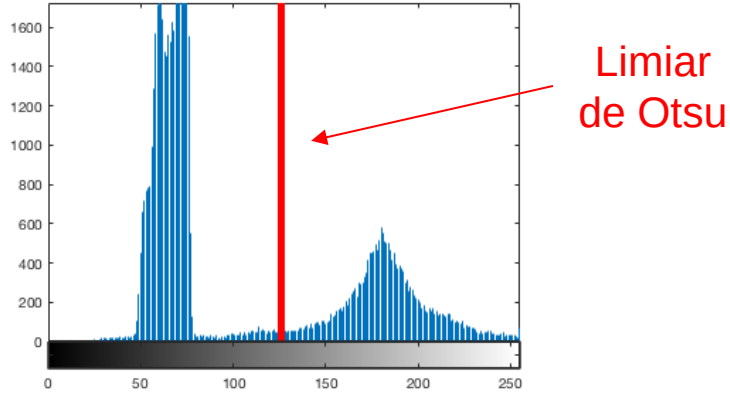
Binarização – Método de Otsu

- Método adaptativo onde o valor de limiar (*threshold*) para binarização é obtido automaticamente
  - Assume que existem duas zonas (classes) na imagem – *foreground* e *background* – com características distintas nos valores de luminância (ou numa componente da cor)
  - O valor de limiar obtido corresponde ao valor que separa as classes minimizando a *variância intra-classe* dos valores do pixels

Imagem original (lum.)



Binarização (Otsu)



Histograma da luminância

  - No OpenCV:

```
img = cv2.imread("images/think.jpg", cv2.IMREAD_GRAYSCALE)
(t, imgBin) = cv2.threshold(img, 0, 255, cv2.THRESH_OTSU)
```

OpenCV – Componentes conexos

Para obter estatísticas sobre as várias regiões a branco (*componentes conexos*) que estão presentes numa imagem binária (imgBin) pode-se utilizar:

```
(numLabels, labels, boxes, centroids) = cv2.connectedComponentsWithStats(imgBin)
```

Significado:

- `numLabels` – número de componentes conexos
- `labels` – Id's dos componentes
- `boxes` – *bounding boxes* - coordenadas do canto superior esquerdo, altura e largura dos retângulos envolventes
- `centroids` – coordenadas dos centróides dos componentes conexos

OpenCV – Convoluções “especializadas”

O OpenCV inclui diversas funções que realizam convoluções “especializadas”, onde não é necessário definir o *kernel*. Exemplos:

```
# blur – aplica um filtro de média com as dimensões indicadas
filtered = cv2.blur(img, (5,5))

# GaussianBlur – aplica um filtro gaussiano com as dimensões indicadas
filtered = cv2.GaussianBlur(img, (5,5))

# Sobel – aplica um filtro de Sobel na direção indicada
Gx = cv2.Sobel(img, -1, dx=1, dy=0)
Gy = cv2.Sobel(img, -1, dx=0, dy=1)

# ...
```

OpenCV – Formatos numéricos em `filter2D(…)`

Por vezes é necessário ter cuidado com o formato numérico resultante de uma convolução com `filter2D(…)`

Assumindo uma imagem `img` com valores inteiros na gama [0, 255], o formato numérico correspondente é 8 bits sem sinal – **`np.uint8`** (numpy) ou **`cv2.CV_8U`** (OpenCV)

Ao realizarmos

```
filtered = cv2.filter2D(img, None, kernel)
```

o parâmetro da função a **`None`** (ou a `-1`) faz com que o formato numérico do resultado seja igual ao de `img`.

Assim, todos os valores resultantes que calhassem fora de [0, 255] ficariam saturados em 0 ou 255, consoante fossem menores que 0 ou superiores a 255, respetivamente.

Mas em muitas situações pretendem-se obter os valores reais do resultado, e não valores restringidos a [0, 255]. Para isso será necessário especificar um formato numérico apropriado no 2º parâmetro da função:

```
filtered = cv2.filter2D(img, cv2.CV_32F, kernel)
```

Neste caso o resultado seria dado numa matriz de *floats* de 32 bits (**`cv2.CV_32F`**, equivalente a **`np.float32`**), permitindo obter os valores reais do resultado e sem restrições à gama [0, 255]

OpenCV – Visualização de matrizes

Em processamento de imagem é muito habitual visualizarmos uma matriz 2D que na realidade não é uma imagem, pois não tem valores de pixel entre 0 e 255 (por exemplo, visualizar a magnitude do gradiente)

Nessas situações, para visualizar o conteúdo de uma matriz `mat` usando `imshow(…)`, é necessário normalizar primeiros os seus valores, de modo a que estes fiquem na gama [0, 255] (inteiros) ou então na gama [0, 1.0] (reais)

Para realizar essa tarefa pode-se usar a função `normalize(…)` do openCV:

```
mat_norm= cv2.normalize(mat, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)    # normaliza para [0, 255], inteiros

mat_norm= cv2.normalize(mat, None, 0, 1.0, cv2.NORM_MINMAX, cv2.CV_32F)    # normaliza para [0, 1.0], float
cv2.imshow("Visualização de mat", mat_norm)
```

Noutras ocasiões, quando se sabe *a priori* que a matriz contém apenas valores positivos ou a zero, para normalizar na gama [0, 1.0] basta dividir todos os valores pelo valor máximo:

```
mat_norm = mat / mat.max()
cv2.imshow("Visualização de mat", mat_norm)
```

Recursos

- Imagens
  - “Think”, https://unsplash.com/s/photos/text
  - “Damas”, https://www.amazon.com/Basic-Checker-Set-Made-USA/dp/B001PF83HG
  - “Lenna”, https://en.wikipedia.org
  - “A persistência da memória, Dali”, https://www.culturagenial.com/a-persistencia-da-memoria-de-salvador-dali/
  - “Castelo Neuschwanstein”, https://www.euronews.com/travel/2021/06/28/14-castles-in-europe-that-are-straight-out-of-a-fairy-tale
  - Método de Otsu, https://blogs.mathworks.com/steve/2016/06/14/image-binarization-otsus-method/
- 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