

# **Assegnamento 1**

## **31-10-2018**

**Corso di Visione Artificiale**  
**A.A. 2018/2019**

# Regole

- L'assegnamento vale **2** punti se consegnato entro **3** settimane
- L'assegnamento vale **1** punto se consegnato oltre la terza settimana

# Regole

- Funzioni OpenCv consentite:
  - cv::Mat

Tutti i costruttori

Operatori =, +, -, \*

clone(), create(...)

data, rows, cols

channels(), type()

elemSize(), elemSize1()

setTo(), zeros()

# Regole

- Funzioni OpenCv consentite:

- I/O

- `imread(...)`

- `imshow(...)`

- `waitKey(...)`

- `namedWindow(...)`

# Esercizio 1

- **Funzione di convoluzione *float***

```
int convFloat(const cv::Mat& image, const  
cv::Mat& kernel, cv::Mat& out)
```

- image: singolo canale uint8
- kernel: singolo canale float32
- out: singolo canale float32



- Il kernel deve essere *simmetrico*, cioè di dimensioni dispari
- Il kernel puo' essere *1-D* o *2-D* indifferentemente
- I pixel per i quali non è possibile applicare il kernel (bordo) vanno messi a zero

# Esercizio 1bis

- **Funzione di convoluzione**

```
int conv(const cv::Mat& image, const  
cv::Mat& kernel, cv::Mat& out)
```

- **image:** singolo canale uint8
  - **kernel:** singolo canale float32
  - **out:** singolo canale uint8; **se necessario,**  
riscalare opportunamente out nel range [0-255]  
(vedi **contrast stretching**)
- 
- **Nota 1:** si ricava direttamente dal precedente  
passando da float ad intero (ed eventualmente  
scalando).

# Esercizio 2

- Scrivere una funzione che generi un **kernel** di blur Gaussiano **1-D orizzontale**

```
int gaussianKernel(float sigma, int radius,  
cv::Mat& kernel)
```

- kernel: singolo canale float32
- sigma: deviazione standard della gaussiana
- radius: raggio del kernel, quindi la dimensione del kernel sarà **(2\*radius)+1**

- Normalizzare il kernel dividendo per la somma dei pesi

# Esercizio 3

- Utilizzando le funzioni precedenti, applicare i seguenti filtri all'immagine lenan.pgm e visualizzare i risultati
  1. Gaussian blur *orizzontale*
  2. Gaussian blur *verticale* 
  3. Gaussian blur *bidimensionale*
- Nota 1: un kernel verticale si ottiene da quello orizzontale trasponendo
- Nota 2: il Gaussian blur bidimensionale si ottiene come filtro separabile dei precedenti

# Esercizio 4

- Utilizzando le funzioni precedenti, applicare all'immagine lenna.pgm i seguenti filtri e visualizzare i risultati:
  1. Filtro *derivativo Gaussiano* orizzontale
  2. Filtro *derivativo Gaussiano* verticale
  3. Filtro Laplaciano
$$\begin{matrix} 0.0, & 1.0, & 0.0 \\ 1.0, & -4.0, & 1.0 \\ 0.0, & 1.0, & 0.0 \end{matrix}$$
- Nota 1: i derivativi Gaussiani possono essere ottenuti come composizione di un Gaussiano e di un derivativo monodimensionale  $[-1,0,1]$ , oppure creando direttamente il kernel DoG.
- Nota 2: le grandezze ottenute saranno sia positive che negative. Per poterle visualizzare, riportare tutto nel range  $[0,255]$ ; 128 rappresenta lo 0,  $[0,127]$  i valori negativi,  $[129,255]$  quelli positivi.

# Esercizio 5

- Utilizzando le funzioni precedenti, scrivere una funzione che calcoli **magnitudo** e **orientazione** [0, 2PI] di **Sobel 3x3** e visualizzare i risultati:

```
int sobel(const cv::Mat& image, cv::Mat&
magnitude, cv::Mat& orientation)
```

- **image:** singolo canale uint8
- **magnitude:** singolo canale float32
- **orientation:** singolo canale float32 tra 0 e 2PI

# Esercizio 5

- Per visualizzare l'orientazione di Sobel con il seguente codice OpenCv:

```
cv::Mat adjMap;  
cv::convertScaleAbs(orientation, adjMap, 255 / 2*M_PI);  
cv::Mat falseColorsMap;  
cv::applyColorMap(adjMap, falseColorsMap,  
cv::COLORMAP_AUTUMN);  
cv::imshow("Out", falseColorsMap);
```

# Esercizio 6

- Realizzare una funzione di **bilinear interpolation**:

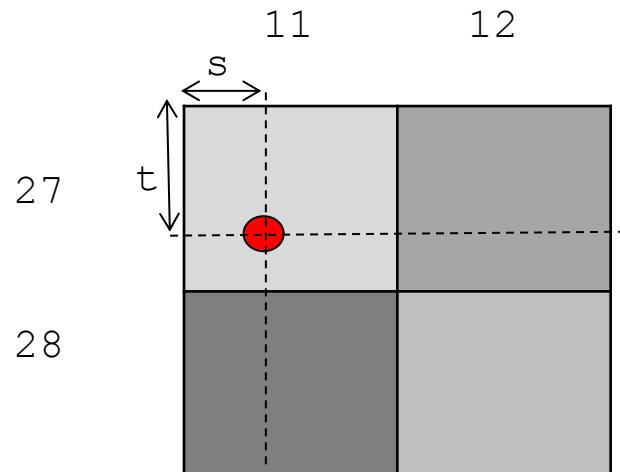
```
float bilinear(const cv::Mat& image, float r, float c)
```

- image: singolo canale uint8
  - $r$  e  $c$  valori float compresi in  $[0, \text{rows}-1]$  e  $[0, \text{cols}-1]$
- Il risultato sarà un singolo valore float ottenuto interpolando i 4 vicini di  $(r,c)$

$$f(I) = (1 - s)(1 - t)f_{00} + s(1 - t)f_{10} + (1 - s)tf_{01} + stf_{11}$$

# Esercizio 6

- Es.  $(r,c) = (27.8, 11.4)$



- $\text{out}(r,c) = \text{in}(11,27)*0.6*0.2 + \text{in}(12,27)*0.4*0.2 + \text{in}(11,28)*0.6*0.8 + \text{in}(12,28)*0.4*0.8$

# Esercizio 7

- Find Peaks of Edge Responses: trovare i picchi (massimi) del gradiente nella direzione **perpendicolare** al gradiente stesso

```
int findPeaks(const cv::Mat& magnitude, const cv::Mat&
               orientation, cv::Mat& out, float th)
```

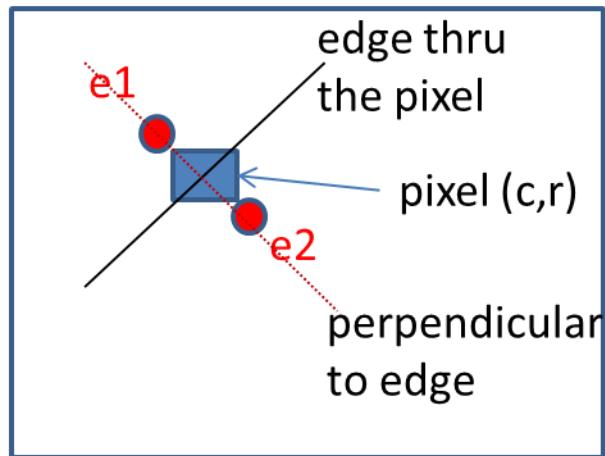


- magnitude: singolo canale float32
- orientation: singolo canale float32
- out: singolo canale float32

- Confrontare ogni pixel dell'immagine magnitudo con i vicini  $e_1$ ,  $e_2$  a distanza **1.0 lungo la direzione perpendicolare al gradiente**. Utilizzare la funzione bilinear del passo precedente.
- Soppressione dei non massimi:

$$out(r, c) = \begin{cases} in(r, c) & se \quad in(r, c) \geq e1 \wedge in(r, c) \geq e2 \wedge in(r, c) \geq th \\ 0 & altrimenti \end{cases}$$

# Esercizio 7



$$e1x = c + 1 * \cos(\theta);$$

$$e1y = r + 1 * \sin(\theta);$$

$$e2x = c - 1 * \cos(\theta);$$

$$e2y = r - 1 * \sin(\theta);$$

Example:  $r=5, c=3, \theta=135$  degrees  
 $\sin \theta = .7071, \cos \theta = -.7071$   
 $e1 = (2.2929, 5.7071)$   
 $e2 = (3.7071, 4.2929)$

# Esercizio 8

- **Soglia con isteresi:**

```
int doubleTh(const cv::Mat& magnitude,  
cv::Mat& out, float th1, float th2)
```

- magnitude: singolo canale float32
- out: singolo canale uint8

- $$out(r, c) = \begin{cases} 255 & se in(r, c) > th1 \\ 128 & se th1 \geq in(r, c) > th2 \\ 0 & se in(r, c) \leq th2 \end{cases}$$

# Esercizio 9

- **Canny Edge Detector su lenna.pgm**

```
int canny(const cv::Mat& image, cv::Mat&
out, float th, float th1, float th2)
```

- image: singolo canale uint8
- out: singolo canale uint8

1. Sobel magnitudo e orientazione
2. Non-Maximum Suppression (findPeaks) della magnitudo
3. Sogliatura con isteresi

# Nota

- Per ottenere un Canny Edge Detector completo sarebbe necessario effettuare il collegamento degli egde (edge linking).
- Esercizio opzionale non valutato.