Elaborazione di Segnali Multimediali

Filtraggio spaziale
Soluzioni

1 Filtri di smoothing

1. *Smoothing seguito da thresholding*.

```
x = np.float64(io.imread('spazio.jpg'))
plt.figure(1); plt.imshow(x, clim=[0,255],cmap='gray');

y = ndi.uniform_filter(x, (15,15))
plt.figure(2); plt.imshow(y, clim=[0,255],cmap='gray');

th = (25/100)*np.max(y)  # soglia = 25 per cento del valore max
z = y>th  # calcolo la maschera
plt.figure(3); plt.imshow(z, clim=[0,1],cmap='gray');

w = z*x;  # risultato dell'elaborazione
plt.figure(4); plt.imshow(w, clim=[0,255],cmap='gray');
```

2. *Denoising*. Aggiungiamo rumore gaussiano ad un'immagine e poi valutiamo l'effetto di un filtro media aritmetica al variare della dimensione della finestra calcolando l'MSE tra immagine originale e ripulita.

```
x = np.float64(io.imread('lena.jpg'));
plt.figure(1); plt.imshow(x, clim=[0,255],cmap='gray');
plt.title('originale');
M,N = x.shape
noisy = x + 20*np.random.randn(M,N)
print('MSE della rumorosa:', np.mean((x-noisy)**2))
plt.figure(2); plt.imshow(noisy, clim=[0,255],cmap='gray');
plt.title('rumorosa');

list_k = [3,5,7,9]
mse = np.zeros(len(list_k))
for index in range(len(list_k)):
    k = list_k[index]
    y = ndi.uniform_filter(noisy, (k,k))
    plt.figure();
    plt.imshow(y, clim=[0,255],cmap='gray');
    plt.title('filtrata per k=%d' % k);
    mse[index] = np.mean((x-y)**2)

plt.figure(10);
plt.plot(list_k,mse,'r-*');
plt.xlabel('dimensione della finestra');
plt.ylabel('MSE');
```

3. *Filtraggio spaziale adattativo*. Innanzitutto scriviamo uno script che richiama la funzione che realizza il filtraggio adattativo (`filtra`) al variare della deviazione standard del rumore e produce il grafico dell'MSE confrontandolo con quello del filtro media aritmetica:

```
M, N = x.shape
sigma = [5,10,15,20,25,30,35]
mse_noisy = np.zeros(len(sigma))
mse_media = np.zeros(len(sigma))
mse_adatt = np.zeros(len(sigma))
for i in range(len(sigma)):
    y = x + sigma[i]*np.random.randn(M,N)
    xr_adatt = filtra(y, sigma[i])
    xr_media = ndi.uniform_filter(y, (7,7))
    mse_noisy[i] = np.mean((y  - x)**2)
    mse_adatt[i] = np.mean((xr_adatt-x)**2)
    mse_media[i] = np.mean((xr_media-x)**2)

plt.figure();
plt.plot(sigma, mse_noisy,'r-o')
plt.plot(sigma, mse_media,'b-o'); plt.plot(sigma, mse_adatt,'g-o')
plt.xlabel('sigma rumore'); plt.ylabel('MSE')
plt.legend(['rumorosa','filtro media', 'filtro adattativo'])
```

Per realizzare il filtraggio il codice può essere scritto in diversi modi. Cominciamo con la soluzione che usa la funzione `generic_filter` specificando la funzione da eseguise su ogni blocco:

```
def filtra(x, sigma):
    sn2 = sigma**2

    def adatt(b):
        vr = np.var(b)
        ml = np.mean(b)
        yb = b[3,3] - (sn2/vr)*( b[3,3] - ml)
        return yb

    y = ndi.generic_filter(x, adatt, (7,7))
    return y
```

In alternativa si possono calcolare la matrice delle medie e delle varianze locali e poi procedere al filtraggio direttamente sull'immagine come matrice:

```
def filtra(x, sigma):
    sn2 = sigma**2
    MED = ndi.uniform_filter(x, (7,7))
    VAR = ndi.uniform_filter(x**2, (7,7)) - (MED**2)

    y = x - (sn2/VAR) * (x - MED)
    return y
```

Confrontate i tempi di elaborazione delle due soluzioni.

2 Esercizi proposti

1. *Rumore sale e pepe.*

```
x = np.float64(io.imread('peppers.png'))
plt.figure(1); plt.imshow(x, clim=[0,255],cmap='gray');
plt.title('originale');

from skimage.util import random_noise
noisy = random_noise(x/255, 's&p') *255
# Nota che: la funzione random_noise richiede l'immaine nel range [0,1]

print('MSE della rumorosa:', np.mean((x-noisy)**2))
plt.figure(2); plt.imshow(noisy, clim=[0,255],cmap='gray');
plt.title('rumorosa');

list_k = [5,7,9]
mse = np.zeros(len(list_k))
for index in range(len(list_k)):
    k = list_k[index]
    y = ndi.median_filter(noisy, (k,k))
    plt.figure();
    plt.imshow(y, clim=[0,255],cmap='gray');
    plt.title('filtrata per k=%d' % k);
    mse[index] = np.mean((x-y)**2)

plt.figure(10);
plt.plot(list_k,mse,'r-*');
plt.xlabel('dimensione della finestra');
plt.ylabel('MSE');
```

2. *Enhancement.*

```
x = np.float63(io.imread('luna.jpg'))

# Di seguito la maschera del filtro che realizza il Laplaciano
# secondo la definizione di derivata indicata nel testo
h = [[0,0,1,0,0],[0,0,0,0,0],[1,0,-4,0,1],[0,0,0,0,0],[0,0,1,0,0]]
h = np.array(h, np.float64)
y = ndi.correlate(x,h)

y_enhanc = x-y
plt.figure(1); plt.imshow(y_enhanc, clim=[0,255],cmap='gray');
```