Elaborazione di Segnali Multimediali

Descrittori locali per la classificazione di immagini

L.Verdoliva, D.Cozzolino

In questa esercitazione affrontiamo il problema della classificazione di immagini, in particolare ci focalizzeremo su immagini con tessiture (texture) diverse e le classificheremo tramite l'uso di descrittori locali (*handcrafted features*). Nella prima parte dell'esercitazione vedremo vari descrittori ad implementeremo un descrittore locale molto diffuso, chiamato Local Binary Pattern (LBP). Quindi useremo tale descrittore per classificare le immagini mediante tecniche di apprendimento automatico (*machine learning*).

1 Descrittori locali

L'obiettivo di un descrittore è quello di rappresentare le caratteristiche più importanti dell'immagine al fine di discriminarlo da altre immagini attraverso una rappresentazione compatta e robusta. I descrittori locali descrivono il comportamento dell'immagine in un intorno del pixel (*patch*) e possono essere calcolati per tutti i pixel dell'immagine (descrittori densi) o solo per alcuni punti salienti (descrittori sparsi). Per ottenere una rappresentazione di tutta l'immagine, l'informazione estratta da tali descrittori viene aggregata ad esempio calcolando l'istogramma.

1.1 Local Binary Pattern

Il Local Binary Pattern (LBP) è un descrittore locale denso utilizzato principalmente per l'analisi delle texture di un'immagine. LBP è calcolato per ogni pixel effettuando i seguenti passi:

1. individuazione di un vicinato (*neighborhood*) costituito da P pixel rispetto al pixel centrale;
2. confronto di ogni pixel del vicinato con il pixel centrale, ottenendo una stringa di P bit dove 1 indica che il pixel del vicinato è maggiore o uguale del pixel centrale, e 0 altrimenti;
3. conversione della stringa di bit in un numero intero da 0 a $2^P - 1$.

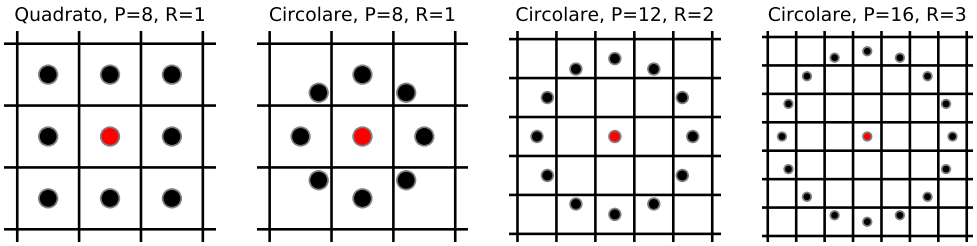

Figura 1: Differenti tipi di vicinato, il pixel di riferimento centrale è in rosso.

.

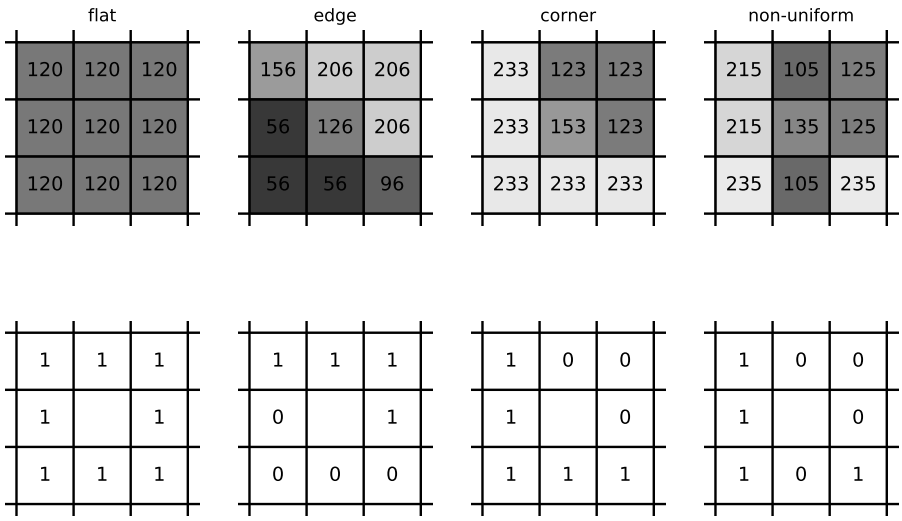

Figura 2: In alto: esempi di patch 3×3 che rappresentano regioni di diversa natura: area piatta (flat), con un bordo (corner), con uno spigolo (corner) e *non-uniform*. In basso: risultato del confronto tra i pixel del vicinato e il pixel centrale.

.

Il vicinato (*neighborhood*) può avere diverse forme (quadrata o circolare) e raggi (R). Alcuni possibili vicinati sono illustrati in figura 1. Il confronto del vicinato con il pixel centrale permette di individuare differenti pattern. Al tale proposito proviamo a calcolare la descrizione per le patch 3×3 riportate nella figura 2 in alto, considerando per semplicità il vicinato quadrato costituito da 8 pixel. Per ogni patch 3×3, gli 8 pixel del vicinato vanno confrontati con il pixel centrale al fine di ottenere una stringa binaria. Il risultato di questa operazione è mostrato in figura 2 in basso. Si può notare come i bit risultano tutti uguali, quando la patch è uniforme, invece per un bordo o uno spigolo si possono identificare due gruppi di bit uguali. Se si identificano più alternanze tra 0 e 1, allora la patch è denominata *non-uniform*. La sequenza di bit così ottenuta viene codificata in decimale ed è assegnata al pixel centrale della patch. Proviamo ad implementare in Python i passi precedentemente descritti su un'immagine con una texture abbastanza marcata. Carichiamo l'immagine dalla libreria di immagini del modulo `skimage.data`:

```
import skimage.data as data
x = np.float64(data.brick())
plt.figure();
plt.imshow(x, clim=[0,255], cmap='gray');
```

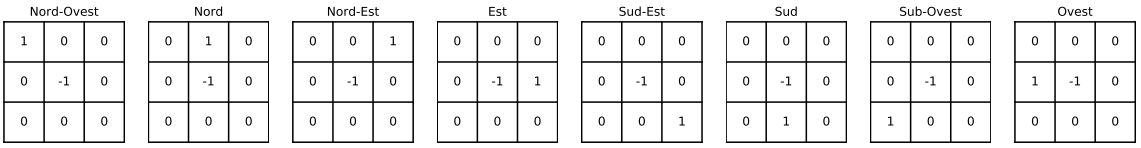

Figura 3: Filtri per implementare LBP.

.

Utilizzando gli 8 filtri riportati in figura 3 possiamo calcolare la differenza di ogni pixel del vicinato con il pixel centrale per tutte le patch dell'immagine.

```
h0 = np.array([[1,0,0],[0,-1,0],[0,0,0]], dtype=np.float64)
h1 = np.array([[0,1,0],[0,-1,0],[0,0,0]], dtype=np.float64)
h2 = np.array([[0,0,1],[0,-1,0],[0,0,0]], dtype=np.float64)
h3 = np.array([[0,0,0],[0,-1,1],[0,0,0]], dtype=np.float64)
h4 = np.array([[0,0,0],[0,-1,0],[0,0,1]], dtype=np.float64)
h5 = np.array([[0,0,0],[0,-1,0],[0,1,0]], dtype=np.float64)
h6 = np.array([[0,0,0],[0,-1,0],[1,0,0]], dtype=np.float64)
h7 = np.array([[0,0,0],[1,-1,0],[0,0,0]], dtype=np.float64)

b0 = ndi.correlate(x, h0) >= 0
b1 = ndi.correlate(x, h1) >= 0
b2 = ndi.correlate(x, h2) >= 0
b3 = ndi.correlate(x, h3) >= 0
b4 = ndi.correlate(x, h4) >= 0
b5 = ndi.correlate(x, h5) >= 0
b6 = ndi.correlate(x, h6) >= 0
b7 = ndi.correlate(x, h7) >= 0
```

Il risultato di ogni filtraggio viene confrontato con zero per ottenere il relativo bit. Infine, la stinga di 8 bit è codificata con un numero intero tra 0 a 255:

```
y = b0 + b1*2 + b2*4 + b3*8 + b4*16 + b5*32 + b6*64 + b7*128

plt.figure(); plt.imshow(y, clim=[0,255], cmap='gray');
plt.title('immagine LBP');
```

La variabile y contiene le descrizioni locali. Possiamo identificare le patch uniformi dell'immagine sapendo che la descrizione relativa è 0 o 255.

```
mappa_uniformi = (y==0) | (y==255)

plt.figure();
plt.imshow(mappa_uniformi, clim=[0,1], cmap='gray');
plt.title('mappa delle patch uniformi');
```

Notate che le patch uniformi sono concentrate all'interno di ogni mattone. In maniera analoga possiamo identificare le patch di bordo caratterizzate da una qualsiasi rotazione circolare della stringa di bit 11110000. Pertanto per le patch di bordo, il valore del descrittore locale può essere uno tra [15, 30, 60, 120, 240, 225, 195, 135], quindi i valori di LBP possono essere diversi anche se descrivono la stessa tipologia di patch. Esistono varianti di LBP che evitano questo comportamento, come *LBP Rotation Invariant* che riduce il numero di descrizioni locali aggregando i pattern che differiscono per una rotazione.

La libreria SKImage fornisce la funzione `skimage.feature.local_binary_pattern` per calcolare LBP con intorno circolare. Tale funzione permette di fissare il numero di pixel del vicinato (P) e il raggio del vicinato (R):

```
from skimage.feature import local_binary_pattern
y = local_binary_pattern(x, P, R)
```

Inoltre con il parametro `method` è possibile specificare una strategia di aggregazione dei pattern. Con `method='ror'` si aggregano i pattern che differiscono solo per rotazione, mentre con `method='uniform'` oltre all'aggrezione per rotazione, si utilizza anche un unico valore del descrittore locale per tutti i pattern *non-uniform*.

Facciamo adesso un esperimento e usiamo LBP per discriminare differenti tessiture e confrontiamo gli istogrammi. Possiamo determinate il tipo di tessiture (mattoni, erba o ghiaia) delle immagini *img1.jpg*, *img2.jpg* e *img3.jpg* in base alle distanze SAD (somma delle differenze assolute) degli istogrammi ottenuti dall'immagine dei descrittori locali LBP rispetto agli istogrammi delle immagini *brick.png*, *grass.png* e *gravel.png*.

1.2 Histogram of Oriented Gradients

Histogram of Oriented Gradient (HOG) è un descrittore locale proposto per il riconoscimento di oggetti basato sulle orientazioni del gradiente. Il descrittore prevede di suddividere l'immagine in piccole regioni spaziali, chiamate "celle". Per ogni cella, viene calcolato il modulo e l'orientazione del gradiente da cui si calcola l'istogramma pesato delle orientazioni. Le celle sono raggruppate in blocchi sovrapposti, gli istogrammi delle celle di ogni blocco vengono normalizzati congiuntamente. Infine, gli istogrammi vengono concatenati per ottenere un unico vettore descrittivo dell'intera immagine. La libreria SKImage fornisce la funzione `skimage.feature.hog` per calcolare HOG:

```
from skimage.feature import hog
y = hog(x, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(3, 3))
```

Tale funzione permette di indicare il numero di orientazioni degli istogrammi, la dimensione in pixel delle celle e il numero di celle per blocco. Inoltre, la funzione `skimage.feature.hog` opzionalmente restituisce una rappresentazione visiva del descrittore HOG. Usate le seguenti istruzioni per ottenere la rappresentazione visiva del descrittore HOG relativa all'immagine *lena.jpg*:

```
x = np.float32(io.imread('lena.jpg'))

from skimage.feature import hog
y, hog_image = hog(x, pixels_per_cell=(16, 16), visualize=True)

plt.figure(1)
plt.subplot(1,2,1)
plt.imshow(x, clim=[0,255], cmap='gray')
plt.subplot(1,2,2)
plt.imshow(hog_image, clim=None, cmap='gray')
plt.show()
```

I descrittori locali possono essere usati per la segmentazione di oggetti o tessiture all'interno di una immagine, a tal fine applicate il seguente codice all'immagine *textures.png* e visualizzate il risultato della segmentazione ottenuta con Kmeans:

```
x = np.float64(io.imread('textures.png'))
y = hog(x, feature_vector=False)

k = 4
M = y.shape[0]
N = y.shape[1]

y = np.reshape(y, (M*N,-1))
centroid, idx, sum_var = k_means(y, k)
idx = np.reshape(idx, (M,N))
```

L'opzione `feature_vector=False` evita la finale concatenazione degli istogrammi preservando la posizione speziale.

1.3 Esercizi proposti

1. *Local Phase Quantization.* Per estrarre caratteristiche legate a periodicità locali, Local Phase Quantization (LPQ) è un descrittore che opera nel dominio di Fourier. Per calcolare il descrittore LPQ, operate su blocchi scorrevoli di dimensioni 9×9 ed effettuare i seguenti passi per ogni blocco:

   (a) Calcolate la trasformata di Fourier bidimensionale del blocco.

   (b) Facendo riferimento alle frequenze più basse (intorno alla componente continua X_{dc}) selezionate i coefficienti complessi X_i per $i = 1, \cdots, 4$:

   $$\begin{bmatrix} X_0 & X_1 & X_2 \\ X_7 & X_{dc} & X_3 \\ X_6 & X_5 & X_4 \end{bmatrix}$$

   Considerando la componente continua al centro del blocco 9×9, i coefficienti da selezionare sono in posizione $[3,4], [3,5], [4,5], [5,5]$.

   (c) A questo punto costruite un vettore di lunghezza 8:

   $$\mathbf{a} = [\mathrm{Re}\{X_1\}, \mathrm{Im}\{X_1\}, \mathrm{Re}\{X_2\}, \mathrm{Im}\{X_2\}, \mathrm{Re}\{X_3\}, \mathrm{Im}\{X_3\}, \mathrm{Re}\{X_4\}, \mathrm{Im}\{X_4\}]$$

   Ustilizate le funzioni `np.real` and `np.imag`.

   (d) Trasformate il vettore $\mathbf{a}$ in un vettore binario $\mathbf{c}$ secondo la seguente regola:

   $$\mathbf{c}(i) = \begin{cases} 1 & \mathbf{a}(i) > 0, \\ 0 & \text{altrimenti} \end{cases}$$

   (e) Convertite il vettore binario $\mathbf{c}$ in un numero decimale: $y = \sum_{i=0}^{7} \mathbf{c}(i)\, 2^i$

   L'istogramma dell'immagine LPQ su 256 valori rappresenta il vettore di feature. Calcolate l'istogramma delle descrizioni locali LPQ per l'immagine (*impronta100.png*).

2 Classificazione tramite apprendimento automatico

In questa sezione vedremo come utilizzare il descrittore LBP per analizzare immagini al microscopio di tessuti tumorali del dataset BreakHis[1]. L'obiettivo è classificare le immagini in tumori benigni o maligni. L'istogramma

[1] https://www.kaggle.com/ambarish/breakhis

estratto da LBP rappresenta il vettore di feature (*feature vector*) che sarà utilizzato dal classificatore per individuare la migliore partizione nello spazio delle feature. In questo esperimento useremo un classificatore lineare (SVM). Per cominciare carichiamo il *training-set* nei file *train_label.npy* e *train_lbp_8_1.npy*.

```
train_label = np.load('train_label.npy')
train_feat  = np.load('train_lbp_8_1.npy')
```

La variabile `train_label` è un vettore di etichette (*label*) per 896 immagini di tessuti tumorali. L'etichetta è 1 se l'immagine è relativa ad un tumore maligno ed è 0 se l'immagine è relativa ad un tumore benigno. La variabile `train_feat` è una matrice 896×256 contenente i vettori di feature (LBP con intorno circolare, `P=8` e `R=1`) per 896 immagini, uno per ogni riga. Notate che è opportuno normalizzare le colonne della matrice, in modo che gli elementi del vettore di feature abbiamo statisticamente media nulla e varianza unitaria.

```
mu    = np.main(train_feat, 0)
sigma = np.std(train_feat, 0)
train_feat = (train_feat - mu)/(sigma + 1e-15)
```

La Support Vector Machine (SVM) lineare è implementata nella libreria SKLearn. L'addestramento viene eseguito tramite il metodo `fit`, a cui bisogna fornire sia il vettore di etichette che la matrice che contiene i vettori di feature.

```
from sklearn.svm import LinearSVC
classifier = LinearSVC().fit(train_feat, train_label) # training-phase
```

Il risultato dell'addestramento è l'oggetto `classifier`. Proviamo a determinare la classe (maligno o benigno) per l'immagine *test_breakhis.png*. A tal fine calcoliamo prima il vettore di feature:

```
x = io.imread('test_breakhis.png')
from skimage.feature import local_binary_pattern
y = local_binary_pattern(x, 8, 1)
feat, b = np.histogram(y.flatten(), bins=np.arange(0,257), density=True)
```

Prima di classificare l'immagine, il vettore di feature va normalizzato con le medie e deviazioni standard precedentemente calcolate e riorganizzato come matrice di una riga.

```
feat = (feat - mu)/(sigma + 1e-15)
feat = np.reshape(feat, (1, -1))
```

Adesso possiamo predire la classe utilizzando il metodo `predict` dell'oggetto `classifier`:

```
predizione = classifier.predict(feat)
if predizione==1:
    print('tumore maligno')
else:
    print('tumore benigno')
```

E' chiaro che la predizione può anche non essere corretta. Pertanto è importante valutare l'affidabilità del risultato ottenuto calcolando le prestazioni su un insieme di immagini di test (*test-set*). Utilizzate le immagini contenute nella archivio *test_set.zip* per valutare l'accuratezza della classificazione, ossia la percentuale delle volte in cui la predizione risulta corretta, ad esempio se su 100 immagini di test il classificatore predice correttamente 80 immagini allora l'accuratezza è del $80/100 = 80\%$.

2.1 Esercizi proposti

1. *Composizione delle caratteristiche.* Provate a cambiare i parametri del descrittore LBP utilizzando 12 vicini e raggio 2. I relativi dati di training sono memorizzati nel file *train_lbp_12_2.npy*. Inoltre provate ad utilizzare insieme le due varianti LBP concatenando i vettori delle feature.

2. *Metriche.* Il modulo `sklearn.metrics` contiene funzioni utili per calcolare diversi indici prestazionali. Per calcolare l'accuratezza provate ad utilizzare la funzione del modulo `accuracy_score` fornendogli il vettore delle etichette vere e il vettore delle etichette predette. Un altra funzione utile del modulo è `confusion_matrix` che restituisce la matrice di confusione. La matrice di confusione è una matrice quadrata dove ogni colonna rappresenta le classi predette dal classificatore, mentre ogni riga rappresenta le classi reali. L'elemento dell'i-sima riga e j-esima colonna è il numero di immagini della classe i che il classificatore ha predetto della classe j. Quindi, i valori sulla diagonalle della matrice di confusione rappresentano le predizioni corrette, i valori fuori dalla diagonale le predizioni errate.