Descrittori locali per la classificazione di immagini

Soluzioni

1 Descrittori locali

1.1 Local Binary Pattern

1. *Confronto istogrammi LBP.* Calcoliamo l'istogramma normalizzato delle descrizioni locali LBP per tre differenti tessiture: mattoni (*brick.png*), erba (*grass.png*) e ghiaia (*gravel.png*).

```
R = 1; P = 8; method = 'uniform'; # settings for LBP

def extract_feature(x):
    y = local_binary_pattern(x,P,R, method=method)
    feat, bins = np.histogram(y.flatten(),
                bins = np.arange(0, np.max(y)+2), density=True)
    return feat

img_brick  = np.float64(io.imread('brick.png'))
img_grass  = np.float64(io.imread('grass.png'))
img_gravel = np.float64(io.imread('gravel.png'))

feat_brick  = extract_feature(img_brick)
feat_grass  = extract_feature(img_grass)
feat_gravel = extract_feature(img_gravel)
```

Adesso valutiamo l'istogramma normalizzato LBP per l'immagine *img1.jpg*, e lo confrontiamo con i tre calcolati precedentemente in base alle distanze SAD (somma delle differenze assolute):

```
x = np.float64(io.imread('img1.jpg'))
feat = extract_feature(x)
dist_brick  = np.sum(np.abs(feat_brick -feat))
dist_grass  = np.sum(np.abs(feat_grass -feat))
dist_gravel = np.sum(np.abs(feat_gravel-feat))
```

```
if dist_brick<dist_grass and dist_brick<dist_gravel:
    print('e'' un muro')
elif dist_grass<dist_brick and dist_grass<dist_gravel:
    print('e'' un prato')
else:
    print('e'' ghiaia')
```

1.2 Esercizi proposti

1. *Local Phase Quantization.* Scriviamo la funzione per il calcolo dei descrittori locali LPQ:

```
def local_phase_quantization(x):
    def local_lpq(b):
        B = np.fft.fft2(b.reshape((9,9)))
        a = [np.real(B[-1,0])>0,np.imag(B[-1,0])>0,
             np.real(B[-1,1])>0,np.imag(B[-1,1])>0,
             np.real(B[ 0,1])>0,np.imag(B[ 0,1])>0,
             np.real(B[ 1,1])>0,np.imag(B[ 1,1])>0]
        d = a[0]+a[1]*2+a[2]*4+a[3]*8+a[4]*16+a[5]*32+a[6]*64+a[7]*128
        return np.uint8(d)

    y = ndi.generic_filter(x, local_lpq, (9,9))
    return y
```

Infine otteniamo l'istogramma delle descrizioni locali LPQ per l'immagine *impronta100.png* con il seguente codice:

```
img = np.float64(io.imread('../impronta100.png')
y = local_phase_quantization(x)
feat, bins = np.histogram(y.flatten(), bins = np.arange(0, 257), density=True)
```

2 Classificazione tramite apprendimento automatico

1. *Esempio codice di test*: Definiamo la funzione di estrazione delle feature data un'immagine:

```
R = 1; P = 8; method = 'default'; # settings for LBP
def extract_feature(x):
    y = local_binary_pattern(x,P,R, method=method)
    feat, bins = np.histogram(y.flatten(),
                bins = np.arange(0, np.max(y)+2), density=True)
    return feat
```

Adesso utilizziamo le immagini contenute nell'archivio *test_set.zip* per eseguite i test:

```
list_feats = list() # lista delle feature da riempire
list_label = list() # lista delle etichette da riempire

from glob import glob # funzione utile per ottenere una lista di file
# estrazione feature per immagini con tumore maligno
list_files_malignant = glob('testset/malignant/*.png')
for filename in list_files_malignant:
    #calcolo feature per un'immagine
    ft = extract_feature(np.float64(io.imread(filename)))
    list_feats.append(ft) #inserimento feature nella lista
    list_label.append(1) #inserimento etichetta nella lista

# estrazione feature per immagini con tumore benigno
list_files_benign    = glob('testset/benign/*.png')
for filename in list_files_benign:
    ft = extract_feature(np.float64(io.imread(filename)))
    list_feats.append(ft)
    list_label.append(0)

feats  = np.stack(list_feats, 0) # conversione lista in array
y_true = np.stack(list_label)    # conversione lista in array

feats = (feats - mu)/(sigma + 1e-15) # normalizzazione
y_pred = classifier.predict(feats) # classificazione

#calcolo dell'accuratezza
acc = np.mean(y_pred==y_true)
```

2.1 Esercizi proposti

1. *Composizione delle caratteristiche.*

```
# Dati di training
train_label = np.load('train_label.npy')
train_feat1 = np.load('train_lbp_8_1_default.npy')
train_feat2 = np.load('train_lbp_12_2_default.npy')

# Concatenamento Feature
train_feat  = np.concatenate((train_feat1,train_feat2), 1)

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

# Training
from sklearn.svm import LinearSVC
classifier = LinearSVC()
classifier.fit(train_feat,train_label)

# definizione funzione per estratte le feature
def extract_fature(x):
    R = 1; P = 8; method = 'default'; # settings for LBP
    y1 = local_binary_pattern(x,P,R, method=method)
    feat1, bins = np.histogram(y1.flatten(),
                bins = np.arange(0, np.max(y1)+2), density=True)

    R = 2; P = 12; method = 'default'; # settings for LBP
    y2 = local_binary_pattern(x,P,R, method=method)
    feat2, bins = np.histogram(y2.flatten(),
                bins = np.arange(0, np.max(y2)+2), density=True)

    return np.concatenate((feat1, feat2), 0)
```

   Adesso per eseguire i test usate lo stesso codice riportato in *Esempio codice di test* (Sez.2.1).

2. *Metriche.* Avendo a disposizione `y_true` il vettore delle etichette vere e `y_pred` il vettore delle etichette predette potete calcolare diversi indici prestazionali tramite le funzioni del modulo `sklearn.metrics`. Esempi:
   `sklearn.metrics.accuracy_score(y_true, y_pred)` restituisce l'accuratezza
   `sklearn.metrics.confusion_matrix(y_true, y_pred)` restituisce la matrice di confusione
   `print(sklearn.metrics.classification_report(y_true, y_pred))` stampa una tabella sintetica con tutti gli indici prestazionali più utilizzati per la classificazione.