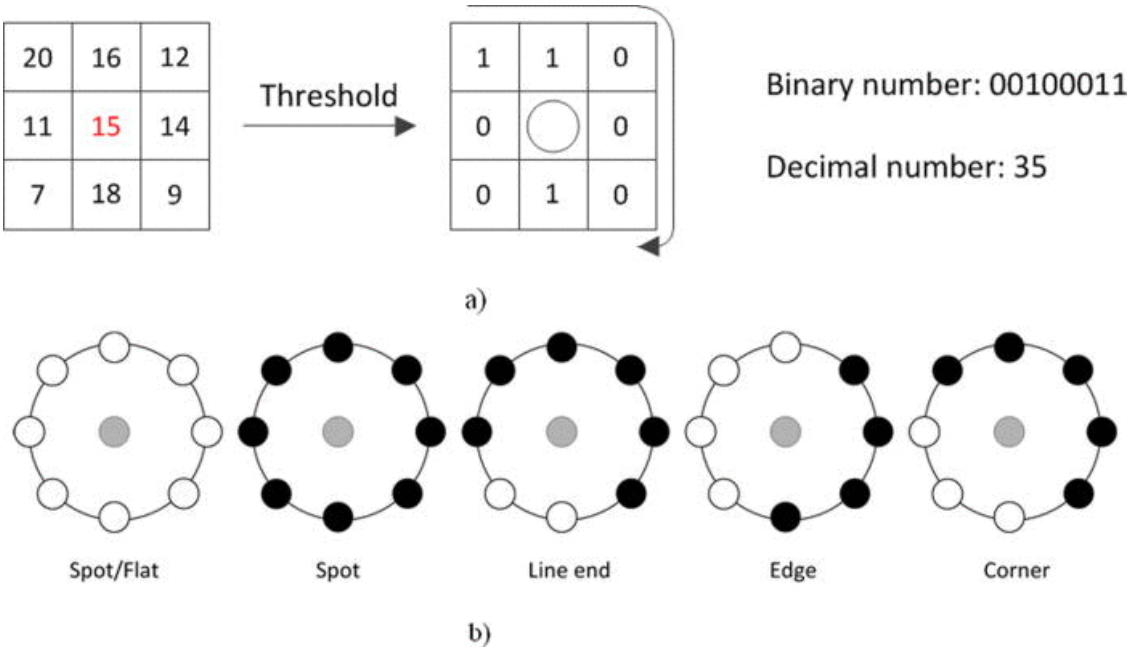
Fabio Merola W82000188 - Local Binary Pattern - Università degli studi di Catania, D.M.I. Informatica(LM-18) - Prof. Sebastiano Battiato

June 22, 2020

In questo elaborato verrà analizzata la funzione `skimage.feature.local_binary_pattern(...)` che mette a disposizione l'implementazione parametrizzata del descrittore in questione. Prima di passare all'analisi però, risulta conveniente fare un breve ripasso osservando degli schemi che riguardano questo descrittore.



In figura *a)* notiamo come viene inizialmente attuato il processo di sogliatura dei pixel adiacenti a quello interessato, il valore del pixel adiacente viene posto ad uno se superiore o uguale a quello del pixel d'interesse, zero altrimenti. In questo esempio si sta optando per raggio dell'intorno pari a uno e numero di pixel vicini considerati pari a otto. Questo setting, considerando di utilizzare solo pattern uniformi(con pixel neri adiacenti) ed invarianti alla rotazione(*00111100* viene inteso come *00001111*), ci porta ad avere le seguenti combinazioni binarie:

- 00000000
- 00000001
- 00000011

- 00000111
- 00001111
- 00011111
- 00111111
- 01111111
- 11111111

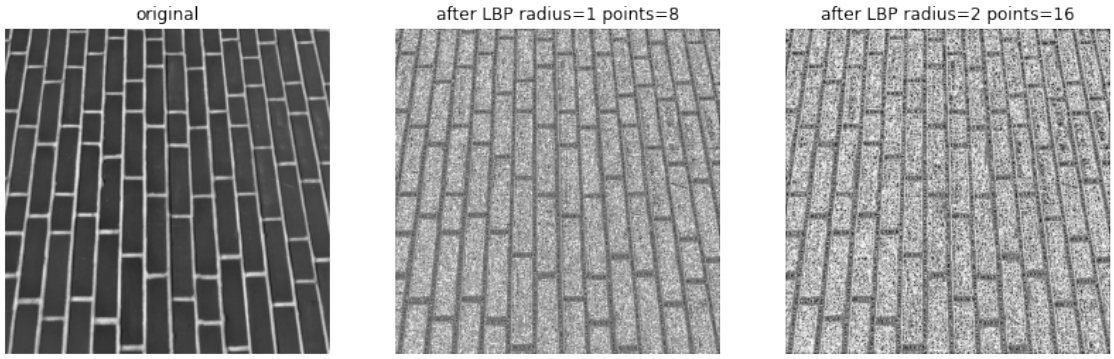
Considerando la figura *b)* possiamo classificare ogni pixel dipendentemente dalla combinazione binaria precedentemente ottenuta, considerando come neri i pixel con valore pari ad uno e bianchi gli altri, ad esempio: - 00000000 → Spot/Flat

- 00001111 → Edge
- 00011111 → Corner
- 00111111 → Line end
- 11111111 → Spot

Una volta trovata la combinaziona binaria per ogni pixel, si può ricostruire l'immagine convertendo il binario in decimale, ottenendo quindi il livello di grigio corrispondene al pixel ed immagini simili alle seguenti:

[4]:
```
import matplotlib.pyplot as plt

from skimage.feature import local_binary_pattern
from skimage import data, io
from skimage.color import label2rgb,rgb2gray


# settings for LBP_1
radius = 1
n_points = 8 * radius

# settings for LBP_2
radius_ = 2
n_points_ = 8 * radius

METHOD = 'uniform'

image = data.brick()  #Load internal library image
#image = io.imread('/path/to/custom_image.jpg')  #Load custom image
#image = rgb2gray(image)  #Need to convert in grayscale if you use a custom␣
 ↪image

lbp_1_8 = local_binary_pattern(image, n_points, radius, METHOD)
lbp_2_16 = local_binary_pattern(image, n_points_, radius_, METHOD)
```

```
imgs = [(image,"original"), (lbp_1_8,"after LBP radius=1 points=8"),␣
 ↪(lbp_2_16,"after LBP radius=2 points=16")]

fig, ax_img = plt.subplots(nrows=1, ncols=3, figsize=(14, 14))
axs = ax_img.flatten()

for img, ax in zip(imgs, axs):
    ax.set_title(img[1])
    ax.imshow(img[0])

for ax in ax_img:
    ax.axis('off')

plt.show()
```



Come visto nel codice precedente, la funzione:

lbp = local_binary_pattern(image, n_points, radius, METHOD)

ritorna in output un immagine e prende in input quattro argomenti:

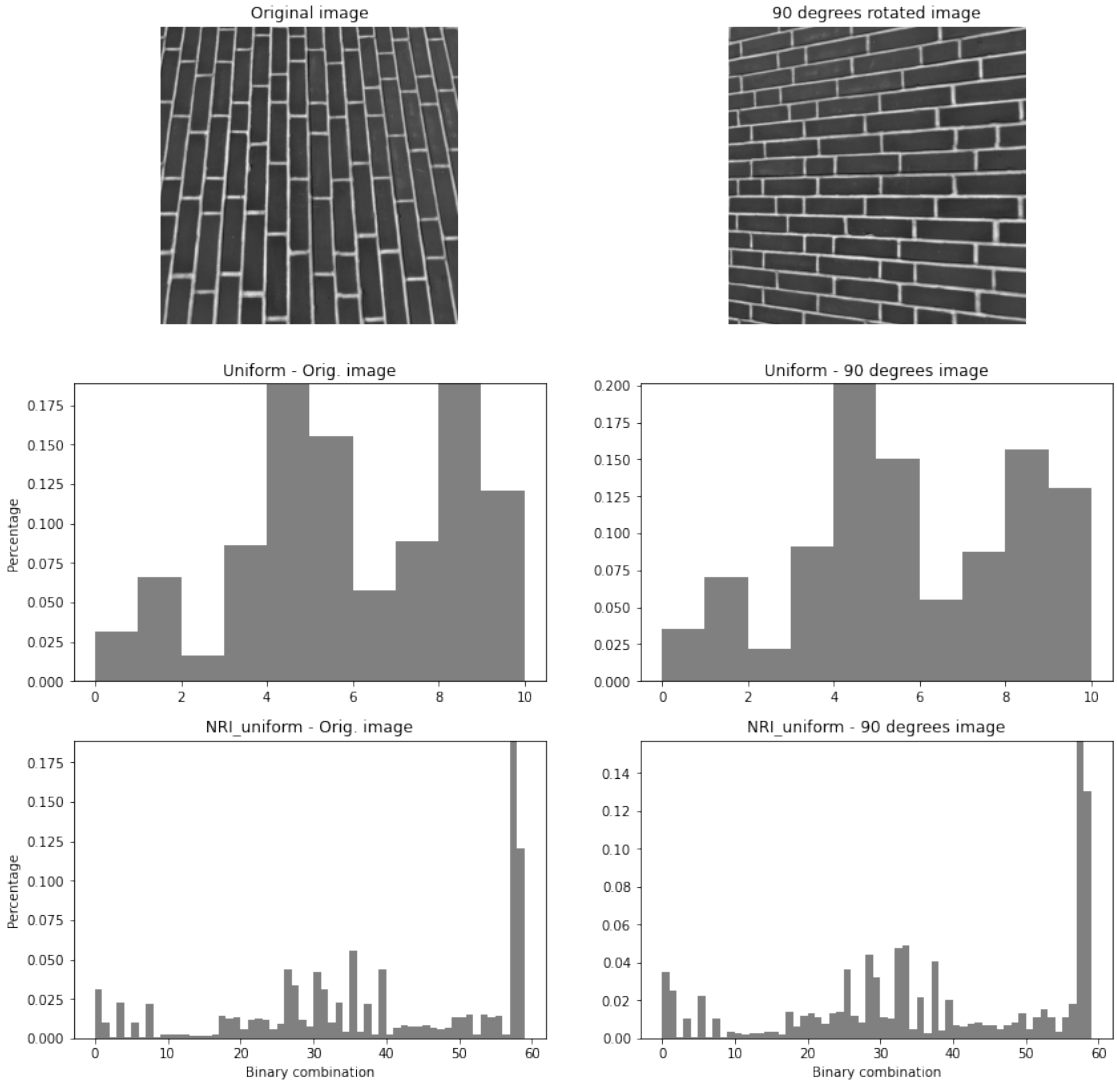
- Image: [`(N, M) array`] L'immagine originale(grayscale) su cui applicare il descrittore;
- N_points: [`int`] Il numero di pixel vicini da prendere in considerazione(generalmente 8*raggio);
- Radius: [`float`] Raggio dell'intorno o, meglio, distanza in pixel dal pixel centrale preso in considerazione ai pixel considerati vicini;
- Method: [`string`] Uno a scelta tra:
  - default: considera anche i pattern non uniformi, non invariante alla rotazione, invariante ai livelli di grigio;
  - ror: estenzione del precedente, invariante alla rotazione, invariante ai livelli di grigio;
  - uniform: considera solo i pattern uniformi, invariante alla rotazione, invariante ai livelli di grigio;

- nri_uniform: estenzione del precedente, non invariante alla rotazione, invariante ai livelli di grigio;
- var: prende in considerazione misurazioni del contrasto locale a pixel, invariante alla rotazione, non invariante ai livelli di grigio;

Ciò che risulta interessante però, quando si parla di LBP, è l'istogramma che deriva dalle immagini sopra. Questo infatti permette di fare texture recognition, face detection, ecc.. Di seguito esempi di istogrammi derivanti dalla stessa immagine(originale e roteata), analizzata con il metodo uniforme invariante alla rotazione e non.

[2]:
```
import numpy as np
from skimage.transform import rotate

def hist(ax, lbp):
    n_bins = int(lbp.max() + 1)
    #print(n_bins)
    return ax.hist(lbp.ravel(), density=True, bins=n_bins, range=(0, n_bins),
                   facecolor='0.5')

# settings for LBP
radius = 1
n_points = 8 * radius
METHOD = 'uniform'
METHOD_nri = 'nri_uniform'

image = data.brick()
r_115_image = rotate(image, angle=90, resize=False)

#LBP uniform method
lbp = (local_binary_pattern(image, n_points, radius, METHOD),"Uniform - Orig.␣
 ↪image")
lbp_75 = (local_binary_pattern(r_115_image, n_points, radius, METHOD),"Uniform␣
 ↪- 90 degrees image")

lbps = [lbp, lbp_75]

#LBP uniform non rotation invariant method
nri_lbp = (local_binary_pattern(image, n_points, radius,␣
 ↪METHOD_nri),"NRI_uniform - Orig. image")
nri_lbp_90 = (local_binary_pattern(r_115_image, n_points, radius,␣
 ↪METHOD_nri),"NRI_uniform - 90 degrees image")

nri_lbps = [nri_lbp, nri_lbp_90]

#Plot images and histograms
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(14, 14))
plt.gray()
```

```
ax[0,0].imshow(image)
ax[0,0].set_title("Original image")
ax[0,1].imshow(r_115_image)
ax[0,1].set_title("90 degrees rotated image")

for _ax in ax[0,range(2)]:
    _ax.axis('off')

for j in [1,2]:
    for i in range(2):
        _lbp = lbps[i] if j == 1 else nri_lbps[i]
        counts, _, bars = hist(ax[j,i], _lbp[0])
        ax[j,i].set_ylim(top=np.max(counts[:-1]))
        ax[j,i].set_title(_lbp[1])

ax[1,0].set_ylabel('Percentage')
ax[2,0].set_ylabel('Percentage')
ax[2,0].set_xlabel('Binary combination')
ax[2,1].set_xlabel('Binary combination')
plt.show()
```



Le due differenze sostanziali tra le due serie di istogrammi sono:

- Il numero di colonne, quindi il numero di possibili combinazioni binarie. Si nota inoltre che, diveramente da quanto detto prima, le combinazioni per il caso uniforme con raggio pari ad uno e vicini pari ad otto, siano dieci e non nove. Questo perchè nell'ultima colonna vengono raggruppate tutte le combinazioni non uniformi. Equivalentemente succede nel caso uniforme ma non invariante alla rotazione dove le posiibili combinazioni sono 58 più una.
- La differenza tra i grafici della prima serie(quella riguardante il metodo uniforme, invariante alla rotazione) risulta inferiore rispetto a quella tra i due istogrammi della seconda serie, questo a sottolineare la differenta tra i due metodi, quindi l'invarianza o meno alla rotazione.

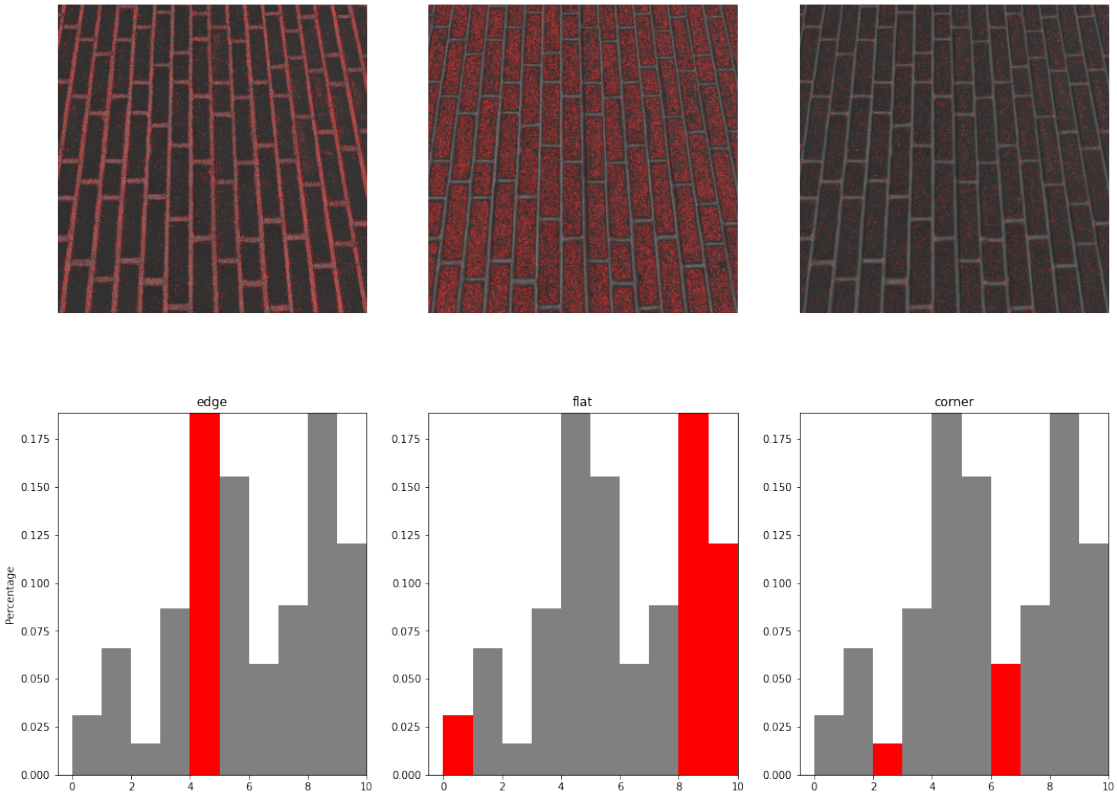
La differenza tra due istogrammi, calcolata con l'apposito algoritmo è ciò che si cela dietro gli algoritmi di face detection, texture recognition, ecc.

Un altro aspetto interessante riguarda la possibilità, correlando immagine ad istogramma, di evi-

denziare pixel appartenenti a pattern diversi(edge, corner, flat, ecc.).

[3]:
```
def overlay_labels(image, lbp, labels):
    mask = np.logical_or.reduce([lbp == each for each in labels])
    return label2rgb(mask, image=image, bg_label=0, alpha=0.5)


def highlight_bars(bars, indexes):
    for i in indexes:
        bars[i].set_facecolor('r')

def hist(ax, lbp):
    n_bins = int(lbp.max() + 1)
    return ax.hist(lbp.ravel(), density=True, bins=n_bins, range=(0, n_bins),
                   facecolor='0.5')

image = data.brick()
lbp = local_binary_pattern(image, n_points, radius, METHOD)

#Plot images and histograms
fig, (ax_img, ax_hist) = plt.subplots(nrows=2, ncols=3, figsize=(18, 14))
plt.gray()

titles = ('edge', 'flat', 'corner')

#Histogram's labels creation in order to make a division of it depending on the␣
 ↪requested pattern
w = width = radius - 1
edge_labels = range(n_points // 2 - w, n_points // 2 + w + 1)
flat_labels = list(range(0, w + 1)) + list(range(n_points - w, n_points + 2))
i_14 = n_points // 4            # 1/4th of the histogram
i_34 = 3 * (n_points // 4)      # 3/4th of the histogram
corner_labels = (list(range(i_14 - w, i_14 + w + 1)) +
                 list(range(i_34 - w, i_34 + w + 1)))

label_sets = (edge_labels, flat_labels, corner_labels)

for ax, labels in zip(ax_img, label_sets):
    ax.imshow(overlay_labels(image, lbp, labels))

for ax, labels, name in zip(ax_hist, label_sets, titles):
    counts, _, bars = hist(ax, lbp)
    highlight_bars(bars, labels)
    ax.set_ylim(top=np.max(counts[:-1]))
    ax.set_xlim(right=n_points + 2)
    ax.set_title(name)
```

```
ax_hist[0].set_ylabel('Percentage')
for ax in ax_img:
    ax.axis('off')
plt.show()
```



Nelle immagini si possono osservare i vari pixel appartenenti ai rispettivi pattern evidenziati in rosso, equivalentemente le colonne negli istogrammi.