Traitement d'images
TP 4 :Les descripteurs de texture GLCM et LBP

Dans ce TP, nous nous focalisons sur les descripteurs de texture. Plus précisément, nous testons l'extraction des deux descripteurs GLCM et LBP.

Nous utilisons dans ce cas la bibliothèque scikit-image qui contient un module feature qui permet l'obtention de plusieurs descripteurs y compris GLCM et LBP.

Donc, pour commencer le TP, il est nécessaire d'importer les bibliothèques à utiliser :

import matplotlib.pyplot as plt

from skimage.feature import graycomatrix, graycoprops # descripteur GLCM

from skimage import data # pour les données images

from skimage.transform import rotate

from skimage.feature import local_binary_pattern # descripteur LBP

from skimage import color

Descripteur GLCM

Dans cette partie, nous illustrons la classification des textures à l'aide des matrices de co-occurrence des niveaux de gris (GLCM). Une GLCM est un histogramme de valeurs de niveaux de gris co-occurrents à un décalage donné sur une image.

Dans ce TP, deux images sont considérées, chacune ayant une texture différente de l'autre. Pour chaque parcelle de chaque image, un GLCM avec un décalage particulier (par exemple distance=1 et angle=0) est calculé. Ensuite, plusieurs caractéristiques des matrices GLCM sont calculées : le contraste, la dissimilarité, la corrélation, l'énergie …. Elles sont représentées graphiquement pour illustrer le fait que les classes forment des groupes dans l'espace des caractéristiques. Dans un problème de classification classique, l'étape finale (non incluse dans cet exemple) consisterait à former un classificateur, tel que la régression logistique, pour classifier d'autres images à partir de leurs textures.

Après chargement des deux images, nous calculons tout d'abords les matrices de co-occurrence GLCM pour les deux images tout en fixant les paramètres du GLCM. Il faut fixer la distance et l'angle du GLCM :

 # calcul des matrices de co-occurrence

```
d=[1,2] # deux distances 1 et 2

angl=[0, np.pi/2] # deux angles

g = graycomatrix(image_array, d, angl, levels=256, normed=True, symmetric=True)
```

Ensuite, des propriétés sont calculées à partir des matrices de co-occurrence :

```
# calcul des propriétés

contrast      = graycoprops(g, 'contrast')

dissimilarity = graycoprops(g, 'dissimilarity')

homogeneity   = graycoprops(g, 'homogeneity')

energy        = graycoprops(g, 'energy')

correlation   = graycoprops(g, 'correlation')

ASM           = graycoprops(g, 'ASM')
```

Les deux images peuvent être comparées en s'appuyant sur ces propriétés.

Faites comparer deux images avec des textures différentes et deux autres images de textures identiques.

Vous pouvez utiliser les images incluses dans le module data de scikit-image :

```
brick=data.brick()
grass=data.grass()
gravel=data.gravel()
```

Descripteur LBP

Supposons que nous ayons deux images, par exemple brick et gravel, et que vous souhaitiez classer une image couleur inconnue en brick ou gravel. Une méthode possible utilisant le LBP consisterait à :

- Convertir les images en couleur en niveaux de gris.
- Calculer les motifs binaires locaux.
- Calculer l'histogramme normalisé des motifs binaires locaux.
- Utiliser des distances pour classifier l'image inconnue.

L'extrait suivant met en œuvre cette approche :

```
def lbp_histogram(color_image):
    img = color.rgb2gray(color_image)
    patterns = local_binary_pattern(img, 8, 1)
    hist, _ = np.histogram(patterns, bins=np.arange(2**8 + 1), density=True)
    return hist

 brick=data.brick()
gravel=data.gravel()
```

```
brick_feats = lbp_histogram(brick)
gravel_feats = lbp_histogram(gravel)

unknown_feats = lbp_histogram(unknown)
```

Ensuite, une manière simple de classifier l'image inconnue est de mesurer la similarité (ou la dissimilarité) entre l'histogramme LBP de l'image inconnue et les histogrammes des images représentant les deux classes considérées. La distance euclidienne entre les histogrammes est une mesure de dissimilarité courante.

```
from scipy.spatial.distance import euclidean
unknown_brick=euclidean(unknown_feats, brick_feats)
unknown_gravel=euclidean(unknown_feats, gravel_feats)
```

On peut aussi visualiser les images ainsi que leurs histogrammes LBP :

```
hmax = max([brick_feats.max(), gravel_feats.max(), unknown_feats.max()])
fig, ax = plt.subplots(2, 3)

ax[0, 0].imshow(brick)
ax[0, 0].axis('off')
ax[0, 0].set_title('brick')
ax[1, 0].plot(brick_feats)
ax[1, 0].set_ylim([0, hmax])

ax[0, 1].imshow(gravel)
ax[0, 1].axis('off')
ax[0, 1].set_title('gravel)
ax[1, 1].plot(gravel_feats)
ax[1, 1].set_ylim([0, hmax])
ax[1, 1].axes.yaxis.set_ticklabels([])

ax[0, 2].imshow(unknown)
ax[0, 2].axis('off')
ax[0, 2].set_title('Unknown (knitwear)')
ax[1, 2].plot(unknown_feats)
ax[1, 1].set_ylim([0, hmax])
ax[1, 2].axes.yaxis.set_ticklabels([])

plt.show(fig)
```