TD2 : TRAITEMENTS SUR HISTOGRAMMES - PROFILS

EXERCICE 1 : Calcul d'un histogramme

Créer un fichier de commandes qui permet de calculer l'histogramme d'une image. On programmera la fonction et on comparera le résultat obtenu avec celui de la fonction de scikit-image (skimage). L'image analysée est l'image en niveaux de gris *IlesLofoten.png*. Compléter le fichier TD2Ex1.py.

- Charger l'image source et la visualiser, afficher les informations principales sur cette image (dimensions, type, etc.)

```
import numpy as np
import skimage as ski
import matplotlib.pyplot as plt

im = ski.io.imread("IlesLofoten.png")

print("Image features:")
print("\tData type........:",type(im))
print("\tPixel type.......:",im.dtype)
print('\tGrayscale range..:[',im.min(),',',im.max(),']')
print("\tImage dimension .:",im.ndim)
print("\tImage size.......:",im.shape)
plt.figure(figsize=(15,5))
plt.imshow(im,cmap='gray')
plt.colorbar()
plt.title(Source image)
plt.show()
```

- Calculer l'histogramme de cette image. On initialisera le tableau `Hist` et on affichera le résultat avec la fonction `plot ou bar`. Commenter le résultat.

- Calculer l'histogramme via la fonction de skimage `ski.exposure.histogram`. Vérifier que les résultats obtenus sont les mêmes que ceux obtenus avec notre code (`np.array_equal`)

- Calculer le tableau `P` des probabilités de chaque niveau de gris.

```
P= --to be completed
fig3 = plt.figure()
plt.bar(range(len(P)),P)
plt.title("Probabilities of gray levels")
plt.xlabel("Gray levels")
plt.ylabel("Gray level probabilities")
```

↳ *Quelle est la probabilité que le niveau de gris d'un pixel soit inférieur ou égal à 100 ?*

- Calculer la fonction de répartition de l'image (i.e l'histogramme cumulé normalisé).

```
F = np.empty_like(P)
for k in range(len(P)):
    F[k]=--to be completed
```

TRAVAIL PERSONNEL

On va réaliser un module de traitement des images à partir de transformations d'histogramme. Ce module sera codé dans le fichier `histogramProcessing.py`. Il ne traitera que des images en niveaux de gris codées sur un octet (`np.uint8`) ou en float (`np.float64`). Les histogrammes seront calculés sur `nbins=256` valeurs de niveaux de gris.

1. Compléter la fonction `myHistogram(im,normalize=True)` qui permet de calculer l'histogramme de l'image im sur 256 niveaux, avec normalisation (probabilités) ou non.
2. Utiliser le programme `testHistogramProcessing.py` pour tester votre fonction. Discuter des différences éventuelles avec le résultat fourni par la fonction scikit-Image `exposure.histogram`.
3. Compléter la fonction `myCumulatedHistogram(im)` qui permet de calculer la fonction de répartition des niveaux de gris.
4. Utiliser le programme `tesHistogramProcessing.py` pour tester votre fonction. Discuter des différences éventuelles avec le résultat fourni par la fonction scikit-Image `exposure.cumulative_distribution`.

EXERCICE 2 : Calibration d'une image

On va programmer la calibration d'une image pour améliorer sa lisibilité et son contraste, et tester la fonction de scikit-Image.

- Charger l'image `fdoeil.png`. Afficher l'image ainsi que ses caractéristiques.
- Calculer et afficher son histogramme

↳ *Cette image vous parait-elle bien contrastée ? Pourquoi ? Quels sont les niveaux minimal et maximal d'intensité dans l'image ?*

- Calibrer cette image en redistribuant la dynamique sur l'intervalle $[0,255]$. Soit `imc` l'image résultat. Afficher l'image résultat `imc` avec son histogramme.

↳ *Est-ce que le résultat est satisfaisant ? Pourquoi ?*

- Comparer avec le résultat de `ski.exposure.rescale_intensity`.

Pour améliorer les résultats, on va redistribuer la dynamique en saturant p% des pixels les plus sombres au noir et p% des pixels les plus clairs au blanc.

- Utiliser la fonction `exposure.cumulative_distribution` pour déterminer l'intervalle des niveaux de gris $[minVal, maxVal]$ qui seront redistribués sur l'intervalle $[0,1]$. Puis réappliquer la transformation. Afficher l'image obtenue avec son histogramme.

```
HistC,BinsC = ski.exposure.cumulative_distribution(im,nbins=-- to be

p = 0.01
ind = np.argwhere(-- to be completed)
minVal = -- to be completed
maxVal = -- to be completed
print("Range for a {0:.2f} % saturation = [{1} ,
{2}]".format(p*100,minVal,maxVal) )

imc = np.ndarray.astype(im,np.int64)
imc = np.round(-- to be completed)   # apply calibration given minVal and
maxVal
imc = np.maximum(imc,-- to be completed)
imc = np.minimum(imc,-- to be completed)
imc = np.ndarray.astype(imc,np.uint8)
```

↳ *Pourquoi transforme-t-on l'image en int64 ?*

↳ *Est-ce que le résultat est plus satisfaisant ? Quel est l'inconvénient ?*

- Utiliser la fonction ski.exposure.rescale_intensity pour réaliser la calibration équivalente. Comparer les résultats après avoir homogénéisé les types de données.

```
imc2 = ski.exposure.rescale_intensity(im,(-- to be completed,-- to be
completed),(-- to be completed,-- to be completed))
imc2 = np.round(imc2)
imc2 = np.ndarray.astype(imc2,np.uint8)

print('Egalité des images = ' + str(np.array_equal(imc,imc2)))
```

EXERCICE 3 : Egalisation d'histogramme

Dans cet exercice, on reprogrammer l'égalisation d'histogramme. On comparera avec la fonction de scikit-Image. On traitera des images de type uint8.

- Charger l'image IlesLofoten.png

```
im = ski.io.imread("IlesLofoten.png")
```

- Programmer la transformation $n' = T(n)$qui permet de réaliser une égalisation d'histogramme de l'image source. On prendra $N = 256$ niveaux de gris ($n \in [0, N[$).

```
N=256
H,W = im.shape
P = np.zeros(N)
T = -- to be completed
```

- Appliquer la transformation pour égaliser l'image.

```
ime = np.empty_like(im)
for i in range(H):
    for j in range(W):
```

```
ime[i,j]=T[im[i,j]]
```

- Afficher l'image source et l'image éqalisée avec leur histogramme et leur fonction de répartition.

↳ *Quel est l'impact l'apparence de l'image et son histogramme ?Le traitement est-il linéaire ? L'ordre des niveaux de gris est-il respecté ?*

- Appliquer le traitement de scikit-Image (`exposure.equalize_hist()`). Faire attention au type des données images. Afficher l'histogramme et la fonction de répartition :

```
imr  = -- to be completed

print("Egalité des images égalisées :",np.array_equal(ime,imr))
```

↳ *Est-ce que les résultats sont identiques ?*

- Appliquer le traitement sur l'image *fdoeil.png*. Commenter.

TRAVAIL PERSONNEL

Compléter le module `hitogramProcessing`.py en programmant les fonctions de calibration et d'égalisation d'histogramme :

5. Compléter la fonction `myCalibration(im,p=0.0)` qui permet de réaliser la calibration d'une image codée en flottants avec saturation des pixels les plus sombres et les plus clairs.
6. Utiliser le programme `tesHistogramProcessing.py` pour tester votre fonction.
7. Compléter la fonction `myEqualization (im)` qui permet de réaliser l'égalisation de l'image `im`.
8. Utiliser le programme `testHistogramProcessing.py` pour tester votre fonction.

EXERCICE 4 : profils cumulés

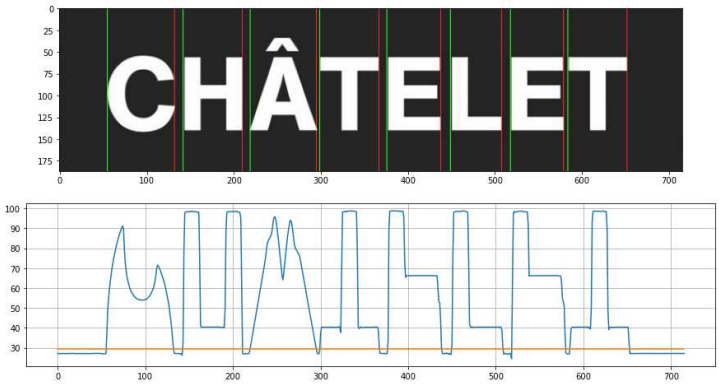
On va Implémenter le calcul d'un profil horizontal cumulé et d'un profil vertical cumulé et appliquer les traitements sur l'image *'texte.png'*. Vous complèterez le fichier `TD2Ex4.py`. On notera `profV` et `profH` les deux profils calculés.

- Charger l'images *'texte.png'* et la transformer en float.
- Calculer les profils `profV` et `profH`.

↳ *Commenter les résultats obtenus*

- Proposer un algorithme qui permet d'isoler les caractères, i.e. de déterminer les ordonnées qui correspondent à des points de séparation entre les caractères. L'implémentation sera faite en travail personnel. Voici le type de résultat attendu :

```
Il y a  8 caractères,
    qui commencent aux indices  [ 55 142 219 299 376 449 518 584]
    et finissent aux indices    [132 210 295 367 438 508 579 652]
```



TRAVAIL PERSONNEL
Programmer l'algorithme de détection des caractères.