TP1
Projection d'histogrammes
Segmentation de Textes en Lignes-Paws

But du TP

Le but de ce TP est d'implémenter la méthode de projection d'histogrammes horizontal et vertical pour l'extraction des lignes à partir d'un texte et ensuite extraire les PAWs (Parts of Arabic Words) à partir des lignes.

Procédure Générale

1. Lecture d'une image de texte
2. La mettre en niveaux de gris
3. La binariser
4. Calculer les projections des histogrammes horizontal et vertical de l'image.
5. Remarquer que :
    a. Appliqués à l'image entière de texte, seule la projection de l'histogramme horizontal peut donner des informations sur la localisation des lignes pour permettre leur extraction.
    b. La projection de l'histogramme vertical n 'est dans ce cas pas d'une grande utilité.
6. Utiliser la projection de l'histogramme horizontal pour :
    a. Trouver le nombre de lignes d'écriture dans le texte
    b. Délimiter ces lignes (début et fin)
    c. Récupérer ces lignes sous formes d'images.
7. Utiliser les images de lignes pour extraire les parties de mots (PAWs).
8. Utiliser la projection de l'histogramme vertical pour :
    a. Trouver le nombre de PAws dans la ligne de texte
    b. Délimiter ces PAWs (début et fin)
    c. Récupérer ces PAWs sous formes d'images pour la prochaine étape : « Extraction de caractéristiques ».

TP : Partie I. Détection et séparation des lignes de texte

1. **Importation des packages nécessaires**

```
# openCV
import cv2
# numpy
import numpy as np
# mayplotlib.pyplot
from matplotlib import pyplot as plt
```

2. **Lecture de l'image de texte**

Par défaut, OpenCV ouvre une image en couleurs.

```
# Lire image de texte et la rendre en nivaux de gris
img=cv2.imread('data/Page3.png')
print(img.shape)
```

(1277, 3940, 3)

```
# rendre l'image en niveaux de gris
img= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

(1277, 3940)

On peut lire directement l'image en niveaux de gris, de deux maniéres :

```
# Lire image de texte directement en nivaux de gris
img=cv2.imread('data/Page3.png', cv2.IMREAD_GRAYSCALE)
print(img.shape)
```

(1277, 3940)

```
# Lire image de texte directement en nivaux de gris
img=cv2.imread('data/Page3.png', 0)
print(img.shape)
```

(1277, 3940)

3. **Binarisation de l'image et l'inverser**

On peut procéder de plusieurs manières :

Remarque :

La fonction cv2.threshold donne deux sorties : le seuil de binarisation et l'image binarisée.

Ci-dessous, on ne récupère que l'image binarisée.

```
#  binarisation seuil global
bw=cv2.threshold(img,130,255,cv2.THRESH_BINARY)[1]
# inverser l'image
bw = cv2.bitwise_not(bw)
```

```
#  binarisation et inversion par un seul appel (seuil global)
bw=cv2.threshold(img,130,255,cv2.THRESH_BINARY_INV)[1]
```

```
#  binarisation et inversion par un seul appel (méthode OTSU)
bw=cv2.threshold(img,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)[1]
```

4. **Calcul des projections des histogrammes horizontal et vertical de l'image**

```
# Calcul projection histogramme horizontal
def histo_h(im):
  h = im.shape[0]
  hist_h = np.zeros((h),np.uint16)
  for i in range(h):
    hist_h[i]=np.count_nonzero(im[i,:])
  return hist_h
```

```
# Calcul projection histogramme vertical
def histo_v(im):
  w = im.shape[1]
  hist_v = np.zeros((w),np.uint16)
  for i in range(w):
    hist_v[i]=np.count_nonzero(im[:,i])
  return hist_v
```

Appel des fonctions histo_h et histo_v pour les calculs :

```
hist_h = histo_h(bw)
hist_v = histo_v(bw)
```

5. Préparation de l'affichage des résultats

Initialiser deux espaces blancs de la taille de l'image de texte pour l'affichage des histogrammes de projection.

```
# Images blanches de la même taille que l'image
im_hist_h = np.zeros(img.shape,np.uint8)
im_hist_h[:,:] = 255
im_hist_v = np.zeros(img.shape,np.uint8)
im_hist_v[:,:] = 255
```

Mettre à zéro (noir), un nombre de pixels égal à celui des pixels allumés dans l'image (par ligne pour l'histogramme horizontal, en colonne pour l'histogramme **vertical).**

```
h,w = img.shape
for i in range(0,h):
  im_hist_h[i,0:hist_h[i]]=0
for i in range(0,w):
  im_hist_v[0:hist_v[i],i]=0
```

Sauvegarde des figures résultats :

- **Création du répertoire si non existant**

```
from genericpath import isdir
from operator import is_not
from os import mkdir
if not(isdir('resultat')):
  mkdir('resultat')
```

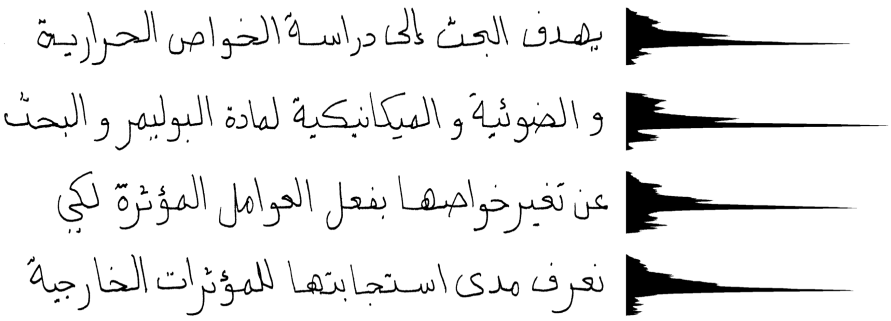
- **Sauvegarde des images avec histogrammes correspondants**

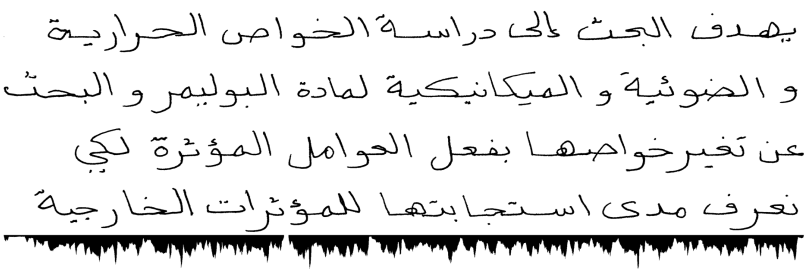
```
cv2.imwrite('resultat/hist_h.png', cv2.hconcat([img, im_hist_h]))
cv2.imwrite('resultat/hist_v.png', cv2.vconcat([img, im_hist_v]))
```

Affichage des histogrammes :

```
cv2.imwrite('horiz.png', im_hist_h)
cv2.imshow('histVert',cv2.hconcat([img, im_hist_h]))
cv2.imwrite('hist_h.png', cv2.hconcat([img, im_hist_h]))
cv2.imshow('histHoris',cv2.vconcat([img, im_hist_v]))
cv2.imwrite('hist_v.png', cv2.vconcat([img, im_hist_v]))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.waitKey(0) sert à garder l'affichage jusqu'à ce que l'utilisateur entre un caractère au clavier.





Remarques :

a. L'histogramme permet clairement de voir la délimitation des lignes d'écriture.
b. L'histogramme vertical ne donne pas d'information particulière sur l'image de texte.

6. Détection des lignes à partir de l'histogramme horizontal.

```
def detection_lignes(im, vh):
  pos_lignes=[]
  lignes=[]
  k=0
  h=len(vh)
  deb =0
  fin =0
  i=0
  while (i<h-1):
    if ((vh[i]==0) & (vh[i+1] > 0)):
      k+=1
      deb =i
      for j in range(deb, h-1):
        if ((vh[j]>0) & (vh[j+1]==0)):
          fin =j
          pos_lignes.append([deb, fin])
          ligne = im[deb:fin,0:]
          lignes.append(ligne)
          break
        else:
           j+=1
      i=fin+1
    else:
        i+=1
  return k, pos_lignes, lignes
```

7. Appels des fonctions histogramme horizontal et détection et séparation des lignes

```
hist_h = histo_h(bw)
nbl, pos_lgs, lgs = detection_lignes(bw, hist_h)
print ('nb_lignes', nbl)
print('loc',pos_lgs)
h,w = lgs[0].shape
print(h,w)
```

8. Sauvegarde des lignes.

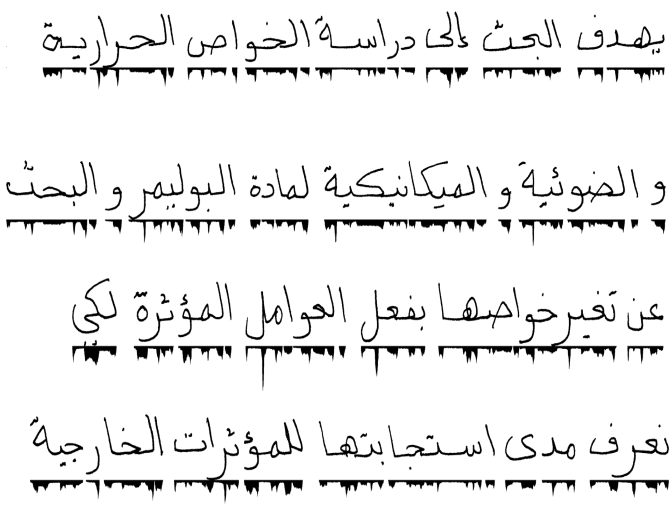
```
for i in range(nbl):
  cv2.imwrite('res/lig'+str(i)+'.png', lgs[i])
```

Appliquons, les projections d'histogramme horizontal et vertical sur les images de lignes de texte :

```
for i in range(nbl):
  hist_h1=histo_h(lgs[i])
  hist_v = histo_v(lgs[i])
  im_hist_h = np.zeros(lgs[i].shape,np.uint8)
  im_hist_h[:,:] = 255
  im_hist_v = np.zeros(lgs[i].shape,np.uint8)
  im_hist_v[:,:] = 255
  cv2.imshow('histVert',cv2.hconcat([~lgs[i], im_hist_h]))
  cv2.imwrite('resultat/hist_h'+str(i)+'.png', cv2.hconcat([~lgs[i], im_hist_h]))
  cv2.imshow('histHoris',cv2.vconcat([~lgs[i], im_hist_v]))
  cv2.imwrite('resultat/hist_v'+str(i)+'.png', cv2.vconcat([~lgs[i], im_hist_v]))
  cv2.waitKey(0)
cv2.destroyAllWindows()
```

Affichons pour voir ce que nous obtenons :

9. La projection verticale de l'histogramme de la ligne d'écriture va maintenant nous donner des informations sur la délimitation des Parties de Mots Arabes (PAWs).



Remarques :

1. Sachant que la ligne de base d'écriture représente la ligne fictive sur laquelle écrirait le scripteur s'il avait devant lui une feuille graduée, celle-ci va être définie comme la ligne contenant le nombre maximum de pixels allumés le long de l'image de la ligne d'écriture.
2. Cette information nous est donc donnée par le pic de l'histogramme horizontal. Calculons les positions des lignes de base de ce texte.

```
lg_base=np.zeros(nbl)
for i in range(nbl):
  hist_h1=histo_h(lgs[i])
  lg_base[i] = np.argmax(hist_h1)
```

3. La bande de base de l'écriture est définie comme étant un intervalle au-dessus et en dessous de la ligne de base d'écriture. Cet intervalle vaut environ 20% de la hauteur de la ligne.

10. Ligne de base :

```
lg_base=np.zeros(nbl)
for i in range(nbl):
  hist_h1=histo_h(lgs[i])
  lg_base[i] = np.argmax(hist_h1)
```

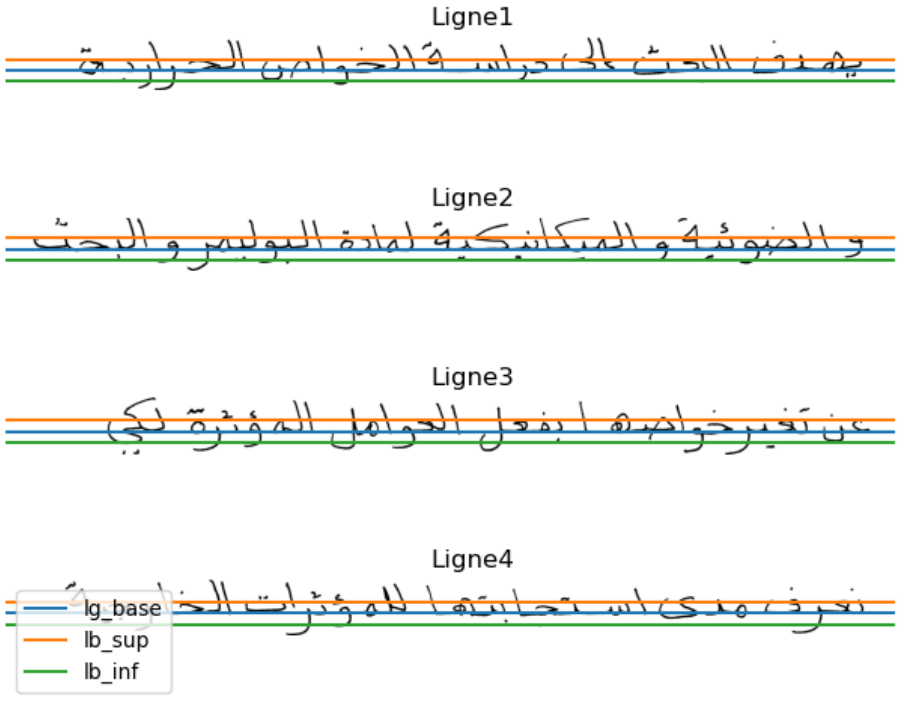
11. Bande de base :

```
def calcul_bd_base(lb, fraction):
  long = (lb[1]-lb[0])*fraction/100
  return long
```

```
bd_base =np.zeros(nbl)
for i in range(nbl):
  bd_base[i]=calcul_bd_base(pos_lgs[i],20)
```

12. Affichage ligne, ligne de base, bande de base :

```
plt.figure(figsize=(8, 6))
for i in range(1,nbl+1):
  plt.subplot(4, 1, i)
  plt.imshow(~lgs[i-1], cmap=plt.cm.gray)
  plt.plot([0,w],[lg_base[i-1],lg_base[i-1]],label='lg_base')
  plt.plot([0,w],[lg_base[i-1]-bd_base[i-1],lg_base[i-1]-bd_base[i-
1]],label='bd_sup')
  plt.plot([0,w],[lg_base[i-1]+bd_base[i-1],lg_base[i-1]+bd_base[i-1]],
label='bd_inf')
  plt.title('Ligne'+str(i))
  plt.axis('off')
plt.legend()
plt.savefig('res.png')
plt.show()
```



TP : Partie II. Détection et séparation des PAWs

Maintenant que nous avons pu récupérer les lignes de texte comme images, voyons ce que nous pouvons faire sur ces lignes.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import os
```

Lisons les lignes de texte :

```
path = 'res'
files = os.listdir(path)
img =[]
for i in range (len(files)):
  img1=cv2.imread(('res/'+files[i]),0)
  img.append(img1)
  cv2.imshow(str(i),img[i])
  cv2.imwrite('resultat/ligne'+str(i)+'.png',img[i])
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Travaillons par exemple avec la dernière ligne du texte :

```
img=cv2.imread('res/lig3.png')
print(img.shape)
# image en niveaux de gris
img= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print(img.shape)
#  binarisation
bw=cv2.threshold(img,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
```

Remarque
Maintenant, nous avons la fonction cv2.threshold avec le paramètre cv2.THRESH_BINARY et non cv2.THRESH-BINARY_INV parce que ce que nous avons sauvegardé pour les images de lignes de texte, la version binarisée inverse (elles sont déjà blanches sur fond noir).

1. Détection et délimitation des PAWs.

```
def detection_paws(im, vh):
  pos_paws=[]
  paws=[]
  k=-1
  h=len(vh)
  deb =0
  fin =0
  i=0
  while (i<h-1):
    if ((vh[i]==0) & (vh[i+1] > 0)):
      k+=1
      deb =i+1
      for j in range(deb, h-1):
        if ((vh[j]>0) & (vh[j+1]==0)):
          fin =j+1
          pos_paws.append([deb, fin])
          paw = im[0:,deb:fin]
          paws.append(paw)
          break
        else:
           j+=1
      i=fin
    else:
        i+=1
  return k+1, pos_paws, paws
```

Appliquons la projection d'histogramme vertical :

```
def histo_v(im):
  w = im.shape[1]
  hist_v = np.zeros((w),np.uint16)
  for i in range(w):
    hist_v[i]=np.count_nonzero(im[:,i])
  return hist_v
```

Appel :

```
hist_v = histo_v(bw)
```

Détection et séparation des images respectives :

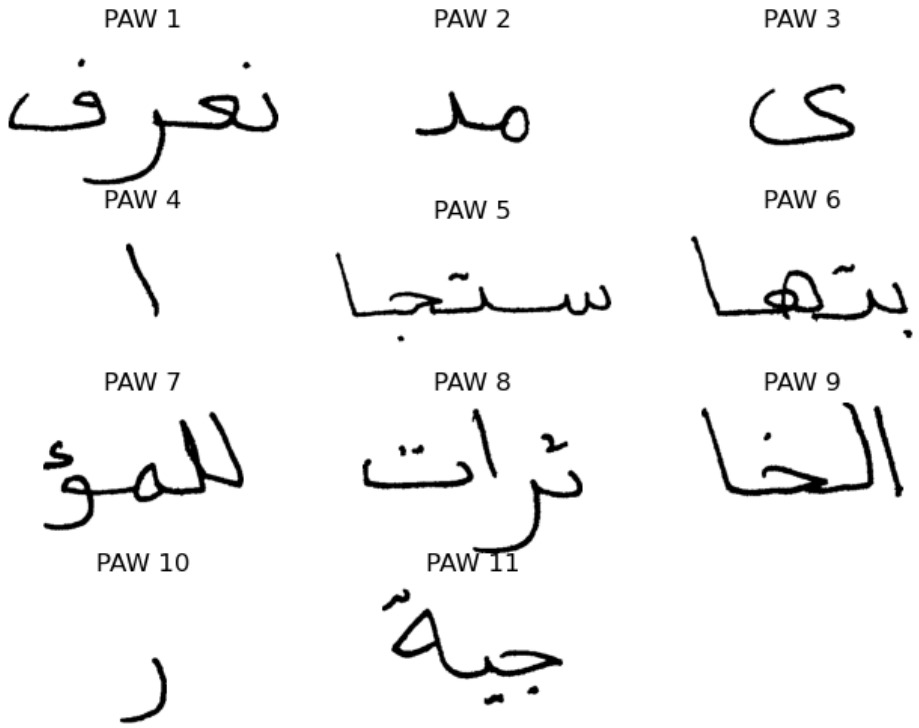
```
nbp, pos_pws, pws = detection_paws(bw, hist_v)
print ('nb_paws', nbp, '\t loc', pos_pws)
```

Sauvegarde des images respectives dans le répertoire correspondant :

```
for i in range(nbp):
  cv2.imwrite('resultat/PAWs/_ligne3_'+str(i)+'.png',pws[i])
```

Affichage :

```
plt.figure(figsize=(8, 6))
for i in range(1,nbp+1):
    plt.subplot(4, 3, i)
    plt.imshow(~pws[-i], cmap=plt.cm.gray)
    plt.title('PAW '+ str(i))
    plt.axis('off')
    plt.savefig('resultat/paws_lig3.png')
plt.show()
```



Remarques:

1. On affiche les PAWs dans le sens inverse de leurs traitements parce le texte arabe s'écrit de la droite vers la gauche et la séparation de la gauche vers la droite.
2. L'affichage a ete fait pour les images inversees pour retrouver les images habituelles noires sur fond blanc avec **plt.imshow(~pws[-i], cmap=plt.cm.gray).**
3. La remarque la plus importante concernant les PAWs détectées est qu'elles ne correspondent toutes aux PAWs réelles de la ligne d'écriture.
    a. Exemples PAW1 et PAW8.
    b. PAW1 correspond à deux PAWs dans la réalité.
    c. PAW8 correspond à trois.
4. Le problème de la méthode de projection d'histogramme est qu'elle ne peut pas résoudre les problèmes d'overlap (chevauchement vertical) entre les PAWs, ce qui est le cas pour les PAWs 1 et 8.