

Rapport de TP2 – Lecture automatique des chiffres par analyse d'image

I. Introduction

Ce TP consiste à réaliser des programmes permettant de réaliser de la reconnaissance de chiffres sur une image à l'aide d'une base de donnée de 10 nombres.

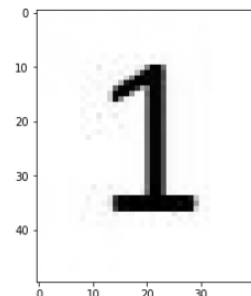
II. Section 1 du TP

1. Question (1).

Le main fonctionne correctement et affiche bien l'image souhaité à l'aide des fonction d'affichages suivantes :

```
# Variables utiles
path_to_assets = '../assets/'
plt.ion() # Mode interactif de matplotlib

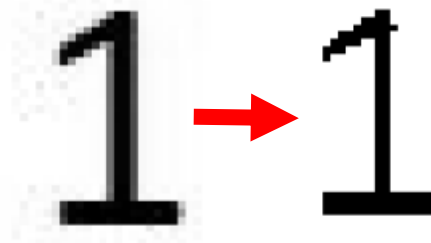
#=====
# Lecture image et affichage
#=====
image = Image()
image.load(path_to_assets + 'test10.JPG')
image.display("Exemple d'image")
```



III. Section 2 du TP

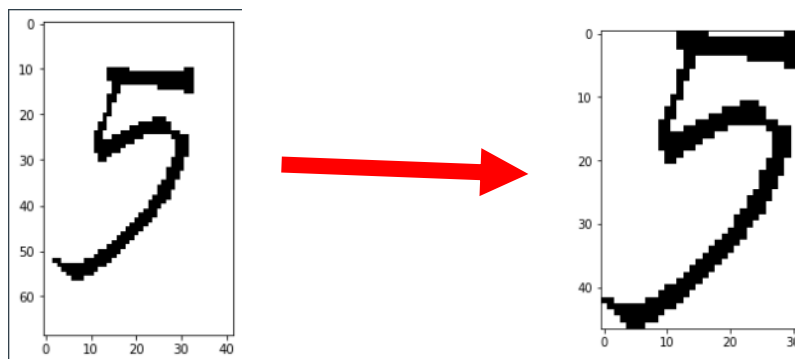
- 1) Dans la classe image, les attributs H et W correspondent respectivement au nombre de pixels de hauteur et de largeur d'une image. L'attribut pixels correspond au tableau numpy de l'image, ainsi pour une image « img » on aura par exemple : `img.pixels[i][j]` qui renverra la valeur entre 0 et 255 du pixel situé à la ligne i et à la colonne j.
- 2) Notre fonction `binarisation(self, S)` prend en compte une image et un seuil S. Ici nous allons créer une nouvelle image de la même taille que l'image entrée en paramètres ayant comme valeur 0 (pixel noir) partout. Par la suite nous parcourons le tableau de l'image entrée en paramètres et comparons chaque pixel avec notre valeur seuil. Si la valeur du pixel est supérieur à S, la valeur du pixel dans notre nouvelle image devient 255 (pixel blanc). L'image que la fonction retourne ne sera donc composé que de pixels de valeur égale à 0 ou 255.

```
def binarisation(self, S):
    im_bin=Image()
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))
    for i in range(self.H):
        for j in range(self.W):
            if (self.pixels[i][j]>S):
                im_bin.pixels[i][j]=255
    return im_bin
```



- 3) Pour la fonction `localisation(self)` on va parcourir l'image en ayant préalablement initialisé des valeurs `lmin`, `lmax`, `cmin`, `cmax` correspondant aux lignes et colonnes maximales. Si un pixel est noir, on va regarder sa position et l'affecter aux variables `lmin`, `lmax`, `cmin` ou `cmax` en fonction de la position du pixel. Si le pixel noir le plus bas sur l'image est à la ligne 48, alors on aura `lmax=47`. On recrée par la suite la nouvelle image allant partant de `lmin` jusqu'à `lmax` et de `cmin` jusqu'à `cmax`.

```
def localisation(self):
    lmin=self.H
    lmax=0
    cmin=self.W
    cmax=0
    for l in range(self.H):
        for c in range(self.W):
            if self.pixels[l][c]==0:
                if l<lmin:
                    lmin=l
                if l>lmax:
                    lmax=l
                if c<cmin:
                    cmin=c
                if c>cmax:
                    cmax=c
    im_bin = Image()
    im_bin.set_pixels(np.zeros((lmax-lmin+1, cmax-cmin+1), dtype=np.uint8))
    for i in range(lmin,lmax+1):
        for j in range(cmin,cmax+1):
            im_bin.pixels[i-lmin][j-cmin]=self.pixels[i][j]
    return im_bin
```



Lors du lancement de `test_image`, la fonction répond à toutes les spécifications demandées.

IV. Reconnaissance automatique de chiffres

La fonction `resize(self, new_H, new_W)` redimensionne l'image pour avoir le nombre de pixels désiré.

Pour cela on utilise la fonction `resize` déjà présente dans python qui nous renvoie un tableau de valeurs réelles, il faut multiplier ce tableau par 255 pour obtenir des entiers, on peut alors recréer l'image correspondante à ce nouveau tableau.

```
def resize(self, new_H, new_W):
    im_bin=Image()
    tab1=resize(self.pixels, (new_H,new_W), 0)
    tab_bin=(np.uint8(tab1*255))
    im_bin.set_pixels(tab_bin)
    return im_bin
```

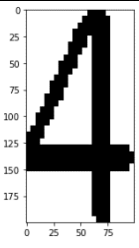
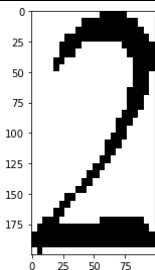
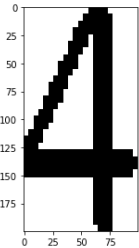
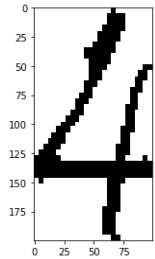
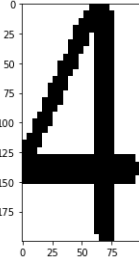
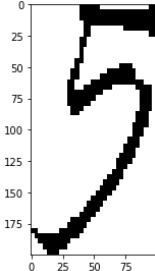
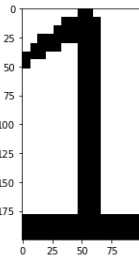
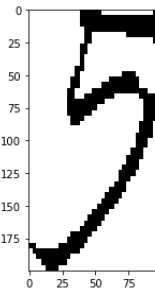
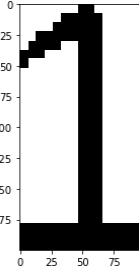
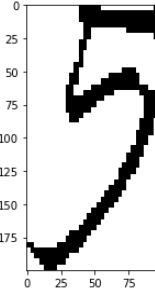
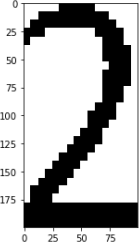
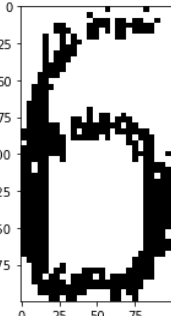
La fonction `similitude(self, im)` compare 2 image pour comparer le niveau de similitude entre ces deux, une valeur comprise entre 0 et 1. Pour cela on va comparer les deux images pixel par pixel de même coordonnées, lorsque la valeur de ce pixel est la même, alors on ajoute 1 à la somme des pixels similaire. On divise alors ce résultat par le nombre de pixel de cette image pour obtenir un ratio de pixels identiques au sein des deux images.

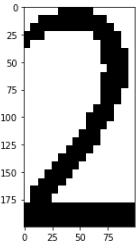
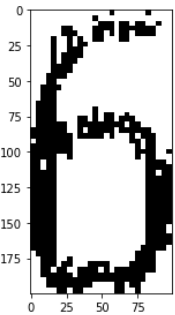
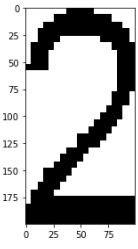
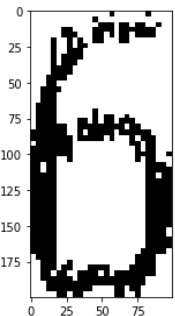
```
def similitude(self, im):  
    n=0  
    for l in range (self.H):  
        for c in range (self.W):  
            if self.pixels[l][c]==im.pixels[l][c]:  
                n+=1  
    return n/(self.H*self.W)
```

Nous réalisons enfin la dernière fonction qui regroupe tout ce que nous avons fait précédemment. La fonction `reconnaissance_chiffre(image, liste_modeles, S)` compare une image avec plusieurs modèles et renvoie le modèle auquel elle ressemble le plus. D'abord nous binarisons puis localisons l'image grâce aux fonctions prévues à cet effet. Puis nous parcourons la liste de model et pour chaque modèle nous redimensionnons l'image et la comparons avec le model. Nous retenons alors l'indice de l'image ayant la similitude la plus élevée et renvoyons son indice.

```
def reconnaissance_chiffre(image, liste_modeles, S):  
    image=image.binarisation(S)  
    image=image.localisation()  
    simi=0  
    for i in range (len(liste_modeles)):  
        im=image.resize(liste_modeles[i].H, liste_modeles[i].W)  
        if im.similitude(liste_modeles[i])>simi:  
            simi=im.similitude(liste_modeles[i])  
            indiceim=i  
    return indiceim
```

Le fichier `test_reconnaissance` confirme le bon fonctionnement de notre programme, il retourne « OK ». Nous pouvons ensuite tester notre programme pour différentes images et valeurs de seuil. Voici un petit tableau récapitulatif de nos tests :

<u>Seuil</u>	<u>Image</u>	<u>Reconnaissance</u>	<u>Seuil</u>	<u>Image</u>	<u>Reconnaissance</u>
50		4	100		2
100		4	100		4
150		4	50		7
50		1	100		7
100		1	150		5
100		2	50		0

150		2	100		0
100		2	150		0

Le seuil marchant le mieux est 150 car c'est le seul qui marche pour la reconnaissance du chiffre 5.

V. Conclusion

En somme, nous avons pu finir ce TP sans rencontrer de gros problèmes. La compréhension de la syntaxe et de l'utilité fonctions au départ est le seul accroc que nous avons eu mais celui-ci a vite été résolu.

On remarque assez rapidement les limites de notre code, il ne peut pas reconnaître plusieurs chiffres sur une même image, et se trompe parfois si l'image de différence beaucoup de celle donné au départ.