LEBRUN Mathilde

HAVELANGE Victor TD1

TP N°2 du 02/12/21

LECTURE AUTOMATIQUE DE CHIFFRES PAR ANALYSE D'IMAGE

I- Introduction

Dans ce TP, nous allons apprendre à représenter des images numériques et à les modifier.

II- Prise en main de l'environnement

Nous avons tout d'abord exécuté le fichier main.py, l'image s'affiche correctement.

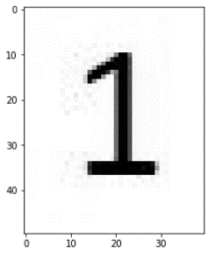


Image non binaire
avec nuances de gris

III- Travail préparatoire

W = width représente la largeur de la grille

H = height représente la hauteur de la grille

2) Ecriture de la méthode binarisation :

On veut définir une nouvelle image binaire avec des cases toutes noires ou toutes blanches.

Soit im_bin.pixels = 255 ou im_bin.pixels = 0

```
def binarisation(self, S):
    im_bin = Image()
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))
    for l in range(self.H):
        for c in range(self.W):
            if self.pixels[l,c]>S:
                im_bin.pixels[l,c] = 255
            else:
                im_bin.pixels[l,c] = 0
    return im_bin
```

3) Méthode de localisation:

Ici, nous voulons recadrer l'image, autrement dit que le chiffre prenne toute l'image pour après la redimensionner et enfin la comparer avec les images de référence.

Nous avons eu deux idées :

- La méthode 4 boucles pour avoir séparément lmin, lmax, cmin et cmax
- La méthode on parcourt une seule fois la grille et on réajuste cmin et lmin en fonction de la précédente colonne ou ligne

Nous avons finalement choisi le deuxième programme, qui était finalement plus simple à coder :

Par exemple, pour les colonnes, on veut connaitre cmin, donc la colonne la plus à gauche comportant au moins 1 pixel noir.

On parcourt la grille avec deux boucles for et dès que l'on tombe sur un pixel noir on affiche cmin = c, et si on tombe sur un c inférieur on remplace cmin par celui-ci.

Raisonnement analogue pour cmax, lmin et lmax.

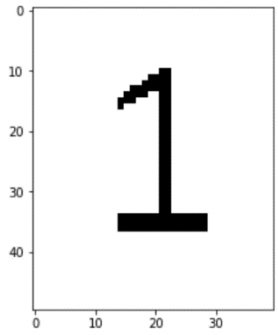
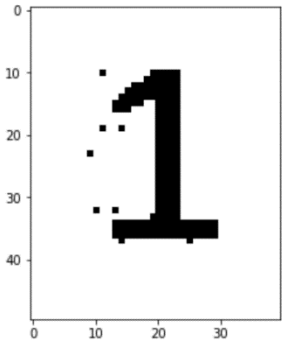
```
def localisation(self):
    #baobab
     c_min= self.W
     c_max=0
     l_min=self.H
     l_max=0
     for l in range(self.H):
         for c in range(self.W):
             if self.pixels[l][c]==0:
                 if c<=c_min:
                    c_min=c
                 if c>=c_max:
                    c_max=c
                 if l<=l_min:
                    l_min=l
                 if l>=l_max:
                    l_max=l

     imagette= Image()
     imagette.set_pixels(self.pixels[l_min:l_max+1,c_min:c_max+1])
     return imagette
```

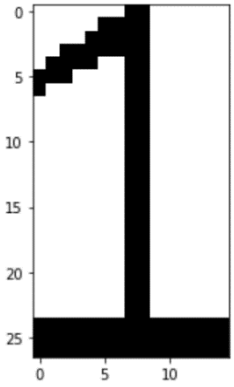
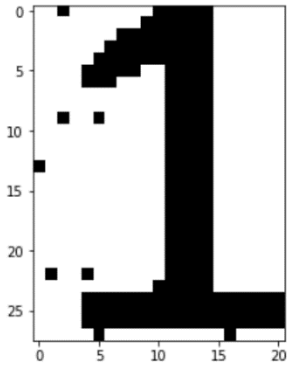
IV – Reconnaissance automatique de chiffre

Seuil : 60 seuil : 240

1) binarisation : résultats

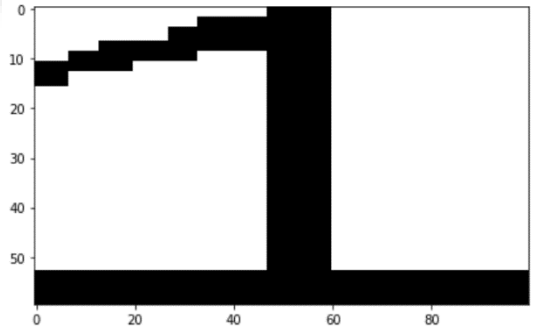



2) localisation : résultats

3) On utilise la fonction np.uint8(pixels_resized*255) avec notre tableau auquel on a déjà appliqué la méthode.

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



POLYTECH ANNECY-CHAMBÉRY

4) Pour la méthode similitude, on suppose en premier lieu que les deux images à comparer ont exactement la même taille.

Ensuite, on réalise une double itération avec l et c dans le tableau afin de voir si self.pixels([i,j] a la même valeur (blanche ou noire) que image2.pixels([i,j].

On note le nombre de pixels identiques et on le divise par le nombre de pixels total dans le tableau (self.W*self.H) pour trouver la proportion de pixels identiques.

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

5) A partir d'une image donnée, on effectue la méthode binarisation et puis la méthode localisation pour obtenir une image contenant seulement des pixels avec seulement 2 valeurs possibles (0 ou 255) et qui est recadrée.

Ensuite, pour chacune des autres images auxquelles on veut la comparer, on la "resize" à la taille de l'autre image et nous réalisons la méthode "similitude" avec ces deux images.

Dans notre cas, nous avons fait une liste contenant la similitude de notre image de départ à chacune des images de chiffres disponibles et nous sortons comme résultat l'indice de l'élément de la liste ayant la plus grande valeur (et donc la plus grande similitude). Dans notre cas, ça nous ressort le chiffre 1, ce qui était bien attendu.

```
Le chiffre reconnu est :  ([0.538, 0.922, 0.68, 0.611, 0.484, 0.592, 0.557, 0.619, 0.555, 0.567], 1)
```

On voit bien que notre image est similaire à 92% par rapport à l'image du chiffre 1.

Notre code pour cette fonction :

```
def reconnaissance_chiffre(image, liste_modeles, S):
    liste=[]
    b=image.binarisation(S)
    l=b.localisation()

    for ref in liste_modeles:
        im_resized=l.resize(ref.H,ref.W)
        s =im_resized.similitude(ref)
        s1 = round(s,3)
        liste.append(s1)
        '''for i in range(len(liste)):
            num = 0
            if liste[i]>liste[num]:
                num=i'''

    return liste.index(max(liste))
```

Ce code nous renvoie juste le chiffre 1. La liste affichée juste avant le code nous montre d'où vient ce chiffre 1.

Conclusion

Dans ce TP, nous avons mis en œuvre les outils de base de la représentation graphique en Python afin de coder un programme permettant la reconnaissance automatique de caractères. Nous avons utilisé plusieurs méthodes comme la binarisation d'une image ayant différents niveaux de gris, le recadrage et le redimensionnement d'image ainsi que la comparaison d'images. Grâce à ces méthodes, nous avons réussi à déterminer à quel chiffre pouvait-on identifier l'image de base que l'on avait reçu.