

Rapport de TP2 – Reconnaissance chiffres

I. Introduction

L'objet de ce TP est d'utiliser la reconnaissance des chiffres par corrélation avec des modèles, ici nous l'appliquerons pour coder les pixels d'une image en noir et blanc. Pour cela nous allons coder les couleurs en binaire, localiser les pixels, et recadrer l'image autour du nouveau caractère.

III. Travail préparatoire

1. Question (1)

Pixel est associé à un tableau 2D vide dont la hauteur est H et la largeur est W, les valeurs de H et W sont initialisées à 0 donc le tableau est pour le moment vide et inexistant.

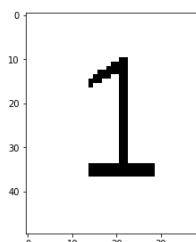
2. Question (2)

On écrit la méthode `binarisation(self,S)` qui code une image initialement codée sur 256 valeur sur 2 bits (le pixel prends pour valeur 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  
            else :  
                im_bin.pixels[i,j]=0  
    return im_bin
```

On parcourt le tableau avec `for i in range(0,self.H)` qui parcourt la hauteur du tableau et `for j in range(0,self.W)` qui parcourt la largeur. L'utilisateur rentre une valeur limite S, si la valeur associée au pixel est supérieure à S, on initialise sa valeur à 255 (elle devient donc blanche), si sa valeur est inférieure à S, on l'initialise à 0(elle devient noire).

En testant notre code on obtient bien l'image uniquement en noir et blanc avec un `S=70` :



3. Question (3)

On cherche maintenant à recadrer la nouvelle image codée en binaire. Pour cela on parcourt une nouvelle fois le tableau en longueur et en largeur avec la même méthodologie que précédemment.

```
def localisation(self):  
    im_loc = Image()  
    c_min = self.w  
    l_min = self.h  
    c_max = 0  
    l_max = 0  
    for i in range(self.H):  
        for j in range(self.W):  
            if self.pixels[i][j] == 0:  
                if i < l_min:  
                    l_min = i  
                if i > l_max:  
                    l_max = i  
                if j < c_min:  
                    c_min = j  
                if j > c_max:  
                    c_max = j  
    im_loc.set_pixels(self.pixels[l_min:l_max,c_min:c_max])  
    return im_loc
```

Lorsqu'on parcourt le tableau, quand on a un pixel noir on enregistre ces coordonnées [i,j] de façon :

- l_min prend la valeur du plus petit i
- l_max du plus grand i
- c_min prend la valeur du plus petit j
- c_max du plus grand j

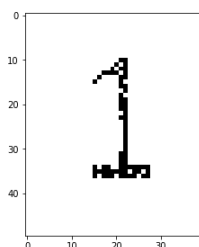
notre fonction retourne l'image seulement entre les nouvelles coordonnées et cela affiche l'image rognée :

IV. Reconnaissance automatique de chiffres

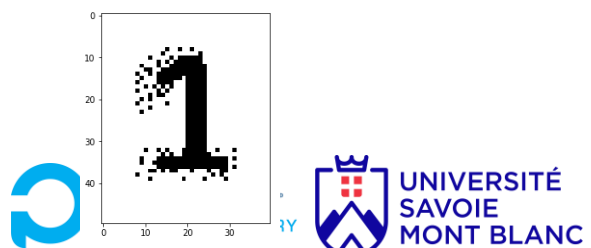
1. Question (1)

Nous changeons la valeur du seuil S dans le fichier main.py on observe que en augmentant la valeur de S, le chiffre s'épaissit (si on s'approche beaucoup de 255, des pixels qui ne font pas partis du caractère deviennent noirs aussi) et lorsqu'on la diminue il s'affine (si on descend en dessous de 10 on a même du mal à reconnaître le caractère).

Exemple pour S=5



Exemple pour S=250

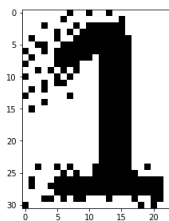


Le module nose2 n'est pas installé sur l'ordinateur, le test sur GitHub n'indique aucune erreur.

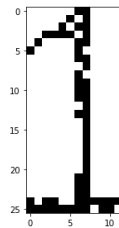
2. Question (2)

En changeant la valeur du seuil, quand on effectue la fonction localisation on obtient bien les mêmes résultats que pour la question 1, mais avec les caractères rognés.

Pour $S = 250$

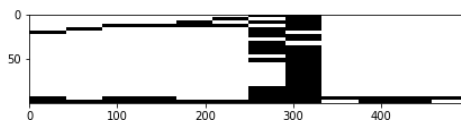


Pour $s = 5$

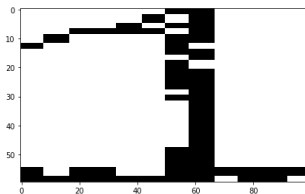


3. Question (3)

Nous avons créé la méthode `resize`, qui permet de redimensionner l'image à partir de coordonnées rentrées par l'utilisateur. Avec les coordonnées de base (100,500) cela affiche :



Avec les coordonnées (60,100) cela affiche :



Après avoir tester notre main sur Github nous n'avons pas d'erreur.

4. Question (4)

Nous avons créé la fonction `similitude` :

Pour cela on initie un compteur qui compte le nombre de pixels identiques entre les deux fonctions


Pour chaque pixel on vérifie qu'il est identique à celui de l'autre image pour les mêmes coordonnées.

Si c'est le cas le compteur augmente d'1.

La fonction renvoie le nombre de pixels identiques sur le nombre de pixel total de l'image.

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

En testant sur github nous n'avons pas d'erreur sur la fonction.

 **Autograding complete**
Points 22/25

5. Question (5)

Pour binariser et localiser l'image on initie im_loc et im_bin de sorte que :
im_bin =binarisation(s) et im_loc=localisation()

On parcourt ensuite la liste liste_modèle pour comparer l'image avec tous les modèles de la liste. Le modèle ayant la similitude la plus importante est retenu et la fonction retourne l'indice de l'image avec laquelle on a la similitude la plus importante, et leur pourcentage de similitude :

```
def reconnaissance_chiffre(image, liste_modeles, S):
    sim = []
    im_bin = image.binarisation(S)
    im_loc = im_bin.localisation()
    for i in liste_modeles:
        im_res = im_loc.resize(i.H, i.W)
        sim.append(im_res.similitude(i))
    return sim.index(max(sim))
```

Ici la fonction renvoie : Le chiffre reconnu est : 2

On peut donc en conclure que l'image avec la plus grande similitude est la 3eme de la liste (à l'indice 2).

6. Question (6)

	Indice	Seuil
Test 1	4	70
Test 2	4	45
Test 3	2	95

On remarque qu'en changeant le seuil, l'image la plus similaire reste la même.