RENAUD Adrien
LE BOS Youen
03/12/2021

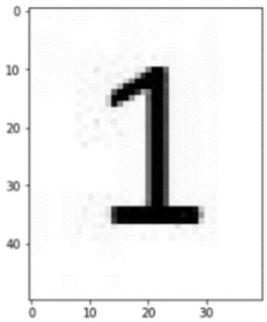
# Rapport de TP2 – Reconnaissance chiffres

## I. Introduction

Ce TP doit nous permettre de programmer avec python une méthode de reconnaissance des chiffres par corrélation avec des modèles. Ici nous l'utiliserons pour coder les pixels d'une image en noir et blanc. Pour cela nous alors coder les couleurs en binaire, localiser les pixels, et recadrer l'image autour du nouveau caractère.

## III. Travail préparatoire

Nous exécutons le programme demandé dans le fichier main.py. On obtient donc :



# 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)

```
    def binarisation(self, S):
        im_bin = Image()
        im_bin.set_pixels(np.zeros((self.H , self.W),dtype=np.vint8))
        for i in range(self.H):
            for k in range(self.W):
                if self.pixels[i,k] >= S :
                    im_bin.pixels[i,k]=255
                else:
                     im_bin.pixels[i,k]=0
        return im_bin

```

On parcourt le tableau avec 2 boucles for. Ensuite, l'utilisateur rentre la valeur de S, et si la valeur du pixel est inférieur à S, alors celui-ci prend la valeur de 0 et la couleur devient noirs. Par ailleurs, quand la valeur du pixel est supérieure à S, celui-ci prend la valeur de 255 et deviens blanc.

L'objectif de cette manœuvre est de rentre plus lisible le chiffre. Ce qui est bien le cas car le chiffre devient tout en noir et blanc et plus net (pour S=70) :

## 3. Question (3)

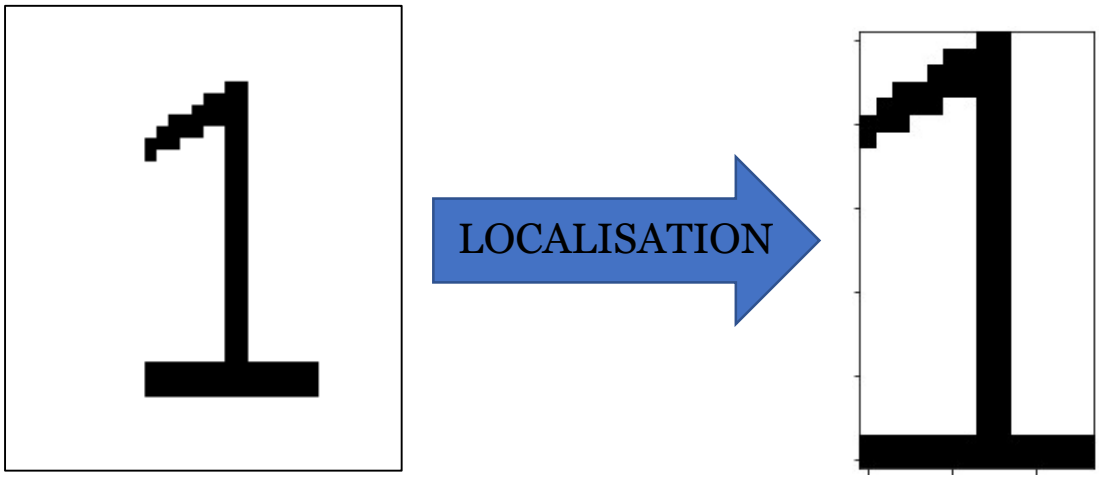
On cherche maintenant à recadrer la nouvelle image codée en binaire. Pour cela on parcourt une nouvelle fois le tableau en longueur et en largeur avec la même méthodologie que précédemment.

```
    def localisation(self):
        im_loc = Image()
        imin = self.H
        imax = 0
        kmin = self.W
        kmax = 0
        for i in range(self.H):
            for k in range(self.W):
                if self.pixels[i][k]==0:
                    if k<=kmin :
                        kmin = k
                    if k>=kmax :
                        kmax = k
                    if i<=imin :
                        imin = i
                    if i>=imax :
                        imax = i
        im_loc.set_pixels(self.pixels[imin:imax,kmin:kmax])
        return im_loc
```

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

imin prend la valeur du plus petit i
imax du plus grand i
kmin prend la valeur du plus petit k
kmax du plus grand k

Voici ce que retourne la fonction :




# 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 et lorsqu'on la diminue il s'affine. On peut justifier cela par le fait que quand le seuil est bas, il y a un manque d'information donc l'image n'est pas complète. Inversement, quand le seuil est très haut, l'image est surchargée et imprécise.

Exemple pour S=5

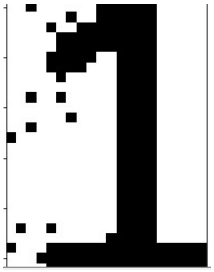
Exemple pour S=245

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

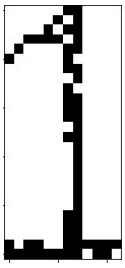
## 2. Question (2)

Nous faisons exactement la même chose que précédemment mais en utilisant la fonction localisation.

Pour S = 245



Pour s = 5



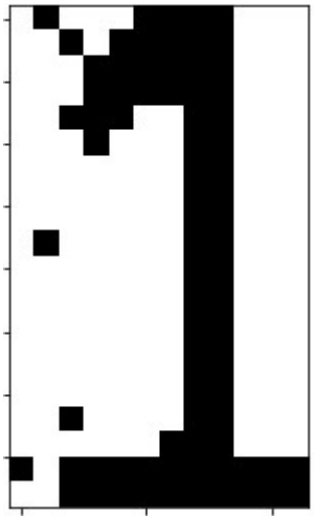
## 3. Question (3)

La méthode resize permet de redimensionner l'image à partir de coordonnées rentrées par l'utilisateur. Cela permet de redéfinir l'échelle de l'image initial.

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

Avec les coordonnées (250,500) cela affiche :

Avec les coordonnées (20,12) cela affiche :



## 4. Question (4)

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

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.

## 5. Question (5)

Pour binariser et localiser l'image, on initie iloc et ibin de sorte que :
ibin =binarisation(s) et iloc=localisation()

On parcourt ensuite la liste liste_modèles à l'aide de deux boucles for pour comparer l'image avec tous les modèles de la liste. Le modèle retenu par le logiciel est celui qui a le plus de point commun avec l'image :

```
def reconnaissance_chiffre(image, liste_modeles, S):
    res = []
    ibin = image.binarisation(S)
    iloc = ibin.localisation()
    for k in liste_modeles:
        ires =iloc.resize(k.H, k.W)
        res.append(ires.similitude(k))
    return res.index(max(res))
```

```
Le chiffre reconnu est :  1
```

## 6. Question (6)

Nous allons utiliser notre fonction afin de tester ses limites. Nous allons ainsi comparer différents résultats :

| | seuil | Résultat attendu | indice |
|---|---|---|---|
| Test 1 | 20 | 4 | 4 |
| Test 1 | 140 | 4 | 4 |
| Test 1 | 240 | 4 | 4 |
| Test 3 | 100 | 2 | 2 |
| Test 3 | 240 | 2 | 2 |
| Test 7 | 136 | 5 | 4 |
| Test 7 | 50 | 5 | 4 |

## 7. Conclusion

Durant ce TP nous avons essayé d'améliorer la qualité d'une image à l'aide de plusieurs fonctions python (binarisation, localisation…). La dernière question nous a permis de tester au global notre étude afin de voir si notre programmation était correcte. Nous avons décelé certaines erreurs néanmoins notre programmation nous semble fiable du fait d'un grand nombre de test et d'une minorité d'erreurs.