DESCOMPS Alexandre
GONAY Arthur
Date du TP (19/11/2021)

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

## I. Introduction

Ce tp nous introduit à la reconnaissance d'une image grâce à la programmation avec pour but d'analyser des images pour reconnaitre des chiffres.

## II. Section 1 du TP

### 1. Question (1).

Il s'agissait ici de binariser une image, c'est-à-dire transformer les différents niveaux de gris en noir ou blanc pour n'avoir que 2 couleurs et faciliter l'analyse de l'image.
On transforme un niveau de gris en noir ou blanc selon un seuil critique S.

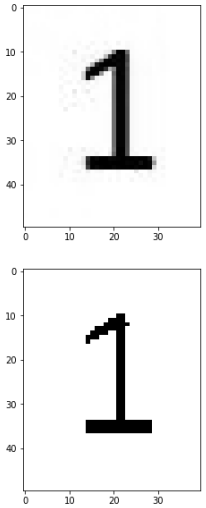
*Code utilisé :*

```
def binarisation(self, S):
    # creation d'une image vide
    im_bin = Image()

    # affectation a l'image im_bin d'un tableau de pixels de meme taille
    # que self dont les intensites, de type uint8 (8bits non signes),
    # sont mises a 0
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))

    # TODO: boucle imbriquees pour parcourir tous les pixels de l'image in
    # et calculer l'image binaire
    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


    pass
```

Pour calculer l'image binaire, on compare les valeurs des pixels de l'image, si ces dernières sont supérieures à S, on leur attribue la valeur de 255 (blanc). Dans le cas contraire, ces pixels auront la valeur de 0 (noir).

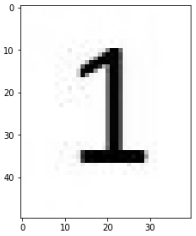
Nous avons affiché quelques résultats à la page suivante.

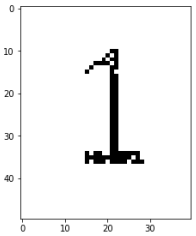
**Avec S = 70 :**



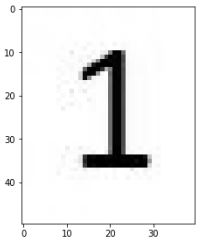


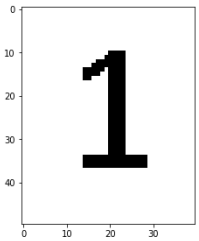
**Avec S = 05 :**





**Avec S = 150 :**





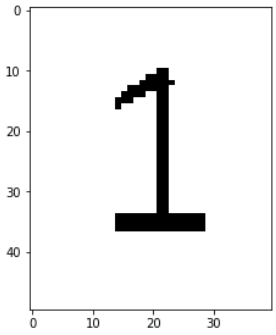
## 2. Question (2).

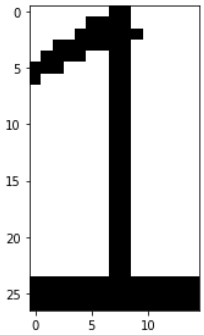
Après avoir binarisé l'image, il fallait la recadrer de façon à ce qu'elle fasse la plus petite taille possible sans que des pixels noirs soient coupés.

Nous avons donc programmé 4 'grosses boucles' qui parcourent successivement les lignes depuis le haut, les lignes depuis le bas, les colonnes depuis la gauche et les colonnes depuis la droite. Nous parcouront ces différents éléments jusqu'à ce que le programme trouve un pixel noir afin de déterminer quelles sont les dimensions idéales de l'image. Nous obtenons donc le code suivant :

```
def localisation(self):
    top = 0
    bottom = 0
    left = 0
    right = 0

    for i in range (self.H):
        for j in range (self.W):
            if self.pixels[i][j] == 0:
                if top == 0:
                    top = i

    for i in range (self.H-1, -1, -1):
        for j in range (self.W):
            if self.pixels[i][j] == 0:
                if bottom == 0:
                    bottom = i+1

    for i in range(self.W):
        for j in range(self.H):
            if self.pixels[j][i] == 0:
                if left == 0:
                    left = i

    for i in range(self.W-1, -1, -1):
        for j in range(self.H):
            if self.pixels[j][i] == 0:
                if right == 0:
                    right = i+1

    im_bin = Image()
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))
    im_bin.pixels=self.pixels[top:bottom,left:right]
    return im_bin
```

La méthode localisation permets d'obtenir un résultat de ce type :



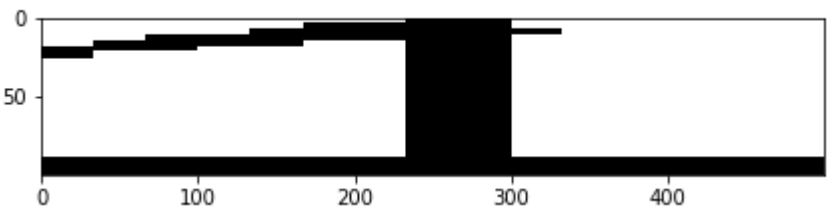




### 3. Question (3).

Nous souhaitons maintenant définir une méthode permettant de redimensionner une image.
D'abord, nous créons une instance d'Image ; ensuite nous définissons les valeurs des nouveaux pixels à l'aide de la fonction « resize » puis nous remplissons notre instance d'image avec ces valeurs obtenues.
Attention à ne pas oublier de se ramener à une image en int avec la commande « np.uint8(pixels_resized*255)

```
def resize(self, new_H, new_W):

    img1 = Image()
    pixels_resized = resize(self.pixels, (new_H,new_W), 0)
    img1.set_pixels(np.uint8(pixels_resized*255))
    return img1
```

Cela nous permet d'obtenir le résultat suivant :



### 4. Question (4).

Ensuite, nous voulons définir une méthode similitude(self,image) afin de mesurer la similitude par corrélation entre 2 images.

Dans notre code ici, nous avons anticipés la question suivante en redimensionnant l'image « self » aux dimensions de l'autre image à comparer.

Ensuite nous additionnons le nombre de pixels identiques, un résultat que nous diviserons par le nombre de pixels total pour avoir la corrélation.

```
def similitude(self, im):
    H = im.H
    W = im.W
    cor = 0
    img1 = self.resize(H,W)
    img2 = im.resize(H,W)
    for i in range(H):
        for j in range(W) :
            if img1.pixels[i][j] == img2.pixels[i][j]:
                cor += 1
    cor = cor/(H*W)

    return cor
```

### 5. Question (5).

On doit maintenant compléter le fichier reconnaissance.py afin de tester notre programme et vérifier son efficacité. Pour ce faire, nous devons binariser puis localiser notre image de base à comparer. Ensuite nous la comparons avec une image de la liste_modele. Notons que le redimensionnement se fait directement dans la méthode similitude. Nous stockons dans une variable le chiffre et l'indice de similitude du modèle avec la plus haute corrélation.

```
S = 70
def reconnaissance_chiffre(image, liste_modeles, S):
    img_bin = image.binarisation(S)
    img_loc = img_bin.localisation()
    similimax = 0
    for i in range (0,9):
        if img_loc.similitude(liste_modeles[i]) > similimax:
            similimax = img_loc.similitude(liste_modeles[i])
            chiffre = i
    return chiffre
```

Bien entendu, tout au long du TP nous avons pu vérifier notre code à l'aide de test_image.py ou test_reconnaissance.py

### 6. Question (6).

| Seuil | test1 | test2 | test3 | test4 | test5 | test6 | test7 | test8 | test9 | test10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 180 | 4 | 1 | 2 | 2 | 2 | 4 | 7 | 4 | 0 | 8 |
| 160 | 4 | 1 | 2 | 2 | 2 | 4 | 2 | 0 | 3 | 4 |
| 140 | 4 | 1 | 2 | 2 | 2 | 4 | 5 | 0 | 3 | 6 |
| 120 | 4 | 1 | 2 | 2 | 2 | 4 | 9 | 6 | 3 | 0 |
| 100 | 4 | 1 | 2 | 2 | 2 | 4 | 7 | 7 | 3 | 6 |
| 80 | 4 | 1 | 2 | 2 | 2 | 4 | 9 | 7 | 3 | 6 |
| 60 | 4 | 1 | 2 | 2 | 2 | 4 | 9 | 7 | 3 | 0 |
| 40 | 4 | 1 | 2 | 2 | 2 | 4 | 7 | 7 | 3 | 6 |
| 20 | 4 | 1 | 2 | 2 | 2 | 4 | 7 | 7 | 3 | 7 |
| valeur réel | 4 | 1 | 2 | 2 | 2 | 4 | 5 | 1 3 5 2 | 1 8 4 5 6 | 6 |

La valeur qui semble le mieux fonctionner pour nous est 140.

## II. Conclusion

Nous avons pu terminer le TP dans les temps, les difficultés majeures étaient de bien comprendre comment passer d'un tableau 2D à une image pour bien gérer les formats et display et la gestion des problèmes d'affichages.

C'était un TP très intéressant qui demande beaucoup d'autonomie et permets de découvrir un projet intéressant.