NICOLAS Thomas
GUERIOT Benjamin

Date du TP (19/11/2021)

Rapport de TP2

Lecture automatique de chiffres par analyse d'image

NICOLAS Thomas

&

GUERIOT Benjamin

Sommaire

NICOLAS Thomas
GUERIOT Benjamin

Introduction

A travers ce TP, nous allons apprendre à manipuler des images en utilisant notamment la librairie numpy. Par ailleurs, nous continuons a nous familiariser avec le langage de programmation python.

Le but de ce TP est des reconnaître automatique des images de chiffres à partir de différents avec des dimensions changeantes.

En amont, nous avons analysé les fonctions présentes dans le fichier **image.py**. Celui-ci contient une classe **Image** qui nous permettra de lire et afficher une image. Pour l'utiliser, il faut initialiser un objet à l'aide de deux valeurs en paramètres qui représentent la hauteur et la largeur de l'image. De plus, il faut initialiser un tableau nommé pixels. Cette classe contient de base quelques méthodes comme **set_pixels(self, tab_pixels)** qui remplir le tableau pixels. En outre, elle contient également une méthode **load** qui lit l'image à partir d'un fichier. Enfin, il y a également une méthode **display** qui affiche une représentation de l'image dans la console.

I/ Travail préparatoire

Cette partie se déroule dans la classe Image.

1)

Nous avons commencé par écrire une fonction **binarisation(self, S)** qui permet de passer d'une image codée sur 256 valeurs à une image avec seulement deux valeurs (0 ou 255). La valeur 0 correspond à du blanc et la valeur 255 à du noir. Pour ce faire, nous comparons chaque pixel de l'image à un seuil choisi qui correspond au S qui est en paramètre.

Nous avons itéré chaque pixels du tableau et les avons comparé avec notre seuil. Les valeurs inférieures au seuil sont remplacées dans un nouvel objet Image par la valeur 0. A contrario, les valeurs supérieures sont remplacées par 255. Ainsi, nous obtenons une image composée uniquement de pixels blancs ou noirs, ce qui correspond bien à une image binaire.

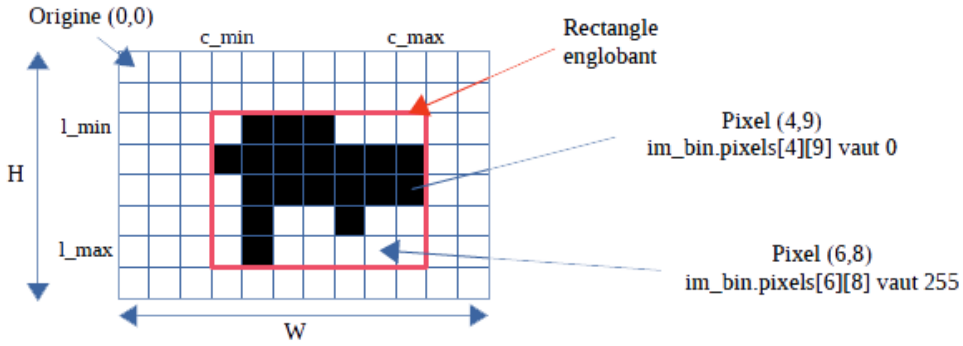
Voici la méthode que nous avons écrite :

```python
#============================================================================
# Methode de binarisation
# 2 parametres :
#   self : l'Image a binariser
#   S : le seuil de binarisation
#   on retourne une nouvelle Image binarisee
#============================================================================
def binarisation(self, S):

    im_bin = Image() # Instanciation d'un objet Image

    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))

    # Itération pour parcourir l'ensemble du tableau pixels
    for i in range(self.H) :
        for j in range(self.W) :
            if  self.pixels[i][j] < S : # On vérifie la condition de seuil
                im_bin.pixels[i][j] = 0 # Les valeurs inférieures deviennent 0
            else:
                im_bin.pixels[i][j] = 255 # Les valeurs supérieures deviennent 255

    return im_bin # On retourne l'Image
```

NICOLAS Thomas
GUERIOT Benjamin

2)

Dans un second temps, il nous a fallu créer la méthode **localisation(self)** qui permet de calculer et retourner l'image recadrée sur le chiffre à identifier.



L'objectif est de déterminer les coordonnés des minimums et maximums du rectangle englobant la forme (en rouge sur le graphique).

Voici la méthode que nous avons réalisé :

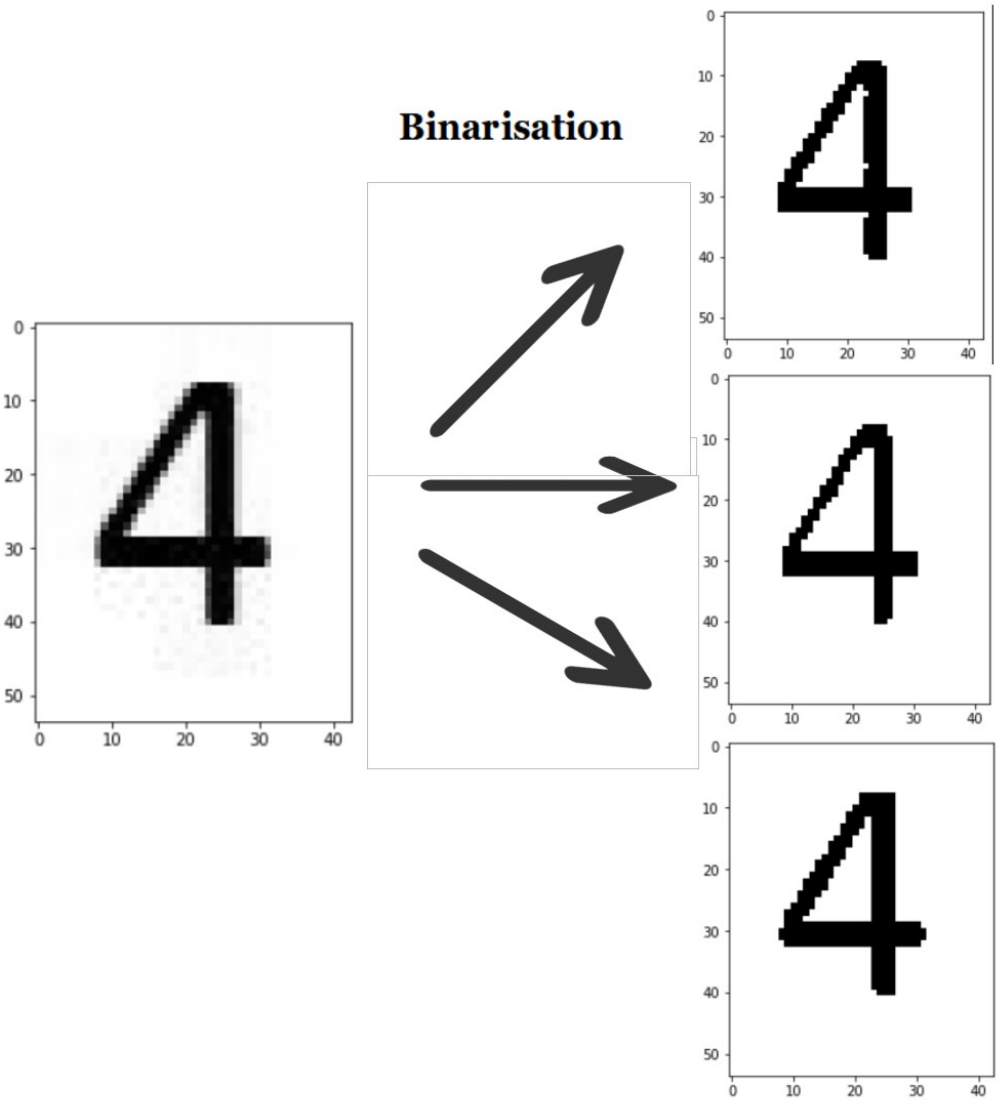
```
#=========================================================================
# Dans une image binaire contenant une forme noire sur un fond blanc
# la methode 'localisation' permet de limiter l'image au rectangle englobant
# la forme noire
# 1 parametre :
#   self : l'image binaire que l'on veut recadrer
#   on retourne une nouvelle image recadree
#=========================================================================
def localisation(self):
    im_loc = self # On copie l'Image original

    # Initialisation des valeurs
    c_max = 0
    c_min= im_loc.W
    l_max = 0
    l_min= im_loc.H

    # Itération pour parcourir l'ensemble du tableau pixels
    for i in range(self.H) :
        for j in range(self.W) :
            # On vérifie si le pixel de coordonnées i et j est égal à 0
            if im_loc.pixels[i][j] == 0:

                # Si la valeur en largeur est supérieure ou égale à c_max
                # on définit c_max comme j
                if j >= c_max:
                    c_max = j

                # Si la valeur en largeur est inférieure à c_min
                # on définit c_min comme j
                if j < c_min:
                    c_min = j

                # Si la valeur en hauteur est supérieure ou égale à l_max
                # on définit l_max comme i
                if i >= l_max:
                    l_max = i

                # Si la valeur en hauteur est inférieure à l_min
                # on définit l_min comme i
                if i < l_min:
                    l_min = i


    new_im = Image() # Instanciation d'un objet Image

    # On définit le tableau pixels de dimensions l_max-l_min et c_max-x_min
    # La fonction np.zeros remplit le tableau numpy de 0
    new_im.set_pixels(np.zeros((l_max-l_min,c_max-c_min),dtype=np.uint8))

    # On change les valeurs du tableau pixels du nouvel objet avec les valeurs
    # composants le rectangle englobant le chiffre
    new_im.pixels = im_loc.pixels[l_min:l_max+1,c_min:c_max+1]

    return new_im # On retourne l'Image
```

NICOLAS Thomas
GUERIOT Benjamin

II/ Reconnaissance automatique de chiffre

1)

Nous avons exécuté le fichier main pour observer les résultats obtenus avec différents seuils pour la méthode binarisation.



S = 70

S = 30

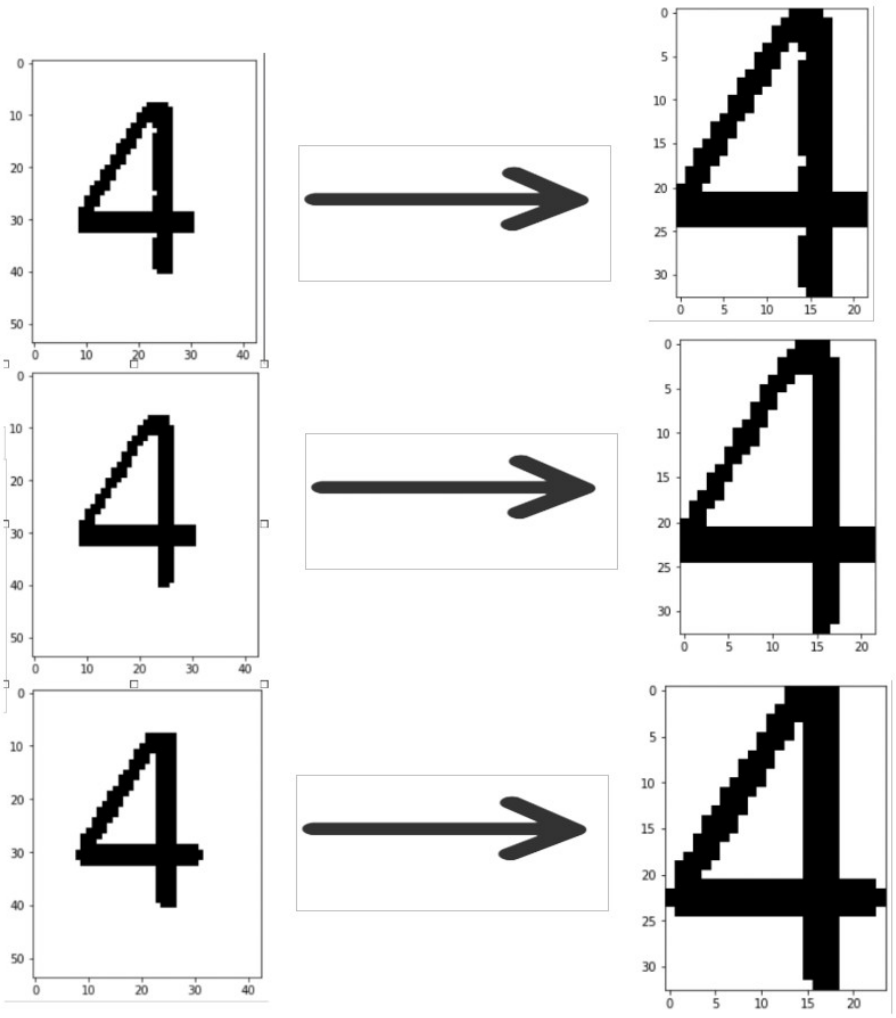
S = 120

Tous les tests portant sur la méthode binarisation ont été réalisé sans aucun problème.

NICOLAS Thomas
GUERIOT Benjamin

2)

Ensuite nous avons observé les résultats obtenus avec différents seuils pour la méthode localisation.



Tous les tests portant sur la méthode localisation ont été réalisé sans aucun problème.

NICOLAS Thomas
GUERIOT Benjamin

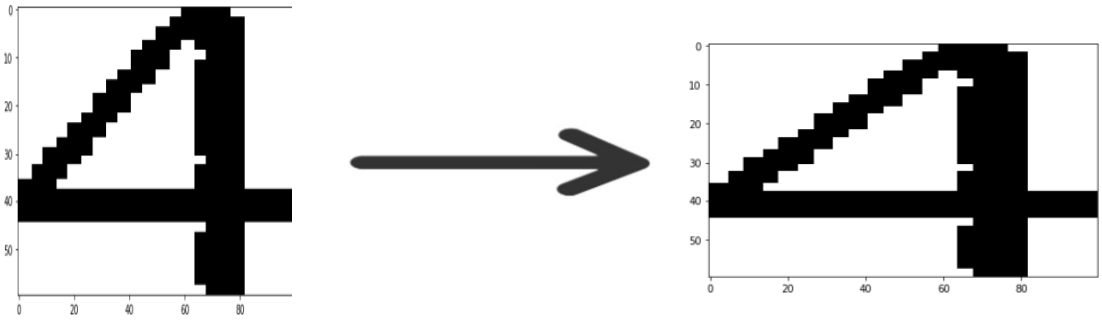
3)

Nous avons ensuite ajouté une fonction **resize(self, new_H, new_W)** qui redimensionne l'image à la taille voulue et renvoie un autre objet de type Image. Cette méthode utilise la fonction resize de la librairie skimage.

Nous avons réalisé la fonction comme ceci :

```
#=============================================================================
# Methode de redimensionnement d'image
#=============================================================================
def resize(self, new_H, new_W):

    im_res = Image() # Instanciation d'un objet Image

    # On crée un tableau numpy 2D basé sur le tableau pixels redimensionné
    # avec la fonction resize
    pixels_resized = resize(self.pixels, (new_H,new_W), 0)

    # Afin de transformer notre tableau numpy en float a un tableau de pixels
    # en int, on utilise la fcontion numpy uint8()
    im_res.set_pixels(np.uint8(pixels_resized*255))

    return im_res # On retourne l'Image
```

Nous avons ensuite lancé le fichier main pour analyser les résultats obtenus. Voici ce que nous obtenons :



NICOLAS Thomas
GUERIOT Benjamin

4)

Il fallait ensuite ajouter une méthode **similitude(self, image)** qui mesure la similitude par corrélation d'images entre l'image représenté par l'objet courant (self) et un objet de type Image entrée en paramètre.

Nous avons réalisé cette méthode comme ceci :

```
#=============================================================================
# Methode de mesure de similitude entre l'image self et un modele im
#=============================================================================
def similitude(self, im):

    # Initialisation des valeurs
    tot = 0
    sim = 0
    hauteur = im.H
    largeur = im.W

    im_sim = self # On copie l'Image original

    # Itération pour parcourir l'ensemble du tableau pixels
    for i in range(hauteur) :
        for j in range(largeur) :
            # On incrémente tot de 1 pour obtenir le total de pixels
            tot += 1

            # On vérifie si les pixels de mêmes coordonnées sont équivalent
            if im_sim.pixels[i][j] == im.pixels[i][j]:
                sim += 1    # On ajoute 1 à similitude

    return sim/tot # On retourne le rapport sim/tot
```

5)

Nous changeons désormais de fichier et nous intéressons à **reconnaissance.py**. Nous écrivons la fonction **reconnaissance_chiffre(image, liste_modeles, S)** qui effectue la reconnaissance de chiffre sur l'image donnée en paramètre. Il faut donc d'abord binariser puis localiser l'image. Puis, il faut calculer la similitude avec tous les modèles sans oublier de redimensionner l'image suivant le modèle. Cette retourne enfin le numéro de modèle avec la plus grande similitude.

Voici notre méthode :

```
def reconnaissance_chiffre(image, liste_modeles, S):

    image_binarisee = image.binarisation(S)     # On binarise l'Image avec le seuil en paramètre
    image_localisee = image_binarisee.localisation()    # On localise l'Image binarisée précedemment obtenue

    # Initialisation des valeurs
    sim = 0
    nb_model = None

    # On parcourt la liste des modeles
    for i in range(len(liste_modeles)):

        # Si la hauteur ou la largeur de l'Image n'est pas égale à celle du modele liste_modeles[i]
        if (image.H != liste_modeles[i].H) | (image.W != liste_modeles[i].W):
                # On redimensionne l'Image localisée
                image_localisee = image_localisee.resize(liste_modeles[i].H,liste_modeles[i].W)

        # On récupère le rapport de similitude entre l'Image et le modele
        c = image_localisee.similitude(liste_modeles[i])

        # On regarde si le nouveau rapport est plus élevé que le précédent
        if sim < c:
            # sim devient le nouveau rapport
            sim = c
            # On sauvegarde l'indice de ce modele
            nb_model = i

    # On affiche le modele reconnu
    liste_modeles[nb_model].display("Modele Reconnu")

    # On retourne l'indice du modele
    return nb_model
```

NICOLAS Thomas
GUERIOT Benjamin

Nous avons lancé le main pour observer le résultat :

```
lecture image : ../assets/test1.JPG (54x43)
lecture image : ../assets/_0.png (32x22)
lecture image : ../assets/_1.png (32x18)
lecture image : ../assets/_2.png (32x20)
lecture image : ../assets/_3.png (32x20)
lecture image : ../assets/_4.png (32x24)
lecture image : ../assets/_5.png (32x20)
lecture image : ../assets/_6.png (32x21)
lecture image : ../assets/_7.png (32x21)
lecture image : ../assets/_8.png (32x22)
lecture image : ../assets/_9.png (32x22)
Le chiffre reconnu est :  4
```

Une fois le programme totalement terminé, nous avons exécuté les deux fichiers tests et il n'y avait aucunes erreurs.

test_Image.py :

```
----------------------------------------------------------
----------------------
Ran 22 tests in 0.152s

OK
```

test_reconnaissance.py :

```
----------------------------------------------------------
----------------------
Ran 3 tests in 0.429s

OK
```

NICOLAS Thomas
GUERIOT Benjamin

6)

Nous avons fait un tableau excel avec différentes valeurs pour observer les chiffres reconnus en fonction du seuil.

	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
25	4	1	2	2	2	4	7	1	4	7
50	4	1	2	2	2	4	7	1	3	0
75	4	1	2	2	2	4	7	1	3	0
100	4	1	2	2	2	4	7	1	3	0
125	4	1	2	2	2	4	9	1	3	6
150	4	1	2	2	2	4	1	1	3	6
175	4	1	2	2	2	4	5	1	3	8
200	4	1	2	2	2	4	7	1	3	8
225	4	1	2	2	2	4	7	1	4	8
250	4	1	4	8	8	4	7	1	9	8
Valeur réelle	4	1	2	2	2	4	5	1-3-5-2	1-8-4-5-6	6

Nous avons remarqué que les seuils les plus appropriés sont ceux entre 125 et 175. Ce qui semble logique car ce sont des valeur proche de la médiane de 256.

Conclusion

Ce TP nous a permis de comprendre comment utiliser numpy ainsi que renforcer nos connaissances dans le domaine de l'orienté objet. Par ailleurs, nous avons complété entièrement le tp qui était très intéressant. Cependant nous avons eu une difficulté lors de la fonction similitude. En effet, nous avions essayé de resize directement dans cette fonction. Malheureusement, nous nous sommes rendu compte qu'il était impossible d'avoir une valeur égale à 1 quand nous importons les 2 même images. Ceci est causé par la fonction resize de numpy qui modifie certaines valeurs de pixels. Nous remarquions que 9 valeurs de pixels avaient changé sur la totalité. Il fallait pour cela ajouter une condition qui vérifie les dimensions des deux images et dans ce cas là, ne pas redimensionner. Mis à part cette erreur, nous n'avons rencontré aucune autre difficulté majeure.