MALACARNE Etienne
DASSEUX Damien
19/11/2021

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

I. Introduction

Dans ce TP, nous devrons reconnaître des caractères dans une image. Pour cela, nous allons devoir créer plusieurs fonctions et effectuer des tests pour voir quelle est la meilleure valeur de seuil .

II. Travail préparatoire

1. Question (1).

Aucune remarque sur cette question.

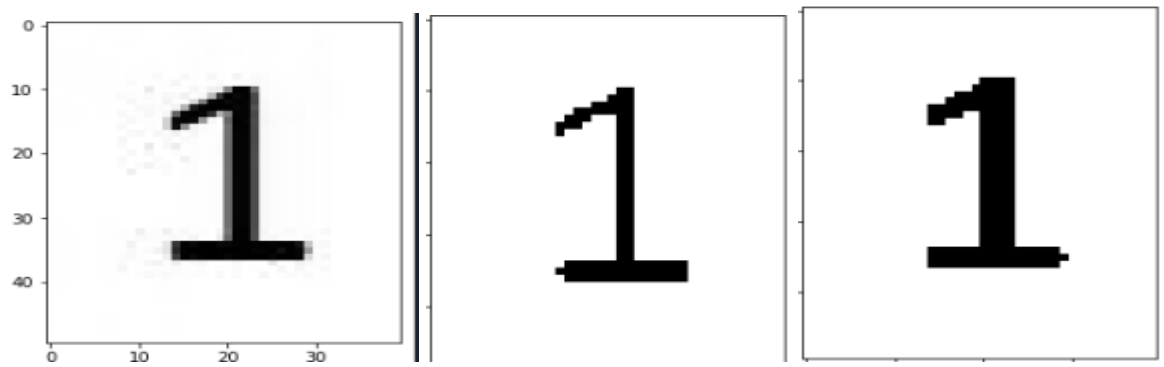
2. Question (2).

On veut une image binaire, c'est à dire une image qui présente en valeur de pixel soit noir (0) soit blanc (255).
Le passage d'une valeur à l'autre est fait à partir d'un seuil.

Une simple double for a été réalisé, il y a sûrement une méthode plus simple avec numpy, mais cette écriture permet une meilleur lecture du code :

```
def binarisation(self, S):
    img_bin = Image()
    img_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))

    for y in range(len(self.pixels)):
        for x in range(len(self.pixels[y])):
            if self.pixels[y][x] >= S: # Est ce que le seuil est inclut ou exclu ?
                img_bin.pixels[y][x] = 255

    return img_bin
```

Voici un exemple de ce que l'on peut obtenir avec l'image de base, une binarisation avec un seuil de 40 et une avec un seuil de 180 :



3. Question (3)

Ici on veut restreindre la zone d'étude. Pour cela il faut définir les limites de l'information sur notre image. Les limites en question sont les pixels des chiffres étudiées, on veut avoir l'image la plus petite en gardant la forme des chiffres, on cherche donc les valeur x_min, x_max, y_min et y_max.

Nous avons commencé avec une solution maison avec des double for.

Nous nous sommes vite rendu compte de la difficulté de cette méthode et nous avons fait des recherches sur numpy.

Nous avons trouvé la méthode numpy.where qui retourne les coordonnées en y et en x de toutes les cases du tableau qui répondait à une condition.

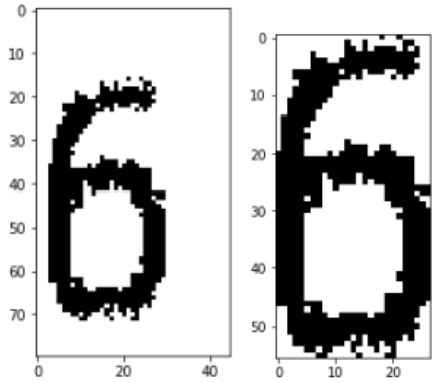
Ensuite, nous avons pris la zone entre x_min, x_max, y_min et y_max pour en faire une nouvelle image.

Au début, une erreur s'était dissimulé dans le code :
self.pixels[y_min:y_max][x_min:x_max]

Il fallait faire :
self.pixels[y_min:y_max,x_min:x_max]

```
def localisation(self):

    x_min = min(np.where(self.pixels==0)[1])
    x_max = max(np.where(self.pixels==0)[1])

    y_min = min(np.where(self.pixels==0)[0])
    y_max = max(np.where(self.pixels==0)[0])



    new_img = Image()
    new_img.set_pixels(np.zeros((y_max-y_min+1, x_max-x_min+1)))
    new_img.pixels=np.array(self.pixels[y_min:(y_max+1),x_min:(x_max+1)])
    return new_img
```

Voici un exemple d'une image avant et après localisation :



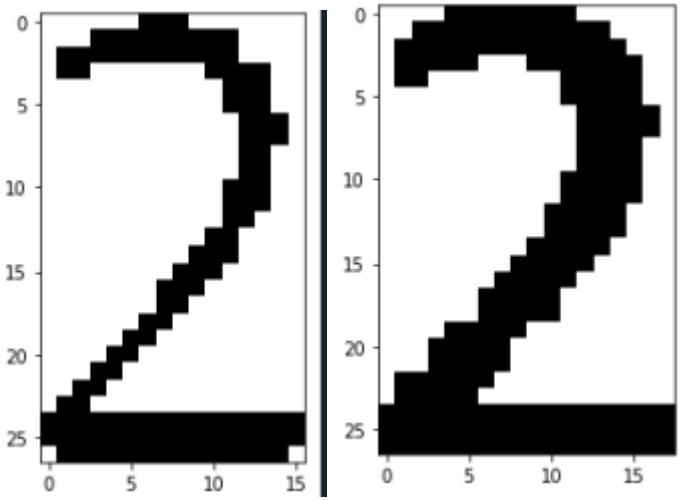
III. Reconnaissance automatique de chiffres

1. Question 1

Les seules erreurs qui sont apparues sont celles sur le resize et la similitude, car ce n'est pas encore fini, mais le reste fonctionne parfaitement.

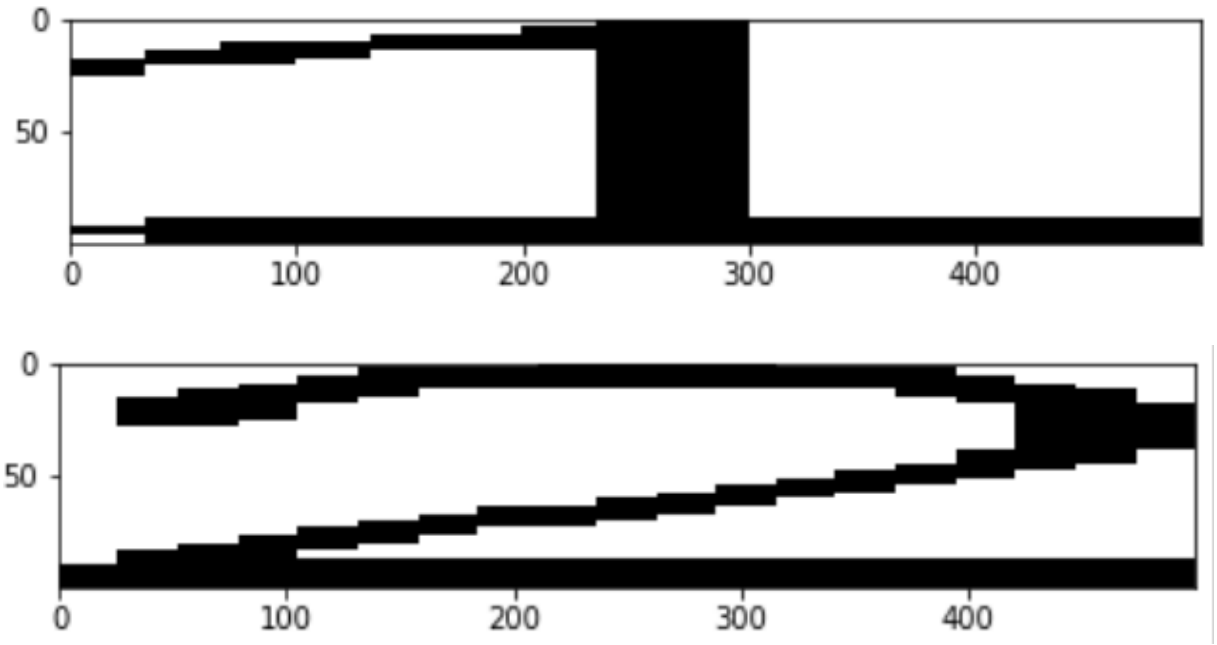
2. Question 2

Plus on augmente le seuil, plus la tolérance est haute pour les pixels noir de l'image (on a plus de pixels noir) et plus le seuil est faible, moins on a de pixels noir, cela peut être considéré comme la "précision" de l'acquisition. Il n'y a pas d'erreur.
Voici un exemple pour deux images, une avec un seuil de 20 et l'autre de 220 :



3. Question 3

Le resize n'a pas posé de problème, au début, nous n'avions pas compris pourquoi les images étaient sur 500 pixels, mais ensuite on a regardé le main et on a compris :
Le but était justement d'obtenir une image très large et assez petite en hauteur



4. Question 4

Encore une fois, une solution numpy a été adoptée.
Le but de la question est de calculer la différence entre notre image et un modèle , pour savoir si elle lui ressemble.

On fait la soustraction entre les 2 images. Cela permet de donner à tous les pixels pareils d'une image à l'autre la valeur 0.

Ensuite, numy propose une méthode "count_non_zero(nparray)", dans notre cas, cela permet de compter tous les pixels différents d'une image à l'autre.
Enfin, il suffit de retourner cette expression :
(nb_pixels-nb_non_zero)/nb_pixels

```
def similitude(self, im):
    img_sub = self.pixels - im.pixels
    nb_non_zero = np.count_nonzero(img_sub)
    nb_pixels = self.pixels.size

    return (nb_pixels-nb_non_zero)/nb_pixels
```

5. Question 5

Dans cette question, il faut faire la reconnaissance du chiffre en utilisant les fonctions coder avant et pouvoir donc dire à quoi le chiffre de notre image ressemble le plus.
La suite d'action à effectuer est la suivante :

- Mettre l'image original en binarisé
- On rogne cette image afin que les chiffre rentre tout pile dans l'image
- On prend une image dans les modèles et on la compare avec notre image rogné
- On garde cette valeur en tête et on répète la dernière étape pour toutes les images modèles.
- Quand c'est fini, on cherche quelle image modèles était la plus proche de notre image rognée.

Ccl : la valeur de l'image modèle est sûrement celle présente dans notre image originale.

```
def reconnaissance_chiffre(image, liste_modeles, S):
    image_original=image.binarisation(S)
    image_original_localisation=image_original.localisation()


    simili=[]
    for x in range(len(liste_modeles)):
        im=image_original_localisation.resize(liste_modeles[x].H,liste_modeles[x].W)

        simili.append(im.similitude(liste_modeles[x]))

        simili_m=max(simili)
        simili_index=simili.index(simili_m)


    return simili_index
```

6. Question 6

On remarque que la photo “test7.JPG” est très mal reconnue avec le seul seuil qui fonctionne , qui est 145. Cette difficulté de reconnaissance est liée à la police d’écriture étant très fine.
Évidemment, cet algorithme ne fonctionne pas lorsqu’il y a plusieurs chiffres dans l’image comme dans l’image “test8” ou “test9”.
Il a aussi du mal lorsque l’image ou est le chiffre est trop nuancé.

Voici nos résultats sous forme de tableau :

		Images :									
		test1.JPG	test2.JPG	test3.JPG	test4.JPG	test5.JPG	test6.JPG	test7.JPG	test8.JPG	test9.JPG	test10.JPG
		Vraie valeurs									
		4	1	2	2	2	4	5	1352	18456	6
		Chiffre reconnu :									
Seuils	20	4	1	2	2	2	4	7	2	3	5
	45	4	1	2	2	2	4	7	2	3	6
	70	4	1	2	2	2	4	9	2	3	6
	95	4	1	2	2	2	4	7	2	3	6
	120	4	1	2	2	2	4	9	2	3	6
	145	4	1	2	2	2	4	5	2	3	6
	170	4	1	2	2	2	4	2	2	3	6
	195	4	1	2	2	2	4	7	2	3	8
	220	4	1	2	2	2	4	2	2	3	8

Comme dit plus haut, la meilleur valeur de seuil est 145 car sur 10 images test, on va pouvoir reconnaître le chiffre présent dans 9 images (enfin, un des 4 chiffres dans l’image test8.JPG).

IV. Conclusion

Nous avons fini le TP. Excepté une erreur d’utilisation de numpy qui nous à un peu ralentis, il n’y a pas eu de difficulté particulière car nous avions déjà fait du traitement d’image avant (même si c’était il y a 8 mois). De ce fait, nous n’avons pas appris beaucoup de choses même si cela reste très intéressant et à permit de nous refaire travailler sur ce sujet que nous avions un peu oublié, notamment l’utilisation de numpy.