HIPPOLITE Axel
WALLERAND Alex
Date du TP (19/11/2021)

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

I. Introduction

Dans ce TP, nous allons écrire un programme permettant de lire un chiffre à partir d'une image.
Nous considérerons seulement des images en niveaux de gris, représenté donc par un tableau de pixels à 2 dimensions.
La technique considérée dans ce TP est la corrélation à partir d'une image modèle. Pour cela, nous devrons dans un premier temps binariser l'image dont on veut lire le chiffre. Ensuite, il faut la recadrer pour ne contenir que le chiffre, ce qu'on appelle la localisation. Puis, il faudra adapter le format de l'image étudiée à celle du modèle. Enfin, nous pourrons calculer la similitude entre l'image et les modèles afin de pouvoir exprimer un résultat.

II. Travail préparatoire

1. Question (1).

On observe la classe Image et on remarque les valeurs de H et W.

2. Question (2).

Pour parcourir l'ensemble des pixels, il faut faire deux boucles for : la 1ère parcourt les valeurs de 0 à la taille H de la hauteur de l'image, la 2nd parcourt les valeurs de 0 à la taille W de la longueur de l'image. Puis on doit vérifier pour chaque itération si la valeur est supérieure à S alors le pixel dans l'image binaire sera blanc (255), sinon il sera noir (0).

```
def binarisation(self, S):
    binaryImage = Image()
    binaryImage.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:
                binaryImage.pixels[i][j] = 255
            else:
                binaryImage.pixels[i][j] = 0
    return binaryImage
```

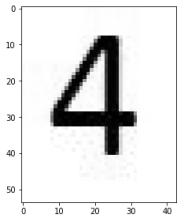
3. Question (3).

Pour cette fonction, on créée d'abord une nouvelle image grâce à la classe Image. On définit les 4 coins du nouvelle encadrement (hmin, hmax, wmin, wmax) à 0. On va ensuite créer (grâce à la fonction np.where()) 2 matrice (de type « Array ») contenant pour la première toutes les colonnes possédant un pixel noir et pour l'autre, toutes les lignes possédant un pixel noir. Pour définir nos limites, on va prendre le maximum et le minimum de chaque matrice : on aura donc la première et la dernière colonne contenant un pixel noir ainsi que la première et la dernière ligne contenant un pixel noir. On redéfinit ainsi les hmin, hmax, wmin, wmax avec ces numéros de colonnes/lignes. On définit ensuite les pixels de notre nouvelle image avec les pixels compris entre ces 4 points. L'image obtenue est alors correctement cadré.

```
def localisation(self):
    locImage = Image()
    wmin = min(np.where(self.pixels==0)[1])
    wmax = max(np.where(self.pixels==0)[1])
    hmin = min(np.where(self.pixels==0)[0])
    hmax = max(np.where(self.pixels==0)[0])
    locImage.set_pixels(self.pixels[hmin:(hmax+1),wmin:(wmax+1)])
    return locImage
```

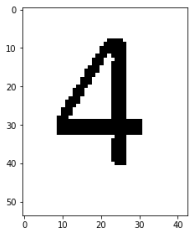
III. Reconnaissance automatique de chiffre

Les images d'origine à identifier ressemble à ça avant traitement :



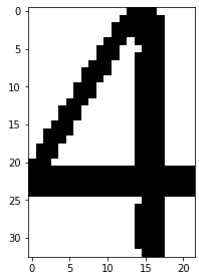
1. Question (1).

On observe que la binarisation est bien effectuée, l'image résultante ne contient que des pixels noirs sur un fond blanc. Nous n'avons pas eu de problèmes de tests unitaires.



2. Question (2).

On observe que la localisation à partir de l'image binarisée est correcte, l'image est correctement recadrée autour du chiffre. Nous n'avons pas eu de problèmes de tests unitaires.



3. Question (3).

Pour cette fonction, on crée d'abord une nouvelle image grâce à la classe Image. On utilise la fonction *resize* de la librairie *skimage* afin d'obtenir un nouvel ensemble de pixel correspond aux nouvelles dimensions passées en paramètres de la fonction. Cet ensemble nous permet ensuite de définir les pixels de notre nouvelle image.

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

4. Question (4).

Pour cette fonction, on crée d'abord une nouvelle image grâce à la classe Image. On initialise une variables *same* à 0, *same* étant le nombre de pixel commun (de même intensité et de même position) aux deux images. On imbrique 2 boucles for pour parcourir tous les pixels en vérifiant si les deux pixels ont la même intensité. Si c'est le cas, on incrémente le compteur *same*.

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

5. Question (5).

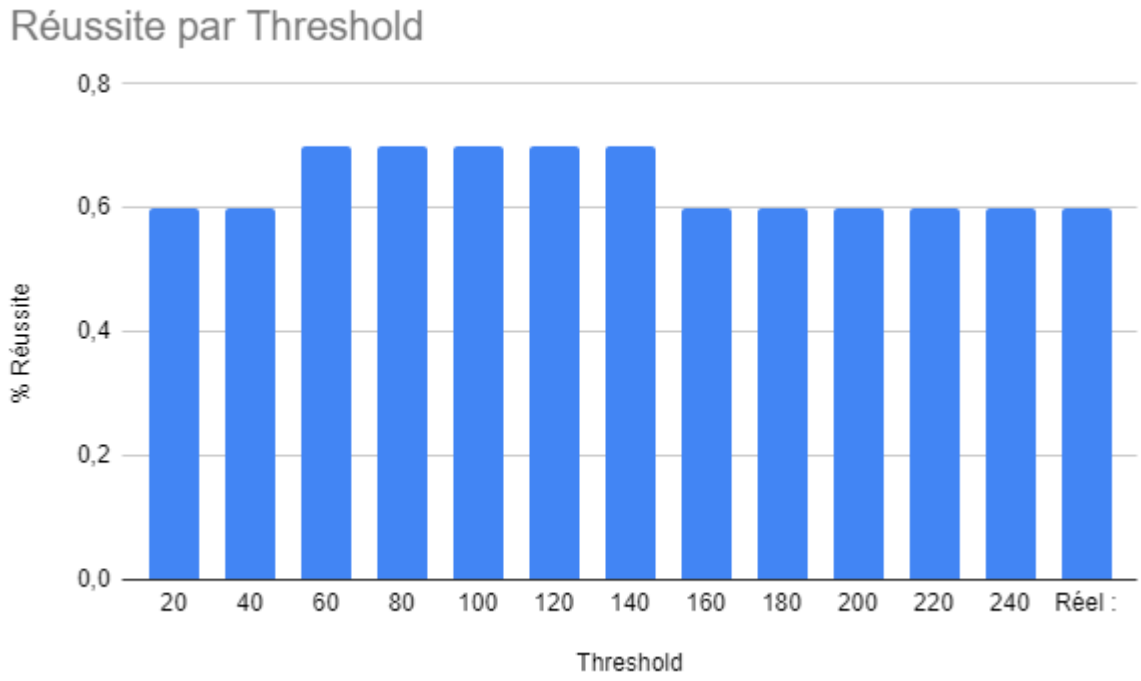
On commence par créer un vecteur qui contiendra l'ensemble des rapports de similitude entre l'image à reconnaitre et les différents modèles. On binarise et on localise ensuite le chiffre de l'image à reconnaitre. En parcourant l'ensemble des images modèles, on va mettre les deux images de la même taille avant de calculer le rapport de similitude et de l'ajouter à notre vecteur. Grâce à la fonction np.where() on obtient l'indice du plus grand rapport qui correspond également au chiffre reconnu.

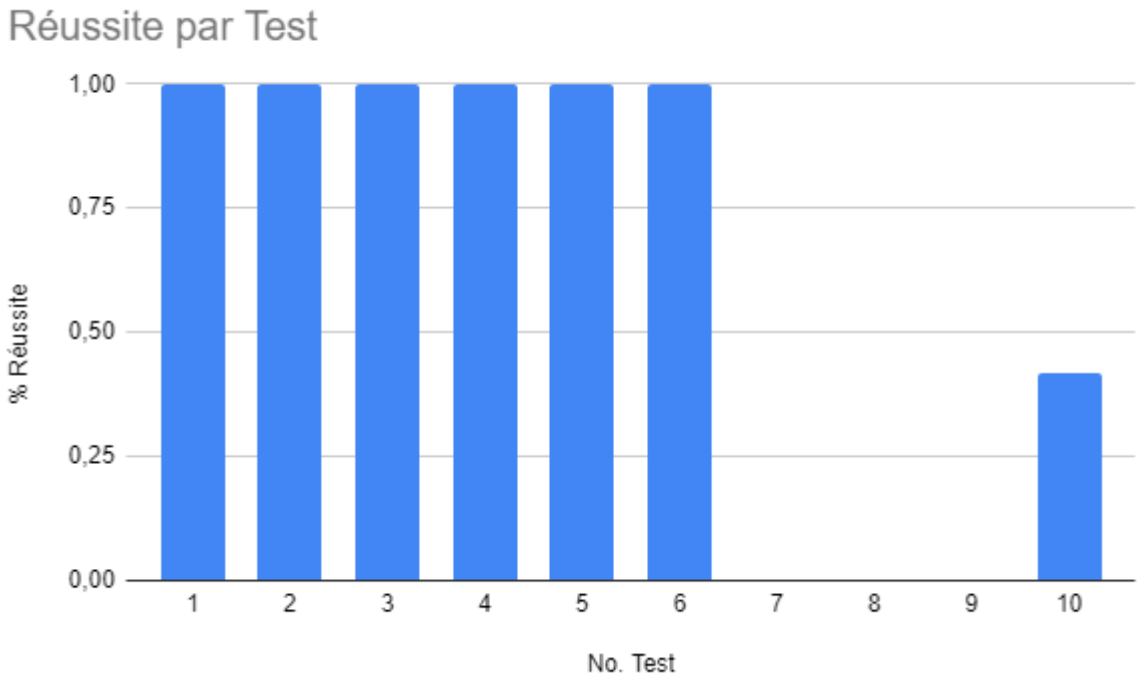
```
def reconnaissance_chiffre(image, liste_modeles, S):
    vec = []
    liste_modeles = lecture_modeles('../assets/')
    image_binarisee = image.binarisation(S)
    image_localisee = image_binarisee.localisation()
    for i in range(len(liste_modeles)):
        image_resize = image_localisee.resize(liste_modeles[i].H, liste_modeles[i].W)
        score = image_resize.similitude(liste_modeles[i])
        vec.append(score)
    result = np.where(vec == np.amax(vec))
    return int(result[0][0])
```

6. Question (6).

Pour cette question, on va tester pour chaque « Image Test » 11 valeurs de seuil (« Threshold ») différentes et noter dans un tableau le résultat de la reconnaissance de l'« Image Test ». Pour meilleur interprétation, on tracera ensuite différents graphiques.

Test :	1	2	3	4	5	6	7	8	9	10	% Réussite :
Threshold :											
20	4	1	2	2	2	4	7	7	7	7	0,6
40	4	1	2	2	2	4	7	7	7	4	0,6
60	4	1	2	2	2	4	7	7	7	6	0,7
80	4	1	2	2	2	4	7	7	7	6	0,7
100	4	1	2	2	2	4	7	7	7	6	0,7
120	4	1	2	2	2	4	7	7	7	6	0,7
140	4	1	2	2	2	4	7	7	7	6	0,7
160	4	1	2	2	2	4	7	7	7	4	0,6
180	4	1	2	2	2	4	7	7	7	8	0,6
200	4	1	2	2	2	4	7	7	7	8	0,6
220	4	1	2	2	2	4	7	7	7	8	0,6
240	4	1	2	2	2	4	7	7	7	8	0,6
Réel :	4	1	2	2	2	4	5	1 - 3 - 5	1 - 8 - 4 - 5 - 6	6	0,6
% Réussite :	1	1	1	1	1	1	0	0	0	0,4166666667	





IV. Conclusion

En fin compte, nous avons avancé relativement vite. Les connaissances nécessaires étaient assez basiques mais cela nous a permis de mettre en place un OCR sans passer par un Réseau de Neurones Convolutifs (comme l'un de nous l'avez déjà fait). Le TP resta très intéressant même si les images de tests contenant plusieurs semblait inapproprié pour un programme ne reconnaissant qu'un seul chiffre à la fois (si ce n'est pour vérifier le non-fonctionnement pour certains cas).