ORSET Emma
ERUAM Hervé
30/11/2021

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

I. Introduction

Dans ce TP, nous allons travailler sur le langage de programmation Python. Nous allons apprendre à lire automatiquement des chiffres par rapport à une image, nous suivrons la méthode de la reconnaissance par corrélation avec des modèles. 5 étapes permettent de comparer deux images.

II. Travail préparatoire

1. Analyse de la classe *Image*

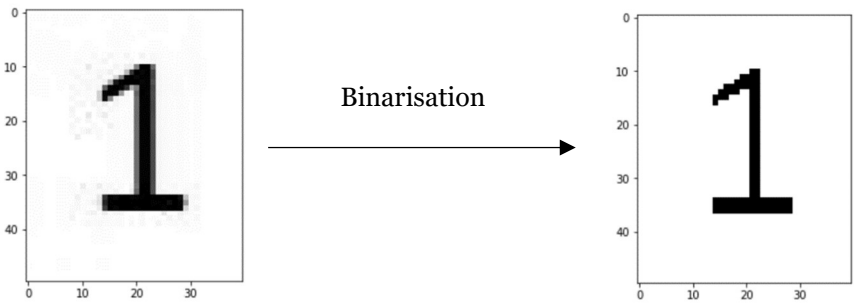
Nous remarquons que **H** indique la hauteur de l'image (*height*) tandis que **W** indique la largeur de l'image (*width*). **Pixels** est un tableau en deux dimensions : une dimension pour **H** et une dimension pour **W**.

2. Etape 1 : Binarisation

Nous avons écrit une méthode ***binarisation(self, S)*** afin de créer une nouvelle image à partir de l'image initiale. La nouvelle image est nette car on ne prend que les pixels qui sont noirs ou blancs (donc pas les gris).

Nous nous sommes concentrés sur le fait de ne pas inverser les indices, mais nous n'avons pas eu de problème à faire cette fonction car la partie la plus difficile était donnée dans le TP.

Ainsi, à cette étape, nous obtenons l'image suivant :

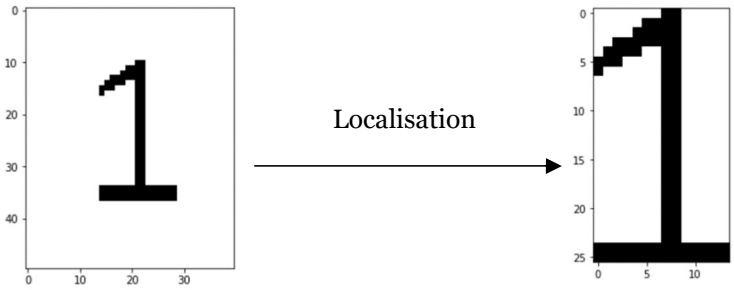


3. Etape 2 : Localisation

Nous avons écrit une méthode localisation(self) qui restreint l'image. Nous renvoyons une nouvelle image qui contient seulement le chiffre (on enlève le surplus de pixels blancs).

Nous avons définis **cmin** qui prend au départ le plus grand pixel, puis nous parcourons les pixels jusqu'à trouver un pixel de couleur. A chaque fois que c'est le cas, nous regardons si la position est inférieure à celle de **cmin**, si c'est le cas, on remplace la valeur **cmin** par la position de ce pixel. Nous avons utilisé le même principe pour **lmin**, **cmax**, **lmax**.

Ainsi, à cette étape, nous obtenons l'image suivante :



III. Reconnaissance automatique de chiffre

1. Test

Nous avons testé notre méthode binarisation, le test a fonctionné.

2. Test

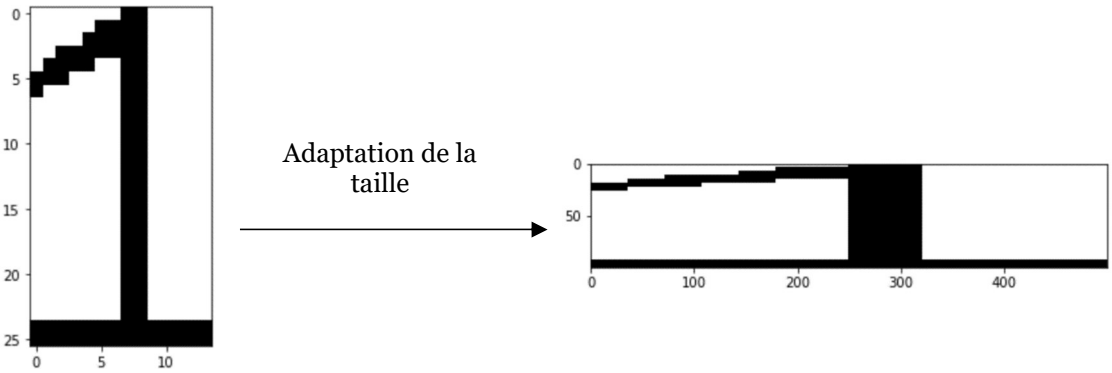
Nous avons testé notre méthode localisation, le test a fonctionné.

3. Etape 3 : Adaptation de la taille au modèle

Nous avons écrit une méthode ***resize(self,new_H,new_W)*** qui redimensionne l'image.

Les fonctions qui permettent de faire cette étape sont données dans le sujet (dans les librairies *skimage* et *numpy*). Nous les avons simplement utilisées après avoir créé une Image.

A cette étape, nous obtenons l'image suivante :



Nous avons lancé le test unitaire et nous n'avons pas eu de problème sur les tests.

4. Etape 4 : Mesure de ressemblance par corrélation

Nous avons écrit une méthode ***similitude(self, image)*** qui renvoie un rapport de similitude entre les deux images, compris entre 0 et 1.

Nous parcourons l'image de long en large, et si le pixel de l'image est égal au pixel de l'image passée en paramètre, on rajoute « un pixel similaire de plus ». Cela nous donnera donc la similitude entre les deux images.

5. Etape 5 : Décision

Nous avons écrit une fonction ***reconnaissance_chiffre(image, liste_modeles, S)*** qui va permettre de reconnaitre le chiffre de notre image à partir d'une base de données d'autres images.

Nous utilisons nos deux fonctions (*binarisation* et *localisation*). Ensuite nous parcourons une liste d'images pour comparer leur similitude avec notre image. Il ne faut pas oublier de créer une image « temporaire » pour ne pas modifier l'image initiale lors de l'utilisation de la méthode *resize*. Ensuite nous regardons la similitude entre notre image et une image de la liste. Nous sauvegardons l'image qui est la plus similaire à notre image, ainsi que son indice. Nous renvoyons ce dernier.

Lors du lancement du test unitaire, nous avions un problème avec cette fonction. En effet nous reprenions l'image passée en paramètre lors du test de similitude entre les images, alors qu'il fallait prendre l'image modifiée par nos fonctions. Nous avons assez vite trouvé le problème.

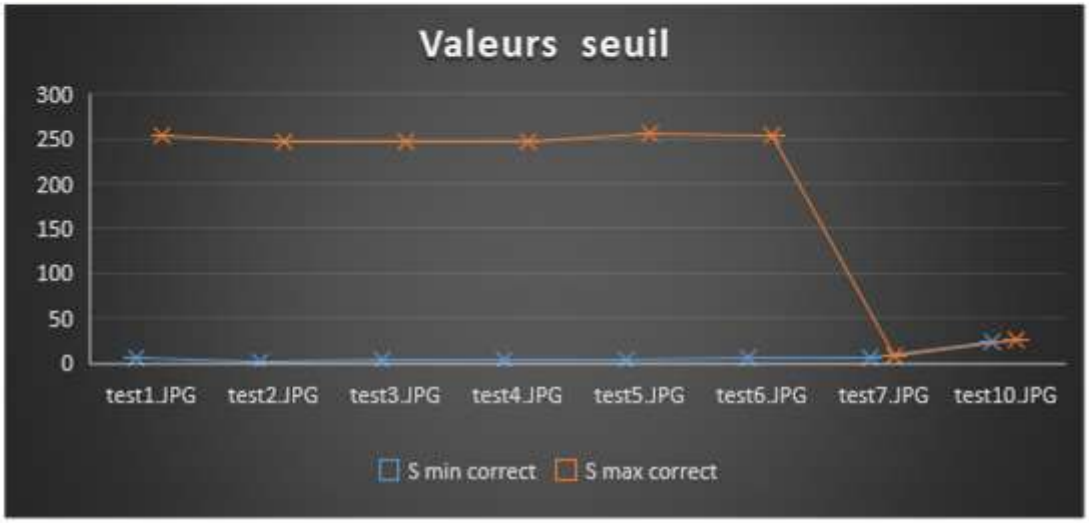
6. Tests de masse

Afin de tester notre fonction *reconnaissance* à plus grande échelle, nous avons effectué plusieurs tests. Les tests 8 et 9 présents dans le dossier assets n'étaient pas conformes au TP, nous les avons donc enlevés. De plus, nous avons modifié la fonction localisation afin de gérer les images toute blanche et toute noire.

Nous avons réduit nos tests afin d'afficher la valeur de **S** seulement quand l'image est reconnue puis non reconnu afin d'avoir un seuil de valeur pour chaque image.

Voici un tableau de nos résultats :

Nom img	attendu	S min correct	S max correct
test1.JPG	**4**	5	253
test2.JPG	**1**	2	248
test3.JPG	**2**	3	246
test4.JPG	**2**	4	246
test5.JPG	**2**	4	255
test6.JPG	**4**	5	254
test7.JPG	**5**	5	7
test10.JPG	**6**	23	25



D'après nos tests, le seuil dépend de la qualité de l'image car nous remarquons que les premières images, qui étaient de bonne qualité, ont un seuil minimum très bas et un seuil maximum très élevé. Pour une image moins nette comme la 7 et la 10, on ne peut pas définir un seuil car la reconnaissance est assez aléatoire. Nous avons laissez nos tests dans notre code pour vérifier cela.

Nous avions un peu de temps donc nous avons testé avec une image d'un 8 écrit au feutre que nous avons pris en photo. Nous l'avons convertie en niveau de gris et au seuil 24, notre code reconnait bien un 8 !

IV. Conclusion

Ce TP était très guidé et très clair donc nous n'avons pas eu de mal à le faire. De plus, nous avions fait le travail préparatoire avant la séance donc nous avons pu finir rapidement le TP et aider nos camarades. Nous avons aussi pris du temps pour la dernière question afin d'automatiser nos tests.