

## Rapport de TP 2

### ■ Introduction

Le TP suivant consiste à binariser une image pour ensuite la localiser et déterminer avec plusieurs exemples d'images le numéro qui correspond à cette image.

### ■ Travail préparatoire (3)

#### Question 2:

Pour la méthode de binarisation, j'ai tout d'abord utilisé une méthode plus algorithmique. Je parcourais à l'aide de deux boucles chaque pixel de l'image, et je modifiais sa valeur selon si il était inférieur ou supérieur/égal au seuil. Respectivement 0 et 255. Cette méthode fonctionnait et donnait les mêmes résultats que la seconde méthode que j'ai utilisé et gardé. Comme indiqué dans le pdf, j'ai décidé d'utiliser des masques numpy afin de binariser l'image.

J'utilise donc un premier masque qui s'appliquera sur les pixels inférieurs à S à l'aide de la méthode `numpy.putmask()`. Je mets comme valeur à appliquer aux éléments correspondants 0, et je fais la même chose avec les pixels supérieurs ou égaux à S, je crée une image à laquelle je donne le tableau de pixels résultant de l'application des masques et je renvoie l'image.

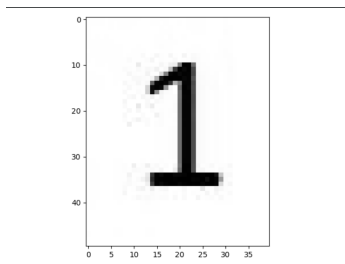


Image avant binarisation

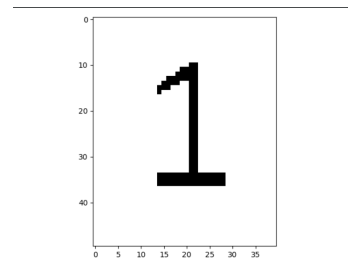


Image après une binarisation avec un seuil de 80

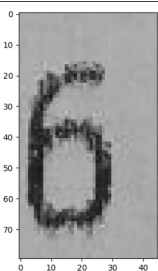


Image avant binarisation

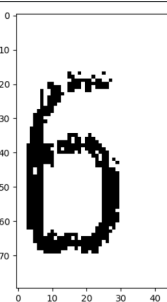


Image après une binarisation avec un seuil de 100

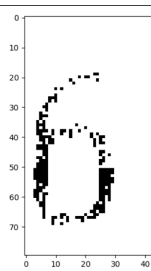


Image après une binarisation avec un seuil de 50

### Question 3 :

Pour la méthode de localisation, j'applique un algorithme permettant de parcourir chaque ligne à la recherche des premiers et derniers pixels noirs de celles-ci. Je récupère ensuite les valeurs  $c_{min}$  et  $c_{max}$  de chaque ligne que je compare aux plus petites et plus grandes déjà rencontrées. J'en profite pour stocker la première ligne  $l_{min}$  et la dernière  $l_{max}$  lorsque respectivement je rencontre le premier pixel noir ou je rencontre un pixel noir sur une ligne (elle devient alors  $l_{max}$  jusqu'à ce qu'on en trouve une autre si il en exista une autre), et je renvoie une nouvelle image avec un tableau que je limite aux valeurs  $pixels[c_{min}:c_{max}+1, l_{min}:l_{max}+1]$

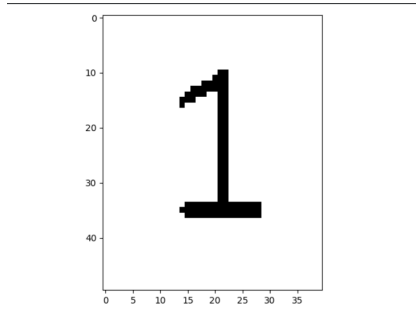


Image binarisée avant localisation

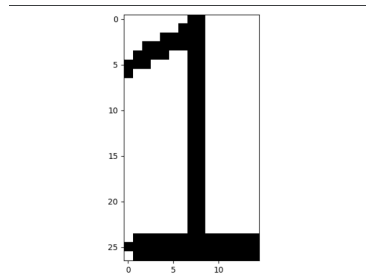


Image binarisée après localisation

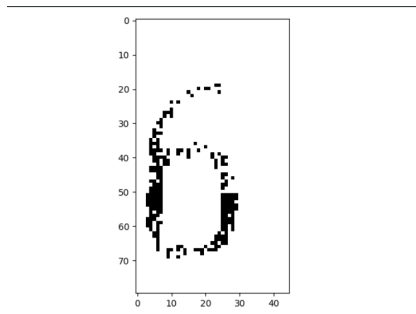


Image binarisée avant localisation

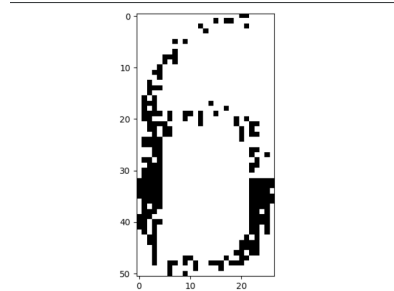


Image binarisée après localisation

## ■ Reconnaissance automatique du chiffre (4)

### Question 1 :

Observable sur la question 2 de la partie précédente

### Question 2 :

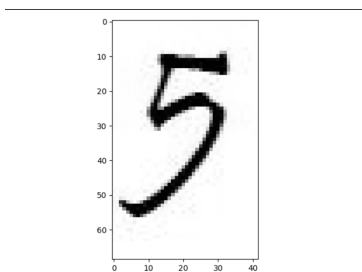


Image avant binarisation

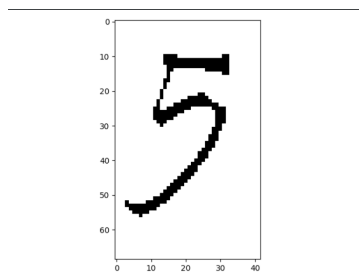


Image après binarisation S=50

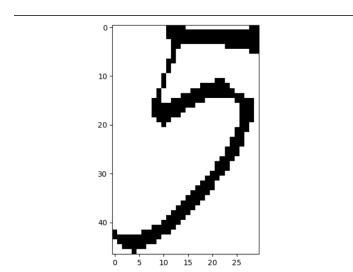
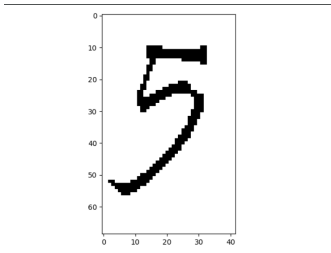
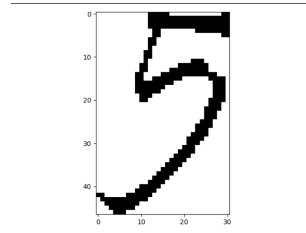


Image localisée



Même image binarisée avec un seuil de 100



Même image binarisée avec un seuil de 100 localisée

### Question 3:

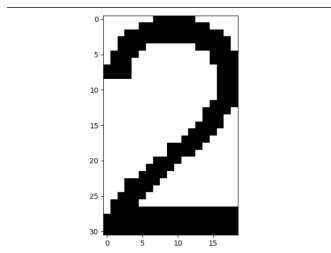


Image après localisation

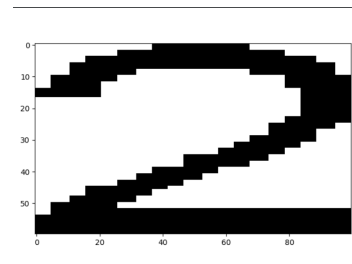


Image localisée après resize

### Question 4:

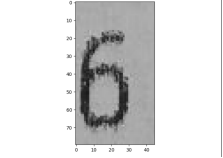
La fonction similitude utilise également les masques numpy. Pour calculer la similitude entre deux images, je calcule d'abord le masque des pixels qui ont la même valeur au même endroit des deux images comparées. Ensuite je calcule le nombre de pixels correspondants et je le divise par Hauteur\*Largeur

### Question 5:

La fonction reconnaissance est simple. Elle binarise l'image à analyser au seuil S donné, la localise, la resize en 40x40. On calcule ensuite pour chaque modèle, auxquels on a préalablement fait passer les mêmes traitements que l'image comparée, et on calcule un par un la similitude. Si elle est supérieure à la similitude maximale observée, alors on sauvegarde sim\_max ainsi que le nombre qui correspond le plus actuellement. Une fois tous les modèles passés, on renvoie le chiffre ayant la correspondance la plus haute

### Question 6:

Image	Seuil 70	Seuil 100	Seuil 150	seuil 200
	2	2	2	2
	4	4	4	4

	0	6	6	8 (image noire après vérification)
---	---	---	---	------------------------------------

On voit donc que lorsque l'on met un seuil trop haut, si l'image n'est pas assez contrastée, on peut se retrouver avec potentiellement des bruits parasites, mais surtout une image noire ou blanche par exemple si le contraste était vraiment trop faible. Cela pourrait se résoudre en augmentant la complexité de la chaîne de travail.

### ■ Conclusion

On voit donc qu'en python, avec l'utilisation de numpy et de scikit image, on peut facilement arriver à un OCR qui donne des résultats prometteurs avec une chaîne de travail relativement simple. L'étape suivante serait d'ajouter d'autres méthodes de reconnaissance, en extrayant de nos images d'autres informations que nous pourrions vectoriser et comparer plus efficacement.