DESAGE Hippolyte

MEYER johan

Date du TP (30/11/2021)

# Rapport de TP2 – Lecture automatique de chiffre par analyse

# d’image.

## Introduction

Le but de ce TP2 est de lire automatiquement des images de chiffre grâce à l’analyse d’image et grâce à des modèles qui nous ont été donné.

## Section 3 du TP

### Question (2).

La fonction binarisation(self,S) est une fonction qui va nous permettre de rendre l’image plus net car elle va mettre totalement en blanc ou totalement en noir les nuances de gris. Tous les pixels qui auront une intensité qui sera inférieur à S (qui est la limite) prendront la valeur 0, c’est-à-dire qu’ils seront mis en noir total. Les pixels qui auront une intensité supérieure à S, seront mis au blanc total avec une intensité de 255. 0 ou 255 sont les seules valeurs que peuvent prendre les pixels.

Le code :

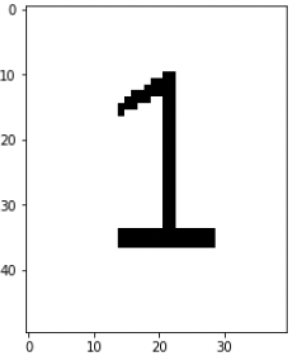
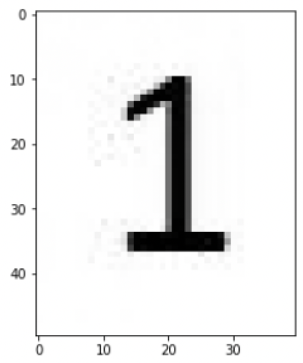
Une image contenant texte

Description générée automatiquement

Le résultat :

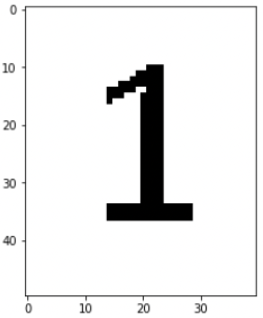
Image obtenue :

Image de départ :



Ici, le traitement de l’image a été réalisé avec un seuil S de 70. Celui-ci peut être modifié pour avoir une limite plus haute ou plus basse. Si le seuil est augmenté, le chiffre que l’on va obtenir sera plus épais car plus de pixels seront mis au noir et inversement si on diminue, le chiffre sera moins épais.

Exemple avec un S de 127 qui est la moitié de 255.



Pour ne pas écraser notre image de départ, le résultat sera donné avec une nouvelle image dédiée à cela.

### Question (3).

La fonction localisation(self) va nous permettre de réduire la taille de l’image à ce qui nous intéresse c’est-à-dire le chiffre. Tous les pixels blancs qui ne seront pas dans les limites fixées seront supprimé. Les limites seront données grâce aux colonnes et aux lignes de pixels, au début de la fonction on définira :

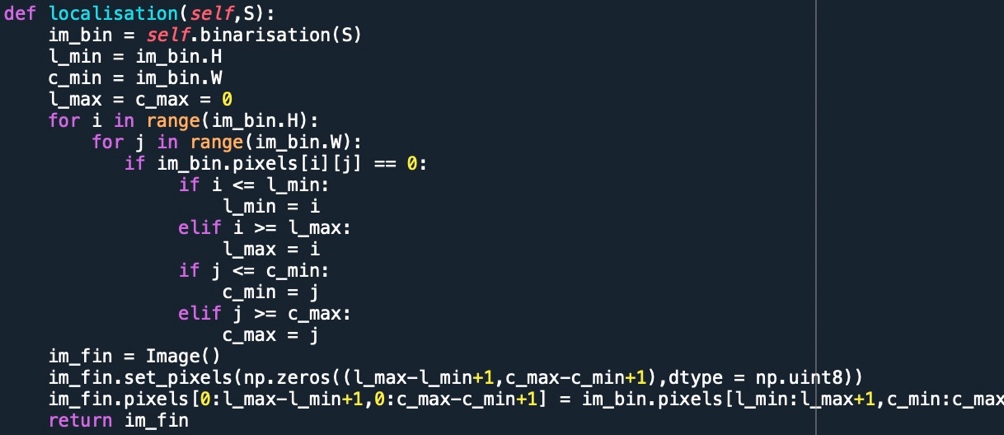
Ligne minimum : ﻿l\_min = im\_bin.H

Colonnes minimum : ﻿c\_min = im\_bin.W

Ligne/Colonnes maximum : l\_max = c\_max = 0

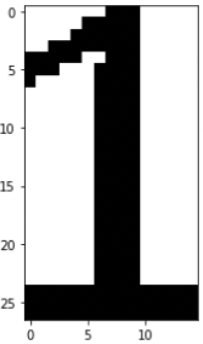
Ensuite en balayant toute l’image grâce à des for, si l’intensité du pixel correspondant à la ligne et à la colonne donnée est de 0 (on rappelle qu’elle sera de 0 ou de 255), les cordonnée du rectangle englobant seront mise à jour.

Le code :



Le résultat :

Avec l’image qui a été créée par la fonction binarisation(), on obtient :

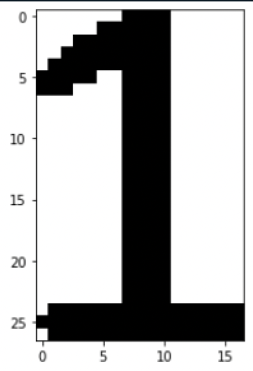
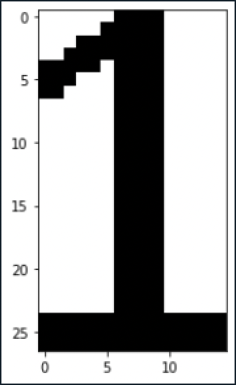
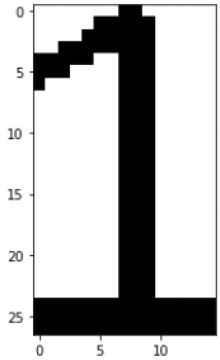
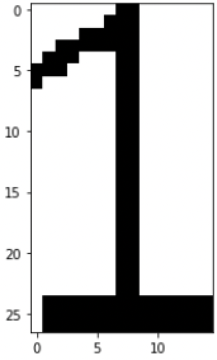


On voit grâce aux graduations que l’image a été rétrécie pour ne garder que l’essentiel.

## Section 2 du TP

### Question (1).

En lançant le test\_Image.py on observe les mêmes résultats que ceux avec notre programme. On peut en effet essayer différents seuils afin de vérifier que notre fonction effectue bien une sorte d’amélioration des contours :



*S = 150*

*S = 230*

*S = 50*

*S = 100*

### Question (2).

Lors des essais de la question 1 nous vont également laisser l’exécution de la méthode localisation. On observe que le résultat est bien celui voulu. En effet pour les valeurs testées le recadrage se fait correctement en fonction des bordures afin que nous n’ayons pas de parties inutiles su l’image.

On aura donc en entrée de la fonction les paramètre de hauteur et de largeur que l’on souhaite donner à l’image. On appliquera donc les nouveau paramètres à l’image.

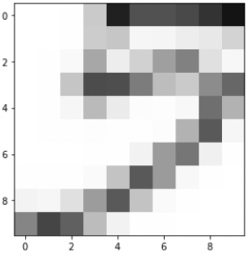
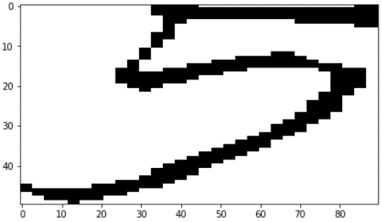
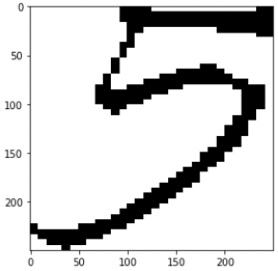
### Question (3).

La fonction resize(self,new\_H,new\_W) nous permet de redimensionner l’image a la taille que l’on souhaite.

Une image contenant texte

Description générée automatiquementLe code :

Comme on recadrera l’image on s’attend à avoir une image de piètre qualité lorsque les dimensions sont petites en effet si on a une image de 10 par 10 on ne pourra pas avoir une image détaillée. On va tester différents cas de figures pour vérifier cela.



*250 par 250*

*10 par 10*

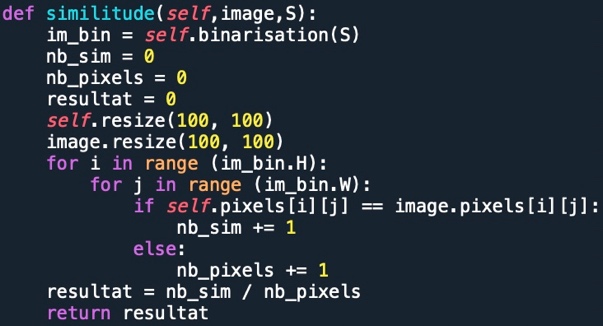
*50 par 90*

On a donc bien le résultat obtenu que l’on vérifie sur le programme teste. En effet l’image est déformée lorsque les dimensions ne sont plus carrées.

### Question (4).

Afin d’avoir la similitude on va déjà recadrer les deux images à la même dimension afin de les comparer pixels par pixels. On va donc avoir deux boucles qui vont nous permettre de sélectionner une ligne pour la première puis une colonne pour la seconde. Ainsi on balayera tous les pixels de l’image. Pour chaque pixel nous allons donc voir s’il s’agit du même que celui de l’image, finalement nous ferons le rapport du nombre de pixels similaires sur le nombre de pixel ce qui nous donnera un nombre représentatif de la ressemblance entre les deux images.

Le code :



### Question (5).

On va maintenant créer la méthode reconnaissance\_chiffre(image, liste\_modeles, S), celle-ci va nous permettre d’utiliser tout ce que nous avons fait jusqu’à maintenant. Nous allons donc redimensionner notre image aux bonnes dimensions pour qu’elle soit comparée avec le modèle. Nous allons donc grâce a la méthode similitude comparer notre image une à une avec chaque modèle afin de voir lequel correspond le plus. Nous serons donc à quelle nombre l’image correspond puisque nous conserverons le passage dans la boucle en corrélations avec le plus grand indice de ressemblance.

Le code :

Une image contenant texte

Description générée automatiquement

Résultat :



### Question (6).

Nous allons donc réaliser des tests pour différents seuils avec des images similaires afin de tirer une conclusion sur quel(s) seuil(s) était le plus adéquat pour notre application.

*Tests pour S = 50*

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro attendu | Image | Résultat donné par le programme | Satisfaction du résultat |
| 2 (test.5) |  | 2 | ✓ |
| 5 (test.7) |  | 7 | ×  Le 5 se rapproche d’un 7 sur l’image |
| 18456 (test9) |  | 3 | ×  Le programme ne permet pas de repérer plusieurs nombres |
| 6 (test10) |  | 0 | ×  Il n’y a pas assez de pixels noirs correspondant au 6 |

*Test pour S = 127*

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro attendu | Image | Résultat donné par le programme | Satisfaction du résultat |
| 2 (test.5) |  | 2 | ✓ |
| 5 (test.7) |  | 9 | ×  Sur cette image, la forme du 5 est très proche de celle d’un 9 |
| 18456 (test9) |  | 3 | ×  Le programme ne permet pas de repérer plusieurs nombres |
| 6 (test10) |  | 6 | ✓ |

*Test pour S = 200*

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro attendu | Image | Résultat donné par le programme | Satisfaction du résultat |
| 2 (test.5) |  | 2 | ✓ |
| 5  (test7) |  | 7 | ×  Sur cette image, la forme du 5 est très proche de celle d’un 7 |
| 18456 (test9) |  | 3 | ×  Le programme ne permet pas de repérer plusieurs nombres |
| 6 (test10) |  | 8 | ×  Il reconnaît un 8 car son image de référence doit être celle qui a le plus de pixels noirs |

On voit donc que pour avoir le meilleur résultat il faut prendre un seuil qui se trouve un peu près au milieu car si on va trop d’un côté ou de l’autre, le logiciel ne permet pas de reconnaître.

Les problèmes de reconnaissance des nombres viennent du fait que le programme compare la position des pixels noir sur l’image que l’on veut reconnaître avec la position des pixels noir sur les images de référence. Par exemple, si on entre une image toute noir dans le programme, il va reconnaître le chiffre correspondant à l’image où il y a le plus de pixels noirs. On pourrait améliorer cela en utilisant plusieurs images avec des formes différentes pour chacun des chiffres ou en comparant la forme globale en plus de la position des pixels.

## Conclusion

Nous avons donc réussi à arriver au bout de ce TP que nous avons tous deux trouvé très intéressant malgré quelques difficultés sur la préparation principalement. En effet les notions apprises nous ont non seulement permis d’acquérir de nouvelles compétences mais également de comprendre un principe que nous utilisons tous les jours. C’est notamment le principe qui se cache derrière une fonctionnalité de Google qui nous indique ce qu’il y a sur une photo donnée. Cependant tout comme ce logiciel, notre programme représente des imperfections, il reconnait mal des chiffres avec une polices distordus ou lorsqu’on a plus d’un chiffre sur une image.