CASTEUR Axel
HERVE Grégoire
Date du TP (10/01/2022)

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

I. Introduction

Les images numériques sont représentées par des tableaux à 2 dimensions, où chaque élément du tableau correspond à un pixel. Chaque pixel a une intensité codée sur 8 bits, ou les valeurs vont de 0 à 255. De manière plus pointue, chaque pixel est défini par trois grandeurs : une intensité Rouge, Vert et Bleu chacune comprises entre 0 et 255.

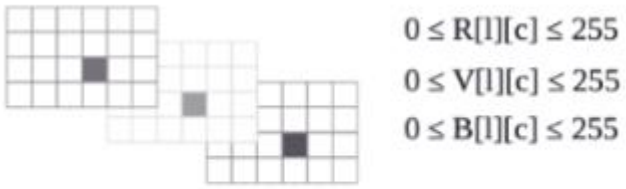
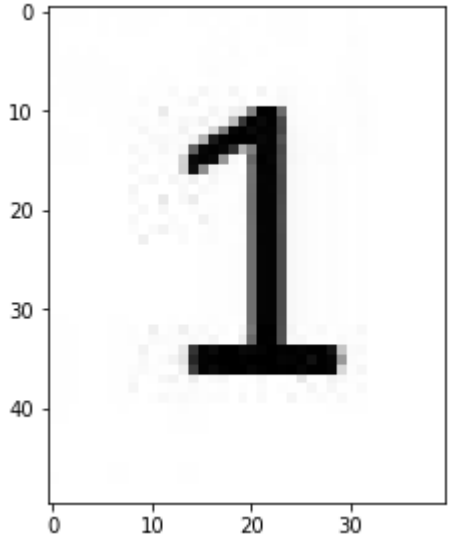


Image RVB couleur

Une fois la première notion comprise, nous allons pouvoir attaquer l'objectif du TP. Celui-ci est d'illustrer, parmi les différentes techniques de lecture automatique de chiffres, une des solutions les plus simples : la reconnaissance par corrélation avec des modèles. Le principe de cette approche est détaillé ci-dessous :

II. Prise en main de l'environnement

Après l'exécution du fichier on obtient l'image suivante :



III. Travail préparatoire

1. Question (1).

Après analyse de la classe image on en déduit que l'attribut H correspond à la hauteur de l'image et l'attribut W correspond à la largeur de l'image. Maintenant concernant l'attribut pixel, il est défini comme un tableau créé

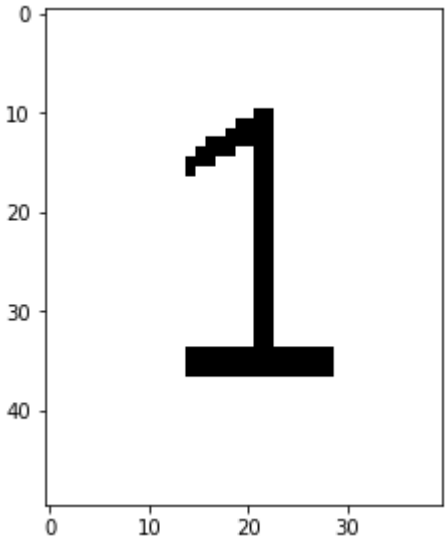
avec tous les pixels constituant l'image. Pour finir, l'attribut Shape de la fonction set pixels permet de connaître la taille de l'image.

2. Question (2)

Pour résoudre la question, nous devons faire une boucle dans une autre qui permettra de naviguer que ce soit en largeur ou en hauteur dans l'image et montrer si la position du pixel [i ; j] est supérieur (dans ce cas il prendra la valeur 255) ou inférieur (il prendra la valeur 0). Voici le code :

```
#=============================================================================
# Methode de binarisation
# 2 parametres :
#   self : l'image a binariser
#   S : le seuil de binarisation
#   on retourne une nouvelle image binarisee
#=============================================================================
def binarisation(self, S):
    im_bin = Image()
    im_bin.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 :
                im_bin.pixels[i,j]=255
            else :
                im_bin.pixels[i,j]=0
    return im_bin
```

On obtient :



Nous sommes passés :

```
FAILED (failures=5, errors=16)
An exception has occurred, use %tb to see the full traceback.
```

```
FAILED (failures=6, errors=8)
An exception has occurred, use %tb to see the full traceback.
```

3. Question (3)

Pour résoudre cette question, nous avons pensé à créer une condition pour tomber sur un pixel de couleur noir soit de valeur 0 sans oublier d'affecter la valeur de l min et c min. De plus on compare les valeurs au fur et à mesure de la boucle pour obtenir les plus petites valeurs.

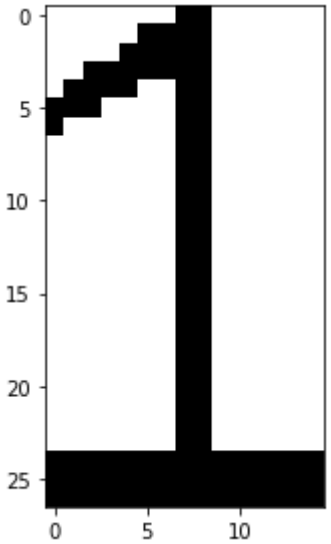
Voici le code :

```
#================================================================================
# Dans une image binaire contenant une forme noire sur un fond blanc
# la methode 'localisation' permet de limiter l'image au rectangle englobant
# la forme noire
# 1 parametre :
#   self : l'image binaire que l'on veut recadrer
#   on retourne une nouvelle image recadree
#================================================================================
def localisation(self):
    l_min=self.H
    l_max=0
    c_min=self.W
    c_max=0

    for i in range(self.H):
      for j in range (self.W):
        if self.pixels[i][j]==0:
            if j<c_min:
                c_min=j
            if j>c_max:
                c_max=j
            if i<l_min:
                l_min=i
            if i>l_max:
                l_max=i
    loc=Image()

    loc.set_pixels(self.pixels[l_min:l_max+1,c_min:c_max+1])
    return loc
```

Nous obtenons :



Nous passons :

```
FAILED (failures=6, errors=8)
An exception has occurred, use %tb to see the full traceback.
```

A :

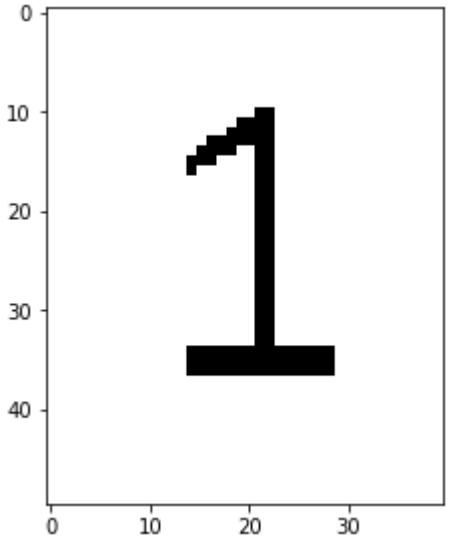
```
FAILED (failures=6, errors=2)
An exception has occurred, use %tb to see the full traceback.
```

IV. Reconnaissance automatique de chiffre

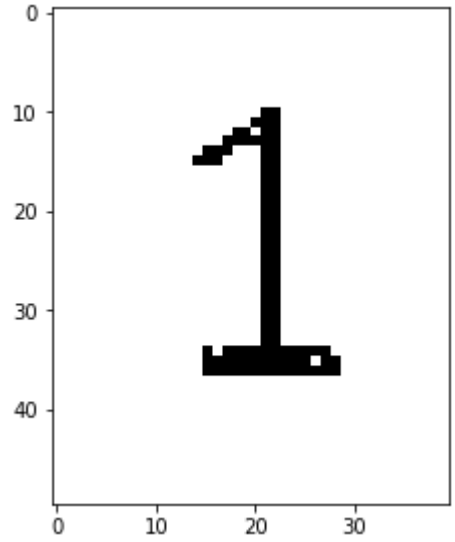
1. Question (1)

Voici nos tests :

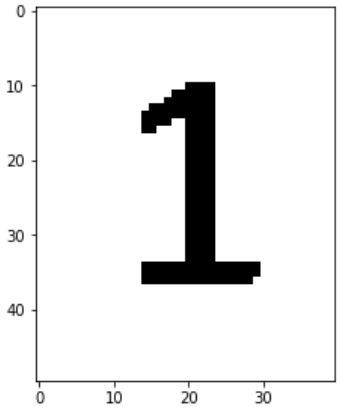
Pour S = 70 :



Pour S = 10 :



Pour S = 200 :



Nous restons au même nombre d'erreur car pas de nouveau code :

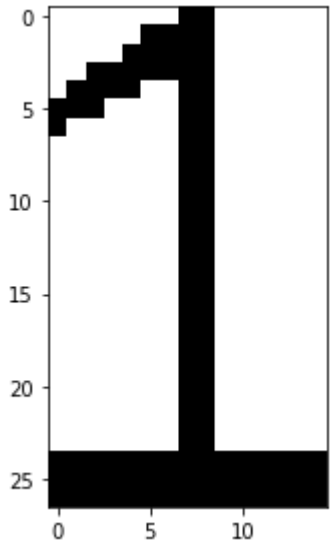
```
FAILED (failures=6, errors=2)
An exception has occurred, use %tb to see the full traceback.
```

Nous remarquons que le 1 est centré et est placé à une certaine distance des axes, au plus le S est grand au plus l'image du chiffre 1 est grande plus il a de pixels, ils sont proportionnels.

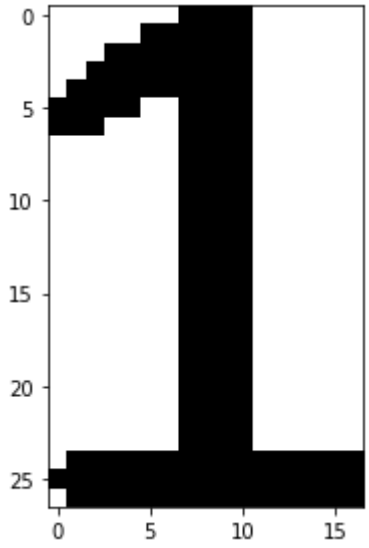
2. Question (2)

Voici nos tests :

Pour S = 70 :



Pour S = 230 :



Nous remarquons que les axes sont en contact avec les pixels, l'image est placée en fonction du pixel tout en évitant des espaces inutiles. Au plus S est grand au plus l'image du chiffre 1 est grande, plus il y a de pixels. S et le chiffre 1 sont proportionnels.

3. Question (3)

On produit une nouvelle image. On récupère le tableau de pixel d'une première image on modifie la taille de ce tableau afin d'obtenir une image redimensionnée. Cela nous retourne la nouvelle image.

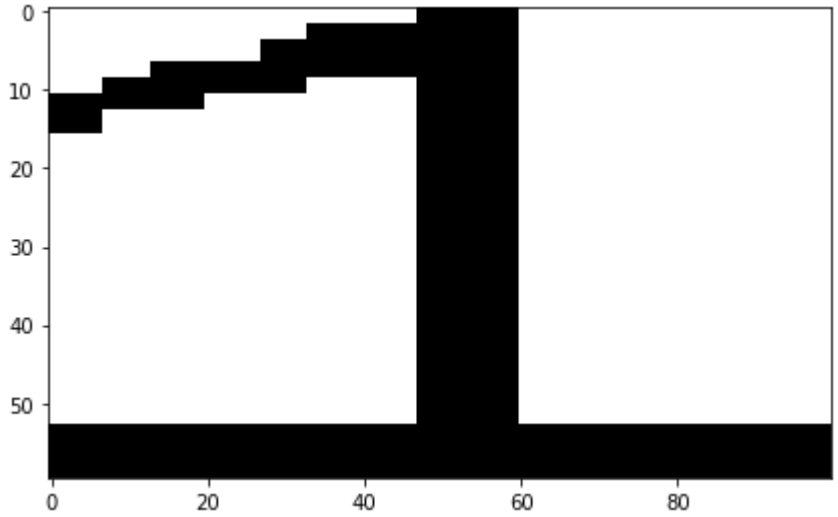
Le code :

```
#=============================================================================
# Methode de redimensionnement d'image
#=============================================================================
def resize(self, new_H, new_W):
    new = Image()
    a = resize(self.pixels, (new_H, new_W), 0)
    new.set_pixels(np.uint8(a*255))
    return new
```

Ensuite on fait un test :

```
#=============================================================================
# Redimensionnement de l'image et affichage
#=============================================================================
image_resizee = image_localisee.resize(60, 100)
image_resizee.display("Image redimensionee")
```

On obtient :



On obtient le chiffre 1 avec une taille différente ! Le nombre d'erreur quant à elle a diminuée :

```
FAILED (failures=4)
An exception has occurred, use %tb to see the full traceback.
```

4. Question (4)

Notre objectif est de comparer deux images en fonction de la valeur des pixels. Comme depuis le début nous faisons une double boucle for ce qui donne :

```
#=============================================================================
# Methode de mesure de similitude entre l'image self et un modele im
#=============================================================================
def similitude(self, im):
    pix = 0
    for i in range(self.H):
      for j in range (self.W):
          if self.pixels[i,j]==im.pixels[i,j]:
              pix+=1
    cor = pix/(self.H*self.W)
    return cor
```

Ensuite concernant nos erreurs nous passons de :

```
FAILED (failures=4)
An exception has occurred, use %tb to see the full traceback.
```

A :

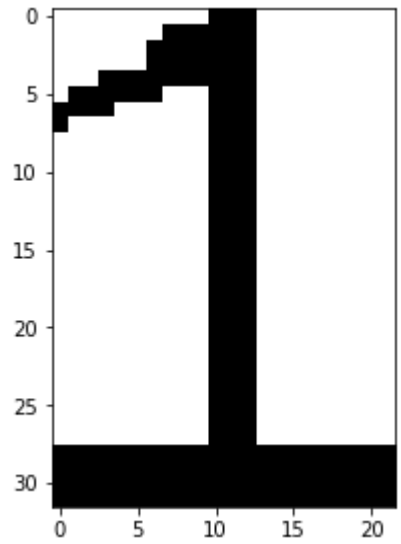
```
Ran 22 tests in 0.574s

OK
```

5. Question (5)

Nous avons écrit la fonction reconnaissance chiffre (image, liste modèles, S) qui va effectuer la reconnaissance de chiffre sur l'image donnée en entrée de la fonction. Pour cela on a dans la fonction tout à tour, binariser l'image et la localiser. Ensuite, nous avons calculé sa similitude à tous les modèles (en redimensionnant l'image à la taille du modèle) et trouver le modèle pour lequel il y a la plus grande similitude tout en renvoyant un entier compris entre 0 et 9. Le code :

```
def reconnaissance_chiffre(image, liste_modeles, S):
    count=[]
    im_b = image.binarisation(S)
    loc = im_b.localisation()
    for i in liste_modeles:
        res = loc.resize(i.H, i.W)
        count.append(res.similitude(i))
    return count.index(max(count))
```




```
Le chiffre reconnu est :  1
```

Concernant Les erreurs sur fonction reconnaissance :

```
Ran 3 tests in 0.227s

OK
```

6. Question (6) :

On a produit le tableau :

Fichier	Max mesure de corrélation	Seuil	Chiffre test	Chiffre reconnu
Test1.JPG	0.8841145833333334	70	4	4
Test1.JPG	0.85546875	30	4	4
Test1.JPG	0.9375	110	4	4
Test4.JPG	0.8125	10	2	2
Test4.JPG	0.8328125	50	2	2
Test4.JPG	0.8265625	90	2	2
Test7.JPG	0.07589285714285714	20	5	7
Test7.JPG	0.08779761904761904	100	5	7
Test7.JPG	0.4659090909090909	6000	5	8

V. Conclusion

Nous avons réussi à compléter les codes manquant de manière à réussir les fichiers test_image et test_reconnaissance. Pour cela nous avons compris les notions (binarisation, recadrage, redimensionnement, recherche de similitude) indispensables à la réalisation des fonctions. Cependant, nous n'avons pas pu déterminer de seuil commun à chaque image test pour obtenir un maximum au niveau de la mesure corrélation. Ce TP nous a aussi permis de conforter certaines notions tout en approfondissant d'autres. Nous avons aussi remarqué une mauvaise corrélation au niveau du test 7 que nous pouvons expliquer par une typographie différente entre l'image test et le modèle.