

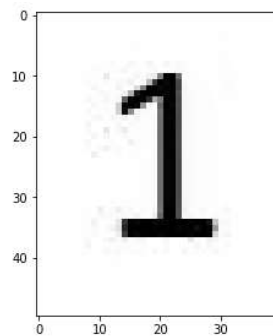
Rapport de TP2 – Lecture automatique de chiffres par analyse d'image

I. Introduction

Le but de ce TP est de créer un code qui va pouvoir reconnaître un chiffre présent sur une image en le comparant à des images de références. Pour cela, nous allons notamment devoir faire du traitement d'image.

II. Prise en main de l'environnement

Pour commencer, nous avons exécuté le fichier main.py et nous avons obtenu l'affichage de cette image. On obtient une image avec des nuances de gris.



III. Travail préparatoire

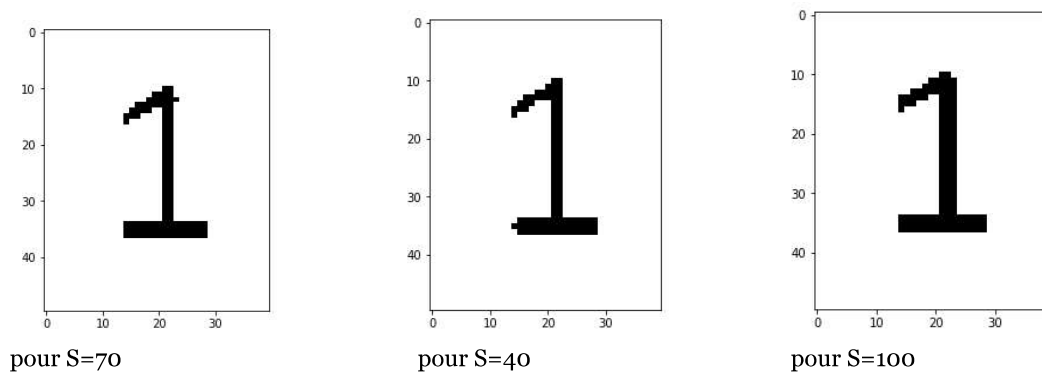
1. Nous avons constaté que W représente la largeur de la grille et H représente la hauteur de la grille
2. La méthode de binarisation :

Ici le but est de créer une nouvelle image constituée uniquement avec du blanc (valeur du pixel : 255) et noir (valeur du pixel : 0). On va alors ici supprimer les nuances de gris.

Pour cela, nous avons parcouru tous les pixels de l'image et comparé la valeur de chaque pixel à la valeur entrée S. Si la valeur de ce pixel est supérieure à S alors on attribue une nouvelle valeur à ce pixel qui sera alors 255. De même lorsqu'elle est inférieure en remplaçant par 0.

```
def binarisation(self, S):  
    im_bin = Image()  
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))  
  
    for l in range(self.H):  
        for c in range(self.W):  
            if self.pixels[l,c]>S:  
                im_bin.pixels[l,c] = 255  
            else:  
                im_bin.pixels[l,c] = 0  
  
    return im_bin
```

Voici les résultats obtenus:



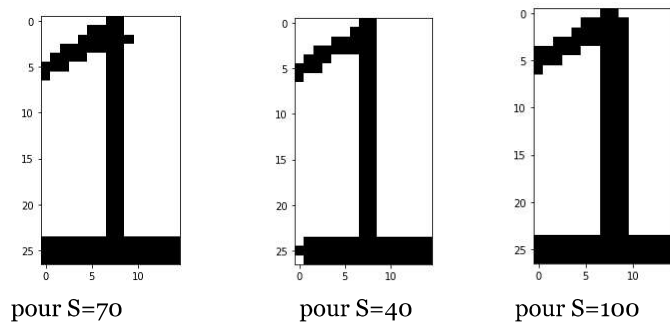
3. La méthode de localisation

Ici nous allons créer une nouvelle image qui va recadrer sur le nombre. Pour ce faire, il faut réussir à trouver les pixels qui sont situés aux extrémités du cadre englobant le nombre.

Pour ce faire, nous avons créé un code qui détermine la position des pixels se situant aux extrémités du rectangle englobant le nombre. Le code parcourt l'ensemble des pixels jusqu'à rencontrer un pixel noir (de valeur = 0). Puis l'ordinateur vérifie si ce pixel est un pixel qui se situe bien plus à l'extrémité que le précédent afin d'avoir un cadre englobant le nombre.

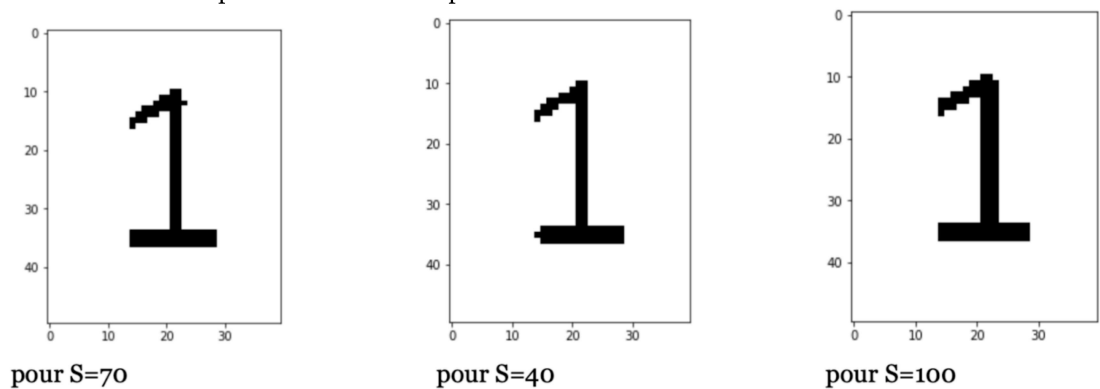
```
def localisation(self):  
    c_min = self.W  
    c_max = 0  
    l_min = self.H  
    l_max = 0  
  
    for l in range (self.H):  
        for c in range (self.W):  
            if self.pixels[l][c]==0:  
                if c<= c_min:  
                    c_min = c  
                if c>= c_max:  
                    c_max = c  
                if l<= l_min:  
                    l_min = l  
                if l>= l_max:  
                    l_max = l  
  
    imag = Image()  
    imag.set_pixels(self.pixels[l_min:l_max+1,c_min:c_max+1])  
    return imag
```

Voici les résultats obtenus :

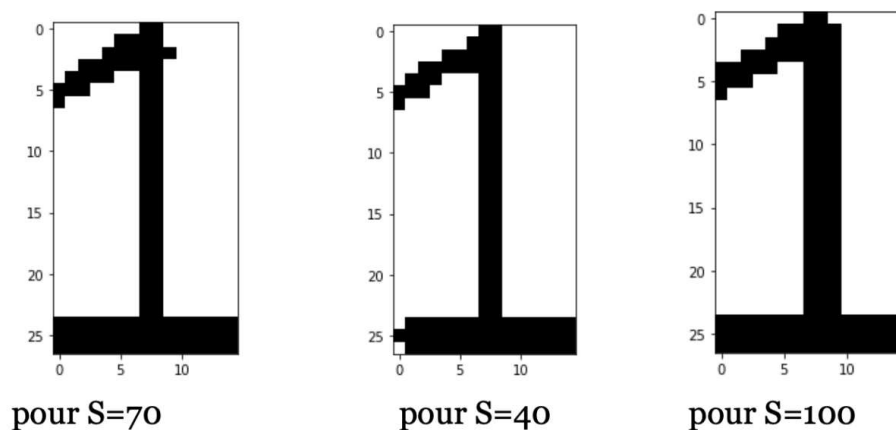


IV. Reconnaissance automatique de chiffre

- Voici les résultats obtenus pour différents seuils pour la méthode binarisation:



- Voici les résultats obtenus pour différents seuils pour la méthode localisation:



Pour les questions 1 et 2, on a des erreurs sur les méthodes resize et similitude. Ces erreurs sont présentes car nous ne les avons pas encore codées.

```
=====
FAIL: test_resize_is_not_none (__main__.Test_Image_resize)
Teste si le résultat de resize renvoie bien quelque chose.
-----
```

```
=====
FAIL: test_similitude_is_not_none
(__main__.Test_Image_similitude)
Teste la méthode similitude ne renvoie pas None (oublie de
return...).
-----
```

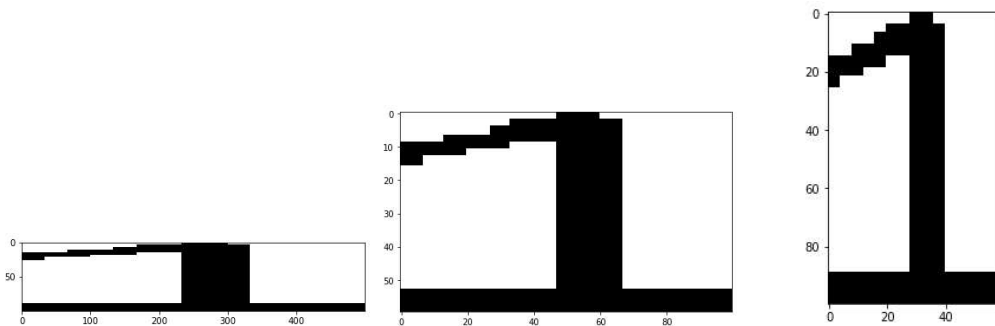
- Le but ici est d'adapter la taille de l'image au modèle. Pour ce faire, on crée une nouvelle image qui sera redimensionnée l'image aux dimensions voulues.

```
def resize(self, new_H, new_W):
    image_resized = Image()

    image_resized.set_pixels(np.uint8(resize(self.pixels, (new_H, new_W), 0)) * 255)

    return image_resized
```

On obtient :



pour les valeurs (100, 500)

pour les valeurs (60, 100)

pour les valeurs (100, 60)

Ici, il ne reste plus qu'une erreur sur la méthode similitude car nous les avons pas encore codées.

```
=====
FAIL: test_similitude_same_image_is_one
(__main__.Test_Image_similitude)
-----
```

4. La méthode similitude:

Le but de la méthode similitude est de déterminer un coefficient qui mesure la similitude entre l'image et une image de référence.

Pour ce faire, on suppose tout d'abord que les deux images ont la même taille. On parcourt les valeurs des pixels des deux images. Le code teste si les deux pixels sont identiques c'est à dire si les pixels (pixels situés sur la même position sur chaque image) ont la même valeur (255 ou 0). On utilise alors un compteur qui va compter le nombre identiques de pixels entre les deux images. Le code va ensuite pouvoir retourner un coefficient de similitude entre les deux images, obtenu en divisant le compteur par le nombre de pixels au total sur l'image.

```
def similitude(self, im):  
    pixel_identique = 0  
  
    for l in range (self.H):  
        for c in range (self.W):  
            if self.pixels[l,c] == im.pixels[l,c]:  
                pixel_identique +=1  
  
    coef = float(pixel_identique/(self.H*self.W))  
    return coef
```

5. Nous allons ici créer un code qui renvoie le nombre reconnu sur l'image en comparant la similitude de l'image à une liste d'images de références de 0 à 9.

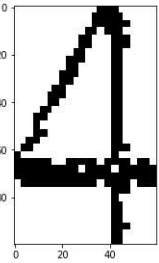
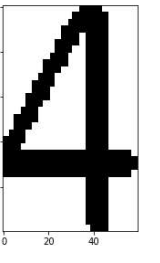
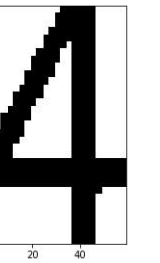
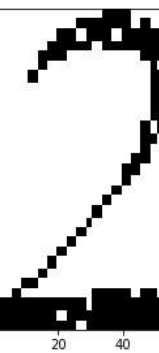
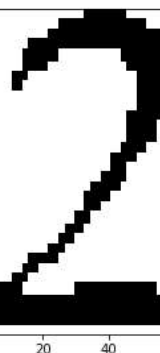
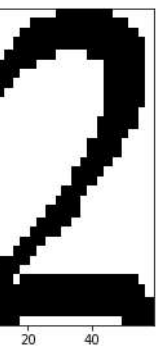

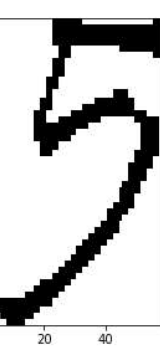
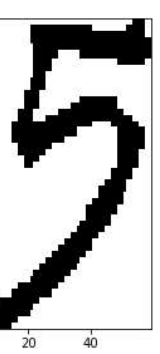


Pour ce faire, nous avons tout d'abord utilisé la méthode de binarisation et de localisation sur l'image. Puis, pour toutes les images de référence, l'image est redimensionnée par rapport à celle-ci (pour avoir deux images de même taille). La similitude est alors calculée pour chaque image de référence et stockée dans une liste. Le code renvoie ensuite le nombre où la similitude est maximale.

```
def reconnaissance_chiffre(image, liste_modeles, S):  
  
    i = image.binarisation(S)  
    imag = i.localisation()  
    liste_simi = []  
  
    for mod in liste_modeles:  
  
        imag_redim = imag.resize(mod.H, mod.W)  
        simi = imag_redim.similitude(mod)  
        liste_simi.append(simi)  
  
    return liste_simi.index(max(liste_simi))
```

Lorsque nous lançons test_reconnaissance.py, il fonctionne bien:

```
-----  
-----  
Ran 3 tests in 0.287s  
  
OK
```

6.

Test n°\Seuil	10	100	200
1	 <p>chiffre reconnu : 4</p>	 <p>chiffre reconnu : 4</p>	 <p>chiffre reconnu : 4</p>
5	 <p>chiffre reconnu : 2</p>	 <p>chiffre reconnu : 2</p>	 <p>chiffre reconnu : 2</p>
7	 <p>chiffre reconnu : 8</p>	 <p>chiffre reconnu : 8</p>	 <p>chiffre reconnu : 8</p>
10	<p>Erreur</p>	 <p>chiffre reconnu : 6</p>	 <p>chiffre reconnu : 6</p>

On note également que le code ne fonctionne pas lorsqu'il y a une succession de nombres notamment sur les tests 8 et 9. On pourrait améliorer notre code pour qu'il arrive à différencier tous les chiffres sur une image.

Pour les tests que nous avons effectués, la variation du seuil S n'influence pas la reconnaissance du chiffre. Mais nous remarquons que pour le test 10 à un seuil trop bas ne permet pas la reconnaissance du chiffre et un chiffre trop élevé montre une image noire. Même si le chiffre est quand même reconnu, nous supposons qu'il faut privilégier un seuil intermédiaire (environ 100-150). Enfin, on note également que le test 7, mais aussi le test 8, ne sont pas reconnus par le code.

V. Conclusion

Dans ce TP, nous avons pu faire du traitement d'image en réalisant notamment la binarisation mais également la localisation. Ce travail préliminaire nous a permis de pouvoir comparer une image à une liste d'image de référence afin de pouvoir déterminer un coefficient de similitude pour chaque image. Ensuite, l'ordinateur a pu renvoyer le nombre reconnu en choisissant celui qui avait le plus de ressemblance à une image. Ce code peut être très utile pour par exemple entrer les chiffres d'une écriture manuscrite automatiquement dans un ordinateur.