SARRE Valentin
Léo MAZZOCHI
Date du TP (02/12/2021)

Rapport de TP2 –Reconnaissance chiffres

I. Introduction

Dans ce TP, nous étudions la reconnaissance d'une image dans le but de les analyser et de reconnaître automatiquement des caractères en analysant une image par différents niveaux de gris. Comme cela est dit dans le TP, il y a 5 étapes :

- La binarisation, qui transforme les niveaux de gris en noir ou blanc.
- La localisation, qui permet de couper l'image afin d'avoir seulement le chiffre
- L'adaptation de la taille au modèle, à la taille des modèles préchargés.
- La mesure de ressemblance permet donner un résultat compris entre 0 et 1 de la ressemblance d'une image aux modèles.

Une décision sera prise afin de choisir le chiffre qui a le plus de corrélation avec l'image.

II. Travail préparatoire

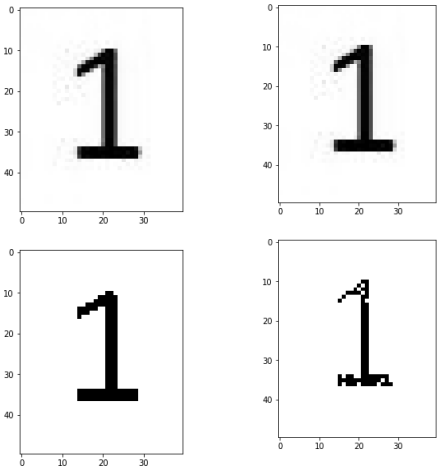
1. La classe image

La classe image permet de créer des attributs H pour la hauteur et W pour la largeur de l'image. Ces dimensions sont le nombre de pixels en hauteur et en largeur. L'attribut pixel est un tableau contenant une valeur comprise entre 0 et 255, 0 représentant le noir et 255 le blanc.

2. Méthode binarisation

L'objectif est de modifier l'image composée de niveau de gris, par des niveaux de noir et de blanc. Pour cela nous créons un nouveau tableau pixel, qui pour chaque pixel qui premier tableau change sa valeur à 255 si la valeur de l'ancien est supérieur l'intensité S défini sinon à 0.

Exemple à S = 100 et S=5 Code



```
def binarisation(self, S):
    # creation d'une image vide
    im_bin = Image()

    # affectation a l'image im_bin d'un tableau de pixels de meme taille
    # que self dont les intensites, de type uint8 (8bits non signes),
    # sont mises a 0
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))

    # TODO: boucle imbriquees pour parcourir tous les pixels de l'image im_bin
    # et calculer l'image binaire
    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
```

Nous pouvons voir une image résultante binarisée, qui ressemble toujours au 1 de l'image initiale.

3. Méthode localisation

Cette méthode-ci va permettre de recadrer l'image. Pour avoir une image de dimensions minimales qui contient le chiffre. Nous avons effectué 2 boucles for qui vont déterminer les limites du chiffre (cmin, cmax, lmin, lmax) afin que tous les pixels noirs du chiffre soient à l'écran.

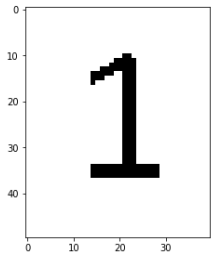
Code

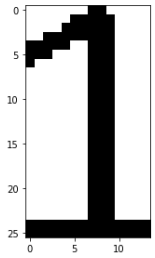
```
def localisation(self):
    cmin=self.W-1
    cmax=0
    lmin=self.H-1
    lmax=0

    for i in range(self.H):
        for j in range (self.W):
            if self.pixels[i][j] == 0 :
                if j> cmax:
                    cmax=j
                if cmin > j:
                    cmin=j
                if lmin>i:
                    lmin =i
                if i>lmax:
                    lmax=i

    im_bin = Image()
    im_bin.pixels=self.pixels[lmin:lmax , cmin:cmax]
    return im_bin
```

Résultat





Le résultat semble correct car le chiffre n'est pas coupé, il ne manque pas de pixel noir et le chiffre est à l'endroit

III. Reconnaissance automatique de chiffre

1. Contrôle du bon fonctionnement de la méthode binarisation et localisation

Pour vérifier qu'il n'y a pas de problème de fonctionnement, nous lançons le programme test_image.py. Les seuls échecs et erreurs affichés ne concernent que les méthodes qui seront à programmer par la suite, qui apparaissent donc comme des erreurs ou échecs car elles ne sont pas encore codées.

2. Méthode resize

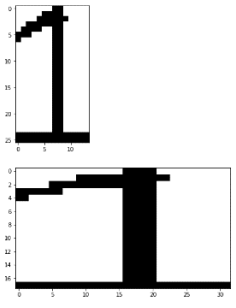
Nous allons programmer une méthode permettant de redimensionner une image avec des dimensions voulues. Pour cela, on utilisera la fonction resize de la librairie skimage.

En entrant les dimensions (18,32), on obtient bien une image dont les dimensions ont été modifiées aux valeurs données.

Code

```
def resize(self, new_H, new_W):
    im = Image()
    redef = resize(self.pixels, (new_H,new_W),0)
    im.set_pixels(np.uint8(redef*255))
    return im
```

Résultat



3. Méthode Similitude

Cette méthode a pour objectif de mesurer un taux de correspondance entre 2 images de même dimensions, qui sera compris entre 0 et 1.

Lorsque nous lançons le programme de test, aucune erreur n'est détectée. Cela veut donc dire que le programme renvoie bien un nombre à virgule compris entre 0 et 1 et qui est bien proportionnel à la corrélation entre deux images.

```
def similitude(self, im):
    R=0
    for i in range (self.H):
        for j in range (self.W):
            if self.pixels[i][j]==im.pixels[i][j]:
                R=R+1
            else:
                R
    return R/(self.H*self.W)
```

4. Méthode Reconnaissance Chiffre

Cette dernière fonction permet de déterminer le chiffre écrit sur l'image entrée en plusieurs étapes.

Tout d'abord la fonction va binariser l'image en utilisant la méthode codée auparavant, puis la localiser. Ensuite pour chaque modèle disponible.
A chaque tour de boucle, la fonction va mettre l'image localisée dans les mêmes dimensions que le modèle comparé dans notre si et calculer la similitude avec ce modèle.
Si la similitude correspondant au modèle est supérieure à la dernière similitude retenue, le programme mémorise le chiffre correspondant au modèle (le numéro du tour i) et met à jour la valeur de la similitude pour les tours de boucle suivants.

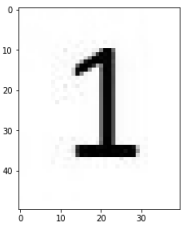
Là aussi, lorsque nous lançons le programme de test, aucune erreur n'est détectée. Cela veut donc dire que le programme détecte le bon chiffre.

Code

```
def reconnaissance_chiffre(image, liste_modeles, S):
    img1 = image.binarisation(S)
    img2 = img1.localisation()
    sim = 0
    chiffre=-1

    for i in range (len(liste_modeles)):
        img3=img2.resize(liste_modeles[i].H,liste_modeles[i].W)
        if img3.similitude(liste_modeles[i])>sim:
            sim = img3.similitude(liste_modeles[i])
            chiffre=i
            print("pour" ,i, "=", sim)
    return chiffre
```

Résultat

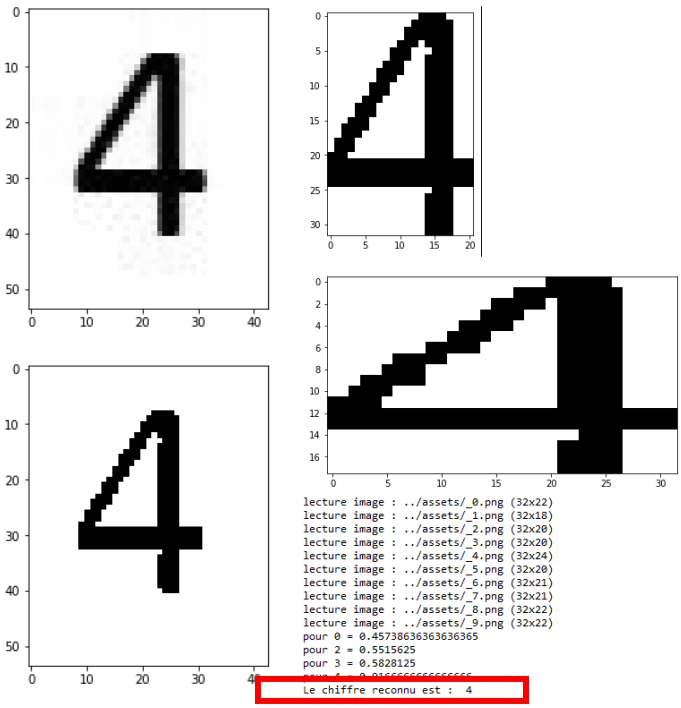


```
lecture image : ../assets/_0.png (32x22)
lecture image : ../assets/_1.png (32x18)
lecture image : ../assets/_2.png (32x20)
lecture image : ../assets/_3.png (32x20)
lecture image : ../assets/_4.png (32x24)
lecture image : ../assets/_5.png (32x20)
lecture image : ../assets/_6.png (32x21)
lecture image : ../assets/_7.png (32x21)
lecture image : ../assets/_8.png (32x22)
lecture image : ../assets/_9.png (32x22)
pour 0 = 0.4034090909090909
pour 1 = 0.9097222222222222
Le chiffre reconnu est :  1
```

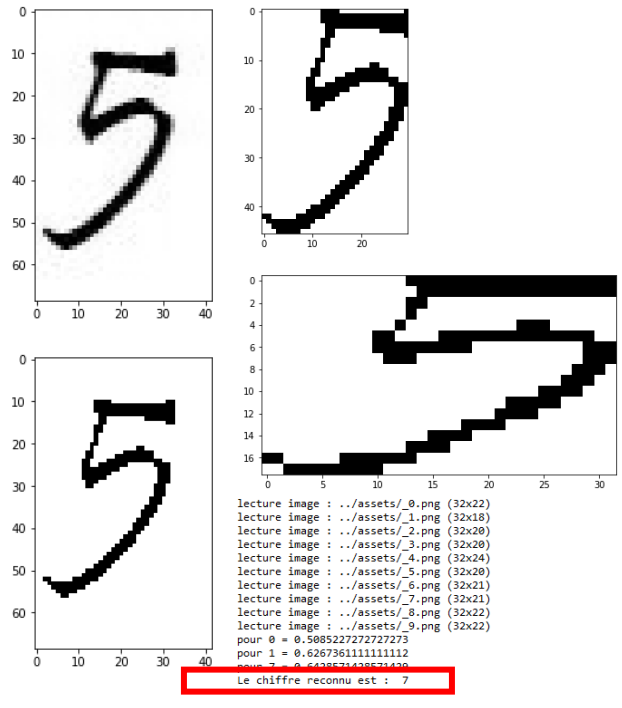
5. Test avec une autre image et détermination du seuil optimal

En modifiant, l'image de test, nous avons pu retrouver le bon chiffre (voir l'exemple si dessous). Mais notre programme à des limites car nous n'avons pas pu retrouver pour chaque image son chiffre correspondant. C'est à dire pour ceux ''trop mal écris" comme le 5 ou encore lorsque plusieurs chiffres étaient sur la même image.

Résultat avec le test1 (chiffre 4)



Résultat avec le test7 (chiffre 5)



Afin de déterminer le seuil optimal à la détection du chiffre, on réalise plusieurs essais répertoriés dans le tableau, la valeur obtenue de chaque test pour chaque seuil (S) est le maximum de similitude obtenu.

	Test1 (chiffre4)	Test2 (chiffre1)	Test3 (chiffre2)	Test6 (chiffre4)
S = 50	0.83	0.913	0.89	0.71
S = 70	0.86	0.915	0.898	0.72
S = 100	0.9375	0.87	0.90	0.714
S = 150	0.915	0.912	0.88	0.71

Nous pouvons conclure de ce tableau que le Seuil (S) le plus pertinent se trouve entre 70 et 100 environ.

IV. Conclusion

Durant ce TP, nous avons pu créer différentes fonctions au sein d'une classe dans le but de retrouver le chiffre de l'image.
Pour faire cela, nous avons dû créer une méthode binarisation qui transforme les couleurs de l'image qui sont sous la forme de nuances de gris en deux valeurs (noir et blanc), et une méthode localisation qui recadre l'image autour du chiffre.
Enfin, pour chacun des modèles, l'image est redimensionnée selon le modèle pour que le programme puisse les comparer et nous afficher le chiffre qui lui semble être le plus ressemblant.