DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1
Date du TP : 15 Novembre 2021.

Rapport de TP2 – LECTURE AUTOMATIQUE de CHIFFRES

II - Prise en main de l'environnement

Il nous est demandé d' exécuter le `main.py`. L'image qui s'affiche est `'1'`.

III - Travail préparatoire

3.1

On nous demande d'analyser la classe image, en particulier les attributs `H` et `W`.

On initialise, au début de la classe image, l'origine de l'image en haut à gauche à la position `(0;0)`.

```
12          self.H = 0
13          self.W = 0
```

Plus tard dans la classe, on voit que `H` et `W` servent à définir la taille du tableau qui contiendra l'image.

3.2

Nous devons écrire un code qui transforme l'image en nuances de gris en une image avec seulement du noir (0) et du blanc (255).

On cherche à tirer les pixel noir et les pixels blancs parmis des nuances de pixels gris. Pour cela, on teste la valeur de chaque pixel. On choisit une valeur S de seuil. Pour les valeurs de pixels ≤ S, les pixels seront remplacés par des pixels noirs, sinon, si leur valeur est > S, ils seront remplacés par des pixels blancs.

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

DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1

3.3

Nous devons écrire un code qui recadre l'image binarisée pour avoir une zone d'image restreinte uniquement au rectangle englobant le chiffre.

On cherche le 1er pixel noir, pour que la ligne de ce pixel devienne la nouvelle ligne 0 dans l'image recadrée. Idem pour les colonnes. On cherche finalement à ce que la dernière ligne soit la ligne max de la nouvelle image, et idem pour les colonnes.

Pour cela, on parcourt toute la grille. On commence par parcourir la 1ère ligne, on teste la valeur du pixel sur chaque colonne. Si la colonne a une valeur de 255, c'est un pixel blanc. Lorsque l'on rencontre un pixel noir (colonne ayant une valeur de 0), on sait qu'on a touché un 'bord' du chiffre … etc.

```
def localisation(self):
    im_loc = Image()
    lmin = self.H-1
    lmax = 0
    cmin = self.W-1
    cmax = 0
    for l in range (len(self.pixels)):
        for c in range (len(self.pixels[0])):
            if l<lmin:
                if self.pixels[l][c] == 0:
                    lmin = l
            elif l>=lmax:
                if self.pixels[l][c] == 0:
                    lmax = l
            if c<cmin:
                if self.pixels[l][c] == 0:
                    cmin = c
            elif c>=cmax:
                if self.pixels[l][c] == 0:
                    cmax = c
    im_loc.set_pixels(self.pixels[lmin:lmax,cmin:cmax])
    return im_loc
```

Une fois les valeurs des lignes et des colonnes aux limites trouvées, on pourra recadrer l'image avec la fonction

DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1

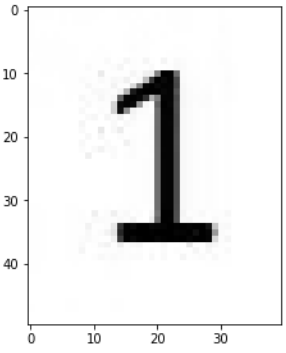
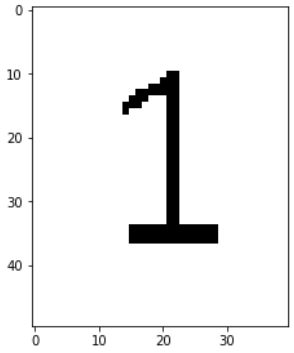
$$\texttt{ima.pixels[}l_{min}\texttt{ : }l_{Max}\texttt{+1 , }c_{min}\texttt{ : }c_{Max}\texttt{+1].}$$

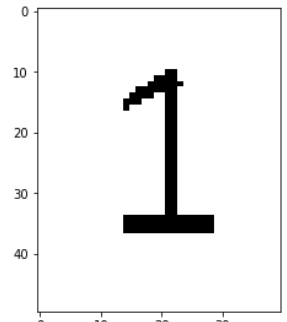
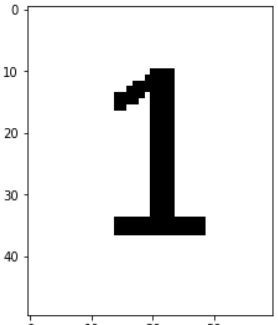
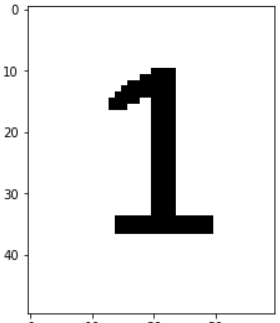
IV - Reconnaissance automatique de chiffre

4.1

Il nous est demandé d'essayer la binarisation avec différentes valeurs de seuil.

On teste et on obtient

Original=>  30=> 

70=>  150=>  225=> 

Lorsque l'on binarise avec différents seuillages on obtient une image définie avec plus ou moins de précision. On remarque que la qualité de l'image binarisée se rapproche de l'image originale quand on binarise plusieurs fois notre image.

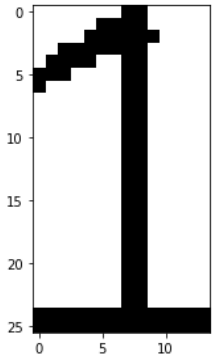
On teste avec `test_Image.py`, **on obtient** aucune erreur.

DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1

4.2

Il nous est demandé d'essayer la localisation avec différentes valeurs de seuil.

On teste et on obtient



On teste avec `test_Image.py`, **on obtient** aucune erreur.

4.3

Il nous est demandé d'utiliser la fonction `resize(im.pixels, (new_H,new_W), 0)` afin de redimensionner l'image.

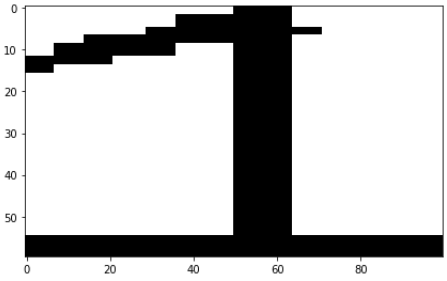
Il faudra renseigner les nouvelles valeurs de `H` et `W` : `new_H` et `new_W`. (On utilise l'image localisée car on sera sûrs de la taille du chiffre, et de ne pas prendre en considération l'encadrement blanc autour).

On utilise la fonction `resize(self,new_H,new_W)` pour dimensionner la nouvelle image à la taille que l'on veut.

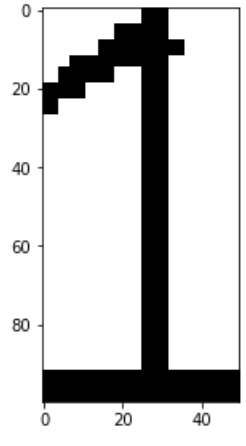
```
def resize(self, new_H, new_W):
    newIm = Image()
    pixels_resized = resize(self.pixels, (new_H,new_W), 0)
    newIm.set_pixels(np.uint8(pixels_resized*255))

    return newIm
```

DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1

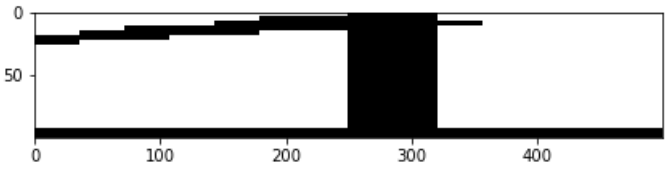
On teste avec les valeurs `(60,100)`, `(100,50)` et `(100,500)` **et on obtient**



<=resize(60,100)



<=resize(100,50)



<=resize (100,500)

Ensuite on re-convertit cette image en int car elle a pu passer en float lors du redimensionnement :

```
np.uint8(pixels_resized*255)
```

On teste avec `test_Image.py`, **on obtient** aucune erreur.

4.4

On nous demande d'ajouter `similitude(self, image)` pour trouver des similitudes entre les images.

On fait une boucle pour tester chaque pixel des deux images. Pour chaque pixel testé, la variable `compte` augmente de 1. Pour chaque pixel similaire d'une image à l'autre, la variable `simi` augmente de 1. À la fin du test, on compare les deux variables en calculant `compte / simi`.

DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1

```python
def similitude(self, im):
    simi = 0
    compte = 0
    for l in range (len(self.pixels)):
        for c in range (len(self.pixels[0])):
            compte += 1
            if self.pixels[l][c] == im.pixels[l][c]:
                simi += 1
    return (simi/compte)
```

4.5

On nous demande d'écrire la fonction `reconnaissance_chiffre(image, liste_modeles, S)`. Pour cela, il faut binariser l'image puis la localiser. Ensuite, il faut tester sa similitude avec chaque chiffre, en redimensionnant l'image pour chaque image de chiffre à comparer. Puis, on testera chaque degré de similitude pour savoir de quel chiffre l'image se rapproche le plus.

```python
def reconnaissance_chiffre(image, liste_modeles, S):
    image = image.binarisation(S)
    image = image.localisation()
    sim = 0
    chiffre = 0
    for i in range (len(liste_modeles)):
        image2 = liste_modeles[i]
        image = image.resize(image2.H, image2.W)
        im = image.similitude(image2)
        if im > sim:
            sim = im
            chiffre = i
    return chiffre
```

4.6

On essaye la fonction de reconnaissance en modifiant l'image de test dans `main.py`.

On teste pour le fichier test2 **et on obtient** :

```
lecture image : ../assets/test2.JPG (50x40)
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)
Le chiffre reconnu est :  1
```

DEPLANO Xavier
YEN-PON Cassilla
MM3-TDA1

On teste pour le fichier test3 **et on obtient** :

```
Le chiffre reconnu est :  2
```

Conclusion

Ce TP nous a permis de comprendre et d'apprendre à traiter des images en python puis les traiter pour reconnaître différents modèles et les comparer. Nous avons pu approfondir nos connaissances sur comment fonctionne une classe et comment écrire des méthodes pour optimiser cette classe.