III – Travail Préparatoire

(1) Pour pouvoir afficher correctement les images ont utilisé des librairies python ci-dessous :

```
from skimage import io
from skimage.transform import resize
import matplotlib.pyplot as plt
import numpy as np
```

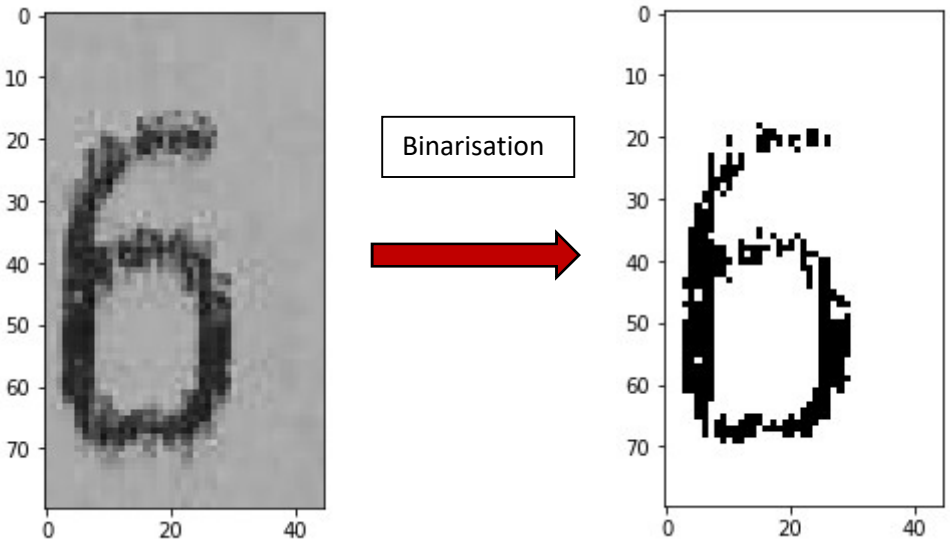
(2). Écrire une méthode **binarisation(self, S)** à la classe **Image** qui permet de passer d'une image codée sur 256 valeurs à une image avec seulement deux valeurs (0 ou 255). Cela se fait en comparant pour chaque pixel de l'image la valeur du pixel à une valeur choisie par l'utilisateur (entrée **S** de la méthode). Ainsi en pratique, il faut itérer sur tous les pixels du tableau numpy et comparer la valeur au seuil. On veut également que le résultat soit donné sous la forme d'une nouvelle image que l'on crée dans la fonction afin de ne pas mmodifier l'image de base.

On écrit la méthode binarisation(self,S)

```
#=========================================================================
# 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 c in range(self.W):
            if self.pixels[i][c]>=S:
                im_bin.pixels[i][c] = 255
            else :
                im_bin.pixels[i][c]=0
    return im_bin
```

On lance les tests et la main pour confirmer les modifications :



(3). Écrire une méthode **localisation(self)** à la classe **Image** calculant et retournant l'image recadrée sur le chiffre à identifier. Le principe de ce recadrage consiste, sur une image binaire **im_bin** de dimension , à déterminer les coordonnées , , et du rectangle englobant la forme noire (valeurs 0) et à construire l'image limitée à ce rectangle englobant

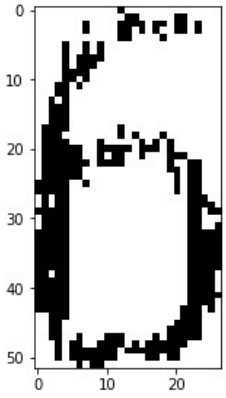
On obtient le code suivant :

```
def localisation(self):
    l_min = self.H -1
    c_min = self.W -1
    l_max = 0
    c_max = 0


    for l in range(self.H):
        for c in range(self.W):
            if self.pixels[l][c] == 0:
                if l < l_min:
                    l_min = l
                if l > l_max:
                    l_max = l
                if c <= c_min:
                    c_min = c
                if c >= c_max :
                    c_max = c

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

En effectuant les tests et exécutant la main on obtient bien la photo du chiffre correctement localisé :



Verifs OK

IV - Reconnaissance automatique de chiffre

(1) & (2) Lancement du main fait auparavant cf capture ci – dessus.

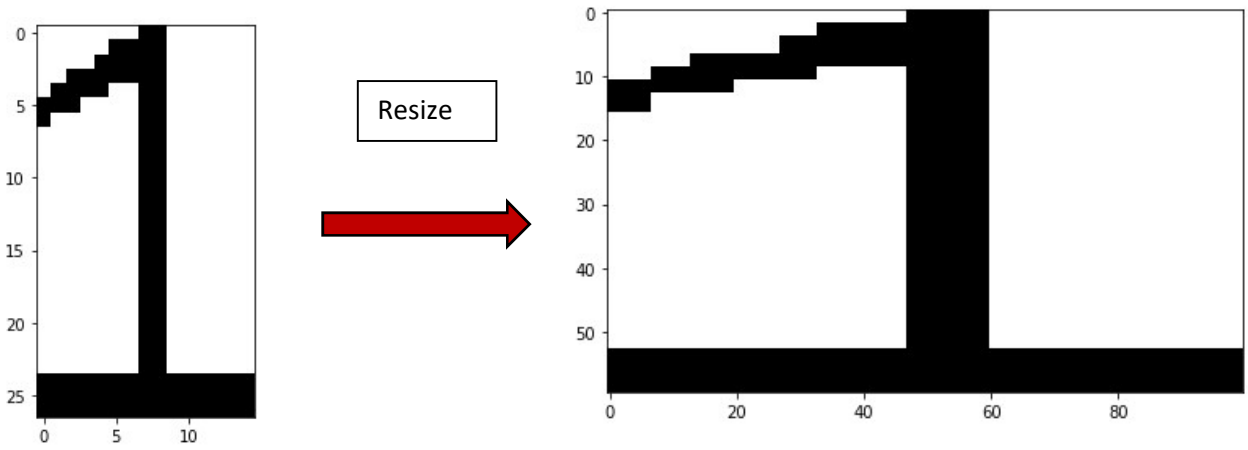
(3). On dispose de la fonction **resize** de la libraire **skimage** permettant de donner à une image des dimensions imposées. Cette fonction s'utilise de la manière suivante

On réalise le programme suivant et on obtient l'image redimensionnée (on a décidé de changer de chiffre pour plus de visibilitée.

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

Pour obtenir l'image redimensionnée il faut utilisé dans le main la commande suivante :

```
image_resizee = image_localisee.resize(60, 100)
image_resizee.display("Image redimensionee")
```



(4). Ajouter à la classe **Image**, la méthode **similitude(self, image)** qui mesure la similitude par corrélation d'images entre l'image représenté par l'objet courant (**self**) et un objet de type **Image** entrée en paramètre. La procédure pour le calcul de similitude est décrite dans la présentation du TP

On a donc le code suivant :

```
def similitude(self, im):
    nombre_similitude = 0
    for x in range(self.H):
        for y in range(self.W):
            if self.pixels[x][y] == im.pixels[x][y]:
                nombre_similitude += 1
        taux = nombre_similitude/(self.H * self.W)
        return taux
```

Le programme reconnaît bien le chiffre 1 :

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

V – Conclusion

Ce TP nous a permis d'analyser des chiffres et d'utiliser les différents outils graphique que peut nous proposer Python.
La partie reconnaissance a été la plus complexe à mettre en place