

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

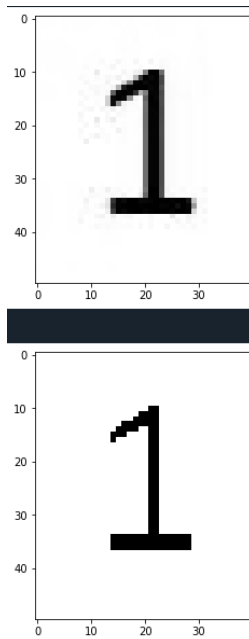
I. Introduction

Dans le cadre du module INFO501 – Numérisation et Algorithmique, nous allons réaliser le deuxième TP qui a pour objectif 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 défini en 5 étapes qui sont les suivantes : La binarisation, Localisation, Adaptation de la taille au modèle, Mesure de ressemblance par corrélation et Décision.

II. Travail Préparatoire

1. Binarisation

```
def binarisation(self, S):  
    """Cette fonction permet de binariser une image  
    selon un seuil S défini par l'utilisateur"""  
    # 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 intensités, de type uint8 (8bits non signes),  
    # sont mises a 0  
    im_bin.set_pixels(np.zeros((self.H, self.W), dtype=np.uint8))  
    h= im_bin.H  
    w= im_bin.W  
    for i in range(h):  
        for j in range(w):  
            if self.pixels[i][j] >= S:  
                im_bin.pixels[i][j]=255  
            else :  
                im_bin.pixels[i][j] = 0  
  
    # TODO: boucle imbriquées pour parcourir tous les pixels de l'image im_bin  
    # et calculer l'image binaire  
  
    return im_bin
```

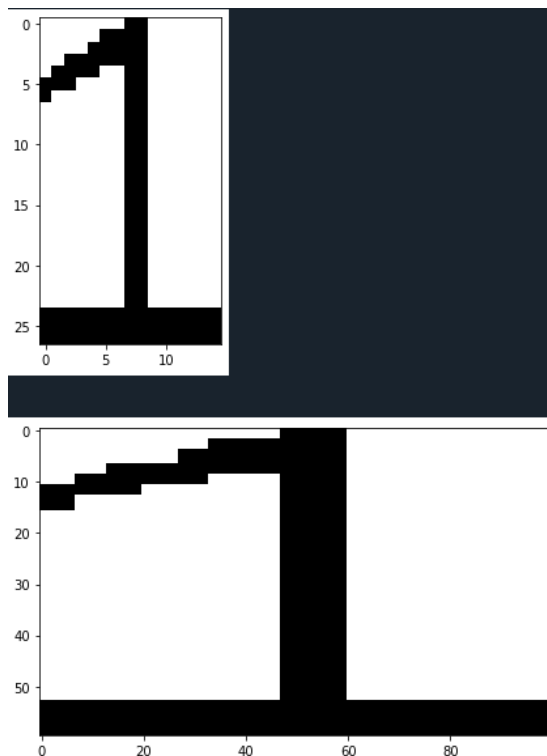


On peut constater que les pixels ayant une couleur trop claire sont devenus blancs et que les plus foncés (à partir du seuil S) sont devenus noirs.

2. Localisation

```
#=====
# 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):
    """Cette fonction permet de recadrer une image autour d'une forme"""
    # 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
    h= self.H
    w= self.W
    l_min = self.H
    c_min = self.W
    l_max= 0
    c_max =0
    #initialisation des coordonnees du futur rectangle englobant la forme noire
    for l in range(h):
        for c in range(w):
            if self.pixels[l][c] == 0:
                if l < l_min:
                    l_min = l
                if c < c_min:
                    c_min = c
                if l > l_max:
                    l_max = l
                if c > c_max:
                    c_max =c
    #on attribue a l'image ses nouvelles dimensions
    im_bin.pixels=self.pixels[l_min:l_max+1,c_min:c_max+1]
    im_bin.H=l_max - l_min
    im_bin.W=c_max - c_min
    return (im_bin)
```

On peut constater que la fonction marche correctement car le chiffre est bien centré sans marges blanches sur les cotés



III. Reconnaissance Automatique de chiffre

1. Redimensionnement

```
#####  
# Methode de redimensionnement d'image  
#####  
def resize(self, new_H, new_W):  
  
    # creation d'une image vide  
    newim = Image()  
    #newim.set_pixels(np.zeros((new_H, new_W), dtype=np.uint8))  
    #redimensionnement et conversion de l'image  
    newim.pixels=resize(self.pixels, (new_H,new_W), 0)  
    newim.pixels=np.uint8(newim.pixels*255)  
    newim.H= new_H  
    newim.W= new_W  
    return newim
```

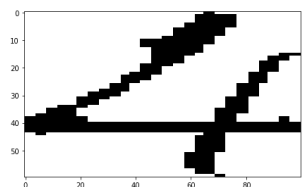
2. Similitude

```
#####  
# Methode de mesure de similitude entre l'image self et un modele im  
#####  
def similitude(self, im):  
    #on redimensionne les images afin qu'elles aient la meme taille  
    image2=im.resize(self.H, self.W)  
    #calcul du nombre total de pixels  
    nb_pix_tot=self.W*self.H  
    #compteur du nombre de pixels similaires  
    nb_pix_sim=0  
  
    h= self.H  
    w= self.W  
  
    for l in range(h):  
        for c in range(w):  
            if self.pixels[l][c]== image2.pixels[l][c]:  
                nb_pix_sim +=1  
  
    return nb_pix_sim/nb_pix_tot
```

3. Reconnaissance

```
def reconnaissance_chiffre(image, liste_modeles, S):  
    #on binarise et localise l'image à comparer  
    im=image.binarisation(S)  
    im=im.localisation()  
  
    #on stocke le taux de similitude le plus important  
    sim_max =0  
    #on stocke l'entier dont le taux de similitude est le plus grand  
    entier_ress = 0  
  
    #on parcourt la liste des images  
    for i in range (len(liste_modeles)):  
        #on redimensionne l'image à comparer avec celle de la liste  
        im=im.resize(liste_modeles[i].H, liste_modeles[i].H)  
  
        #si le taux de similitude est supérieur, on l'affecte à sim_max  
        if sim_max<im.similitude(liste_modeles[i]) :  
            sim_max = im.similitude(liste_modeles[i])  
            #on stocke le chiffre correspondant à l'image la plus proche  
            entier_ress = i  
  
    #on renvoie le chiffre trouvé  
    return entier_ress
```

Nous avons également testé avec d'autres images de référence comme le 4 ci-dessous :



```
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 : 4
```

Nous n'avons pas mis d'autres test mais la reconnaissance fonctionne avec les chiffres de 1 à 9