Rapport TP2 Python

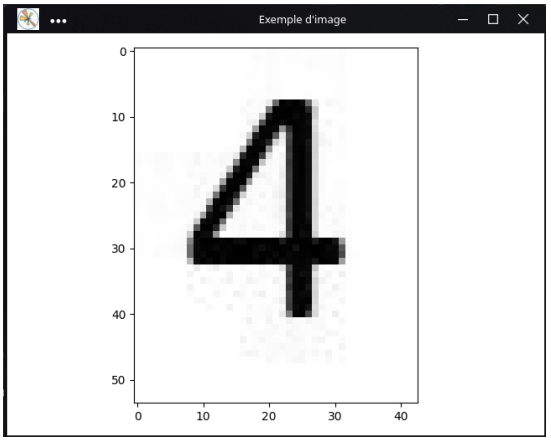
Andrew Mary Huet de Barochez

Calvin Ngor

I. Travail préparatoire

1. Analyse du code

On peut observer dans la méthode __init__ que les différents attributs de la classe sont instanciés avec des valeurs par défaut. Ces différents attributs seront ensuite remplis lors de l'importation d'une image à l'aide de la méthode load ou set_pixels.



Capture 1. Exemple d'un affichage d'image

2. Fonction binarisation

Pour cette fonction il suffit d'instancier un nouvel objet image dont on va remplir le tableau de pixel par des zéros. Pour cela il faut utiliser la fonction 'zeros' de numpy permettant l'instanciation d'un tableau avec des dimensions données, dont les valeurs seront toutes à zéro.

Ensuite on parcourt ce tableau à deux dimensions à l'aide de deux boucles for. A chaque pixel on compare la valeur de ce dernier et si elle est en dessous du seuil on attribue au pixel du nouveau tableau la valeur 0. Quand la valeur est supérieure au seuil on lui attribue 255.
Python nous permet d'écrire le 'if' 'else' et l'affectation de la variable en une ligne.

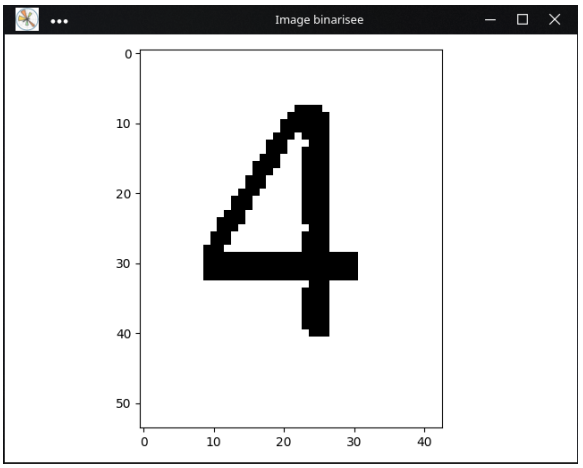
```python
def binarisation(self, S):
    # creation d'une image vide
    im_bin = Image()

    # 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))

    # Parcours du tableau
    for i in range(self.H):
        for j in range(self.W):
            # si la valeur du pixel actuel est en dessous du seuil, affecte 0, sinon 255
            im_bin.pixels[i][j] = 0 if self.pixels[i][j] < S else 255

    return im_bin
```

Code 1. fonction binarisation.



Capture 2. Exemple d'une image binarisé avec un seuil de 70.

3. Fonction localisation

Cette fonction est probablement celle qui nous à pris le plus de temps. La raison de cette perte de temps est due à la méthode que nous avons utilisée pour trouver c_min, c_max, l_min et l_max.

Nous sommes partis du principe que pour trouver ces différentes valeurs nous devions parcourir 4 fois le tableau d'un ordre différent. Nous avons donc codé un algorithme parcourant le tableau de gauche à droite pour calculer c_min, de haut en bas pour l_min etc. Pendant le parcours nous ajoutons les valeurs des variables pour chaque ligne ou colonne pour ensuite récupérer la plus petite valeur.

Cela nous à posé plusieurs problèmes. Premièrement nous avions à maintenir la cohérence de ces 4 bouts de code. Secondement les valeurs max étaient égales à la taille du max et non à sa position.

```
c_min = []
c_max = []
l_min = []
l_max = []

for i in range(self.H):
    for j in range(self.W):
        if self.pixels[i][j] == 0:
            c_min.append(j)
            break

for i in range(self.H - 1, 0, -1):
    for j in range(self.W):
        if self.pixels[i][j] == 0:
            c_max.append(j)
            break

for i in range(self.W):
    for j in range(self.H):
        if self.pixels[j][i] == 0:
            l_min.append(j)
            break

for i in range(self.W - 1, 0, -1):
    for j in range(self.H):
        if self.pixels[j][i] == 0:
            l_max.append(j)
            break

c_min = min(c_min)
c_max = min(c_max)
l_min = min(l_min)
l_max = min(l_max)
```

Code 2. Très mauvaise implémentation de la fonction localisation

Nous nous sommes ensuite rendu compte que ces valeurs pouvaient s'obtenir en seulement un parcours du tableau. Nous avons donc refactorisé la fonction et cette fois-ci nous déterminons les différentes valeurs sur la même boucle. Les valeurs vont donc se mettre à jour au fur à mesure du parcours du tableau.

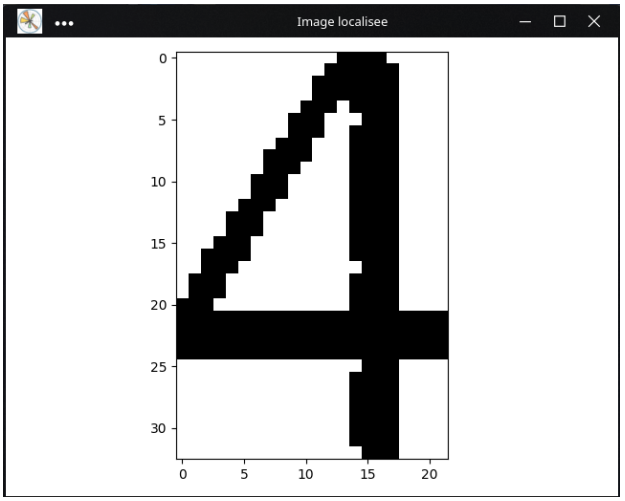
Finalement nous déclarons un nouvel objet Image et on appelle la fonction set_pixels en donnant le tableau de notre objet découpé selon les différentes valeurs que nous avons déterminé.

```python
def localisation(self):
    # Instanciation des variables de découpe
    c_min = self.W - 1
    c_max = 0
    l_min = self.H - 1
    l_max = 0

    for i in range(self.H):
        for j in range(self.W):
            # Si le pixel actuel est noir
            if self.pixels[i][j] == 0:
                # Si la valeur de j est inférieure à c_min cela signifie
                # que l'on rencontre un pixel noir à un index inférieur
                # que la dernière valeur de c_min stockée précédemment.
                if j < c_min:
                    c_min = j
                if j > c_max:
                    c_max = j
                if i < l_min:
                    l_min = i
                if i > l_max:
                    l_max = i

    # On instacie un nouvel objet Image
    img = Image()
    # On rempli l'objet avec le nouveau tableau découpé
    img.set_pixels(self.pixels[l_min:l_max + 1, c_min:c_max + 1])
    return img
```

Code 3. Implémentation de la fonction localisation

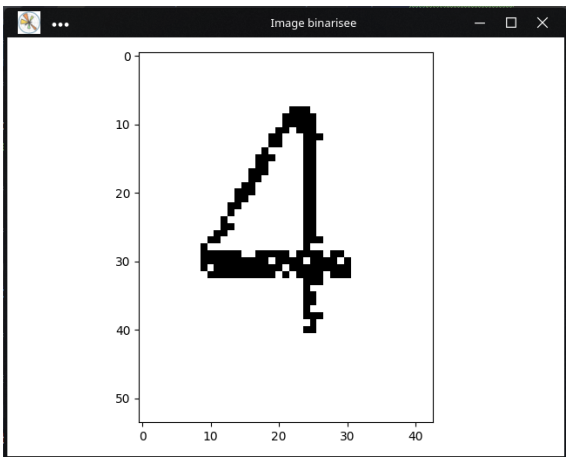


Capture 3. Exemple d'une image localisée.

II. Reconnaissance automatique de chiffre

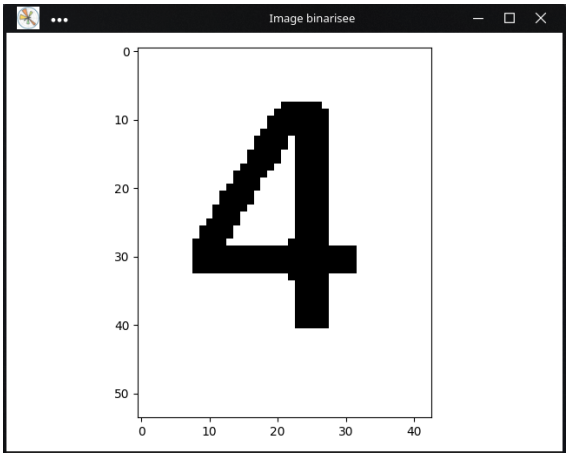
1. Variation du seuil et tests

Avec un seuil plus petit on remarque que l'image est plus précise mais perd des pixels (Voir capture 4).



Capture 4. Exemple d'une image binarisée avec un seuil de 10

Avec un seuil plus grand on peut observer que l'image est moins précise car elle accepte les pixels avec une faible intensité de noir (Voir capture 5).

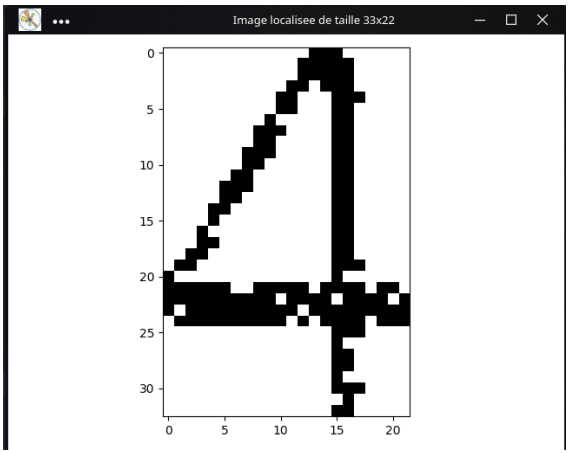


Capture 5. Exemple d'une image binarisée avec un seuil de 230.

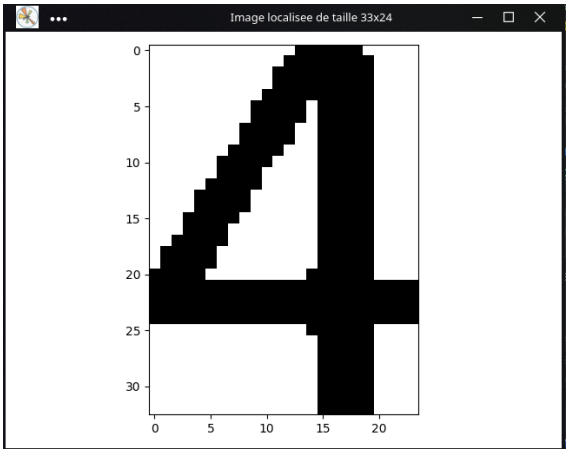
Concernant les tests de la fonction localisation ils passaient tous à l'exception de celui qui vérifie que l'image retournée n'est pas égale à l'image binarisé. En effet nous retournions self dans la fonction localisation au lieu d'instancier une autre Image.

2. Variation des seuils et localisation

La variation des seuils impacte légèrement le redimensionnement localisé des images étant donné que comme vu précédemment il y a plus ou moins de pixels en fonction du seuil (Voir capture 6 et 7).



Capture 6. Exemple d'une localisation avec un seuil de 10.
Les dimensions de l'image sont 33x22.



Capture 7. Exemple d'une localisation avec un seuil de 230.
Les dimensions de l'image sont 33x24.

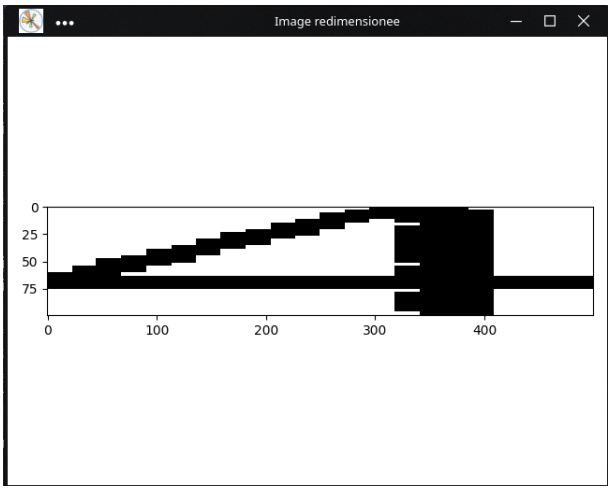
Pas de problèmes au niveau des tests.

3. Fonction resize

La fonction resize n'a pas été très dure à implémenter étant donné que le sujet nous donnait beaucoup d'indications. Il faut donc instancier un nouvel objet et utiliser la fonction resize pour redimensionner le tableau. Ensuite la fonction np.uint8 convertit les types de ce tableau en entier non signés sur 8 bits.

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

Code 4. Implémentation de la fonction resize.



Capture 8. Exemple d’une image redimensionnée.

4. Fonction similitude

Pour cette fonction, ce fut également plutôt simple. On instancie un compteur qui sera incrémenté à chaque fois que les deux images possèdent un même pixel à la même position. Ce compteur sera ensuite divisé par la dimension de l’image afin d'obtenir un chiffre en 0 et 1. Le seul problème que nous avons eu avec cette fonction concerne le redimensionnement, en effet nous avions oublié d'affecter le résultat de ce dernier à la variable ‘im’. Cette erreur d'inattention ne provoquait visiblement pas d’erreur, mais par contre elle nous donnait de mauvaises prédictions. Nous avons donc mis un peu de temps à voir d’où venait le problème.

```python
def similitude(self, im):
    # Si les deux images ne sont pas de même taille
    if not (im.W == self.W and im.H == self.H):
        # On redimensionne l'autre image
        im = im.resize(self.H, self.W)
    cpt = 0

    for i in range(self.H):
        for j in range(self.W):
            # Si les pixels de même position possèdent la même valeur
            if self.pixels[i][j] == im.pixels[i][j]:
                cpt += 1

    return cpt / (self.W * self.H)
```

Code 5. Implémentation de la fonction similitude

5. Fonction reconnaissance chiffre

Pour cette dernière fonction on commence par effectuer une binarisation et une localisation sur les images. Ensuite on appelle la fonction ‘Similitude’ sur chaque image du modèle et on stocke le résultat dans un tableau. Finalement on retourne l’index de la similitude la plus grande car les index correspondent aux chiffres.

```python
def reconnaissance_chiffre(image, liste_modeles, S):
    # Binarisation & localisation
    for model in liste_modeles:
        model = model.binarisation(S)
        model = model.localisation()

    image = image.binarisation(S)
    image = image.localisation()

    # Stockage de toutes les similitudes
    simis = [image.similitude(model) for model in liste_modeles]

    # Trouve l'index avec la plus grande similitude
    best_ind = 0
    best = 0
    for i, simi in enumerate(simis):
        if simi > best:
            best_ind = i
            best = simi

    return best_ind
```

Code 6. Implémentation de la fonction reconnaissance_chiffre

Comme nous pouvons le voir sur la capture 9, le fichier test1.jpg est bien reconnu comme étant un 4. Nous avons également fait d’autres essais qui fonctionnaient eux aussi.

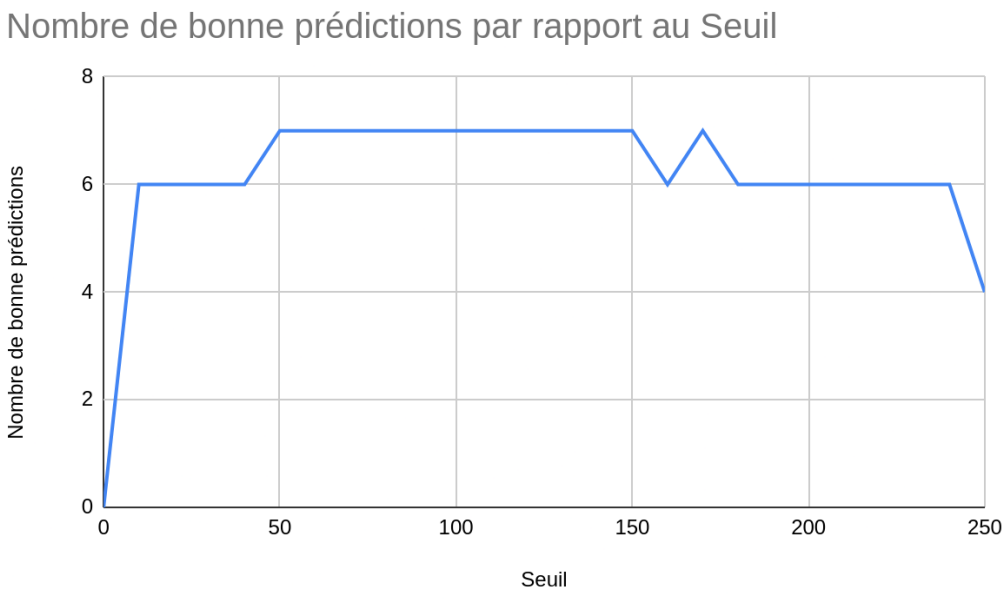
```
lecture image :  ../assets/test1.JPG (54×43)
lecture image :  ../assets/_0.png (32×22)
lecture image :  ../assets/_1.png (32×18)
lecture image :  ../assets/_2.png (32×20)
lecture image :  ../assets/_3.png (32×20)
lecture image :  ../assets/_4.png (32×24)
lecture image :  ../assets/_5.png (32×20)
lecture image :  ../assets/_6.png (32×21)
lecture image :  ../assets/_7.png (32×21)
lecture image :  ../assets/_8.png (32×22)
lecture image :  ../assets/_9.png (32×22)
Le chiffre reconnu est :  4
```

Capture 9. Exemple d'un test avec le chiffre 4 qui est bien reconnu.

6. Essai de la fonction

Pour la dernière partie nous avons simplement appelé la fonction reconnaissance de chiffre en augmentant à chaque tour de boucle le seuil et nous avons compté le nombre de bonnes prédictions totales sur les images de test.

Voici donc une représentation sous forme de graphique (voir Graphique 1), car nous pensons qu'il s'agit d'un format plus adapté et plus visuel qu'un tableau. On peut constater qu'avec un seuil de 0 on à aucune bonne réponse ce qui est normal étant donné que l'image est vide. Les meilleurs résultats sont atteints avec un seuil entre 50 et 150. Un jeu de données d'image plus important nous permettrait de mieux évaluer le seuil idéal.



Graphique 1. Nombre de bonnes prédictions par rapport au Seuil.

```
fichiers = [('test1.JPG', 4), ('test2.JPG', 1),
            ('test3.JPG', 2), ('test4.JPG', 2),
            ('test5.JPG', 2), ('test6.JPG', 4),
            ('test7.JPG', 5), ('test10.jpg', 6)]
# Chargement des images
liste_imgs = []
for fichier, nbr in fichiers:
    model = Image()
    model.load(path_to_assets + fichier)
    liste_imgs.append((model, nbr))

res = []
seuils = [*range(0, 256, 10)]
# Evolution du seuil
for s in seuils:
    l = []
    for img, c in liste_imgs:
        chiffre = reconnaissance_chiffre(img, liste_modeles, s)
        l.append(chiffre == c)
        res.append(l)
# Cumul des bonnes réponse
res_sim = []
for s in res:
    res_sim.append(sum(s))

print(seuils) # Liste des seuils
print(res_sim) # Liste des résultats
```

Code 7. Script calculant la performance de l'algorithme en fonction du seuil

III. Conclusion

Pour conclure le TP fonctionne entièrement et nous passons tous les tests. J'ai pu expliquer le fonctionnement de numpy à Calvin et nous avons tous deux appris à utiliser la librairie skimage. Les difficultés rencontrées étaient majoritairement liées à des problèmes d'algorithmique, de réflexion et de manipulation d'indice.