TP 5
k moyennes

L'algorithme des k-moyennes peut être utilisé pour réduire le nombre de couleurs d'une image sans que cela ne nuise trop à sa qualité. L'espace de couleur du système RGB de codage des couleurs consiste à représenter une couleur par trois nombres entiers compris entre 0 et 255 qui représentent les intensités respectives du rouge, du vert et du bleu dans la couleur à afficher. Cela donne 256 couleurs possibles, soit environ 16 millions. L'œil humain non-entraîné peine à distinguer autant de couleurs et il est donc possible de remplacer deux couleurs proches par une seule sans grande perte de qualité ; ce peut être utile à des fins de compression ou pour permettre l'affichage optimal d'une image sur un écran ou une imprimante n'offrant pas une grande variété de couleurs. L'algorithme des k-moyennes permet de trouver, pour chaque pixel d'une image, parmi une liste de$k$ couleurs définies, la couleur qui est la plus proche.

L'algorithme de réduction d'image nécessite plusieurs itérations. D'abord on initialise en choisissant une liste de k couleurs, et on crée une nouvelle image en remplaçant chaque pixel de l'image d'origine par la couleur dont il est le plus proche. À chaque itération, ensuite, on récupère tous les pixels d'une même couleur pour créer une partition. Pour chacune des k partitions obtenues, on calcule la moyenne des couleurs, ce qui donne une nouvelle liste de k couleurs. On remplace alors chaque pixel de l'image d'origine par la couleur dont il est le plus proche, ce permet d'obtenir une nouvelle image.

Le partitionnement des pixels puis le remplacement de leur couleur par une couleur moyenne est itéré jusqu'à ce que l'image ne soit plus modifiée par le procédé.

Les paramètres qui influent sur le résultat sont :

— la valeur de k ;
— le choix d'initialisation des couleurs ;
— l'arrêt éventuel de l'exécution avant que les couleurs ne soient complètement stabilisées.

1 Importation d'une image

On peut représenter une image (noire et blanc ou couleur) grâce à une matrice ou chaque élément a_{ij} représente une case ou un pixel de cordonnée (i, j) qui est de couleur la valeur a_{ij} qui est un vecteur de dimension 3, `np.array[r,bg]` d'entiers entre 0 et 255, (rouge, vert bleu). On importera les modules, `numpy`, `numpy. random` et `PIL` pour traiter les images.

```
>>>import numpy as np
>>> import numpy.random as rd
>>> import PIL.Image
>>> photo=PIL.Image.open('chemin.photo.png') # ouvre le fichier photo sous Pythor
# il faut rentrer l'adresse du chemin par exemple:
>>> photo=PIL.Image.open('C:\\Users\\eleve\\Documents\\Python\\photo.png')
>>> photo.show() # affiche l'image
>>> photo.size # indique la taille
(201,251)
```

Il faut, pour travailler sur une photo, d'abord la transformer en tableau (matrice). On utilise, pour cela, la bibliothèque `numpy` :

```
>>> import numpy as np
>>> phototab=np.array(photo) # Transforme la photo en tableau
>>> photo2=PIL.Image.fromarray(phototab) # Transforme un tableau en image
>>> photo2.save('photo.jpg') # Permet de sauvegarder une image sous un format au
```

Attention la commande `np.shape(phototab)` renvoie un uplet de trois entiers $(n, p, 3)$ (lignes, colonnes, couleurs).

Pour rechercher ou déplacer des fichiers avec Python : Les commandes qui suivent doivent être lancées après `import os`.

— savoir quel est le répertoire courant : `os.getcwd()`

— connaître le contenu du répertoire courant : `os.system('truc')` où `truc` est la commande système de l'O.S. [1] sous lequel **Python** est exécuté [2].

— changer de répertoire courant : `os.chdir('chemin_du_repertoire')`

On importera l'image, qu'on stockera dans `photo` , puis qu'on transformera en tableau dans la variable `phototab`.

2 Compression d'image via k_moyenne

À l'initialisation, une palette de k couleurs est choisie aléatoirement et chaque pixel est rattaché à la couleur de la palette de laquelle il est le plus proche. Lors d'une itération, on remplace chaque couleur de la palette par la «moyenne» des couleurs des pixels qui y sont rattachés. On rattache à nouveau chaque pixel à la couleur la plus proche dans la palette.

On poursuit jusqu'à la convergence, c'est-à-dire jusqu'à ce qu'une itération ne modifie plus les couleurs de la palette.

1. Programmer la fonction `dist(x,y)` prenant comme paramètres deux triplet type `array` d'entier, correspondant à deux couleurs, et renvoyant la distance Euclidienne canonique entre les deux couleurs :

$$dist(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}$$

2. Programmer la fonction `barycentre(phototab ,xi)` prenant comme paramètres la photo (sous forme de tableau), `phototab`, et une liste `xi` de pixels (c-a-d une liste de uplet (i, j)) et renvoyant la couleur moyenne, c-a-d la liste `array` des parties entières des moyennes de rouge, vert, bleu des pixels de xi.

```
def barycentre(X,xi):
    """ X tableau de couleur, xi liste de pixels de X.
    Renvoie le barycentre des X[k] pour k dans xi """
    s=np.array([0,0,0])
    ..............................
    ..........................
    return ..........
```

3. Avant de démarrer, on donne une étiquette aléatoire parmi $\{0, ...k-1\}$ à chaque pixel de l'image. La moyenne des couleurs des pixels étiquetés par 0 donne notre première moyenne initiale, la moyenne des couleurs de pixels étiquetés par 1 donne notre deuxième moyenne initiale, etc. En pratique, cela donne des moyennes très proches et «centrales» par rapport aux données. On utilisera la commande :

   `rd.randint(0,k,(n,p))` qui renvoie un tableau `numpy` aléatoire de taille $n \times p$ composé uniformément d'entier de 0à $k-1$.

   Programmer une fonction `initialisation(n,p,k)` renvoyant une liste `part` de k listes d'uplet (i, j) correspondant aux $n \times p$ pixels de l'image (donc du tabkleau) `phototab`)

```
def initialisation(n,p,k):
    """ initialisation de l'algorithme : renvoie une partition de [n*p]
    en k morceaux non vides."""
    assert n*p>k
    part=[[] for i in range(k)]
    ..................
    return part
```

1. *Operating System* ou système d'exploitation
2. ainsi, toutes les commandes systèmes peuvent être effectuées à partir de **Python**, effacement de fichiers, de répertoires, changement de répertoire, etc. : DANGER!!!!

4. Programmer une fonction `Newphoto(phototab,part)` prenant comme paramètres le tableau de l'image initiale et la liste `part` des partitions des pixels (par exemple obtenue par initialisation) et renvoyant le tableau de l'image correspondant, c-a-d le tableau dont les couleurs de chacun des pixels de `part[i]` soient la moyenne des couleurs de tous les pixels de la partition `part[i]` (on pourra appeler les fonctions précédentes).

   Tester sur l'initialisation, puis afficher l'image obtenue.

5. Programmer la fonction `plus_proche(p,M)` prenant comme paramètres une couleur `p=(r,g,b)`, la k liste des moyennes des couleurs de chaque partition et renvoyant la classe (l'indice de la partition) la plus proche de p. Par exemple si p est plus proche de la la couleur de $M[i]$ la fonction renvoie i.

6. On se lance dans le programme, à proprement parler des k moyennes. On rappelle le principe théorique de l'algorithme :

   Soit X un ensemble fini de points d'un espace euclidien, et k un entier inférieur au cardinal de X. L'algorithme consiste à trouver une partition de X en sous-ensembles $X_0, \dots, X_{k-1}$ non vides qui minimise la quantité $\sum_{i=0}^{k-1} \sum_{x \in X_i} \| x - m_i \|^2$, avec m_i le barycentre des points de X_i.

   Choisir k points qui représentent la position moyenne des partitions $m_1^{(1)}, \dots, m_k^{(1)}$ initiales (au hasard par exemple) ;

   Répéter jusqu'à ce qu'il y ait convergence :

   — affecter chaque observation à la partition la plus proche :

   $$S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\| \leq \|\mathbf{x}_j - \mathbf{m}_s^{(t)}\| \ \forall \ s = 1, \dots, k \right\},$$

   — mettre à jour la moyenne de chaque cluster :

   $$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j.$$

   Programmer la fonction `k_moyennes(phototab,k)` prenant comme paramètre le tableau de la photo originale, l'entier k du nombre de couleurs (donc de partition) choisis et renvoyant la liste des k partitions, c-a-d une liste de liste de 2-uplets correspondants aux coordonnées des pixels.

   On pourra compléter le programme suivant, on prendra garde à anticiper la condition d'arrêt :

```
def k_moyennes(X,k):
    n,p,c=np.shape(X)
    assert n>=k
    ..........................
```

   Tester le programme pour $k = 16$, 6 et 20.

7. Le choix de la distance peut conditionner le type de couleur choisit, par exemple avec :

   $$dist(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + 0.01(x_3 - y_3)^2}$$

   Consiste à donner moins d'importance au bleu et donc tourner le bleu en gris. Faites un test et amusez vous.

8. Le choix de k est prépondérant, un choix inadéquat de k, correspond à un sous/sur apprentissage. Mais comment savoir quel k choisir, si on ignore k à l'avance ? Un moyen empirique consiste à mesure pour différentes valeur de k, l'écart (ou la proportion) de changement entre l'image initiale et la nouvelle image. par exemple :

   $$R(k) = \frac{1}{np} \sum_{i=0}^{n} \sum_{i=1}^{p} \| X[i, j] - Xk[i, j] \|^2$$

   où X est le tableau de l'image initiale et Xk celle obtenues par les k moyennes.

   Proposer une représentation graphique de $R(k)$ pour des entiers k de 5 à 30.