PIXELISATION D'UNE IMAGE SELON UNE PALETTE DE 18 COULEURS PREDEFINIES

ROCHER Julie et LIXI Valentin

I. Simplification de l'image

La première ligne de code charge dans notre programme le module PIL. Il va nous servir à importer l'image choisie avec :

img=Image.open(«»)

cette variable va permettre de définir la hauteur et la largeur de l'image :

l=img.width

h=img.height

On cherche dans un premier temps à simplifier l'image en fusionnant des groupes de 4 pixels pour n'en former qu'1 seul.

Pour ce faire, il faut que la hauteur ainsi que la largeur de l'image soient des multiples de 4 : on va supprimer entre 0 et 3 lignes et/ou colonnes de pixels pour y parvenir (c'est très peu, ça ne changera pas l'image).
Nous devons ensuite calculer la moyenne des composantes R,G,B des 4 pixels pour en former qu'un seul avec R=(R1+R2+R3+R4)/4, G=(G1+G2+G3+G4)/4 et B=(B1+B2+B3+B4)/4, arrondis à l'unité. La nouvelle image apparaîtra 4 fois plus petite.

II. Colorisation de l'image

Après avoir comprimé l'image, le but est de réduire le nombre de couleurs qui la composent, en la recolorisant avec une palette limitée de 18 couleurs prédéfinies (couleurs et nombre de couleurs modifiable), pour pouvoir l'afficher dans certains cas ou le nombre de couleurs serait limité.

Pour pouvoir utiliser dans notre programme les 18 couleurs ci-dessous, on les a enregistrées sous la forme de listes allant de c1 à c18, contenant chacune 3 valeurs qui correspondent à leurs composantes R,G,B.

C1	C2	C3	C4	C5	C6
C7	C8	C9	C10	C11	C12
C13	C14	C15	C16	C17	C18

Il s'agit maintenant de trouver comment comparer la couleur d'un pixel p à toutes les couleurs de la palette afin de choisir les meilleures composantes R,G,B à enregistrer dans p. Nous avons réfléchi à 2 moyens :

1) En calculant la distance euclidienne, sans bibliothèque

La première idée pour essayer de comparer un pixel p a toutes les couleurs de la palette serait de calculer la distance euclidienne entre les valeurs R,G,B du pixel et celles de chaque couleur.

Si on calcule la distance euclidienne entre p et C1, on a :

*d1=math.sqrt((C1[0]-r)**2+(C1[1]-v)**2+(C1[2]-b)**2)*

Pour répéter cette opération pour chaque couleur, on pourrait tenter de créer une seconde image de 6x3 pixels, où chaque pixel contiendrait les composantes R,G,B d'une couleur (comme le tableau ci-dessus). Cette sorte de tableau permettrait de créer une boucle avec :

for y in range(3):

for x in range(6):

en comparant le pixel p à chaque pixel de cette image.

Il faut ensuite comparer toutes les distances euclidiennes d, prendre la plus basse et remplacer les composantes R,G,B du pixel p par celles de la couleur C associée à d.
On ne sait pas encore comment s'y prendre.

2) En calculant la métrique de distance Delta-E, avec la bibliothèque colormath et le CIE Chromaticity Diagram

Bien que les valeurs RGB soient un moyen pratique de représenter les couleurs sur ordinateur, nous ne percevons pas les couleurs de la même manière qu'eux. Ce qui nous semble être des couleurs identiques peut nous donner une distance euclidienne supérieure à la distance euclidienne des couleurs qui nous semblent différentes.

Nous allons donc utiliser la métrique de distance Delta-E qui utilise l'espace colorimétrique CIE Lab qui perçoit les couleurs d'une façon beaucoup plus 'humaine'.
Les commandes dont on va avoir besoin se trouvent dans la bibliothèque colormath.

Il suffit ensuite de réécrire les mêmes lignes de code que lorsqu'on calcule la distance euclidienne, c'est-à-dire :
créer un tableau de 6x3 pixels, trouver les métriques de distance Delta-E, les comparer, prendre la plus basse et remplacer les composantes R,G,B du pixel p par celles de la couleur C associée à delta_e.