Programme principal

Il s'agit maintenant d'articuler toutes les «briques» de notre programme.

Étapes-clés

- 1. Chaque groupe doit disposer:
 - d'une image de base (disponibles sur Moodle);
 - d'un enregistrement sonore (disponibles sur Moodle);
 - d'une vidéo et de l'audio extrait de cette vidéo (disponibles sur https://methotapes.com/BILANS_CLIMATIQUES/uploads/video/);
- 2. On commencera par créer 4 variables contenant les noms de ces fichiers.
- 3. Le protocole de création du GIF comporte 25 itérations du même procédé:
 - extraire une image de la vidéo, puis sa couleur dominante;
 - extraire les données du son de l'enregistrement et du son de la vidéo (qui en fait se font au préalable);
 - extraire la zone de l'image de base;
 - · lui appliquer le filtre
 - · enregistrer l'image
- 4. Une fois les 25 images créées, lancer la création du GIF:
 - redimensionner les 25 images à la bonne taille 500x500;
 - créer le GIF!

6 Redimensionnement des images

Le logiciel (Open-source) ImageMagick permet de faire à peu près tout ce qu'on veut comme retouche d'images... en ligne de commande dans un terminal.

Par exemple, pour redimensionner une image monimage.png en créant une nouvelle image:

Bash convert -resize 500x500 monimage.png monimageredimensionnee.png

Ou bien en écrasant l'image:

Bash mogrify -resize 500x500 monimage.png

b Shell et Python

En Python, on peut exécuter une ligne de commande du shell depuis un programme. Pour cela il faut utiliser le module os et la fonction system:

```
1 import os
2 os.system('la commande en chaine de caractères')
```

On s'en servira pour:

- extraire l'audio de la vidéo avec le logiciel ffmpeg
- redimensionner avec ImageMagick les images générées en 500x500 pixels pour ensuite les ajouter au GIF.

Voici le fichier/module composé de vos fonctions qu'il faudra importer dans le programme principal.

```
1
     import scipy.io.wavfile as wave
 2
     import math
 3
     import numpy
 4
     import imageio
 5
     6
 7
 8
     def spectre(data: list, rate: int, debut: float, duree: float) -> list:
        1.1.1
 9
        Renvoie le spectre correspondant à un intervalle du signal.
10
11
12
        data: le signal d'un canal
13
        rate: la fréquence d'échantillonnage
14
        debut: le début de l'intervalle à étudier (en secondes)
15
        duree: la durée de l'intervalle à étudier (en secondes)
        1.1.1
16
17
        start = int(debut * rate)
        stop = int((debut+duree) * rate)
18
19
        s = numpy.absolute(numpy.fft.fft(data[start:stop]))
20
        s = s / s.max()
21
        return [math.log10(i) for i in s if i != 0]
22
23
     def volumes_min(son)->list: #Mélodie
24
        1.1.1
25
26
        Prend en paramètre un son qui correspond
27
        et renvoie une liste de 25 valeurs qui
28
        correspondent aux volume minimaux de
29
        chaque 1/25eme de la durée totale du son.
        1.1.1
30
31
        rate, echantillon = wave.read(son)
32
        cd = [elt[1] for elt in echantillon]
        duree = len(echantillon)/rate
33
34
        duree_intervalle = duree//25
35
        depart = 0.0
36
        volumes_mini = []
37
        for k in range (25):
38
             s = spectre(cd, rate, depart, duree_intervalle)
39
            volumes_mini.append(min(s))
40
            depart += duree_intervalle
41
         return volumes_mini
42
43
     def pourcentage (lst:list)-> list: #Mélodie
         1 \ 1 \ 1
44
45
         Prends en paramètre la liste des volumes minimaux
46
         et renvoie la liste contenant les pourcentages de
47
        chaque point dans l'intervalle minimal-maximal de
48
        la liste de volumes minimaux.
         1.1.1
49
50
        lst_pourcentage = []
51
        maxi = max(lst)
52
        mini = min(lst)
53
        for elt in lst:
             p = ((elt-mini)/(maxi-mini))*100
54
55
             lst_pourcentage.append(int(p))
56
         return lst_pourcentage
57
```

```
58
      59
 60
      # Extraction de la zone
 61
 62
      def extraction_zone(img: list, S: int, Vson: float, Vvideo: float) -> list: #Manon
 63
 64
         Renvoie une image constituée des pixels de l'image img, de taille W avec le décalage x, y
 65
         donnés par les paramètres S, Vson et Vvideo.
 66
 67
         W = int(500 * S/100)
 68
         x = int((500-W) * Vson)
 69
         y = int((500-W) * Vvideo)
 70
         #On créait une image vide pour pouvoir y placer la zone extraite
 71
         zone = numpy.zeros((W, W, img.shape[2]), dtype=numpy.uint8)
 72
         for i in range(W):
 73
             for j in range(W):
 74
                 zone[i][j] = img[i+y][j+x]
 75
         return zone
 76
 77
      def extraction(image_AP: list, S: int, Vson: float, Vvideo: float) -> list: #Jules
 78
 79
 80
         Créer une image vide, sélectionne une zone dans l'image
 81
 82
          en paramètre et copie les pixels de la zone dans l'image vide.
 83
 84
 85
         W = int(500 * S/100)
 86
         x = int((500-W) * Vson)
 87
         y = int((500-W) * Vvideo)
 88
 89
         zone = numpy.zeros([W,W,3], dtype=numpy.uint8)
 90
 91
         for i in range(W):
 92
             for j in range(W):
 93
                 zone[i][j] = image\_AP[y+i][x+j]
 94
 95
         imageio.imsave("monimage.jpg", zone)
 96
          return zone
 97
 98
      # Recherche de la couleur dominante
 99
100
      def color_dom(image): #Yanis
101
102
         # Cherche le pixel de l'image img qui revient le plus grand nombre de fois dans img.
103
         hauteur = image.shape[0]
104
         largeur = image.shape[1]
105
         pixels = {}
         for i in range(hauteur):
106
             for j in range(largeur):
107
                 if tuple(image[i][j]) in pixels:
108
109
                     pixels[tuple(image[i][j])] += 1
                 else:
110
                     pixels[tuple(image[i][j])] = 1
111
112
         # recherche du pixel ayant la plus grande occurence
113
         grand = ''
114
         nmax = 0
115
116
         for k, v in pixels.items():
117
             if v > nmax:
118
                 grand = k
119
                 nmax = v
120
         return grand
121
122
      def couleur_dominante(image) -> tuple: # Manon
```

```
# On parcous l'image pour savoir combien de fois les couleurs apparaîssent dans le tableu.
123
          dico_RGB = {}
124
125
          for i in range(img_test.shape[0]):
             for j in range(img_test.shape[1]):
126
127
          # On utilise un tuple pour qu'il soit possible de parcourir l'image.
128
                  t = tuple(img_test[i][j])
129
                  if t in dico_RGB:
130
                      dico_RGB[t] += 1
131
                  else:
                      dico_RGB[t] = 1
132
133
          # On recherche la couleur qui revient le plus.
134
135
          clr_max_apparition = 0
          for clr, n in dico_RGB.items():
136
              if n > clr_max_apparition:
137
138
                  clr_max_apparition = n
                  couleur = clr
139
140
141
          return couleur
142
143
      # Application du filtre
144
145
146
      la fonction f permet de superposer une teinte sur une couche
147
      def f(a,b): #Florette
148
149
         a=a/255
150
          b=b/255
151
         if a < 0.5:
152
             r= 2*a*b
153
          else:
154
             r = 1-2*(1-a)*(1-b)
155
          return int(r*255)
156
157
158
          Renvoie une nouvelle image, donnée par l'application d'une filtre coloré à l'image de départ
159
160
161
      def filtre(img : list, couleur: tuple): #Florette
162
          img_filtree = numpy.zeros((img.shape[0],img.shape[1],3), dtype=numpy.uint8)
163
          for i in range(img.shape[0]):
164
              for j in range(img.shape[1]):
165
                  r=img[i][j][0]
166
                  g=img[i][j][1]
167
                  b=img[i][j][2]
168
                  img_filtree[i][j] = (f(r,couleur[0]), f(g,couleur[1]), f(b,couleur[2]))
169
          imageio.imsave("imageavecfiltre.jpg", img_filtree)
170
          return img_filtree
171
172
      def f(a, b): #Dawson
173
          a = a / 255
174
          b = b / 255
175
          if a < 0.5:
              r = 2 * a * b
176
177
              r = 1-2*(1-a)*(1-b)
178
179
          return int(r*255)
180
181
182
      def filtre(img: list, couleur: tuple): #Dawson
183
          Renvoie une nouvelle image, donnée par l'application d'une filtre coloré à l'image de départ
184
185
      ima.
          1.1.1
186
187
          img_filtree = numpy.zeros((img.shape[0], img.shape[1],3), dtype=numpy.uint8)
```

```
188
      for i in range(img.shape[0]):
189
             for j in range(img.shape[1]):
190
                 r = img[i][j][0]
                  g = img[i][j][1]
191
192
                  b = img [i][j][2]
                  img\_filtree[i][j] = [f(r,couleur[0]), \ f(g,couleur[1]), \ f(b,couleur[2])]
193
194
         return img_filtree
195
196
      # Conversion RGB -> HSL
197
198
199
      def convertisseur (r,g,b :int)-> int: #Safia
200
201
         Convertit la couleur RGB en HSL
202
203
         r, g, b = r/255, g/255, b/255
204
         Cmax=max(r,g,b)
205
         Cmin=min(r,g,b)
         Cdiff=Cmax-Cmin
206
207
         if Cmax = = 0:
208
             T=0
         elif Cmax == r:
209
210
             T=(((g-b)/Cdiff)) \% 6
         elif Cmax == g:
211
            T = (((b-r)/Cdiff) + 2) \% 6
212
213
         elif Cmax == b:
214
             T = (((r-g)/Cdiff) + 4) \% 6
         if Cmax == 0:
215
216
             T = 0
         t=60*T
217
218
          L=1/2*(Cmax+Cmin)
219
         if L==1:
220
             S=0
221
          else:
             S=Cdiff/(1-abs(2*L-1))
          return t, S*100, L*100
```

v

```
🐁 Script Python
```

```
# Import des modules
1
 2
3
4
    # Chargement des fichiers de données
5
6
    video = 'data/video_station1_groupe5.mp4'
7
    son = 'son_station1_groupe5.wav'
    image = imageio.imread('images/Gr5_Sp1_Florette2.jpg')
8
    audio = 'audio.wav'
9
10
    # Extraction du son de la video
11
12
    os.system('ffmpeg -i ' + + ' ' + )
13
14
    # Extraction des volumes du son et de l'audio
15
16
17
    v_son =
18
    v_video =
19
    # Création du reader de la vidéo
20
21
    reader = imageio.get_reader(video)
22
23
     # Boucle principale
24
25
26
     for k in range(25):
         # extraction de l'image de la video
27
28
         # extraction de la couleur dominante, convertie en hsl
29
30
31
         # extraction de la zone
32
         # application du filtre
33
34
35
         # enregistrement de l'image
36
37
38
    # Création du GIF
    with imageio.get_writer('GIF_ultime.gif', mode='I') as writer:
39
40
        for k in range(25):
41
             #redimensionnement de l'image en ligne de commande
42
43
             #lecture de la k-ième image du gif
44
45
             # ajout de l'image au writer
```