Images_coloration_python

October 1, 2021

[ ]:
```
%reset -f
```

[ ]:
```
# PART 1 - IMPORTING PYTHON LIBRARIES AND MOUNTING DRIVE

import cv2
from google.colab.patches import cv2_imshow
import numpy as np
from numpy import genfromtxt
import pandas as pd
import csv
import os, math
from scipy import ndimage
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from google.colab import drive

#Mounting Google Drive to access files
drive.mount('/content/drive', force_remount=False)
```

```
Mounted at /content/drive
```

[ ]:
```
# PART 2 - LOADING IMAGE

path_base = '/content/drive/MyDrive/Internship Burapha University/#Urban␣
↪Cooling - Manuel/6. Results/Image_1'

# Loading canals and forests positions from csv file
px_to_remove = 32 #Cropping matrix
mat_position_to_crop = np.loadtxt(open(path_base + '/4. Combinated images and␣
↪matrixes/Image_1_combination_clean.csv', "rb"), delimiter=",", skiprows=1)
mat_position = mat_position_to_crop[0:len(mat_position_to_crop)-px_to_remove, 0:
↪len(mat_position_to_crop[0])-px_to_remove]
rows_number, columns_number = len(mat_position), len(mat_position[0])

# Re-associating the correct number to each class knowing 1 = city, 2 = canal,␣
↪3 = rest
mat_position[mat_position == 2] = 1
```

```
mat_position[mat_position == 6] = 2
mat_position[mat_position == 4] = 3
mat_position[mat_position == 0] = 1

# Loading image
original_img = cv2.imread(path_base + '/1. Original Image/Image 1.jpg')[0:
↪rows_number, 0:columns_number]
imgR = cv2.imread(path_base + '/1. Original Image/Image 1.jpg')[0:rows_number,␣
↪0:columns_number]
imgG = cv2.imread(path_base + '/1. Original Image/Image 1.jpg')[0:rows_number,␣
↪0:columns_number]
imgB = cv2.imread(path_base + '/1. Original Image/Image 1.jpg')[0:rows_number,␣
↪0:columns_number]
width,height,nb_colors = original_img.shape
img = np.zeros((width,height,nb_colors))
index_save = 0 #index to save results at the end of this code without erasing␣
↪previous results

# Extraction of R G B channels
# Set blue and green channels to 0
imgR[:,:,0] = 0         # X,Y,Z     # 0=blue, 1=green, 2=red
imgR[:,:,1] = 0

# Set blue and green channels to 0
imgG[:,:,0] = 0
imgG[:,:,2] = 0

# Set red and green channels to 0
imgB[:,:,1] = 0
imgB[:,:,2] = 0

# Image reconstrution
img[:,:,0] = imgB[:,:,0]
img[:,:,1] = imgG[:,:,1]
img[:,:,2] = imgR[:,:,2]

# Creating a folder for the results and defining the path
path_result = path_base + '/5. Post-Processed images'
list = os.listdir(path_result)
nb_folders = len(list) #We take a look at the number of tests already existing␣
↪in the destination folder
folder_name = 'Display_'+ str(nb_folders+1) #We assign a new number to the␣
↪current test
index = 1
os.mkdir(path_result + '/' + folder_name)
path_display = path_result + '/' + folder_name
```

```python
# Saving RGB channels into the previous folder
dR = pd.DataFrame(imgR[:,:,2])
dR.to_csv(path_display + '/R.csv',header=False, index=False)
dG = pd.DataFrame(imgG[:,:,1])
dG.to_csv(path_display + '/G.csv',header=False, index=False)
dB = pd.DataFrame(imgB[:,:,0])
dB.to_csv(path_display + '/B.csv',header=False, index=False)

# Plot original image and RGB channels just to show we can sucessfully extract␣
↪them
cv2_imshow(original_img)
cv2_imshow(imgR)
cv2_imshow(imgG)
cv2_imshow(imgB)
#cv2_imshow(img)
cv2.waitKey()

print('Matrix dimensions =', rows_number, 'x', columns_number)

# Plot prediction matrix
my_data_to_crop = genfromtxt(path_base + '/4. Combinated images and matrixes/
↪Image_1_combination_clean.csv', delimiter=',')
my_data  = my_data_to_crop[0:rows_number, 0:columns_number]
my_data[my_data == 2] = 1
my_data[my_data == 6] = 2
my_data[my_data == 4] = 3
my_data[my_data == 0] = 1
plt.imshow(my_data, interpolation='nearest')
plt.show()
```

[ ]:
```python
#PART 3 - URBAN COOLING SCIENTIFIC DATA USED IN THIS STUDY

#Canals temperature diminution:
#0-30m: 1.2°C

#Forests temperature diminution:
#Remaining forest:          #Street trees:              #Isolated trees:
#(more than 5 trees)        #(from 2 to 5 trees)        #(0 to 2 trees)
#0-50:     1.32°C           #0-50:     1.01°C           #0-50:      0.58°C
#50-100:   1.14°C           #50-100:   0.94°C           #50-100:    0.52°C
#100-150:  0.57°C           #100-150:  0.73°C           #100-150:   0.22°C
#150-200:  0.30°C           #150-200:  0.24°C           #150-200:   0.02°C
#200-250:  0.16°C           #200-250:  0°C              #200-250:   0°C
#250-300:  0.13°C           #250-300:  0°C              #250-300:   0°C
#Average reception of carbon dioxide (CO2) per tree: 21 kg/year
```

```
#Average tree surface in Bangkok: 60.73m^2 based on a study on the three most␣
↪common tree species in Bangkok:
#Polyalthia longifolia Sonn          (representing 15.7% of Bangkok trees) ->␣
↪Average diameter = 2.7432m -> Average surface =   5.91m^2
#Mangifera indica L.                 (representing 13.0% of Bangkok trees) ->␣
↪Average diameter =     10m -> Average surface =  78.74m^2
#Pithecellobium dulce (Roxb.) Benth.  (representing 5.4%  of Bangkok trees) ->␣
↪Average diameter =     15m -> Average surface = 176.71m^2
#Considering these are the only species of trees in Bangkok, 15.7% becomes 46.
↪04%, 13.0% becomes 38.12%, 5.4% becomes 15.84%
#Average surface of a tree in Bangkok according to this data: ((46.04/100)*5.
↪91)+((38.12/100)*78.74)+((15.84/100)*176.71) = 60.727516 m^2

# Defining sizes of "circles" to color around forest and canal areas respecting␣
↪ri+1 > ri
R_rf = np.array([50, 100, 150, 200, 250, 300],ndmin=2)    #"rf" for remaining␣
↪forest
R_st = np.array([50, 100, 150, 200, 250, 300],ndmin=2)    #"st" for street trees
R_it = np.array([50, 100, 150, 200, 250, 300],ndmin=2)    #"it" for isolated␣
↪trees
R_ca = np.array([30],ndmin=2)                             #"ca" for canals

# Defining temperature diminution influence of each forest and canal "circle"
T_rf = np.array([1.32, 1.14, 0.57, 0.30, 0.16, 0.13],ndmin=2)
T_st = np.array([1.01, 0.94, 0.73, 0.24, 0.00, 0.00],ndmin=2)
T_it = np.array([0.58, 0.52, 0.22, 0.02, 0.00, 0.00],ndmin=2)
T_ca = np.array([1.2],ndmin=2)

# Defining the average tree surface and its carbon reception
surface_tree = 60.73 #Tree surface in m^2
carbon_tree = 21      #Reception of carbon dioxide (CO2) per year and per tree

# Defining a scale between meters and pixels
scale = 1668/200 #200m = 1668 pixels

print('Done')
```

Done

[ ]:
```
# PART 4 - FINDING CLUSTERS

# Finding clusters in position matrix
def find_clusters(array, width, height):
    clustered = np.empty_like(array)
    unique_vals = np.unique(array)
    cluster_count = 0
    print(unique_vals)
```

```python
    labelling = np.zeros((len(unique_vals),width,height)) #labelling = 3D array, each level [n,:,:] containing all pixel positions and associated cluster number of class number n
    #City class number = 1, Canal class number = 2, Forest class number = 3
    for val in unique_vals:
        labelling[int(val-1)], label_count = ndimage.label(array == val)
        for k in range(1, label_count + 1):
            clustered[labelling[int(val-1)] == k] = cluster_count
            cluster_count += 1
    return clustered, cluster_count, labelling, len(unique_vals)

clusters, cluster_count, cluster_position, val_number = find_clusters(mat_position, width, height)
print("Found {} clusters:".format(cluster_count))
ones = np.ones_like(mat_position, dtype=int)
cluster_sizes = ndimage.sum(ones, labels=clusters, index=range(cluster_count)).astype(int)
com = ndimage.center_of_mass(ones, labels=clusters, index=range(cluster_count))

# Creating a matrix to collect information about the size and position of center of the different clusters #cluster_count = nb of clusters
Clust = np.zeros((cluster_count,4))
for i, (size, center) in enumerate(zip(cluster_sizes, com)):
    print("Cluster #{}: {} elements at {}".format(i, size, center))
    Clust[i,0] = i
    Clust[i,1] = round(size)
    Clust[i,2] = round(center[0]) #horizontal coordinate of center of cluster
    Clust[i,3] = round(center[1]) #vertical coordinate of center of cluster
Clust = Clust.astype(int)

# Saving pixel position inside the clusters under csv matrixes
for l in range(0, val_number):
    dcluster_position = pd.DataFrame(cluster_position[l,:,:])
    dcluster_position.to_csv(path_display + '/cluster_position_' + str(l+1) + '.csv',header=False, index=False)

# Creating a matrix with the same dimension of the prediction matrix with the number of the corresponding cluster on each pixel
which_clust = np.zeros((width,height))

# Creating an array with the number of cluster for dimension to explicit what class is associated to this cluster
max_city = int(np.amax(cluster_position[0,:,:]))
max_canal = int(np.amax(cluster_position[1,:,:]))
max_forest = int(np.amax(cluster_position[2,:,:]))
max_tot = max_city + max_canal + max_forest
```

```python
which_class = np.zeros((max_tot))
print('\nThere are:\n',max_city,'cluster(s) of city\n', max_canal,'cluster(s) of canals\n', max_forest,'cluster(s) of forests')

# Filling the previous two matrixes
for a in range(1, max_city+1):
  which_clust[cluster_position[0,:,:] == a] = a-1
  which_class[a-1] = 1 #city
for b in range(1, max_canal+1):
  which_clust[cluster_position[1,:,:] == b] = max_city+b-1
  which_class[max_city+b-1] = 2 #canal
for c in range(1, max_forest+1):
  which_clust[cluster_position[2,:,:] == c] = max_city+max_canal+c-1
  which_class[max_city+max_canal+c-1] = 3 #forest

# Adding to the Clust matrix the associated class number to each cluster and displaying this matrix
which_class = which_class.astype(int)
Clust = np.concatenate((Clust,which_class[:,None]),axis=1)
print('\nThe following matrix shows the number of the cluster, its size, its center x and y positions and its associated class:\n', Clust)
print('assuming 1 = city, 2 = canal, 3 = forest')

# Saving the matrix with the number of the corresponding cluster on each pixel
dwhich_clust = pd.DataFrame(which_clust)
dwhich_clust.to_csv(path_display + '/which_clust.csv',header=False, index=False)

# For Part 6, defining which temperature diminution value to use according to the forest area size (isolated tree or street trees or remaining forests)
which_coef = np.ones((width, height))

# Studying the size of the clusters
for k in range(0, cluster_count):
  if (Clust[k,4] == 3): #If the cluster is a cluster of forest
    clust_size_sqrt = round((200/1668)*np.sqrt(Clust[k,1]))
    clust_size = np.square(clust_size_sqrt)
    surface_tree = 60.73 #Tree surface in m^2
    nb_tree = float(round(10*float(clust_size/surface_tree))/10) #Truncating the float number to 1 significant digit

    # Deciding which temperature diminution value to use for each pixel depending of the size of its cluster
    if (nb_tree > 5):
      which_coef[which_clust == k] = 1
    if (2 <= nb_tree <= 5):
      which_coef[which_clust == k] = 2
```

```
    if (0 < nb_tree < 2):
      which_coef[which_clust == k] = 3
which_coef = which_coef.astype(int)

# Saving the previous matrix
dwhich_coef = pd.DataFrame(which_coef)
dwhich_coef.to_csv(path_display + '/which_coef.csv',header=False, index=False)
```

```
[1. 2. 3.]
Found 63 clusters:
Cluster #0: 2067678 elements at (664.1320998724173, 1447.2088835882569)
Cluster #1: 3 elements at (0.0, 598.0)
Cluster #2: 3 elements at (0.0, 606.0)
Cluster #3: 3 elements at (14.0, 598.0)
Cluster #4: 3 elements at (14.0, 606.0)
Cluster #5: 2 elements at (15.0, 0.5)
Cluster #6: 2 elements at (15.0, 12.5)
Cluster #7: 1 elements at (18.0, 63.0)
Cluster #8: 2 elements at (20.0, 0.5)
Cluster #9: 2 elements at (20.0, 12.5)
Cluster #10: 1 elements at (22.0, 63.0)
Cluster #11: 2 elements at (25.0, 12.5)
Cluster #12: 2 elements at (30.0, 12.5)
Cluster #13: 2 elements at (57.0, 48.5)
Cluster #14: 2 elements at (62.0, 48.5)
Cluster #15: 2 elements at (79.5, 2527.0)
Cluster #16: 2 elements at (84.5, 2527.0)
Cluster #17: 3 elements at (93.0, 2527.0)
Cluster #18: 256 elements at (342.5, 231.5)
Cluster #19: 6 elements at (383.5, 759.0)
Cluster #20: 6 elements at (383.5, 766.0)
Cluster #21: 6 elements at (397.5, 759.0)
Cluster #22: 6 elements at (397.5, 766.0)
Cluster #23: 1 elements at (399.0, 996.0)
Cluster #24: 1 elements at (399.0, 1000.0)
Cluster #25: 1 elements at (399.0, 1004.0)
Cluster #26: 7430 elements at (506.3520861372813, 2554.7600269179)
Cluster #27: 6 elements at (494.0, 2482.5)
Cluster #28: 7448 elements at (594.3088077336198, 2400.623523093448)
Cluster #29: 9 elements at (563.0, 2303.0)
Cluster #30: 3 elements at (587.0, 2303.0)
Cluster #31: 3 elements at (593.0, 2303.0)
Cluster #32: 4 elements at (623.5, 288.5)
Cluster #33: 4 elements at (629.5, 288.5)
Cluster #34: 4 elements at (635.5, 288.5)
Cluster #35: 4 elements at (641.5, 288.5)
Cluster #36: 7 elements at (690.0, 1583.0)
Cluster #37: 60 elements at (808.5, 333.0)
```

```
Cluster #38: 4 elements at (816.5, 176.0)
Cluster #39: 4 elements at (823.5, 176.0)
Cluster #40: 58232 elements at (1063.372990795439, 97.65153180381921)
Cluster #41: 4 elements at (943.5, 208.5)
Cluster #42: 4 elements at (951.5, 208.5)
Cluster #43: 4 elements at (957.5, 208.5)
Cluster #44: 4 elements at (963.5, 208.5)
Cluster #45: 2 elements at (968.0, 208.5)
Cluster #46: 2 elements at (974.0, 208.5)
Cluster #47: 1 elements at (1186.0, 378.0)
Cluster #48: 1 elements at (1186.0, 384.0)
Cluster #49: 326384 elements at (647.1230238001863, 202.40672030491692)
Cluster #50: 426687 elements at (246.97639487493174, 1018.0501772962382)
Cluster #51: 70 elements at (113.0, 184.5)
Cluster #52: 256 elements at (182.5, 1511.5)
Cluster #53: 10143 elements at (663.4031351671103, 1568.675145420487)
Cluster #54: 1536 elements at (910.5, 1767.5)
Cluster #55: 256 elements at (1014.5, 1847.5)
Cluster #56: 17364 elements at (93.1846348767565, 2270.173923059203)
Cluster #57: 189722 elements at (464.0891778496959, 2457.735945225119)
Cluster #58: 512 elements at (542.5, 2599.5)
Cluster #59: 10496 elements at (1124.5487804878048, 1151.3048780487804)
Cluster #60: 1040 elements at (1186.2230769230769, 631.6538461538462)
Cluster #61: 772 elements at (1190.6088082901554, 279.9300518134715)
Cluster #62: 512 elements at (1190.5, 1919.5)

There are:
 49 cluster(s) of city
 7 cluster(s) of canals
 7 cluster(s) of forests

The following matrix shows the number of the cluster, its size, its center x and
y positions and its associated class:
 [[      0 2067678     664    1447       1]
 [      1       3       0     598       1]
 [      2       3       0     606       1]
 [      3       3      14     598       1]
 [      4       3      14     606       1]
 [      5       2      15       0       1]
 [      6       2      15      12       1]
 [      7       1      18      63       1]
 [      8       2      20       0       1]
 [      9       2      20      12       1]
 [     10       1      22      63       1]
 [     11       2      25      12       1]
 [     12       2      30      12       1]
 [     13       2      57      48       1]
 [     14       2      62      48       1]
```

```
[      15       2      80    2527        1]
[      16       2      84    2527        1]
[      17       3      93    2527        1]
[      18     256     342     232        1]
[      19       6     384     759        1]
[      20       6     384     766        1]
[      21       6     398     759        1]
[      22       6     398     766        1]
[      23       1     399     996        1]
[      24       1     399    1000        1]
[      25       1     399    1004        1]
[      26    7430     506    2555        1]
[      27       6     494    2482        1]
[      28    7448     594    2401        1]
[      29       9     563    2303        1]
[      30       3     587    2303        1]
[      31       3     593    2303        1]
[      32       4     624     288        1]
[      33       4     630     288        1]
[      34       4     636     288        1]
[      35       4     642     288        1]
[      36       7     690    1583        1]
[      37      60     808     333        1]
[      38       4     816     176        1]
[      39       4     824     176        1]
[      40   58232    1063      98        1]
[      41       4     944     208        1]
[      42       4     952     208        1]
[      43       4     958     208        1]
[      44       4     964     208        1]
[      45       2     968     208        1]
[      46       2     974     208        1]
[      47       1    1186     378        1]
[      48       1    1186     384        1]
[      49  326384     647     202        2]
[      50  426687     247    1018        2]
[      51      70     113     184        2]
[      52     256     182    1512        2]
[      53   10143     663    1569        2]
[      54    1536     910    1768        2]
[      55     256    1014    1848        2]
[      56   17364      93    2270        3]
[      57  189722     464    2458        3]
[      58     512     542    2600        3]
[      59   10496    1125    1151        3]
[      60    1040    1186     632        3]
[      61     772    1191     280        3]
[      62     512    1190    1920        3]]
```

assuming 1 = city, 2 = canal, 3 = forest

```
[ ]: # PART 5 -  CREATING MATRIXES FOR FORESTS AND CANALS CONTAINING ALL "CIRCLES"␣
     ↪INFORMATION (SIZE, COLOR, ASSOCIATED TEMPERATURE DIMINUTION)

     # Color definition for display using rgb code #f means forest, c means canal
     c0r_c, c0g_c, c0b_c = 0, 76, 153 #blue  rgb code
     c0r_f, c0g_f, c0b_f = 0, 102, 0   #green  rgb code

     # Color definition for shading
     inicolor_f = [255,215,0]  #Initial RGB color #[255,215,0]  = Yellow
     fincolor_f = [255,127,80] #Final RGB color   #[255,127,80] = Orange
     inicolor_c = [218,165,32] #Initial RGB color #[218,165,32] = Golden yellow
     fincolor_c = [184,184,11] #Final RGB color   #[184,184,11] = Dark golden yellow

     # Creating a matrix R_f for forest "circles" sizes information
     R_f_meters_T = np.concatenate((R_rf,R_st,R_it))
     R_f_T = np.matrix.round(R_f_meters_T*scale) #Conversion meters to pixels
     R_f = np.matrix.transpose(R_f_T)

     # Creating a matrix R_c for canal "circles" sizes information
     R_c_meters_T = R_ca
     R_c_T = np.matrix.round(R_c_meters_T*scale) #Conversion meters to pixels
     R_c = np.matrix.transpose(R_c_T)

     # Creating a matrix T_f for forest temperature diminution information
     T_f_T = np.concatenate((T_rf,T_st,T_it))
     T_f = np.matrix.transpose(T_f_T)

     # Creating a matrix T_c for canal temperature diminution information
     T_c_T = T_ca
     T_c = np.matrix.transpose(T_c_T)
     #print(R_f,'\n\n',T_f,'\n\n',R_c,'\n\n',T_c)

     # Defining the number of tree categories (remaining forest, street trees,␣
     ↪isolated trees)
     nb_tree_types = len(R_f[0])

     # Number of outlines for forests and canals areas = number of steps for shading
     step_f = len(R_f)
     step_c = len(R_c)

     # Creating matrixes that we will use in a loop to create the shading
     shade_f = np.zeros((3,1))                #3 rows because 3 channels (RGB)
     mat_shade_f_T = np.zeros((3,step_f))
     shade_c = np.zeros((3,1))
     mat_shade_c_T = np.zeros((3,step_c))
```

```
# Loop to shade step by step the "circles" color from original color to final␣
 ↪color
if (step_f > 1): #Loop for forest circles
  for i in range(0,3): #0=R, 1=G, 2=B
    if (inicolor_f[i]  >= fincolor_f[i]):
      shade_f[i] = inicolor_f[i] + (inicolor_f[i] - fincolor_f[i])/(step_f-1)
      for j in range(0,step_f):
        shade_f[i] = shade_f[i] - (inicolor_f[i] - fincolor_f[i])/(step_f-1)
        mat_shade_f_T[i,j] = shade_f[i]
    else:
      shade_f[i] = inicolor_f[i] - (fincolor_f[i] - inicolor_f[i])/(step_f-1)
      for j in range(0,step_f):
        shade_f[i] = shade_f[i] + (fincolor_f[i] - inicolor_f[i])/(step_f-1)
        mat_shade_f_T[i,j] = shade_f[i]
else: #If there is only one circle, the color to display = final color
  for i in range(0,3):
    shade_f[i] = fincolor_f[i]
    for j in range(0,step_f):
      mat_shade_f_T[i,j] = shade_f[i]

if (step_c > 1): #Loop for canal circles
  for i in range(0,3):
    if (inicolor_c[i]  >= fincolor_c[i]):
      shade_c[i] = inicolor_c[i] + (inicolor_c[i] - fincolor_c[i])/(step_c-1)
      for j in range(0,step_c):
        shade_c[i] = shade_c[i] - (inicolor_c[i] - fincolor_c[i])/(step_c-1)
        mat_shade_c_T[i,j] = shade_c[i]
    else:
      shade_c[i] = inicolor_c[i] - (fincolor_c[i] - inicolor_c[i])/(step_c-1)
      for j in range(0,step_c):
        shade_c[i] = shade_c[i] + (fincolor_c[i] - inicolor_c[i])/(step_c-1)
        mat_shade_c_T[i,j] = shade_c[i]
else: #If there is only one circle, the color to display = final color
  for i in range(0,3):
    shade_c[i] = fincolor_c[i]
    for j in range(0,step_c):
      mat_shade_c_T[i,j] = shade_c[i]

# Rounding and transposing the shading matrixes
mat_shade_f = np.matrix.round(np.matrix.transpose(mat_shade_f_T))
mat_shade_c = np.matrix.round(np.matrix.transpose(mat_shade_c_T))

# Gathering all the information in one matrix Om_f for forest and Om_c for␣
 ↪canals
Om_f_T = np.concatenate((mat_shade_f_T, R_f_T, T_f_T))
```

```
Om_f = np.matrix.transpose(Om_f_T) # Forests outline matrix #(number of␣
↪outlines, thickness and rgb channels)
#Om_f = Om_f.astype(int) #Convert float matrix to int matrix
#print('Om_f matrix: RGB channels ( 3 columns ), Outline sizes␣
↪(',nb_tree_types,'columns ), Temperature diminutions␣
↪(',nb_tree_types,'columns )')
Om_c_T = np.concatenate((mat_shade_c_T, R_c_T, T_c_T))
Om_c = np.matrix.transpose(Om_c_T) # Canals outline matrix #(number of␣
↪outlines, thickness and rgb channels)
#Om_c = Om_c.astype(int) #Convert float matrix to int matrix
#print('Om_c matrix: RGB channels ( 3 columns ), Outline sizes ( 1 column  ),␣
↪Temperature diminutions ( 1 column  )')

# Print validation of model setting
print('Model setting done.\n')
print('Forest RGB channels:\n',mat_shade_f,'\n\nForest circle sizes (m):\n',(np.
↪matrix.round(R_f*200/1668)).astype(int),'\n\nForest temperature diminutions␣
↪(°C):\n',T_f,'\n\n')
print('Canal RGB channels:\n',mat_shade_c,'\n\nCanal circle sizes (m):\n',(np.
↪matrix.round(R_c*200/1668)).astype(int),'\n\nCanal temperature diminutions␣
↪(°C):\n',T_c)
```

```
Model setting done.

Forest RGB channels:
 [[255. 215.   0.]
 [255. 197.  16.]
 [255. 180.  32.]
 [255. 162.  48.]
 [255. 145.  64.]
 [255. 127.  80.]]

Forest circle sizes (m):
 [[ 50  50  50]
 [100 100 100]
 [150 150 150]
 [200 200 200]
 [250 250 250]
 [300 300 300]]

Forest temperature diminutions (°C):
 [[1.32 1.01 0.58]
 [1.14 0.94 0.52]
 [0.57 0.73 0.22]
 [0.3  0.24 0.02]
 [0.16 0.   0.  ]
 [0.13 0.   0.  ]]
```

```
Canal RGB channels:
 [[184. 184.  11.]]

Canal circle sizes (m):
 [[30]]

Canal temperature diminutions (°C):
 [[1.2]]
```

[ ]:
```
# PART 6 - COLORATION

# Extracting number of pixels of forest and number of pixels of canal
tot_f, tot_c = 0,0
for k in range (0,cluster_count):
  if (Clust[k,4] == 2):
    tot_c = tot_c + Clust[k,1]
  if (Clust[k,4] == 3):
    tot_f = tot_f + Clust[k,1]

# Extracting pixel positions of forests and canals
print('Extracting canal and forest pixels positions...')
forest = np.zeros((tot_f,2))
canal = np.zeros((tot_c,2))
m = 0
n = 0
l = 0
for i in range(0, width, 1):   #start, stop, step
  for j in range(0, height, 1):
    if (mat_position[i,j] == 3):
      forest[m,0] = i
      forest[m,1] = j
      m = m+1
    if (mat_position[i,j] == 2):
      canal[n,0] = i
      canal[n,1] = j
      n = n+1
forest = forest.astype(int)
canal = canal.astype(int)
dforest = pd.DataFrame(forest)
dforest.to_csv(path_display + '/position_forest.csv',header=False, index=False)
dcanal = pd.DataFrame(canal)
dcanal.to_csv(path_display + '/position_canal.csv',header=False, index=False)

# Creating a matrix with a Treated/Non-treated parameter for each pixel to␣
↪avoid treating the same pixels several times
```

```
pixel_already_treated = np.zeros((width, height)) #0 means non-treated, other␣
↪means treated

# Associating maximum temperature diminution +100 to forest and canal areas␣
↪pixels in pixel_already_treated to be sure they won't be colored as "circles"
pixel_already_treated[mat_position==3] = max(np.amax(T_f),np.amax(T_c)) + 100
pixel_already_treated[mat_position==2] = max(np.amax(T_f),np.amax(T_c)) + 100

# Extracting forest edges to gain time for the next step
# We create a matrix with i,j coordonates of "edge pixels" longer than␣
↪necessary because we don't know yet the lenght and then we crop the empty␣
↪rows
edge_f = np.zeros((tot_f,2)) # We use tot_f here because we know tot_f is␣
↪obviously superior to the number of "edge pixels" (that we don't know yet)
f = 0
print('Extracting forest edges...')
for m in range(0, tot_f, 1):   #start, stop, step
    i = forest[m,0]
    j = forest[m,1]
    cond = 0 # This condition is aimed at gaining time by quiting the loop when␣
↪at least one pixel of non-canal and non-forest is found next to the current␣
↪forest pixel
    a = 0
    while (a < 360) and (cond != 1):
      alpha = np.radians(a)
      x = i+ round(np.cos(alpha))
      y = j+ round(np.sin(alpha))
      a = a +1
      if ((0 < (x) < width) and (0 < (y) < height)
      and (pixel_already_treated[x, y] == 0)):  # If there is a pixel which is␣
↪neither canal nor forest just next to the current forest pixel then this is␣
↪an edge pixel
        edge_f[f,0] = i
        edge_f[f,1] = j
        f = f +1
        cond = 1
        #imgR[:,:,2][i,j] = c0r_c #useful to color the edge to check if it works
        #imgG[:,:,1][i,j] = c0g_c
        #imgB[:,:,0][i,j] = c0b_c

# Extracting canal edges to gain time for the next step
# We create a matrix with i,j coordonates of "edge pixels" longer than␣
↪necessary because we don't know yet the lenght and then we crop the empty␣
↪rows
edge_c = np.zeros((tot_c,2)) # We use tot_c here because we know tot_c is␣
↪obviously superior to the number of "edge pixels" (that we don't know yet)
```

```
c = 0
print('Extracting canal edges...')
for n in range(0, tot_c, 1):   #start, stop, step
    i = canal[n,0]
    j = canal[n,1]
    cond = 0 # This condition is aimed at gaining time by quiting the loop when␣
 ↪at least one pixel of non-canal and non-forest is found next to the current␣
 ↪forest pixel
    a = 0
    while (a < 360) and (cond != 1):
      alpha = np.radians(a)
      x = i+ round(np.cos(alpha))
      y = j+ round(np.sin(alpha))
      a = a +1
      if ((0 < (x) < width) and (0 < (y) < height)
      and (pixel_already_treated[x, y] == 0)):  # If there is a pixel which is␣
 ↪neither canal nor forest just next to the current forest pixel then this is␣
 ↪an edge pixel
        edge_c[c,0] = i
        edge_c[c,1] = j
        c = c +1
        cond = 1
        #imgR[:,:,2][i,j] = c0r_c #useful to color the edge to check if it works
        #imgG[:,:,1][i,j] = c0g_c
        #imgB[:,:,0][i,j] = c0b_c

# Cropping all empty rows in order to keep just the "edge pixels" positions
edge_f = edge_f.astype(int)
mask_f = edge_f == 0
rows_f = np.flatnonzero((~mask_f).sum(axis=1))
cols_f = np.flatnonzero((~mask_f).sum(axis=0))
edge_f = edge_f[rows_f.min():rows_f.max()+1, cols_f.min():cols_f.max()+1]

# Cropping all empty rows in order to keep just the "edge pixels" positions
edge_c = edge_c.astype(int)
mask_c = edge_c == 0
rows_c = np.flatnonzero((~mask_c).sum(axis=1))
cols_c = np.flatnonzero((~mask_c).sum(axis=0))
edge_c = edge_c[rows_c.min():rows_c.max()+1, cols_c.min():cols_c.max()+1]

# Saving previous matrixes
dedge_f = pd.DataFrame(edge_f)
dedge_f.to_csv(path_display + '/edge_f.csv',header=False, index=False)
dedge_c = pd.DataFrame(edge_c)
dedge_c.to_csv(path_display + '/edge_c.csv',header=False, index=False)

# Extracting forest circle pixels
```

```
d_f_m1 = 0
for l in range(0,step_f,1):
  print('Coloration of forest circle number',l+1,'/',step_f,'...')
  for m in range(0, f):
    i = edge_f[m,0]
    j = edge_f[m,1]
    d_f = int(Om_f[l, 2+which_coef[i,j]])
    if (l >= 1):
      d_f_m1 = int(Om_f[l-1, 2+which_coef[i,j]])
    t_f = Om_f[l, 2+nb_tree_types+which_coef[i,j]]
    for d in range(d_f_m1,d_f):
      a = 0
      while (a < 360):
        alpha = np.radians(a)
        x = i+ round(d *np.cos(alpha))
        y = j+ round(d *np.sin(alpha))
        a = a + 1 #Increase a < 360 to gain execution rapidity but losing␣
↪precision of coloration
        if ((0 < (x) < width) and (0 < (y) < height)
        and (pixel_already_treated[x, y] < t_f)):
          pixel_already_treated[x, y] = t_f
  dpixel_already_treated = pd.DataFrame(pixel_already_treated)
  dpixel_already_treated.to_csv(path_display + '/
↪pixel_already_treated_'+str(l+1)+'.csv',header=False, index=False)
  print('pixel_already_treated_',l+1, 'saved')

# Extracting canal circle pixels
d_c_m1 = 0
for l in range(0,step_c):
  print('Coloration of canal circle number',l+1,'/',step_c,'...')
  for n in range(0, c, 1):
    i = edge_c[n,0]
    j = edge_c[n,1]
    d_c = int(Om_c[l, 3])
    if (l>=1):
      d_c_m1 = int(Om_c[l-1, 3])
    t_c = Om_c[l, 4]
    for d in range(d_c_m1,d_c):
      a = 0
      while (a < 360):
        alpha = np.radians(a)
        x = i+ round(d *np.cos(alpha))
        y = j+ round(d *np.sin(alpha))
        a = a + 1 #Increase a < 360 to gain execution rapidity but losing␣
↪precision of coloration
        if ((0 < (x) < width) and (0 < (y) < height)
        and (pixel_already_treated[x, y] < t_c)):
```

```
            pixel_already_treated[x, y] = t_c
    dpixel_already_treated = pd.DataFrame(pixel_already_treated)
    dpixel_already_treated.to_csv(path_display + '/
 ↪pixel_already_treated_'+str(step_f+l+1)+'.csv',header=False, index=False)
    print('pixel_already_treated_',step_f+l+1, 'saved')

# Coloring forest "circles"
for u in range(0,step_f):
    for v in range(0, nb_tree_types):
        if (Om_f[u, 3+nb_tree_types+v] != 0):
            imgR[:,:,2][pixel_already_treated == Om_f[u, 3+nb_tree_types+v]] =␣
 ↪Om_f[u, 0]
            imgG[:,:,1][pixel_already_treated == Om_f[u, 3+nb_tree_types+v]] =␣
 ↪Om_f[u, 1]
            imgB[:,:,0][pixel_already_treated == Om_f[u, 3+nb_tree_types+v]] =␣
 ↪Om_f[u, 2]

# Coloring canal "circles"
for u in range(0,step_c):
    if (Om_c[u, 4] != 0):
        imgR[:,:,2][pixel_already_treated == Om_c[u, 4]] = Om_c[u, 0]
        imgG[:,:,1][pixel_already_treated == Om_c[u, 4]] = Om_c[u, 1]
        imgB[:,:,0][pixel_already_treated == Om_c[u, 4]] = Om_c[u, 2]

# Re-associating the correct temperature diminution to forest and canal areas␣
 ↪pixels in pixel_already_treated after they didn't get colored as "circles"
pixel_already_treated[mat_position==3] = np.amax(T_f)
pixel_already_treated[mat_position==2] = np.amax(T_c)

# Forests and canals coloration
imgR[:,:,2][mat_position==3] = c0r_f
imgG[:,:,1][mat_position==3] = c0g_f
imgB[:,:,0][mat_position==3] = c0b_f
imgR[:,:,2][mat_position==2] = c0r_c
imgG[:,:,1][mat_position==2] = c0g_c
imgB[:,:,0][mat_position==2] = c0b_c

#Saving previous matrixes
dpixel_already_treated = pd.DataFrame(pixel_already_treated)
dpixel_already_treated.to_csv(path_display + '/pixel_already_treated_final.
 ↪csv',header=False, index=False)
```

```
Extracting canal and forest pixels positions…
Extracting forest edges…
Extracting canal edges…
Coloration of forest circle number 1 / 6 …
Coloration of forest circle number 2 / 6 …
```

```
Coloration of forest circle number 3 / 6 …
Coloration of forest circle number 4 / 6 …
Coloration of forest circle number 5 / 6 …
Coloration of forest circle number 6 / 6 …
Coloration of canal circle number 1 / 1 …
```

[ ]:

```
# PART 7 - IMAGE RECONSTRUCTION AND PLOT

# Erasing colors on canals and forests
#ih = cv2.imread(path_base + '/Image 1.jpg')[0:rows_number, 0:columns_number]
#ihR = cv2.imread(path_base + '/Image 1.jpg')[0:rows_number, 0:columns_number]
#ihG = cv2.imread(path_base + '/Image 1.jpg')[0:rows_number, 0:columns_number]
#ihB = cv2.imread(path_base + '/Image 1.jpg')[0:rows_number, 0:columns_number]
#wid,hei,nb_co = ih.shape
#imge = np.zeros((wid,hei,nb_co))
#ihR[:,:,0] = 0         # X,Y,Z     # 0=blue, 1=green, 2=red
#ihR[:,:,1] = 0
#ihG[:,:,0] = 0
#ihG[:,:,2] = 0
#ihB[:,:,1] = 0
#ihB[:,:,2] = 0
#imgR[:,:,2][mat_position==3] = ihR[:,:,2][mat_position==3]
#imgG[:,:,1][mat_position==3] = ihG[:,:,1][mat_position==3]
#imgB[:,:,0][mat_position==3] = ihB[:,:,0][mat_position==3]
#imgR[:,:,2][mat_position==2] = ihR[:,:,2][mat_position==2]
#imgG[:,:,1][mat_position==2] = ihG[:,:,1][mat_position==2]
#imgB[:,:,0][mat_position==2] = ihB[:,:,0][mat_position==2]

# Image reconstrution
img[:,:,0] = imgB[:,:,0]
img[:,:,1] = imgG[:,:,1]
img[:,:,2] = imgR[:,:,2]

# Transparency of display
alpha = 0.2 # transparency to change # alpha between 0 et 1
beta = (1 - alpha)

# Overlaying the two images with transparency
cv_ori = original_img.astype(np.uint8)
cv_img = img.astype(np.uint8)
added_image = cv2.addWeighted(cv_ori,alpha,cv_img,beta,0)

# Plot
print('\n','Original image:')
cv2_imshow(original_img)
cv2.waitKey()
#print('\n','Recolored image without transparency:')
```

```
#cv2_imshow(cv_img)
#cv2.waitKey()
print('\n','Recolored image with transparency at',str(int(alpha*100)),'%.')
cv2_imshow(added_image)
cv2.waitKey()

# Saving result image
result = cv2.imwrite(path_display + '/image_' + str(index_save) + '_Combined.
↪jpg', added_image)
if (result == True):
  print('Combined image saved successfully')
else:
  print('Combined in saving file')
index_save = index_save + 1
```

[ ]:

```
# PART 8 - WRITE CO2 CAPTATION ON THE PREVIOUS IMAGE WITH CV2

# Reinitialising the image to the Part 7 one
alpha = 0.1
beta = 1 - alpha
cv_ori = original_img.astype(np.uint8)
cv_img = img.astype(np.uint8)
added_image = cv2.addWeighted(cv_ori,alpha,cv_img,beta,0)

# Defining the font parameters
FONT = cv2.FONT_HERSHEY_SIMPLEX
FONT_SCALE = 1      #Font size
FONT_THICKNESS = 2   #Font thickness
label_color = (0, 0, 0)

# Defining parameters for the rectangle around the text
color = (255, 255, 255) #BGR
thickness = -1 # Line thickness in px, -1 = filled

# Writing the CO2 captation of each cluster on its center
tot_carbon = 0
for k in range(0, cluster_count):
  if (mat_position[Clust[k,2],Clust[k,3]] == 3):  #If the cluster is a cluster␣
↪of trees

    # Calculating the number of trees within the area
    clust_w, clust_h = round((200/1668)*np.sqrt(Clust[k,1])), round((200/
↪1668)*np.sqrt((Clust[k,1])))
    clust_area = clust_w*clust_h
    print('Cluster surface:', clust_area , 'm^2')
    nb_tree = float(round(10*float(clust_area/surface_tree))/10) #Truncating␣
↪the float number to 1 significant digit
```

```
    print('Considering an average tree surface of 60.73m^2 in Bangkok, the␣
↪estimation of the number of trees within the area is: ',nb_tree,'tree(s).')
    carbon_area = carbon_tree * nb_tree
    print('Considering an average CO2 reception around 21 kilograms per year␣
↪and per tree, the estimation of the CO2 reception within the area is:
↪',carbon_area,'kg/year.')
    tot_carbon = tot_carbon + carbon_area

    # Calculating the position to display the text
    label = str(carbon_area) + 'kg/year' #Text to display
    (label_height, label_width), baseline = cv2.getTextSize(label, FONT,␣
↪FONT_SCALE, FONT_THICKNESS)
    x = round(Clust[k,3] - label_height/2)     #left
    y = round(Clust[k,2] + label_width/2)    #low (by default it is writting in␣
↪(0,label_height) so on its lower left corner)

    # The following conditions are aimed at displaying the text within the␣
↪image frame
    if (x < 0):
      x = 0
    elif (round(x + label_height) > height):
      x = height - label_height
    if (y < 0):
      y = label_width
    elif (y - label_width > width):
      y = Clust[k,2]

    # Draw a rectangle around the text
    start_point = (x, y-label_width - 2)     #represents the top left corner of␣
↪rectangle
    end_point = (x+label_height, y + 4)   #represents the bottom right corner of␣
↪rectangle
    cv2.rectangle(added_image, start_point, end_point, color, thickness)

    # Write text according to the previous information
    cv2.putText(added_image, label, (x, y), FONT, FONT_SCALE, label_color,␣
↪FONT_THICKNESS)

# Display the result image
cv2_imshow(added_image)
cv2.waitKey(0)

# Saving result image
resul = cv2.imwrite(path_display + '/image_' + str(index) +␣
↪'_Combined_with_text.jpg', added_image)
if (resul == True):
```

```
  print('Combined image saved successfully')
else:
  print('Error in saving combined image')
index = index + 1
```

[ ]:
```
# PART 9 - CALCULATING TOTAL CO2 RECEPTION AND MEAN TEMPERATURE DIMINUTION

# tot_carbon is equal to the total CO2 reception of the whole area of the␣
↪picture
print('Considering an average CO2 reception around 21 kilograms per year and␣
↪per tree, the estimation of the CO2 reception within the total area of the␣
↪picture is:',round(tot_carbon),'kg/year.')

# We created the matrix pixel_already_treated so that in every pixel of the␣
↪matrix, there is the corresponding temperature diminution
# so the mean dtemperature diminution on the picture is the mean value of our␣
↪matrix
mean_temp = float(round(100*float(pixel_already_treated.mean()))/100)␣
↪#Truncating the float number to 1 significant digit
print('The mean urban cooling thanks to forests and canals within the total␣
↪area of the picture is', mean_temp,'°C.')
```