comparator.py

```python
import numpy as np
import encoder
import sys, os
from pathlib import Path
from PIL import Image
import cv2
import pandas as pd
import time as t
import shutil
import utils
import matplotlib.pyplot as plt
from encoder import DCT, padding
from scipy.fftpack import dct

LIM = 25  # number of files to test to


def compare(
    quality=None,
    dataDirectory=None,
    outputDirectory=None,
    subsample=None,
    useStdHuffmanTable=None,
    DeleteFilesAfterward=True,
):
    if quality is None:
        quality = np.random.randint(0, 101)
    if subsample is None:
        subsample = "4:2:2"
    if dataDirectory is None:
        dataDirectory = "./data/datasetBmp"
    if useStdHuffmanTable is None:
        useStdHuffmanTable = False
    outputDirectory = f"./data/treated/quality{quality}-subsample{subsample}-stdHf{useStdHuffmanTable}"
    i = 0
    stat = np.zeros(
```

```
        (LIM, 3), dtype=object
    )  # first parameter is size before compression, second after, and third the time to achieve compression
    for filename in os.listdir(dataDirectory):
        if i >= LIM:
            break
        f = os.path.join(dataDirectory, filename)
        if not os.path.exists(outputDirectory):
            os.makedirs(outputDirectory)
        f_out = os.path.join(outputDirectory, filename + ".jpg")
        if os.path.isfile(f):
            previousSize = os.stat(f).st_size
            image = Image.open(f)
            time = t.time_ns()
            encoder.write_jpeg(
                f_out, np.array(image), quality, subsample, useStdHuffmanTable
            )
            time = t.time_ns() - time
            newSize = os.stat(f_out).st_size
            stat[i][0] = previousSize
            stat[i][1] = newSize
            stat[i][2] = time
        i += 1
    if DeleteFilesAfterward:
        shutil.rmtree(outputDirectory)
    write_stat("./data/treated/stat.txt", stat, quality, subsample, useStdHuffmanTable)


def write_stat(statFile, stat, quality, subsample, standHuffTables):
    with open(statFile, "a+") as f:
        f.write("\n" * 2)
        f.write("New sample \n")
        f.write(f"Size of sample : {LIM} images \n")
        f.write(
            f"Parameters of compression : (quality) {quality}, (subsample) {subsample}, (usage of standard HuffTables) {'Yes' if standHuffTables else 'No'} \n"
```

```
71          )
72          avgPreviousSize = np.average(stat[:, 0])
73          avgNewSize = np.average(stat[:, 1])
74          f.write(
75              f"Average size of image before
compression : {avgPreviousSize} bytes \n"
76          )
77          f.write(f"Average size of images after
compression : {avgNewSize} bytes \n")
78          f.write(f"Ratio is {avgPreviousSize /
avgNewSize:.2f}")
79
80
81  def write_stat_csv(outputDirectory, stat, quality,
subsample, standHuffTables):
82      pass
83
84  def energyCompaction(imgPath):
85      img = cv2.imread(imgPath)
86
87      imgYCrCB = cv2.cvtColor(
88          img, cv2.COLOR_RGB2YCrCb
89      )  # Convert RGB to YCrCb (Cb applies V, and Cr
applies U).
90
91      Y, Cr, Cb = cv2.split(padding(imgYCrCB, 8, 8))
92      Y = Y.astype('int') - 128
93      blocks_Y = utils.divide_blocks(Y, 8, 8)
94      dctBlocks_Y = np.zeros_like(blocks_Y)
95      for i in range(len(blocks_Y)):
96          dctBlocks_Y[i] = dct(dct(blocks_Y[i], axis=0,
norm="ortho"), axis=1, norm="ortho")
97      avg_Y = utils.averageMatrix(blocks_Y)
98      avgDct_Y = utils.averageMatrix(dctBlocks_Y)
99
100     x = np.random.randint(blocks_Y.shape[0])
101     arr1 = blocks_Y[x]
102     arr2 = dctBlocks_Y[x]
103
104     fig, (ax1, ax2 )= plt.subplots(1, 2)
105
106     valueMax, valueMin = max(np.max(arr1),
np.max(arr2)), min(np.min(arr1), np.min(arr2))
```

```
107     # fig.suptitle('Matrice de la luminance de
"villeLyon.jpg"')
108
109     ax1.matshow(arr1, cmap="cool", vmin=valueMin,
vmax=valueMax)
110     ax1.set_title('avant DCT')
111
112     ax2.matshow(arr2, cmap="cool", vmin=valueMin,
vmax=valueMax)
113     ax2.set_title('après DCT')
114
115
116     for i in range(arr1.shape[0]):
117         for j in range(arr1.shape[1]):
118             cNormal= int(arr1[i, j])
119             cDct = int(arr2[i, j])
120             ax1.text(i, j, str(cNormal), va='center',
ha='center')
121             ax2.text(i, j, str(cDct), va='center',
ha='center')
122     plt.savefig('./data/energyCompaction.png',
transparent=True)
123
124
125 def rgbToYCbCr_channel_bis():
126     img = cv2.imread("./data/villeLyon.jpg")  # Read
input image in BGR format
127
128     imgYCrCB = cv2.cvtColor(
129         img, cv2.COLOR_BGR2YCrCb
130     )  # Convert RGB to YCrCb (Cb applies V, and Cr
applies U).
131
132     Y, Cr, Cb = cv2.split(imgYCrCB)
133
134     # Fill Y and Cb with 128 (Y level is middle gray,
and Cb is "neutralized").
135     onlyCr = imgYCrCB.copy()
136     onlyCr[:, :, 0] = 128
137     onlyCr[:, :, 2] = 128
138     onlyCr_as_bgr = cv2.cvtColor(
139         onlyCr, cv2.COLOR_YCrCb2BGR
```

```
    )  # Convert to BGR - used for display as false color

    # Fill Y and Cr with 128 (Y level is middle gray, and Cr is "neutralized").
    onlyCb = imgYCrCB.copy()
    onlyCb[:, :, 0] = 128
    onlyCb[:, :, 1] = 128
    onlyCb_as_bgr = cv2.cvtColor(
        onlyCb, cv2.COLOR_YCrCb2BGR
    )  # Convert to BGR - used for display as false color

    cv2.imshow("img", img)
    cv2.imshow("Y", Y)
    cv2.imshow("onlyCb_as_bgr", onlyCb_as_bgr)
    cv2.imshow("onlyCr_as_bgr", onlyCr_as_bgr)
    cv2.waitKey()
    cv2.destroyAllWindows()

    cv2.imwrite("./data/treated/villeLyon_Y.jpg", Y)
    cv2.imwrite("./data/treated/villeLyon_Cb.jpg", onlyCb_as_bgr)
    cv2.imwrite("./data/treated/villeLyon_Cr.jpg", onlyCr_as_bgr)


if __name__ == "__main__":
    # compare()
    # rgbToYCbCr_channel_bis()
    energyCompaction("./data/villeLyon.jpg")
    test = np.array([[93, 90, 83, 68, 61, 61, 46, 21],
                     [102, 92, 95, 77, 65, 60, 49, 32],
                     [69, 55, 47, 57, 65, 60, 72, 65],
                     [55, 55, 40, 42, 23, 1, 11, 38],
                     [55, 57, 47, 53, 35, 59, -2, 26],
                     [64, 41, 42, 55, 60, 57, 25, -8],
                     [77, 87, 58, -2, -5, 14, -10, -35],
                     [38, 14, 33, 33, -21, -23, -43, -34]])
```

```
174     print(dct(dct(test, axis=0, norm="ortho"), axis=1, norm='ortho'))
```