import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from scipy import ndimage

from scipy import optimize

import numpy as np

import math

# Manipulate channels

def get\_greyscale\_image(img):

return np.mean(img[:,:,:2], 2)

def extract\_rgb(img):

return img[:,:,0], img[:,:,1], img[:,:,2]

def assemble\_rbg(img\_r, img\_g, img\_b):

shape = (img\_r.shape[0], img\_r.shape[1], 1)

return np.concatenate((np.reshape(img\_r, shape), np.reshape(img\_g, shape),

np.reshape(img\_b, shape)), axis=2)

# Transformations

def reduce(img, factor):

result = np.zeros((img.shape[0] // factor, img.shape[1] // factor))

for i in range(result.shape[0]):

for j in range(result.shape[1]):

result[i,j] = np.mean(img[i\*factor:(i+1)\*factor,j\*factor:(j+1)\*factor])

return result

def rotate(img, angle):

return ndimage.rotate(img, angle, reshape=False)

def flip(img, direction):

return img[::direction,:]

def apply\_transformation(img, direction, angle, contrast=1.0, brightness=0.0):

return contrast\*rotate(flip(img, direction), angle) + brightness

# Contrast and brightness

def find\_contrast\_and\_brightness1(D, S):

# Fix the contrast and only fit the brightness

contrast = 0.75

brightness = (np.sum(D - contrast\*S)) / D.size

return contrast, brightness

def find\_contrast\_and\_brightness2(D, S):

# Fit the contrast and the brightness

A = np.concatenate((np.ones((S.size, 1)), np.reshape(S, (S.size, 1))), axis=1)

b = np.reshape(D, (D.size,))

x, \_, \_, \_ = np.linalg.lstsq(A, b)

#x = optimize.lsq\_linear(A, b, [(-np.inf, -2.0), (np.inf, 2.0)]).x

return x[1], x[0]

# Compression for greyscale images

def generate\_all\_transformed\_blocks(img, source\_size, destination\_size, step):

factor = source\_size // destination\_size

transformed\_blocks = []

for k in range((img.shape[0] - source\_size) // step + 1):

for l in range((img.shape[1] - source\_size) // step + 1):

# Extract the source block and reduce it to the shape of a destination block

S = reduce(img[k\*step:k\*step+source\_size,l\*step:l\*step+source\_size], factor)

# Generate all possible transformed blocks

for direction, angle in candidates:

transformed\_blocks.append((k, l, direction, angle, apply\_transformation(S, direction, angle)))

return transformed\_blocks

def compress(img, source\_size, destination\_size, step):

transformations = []

transformed\_blocks = generate\_all\_transformed\_blocks(img, source\_size, destination\_size, step)

i\_count = img.shape[0] // destination\_size

j\_count = img.shape[1] // destination\_size

for i in range(i\_count):

transformations.append([])

for j in range(j\_count):

#print("{}/{} ; {}/{}".format(i, i\_count, j, j\_count))

transformations[i].append(None)

min\_d = float('inf')

# Extract the destination block

D = img[i\*destination\_size:(i+1)\*destination\_size,j\*destination\_size:(j+1)\*destination\_size]

# Test all possible transformations and take the best one

for k, l, direction, angle, S in transformed\_blocks:

contrast, brightness = find\_contrast\_and\_brightness2(D, S)

S = contrast\*S + brightness

d = np.sum(np.square(D - S))

if d < min\_d:

min\_d = d

transformations[i][j] = (k, l, direction, angle, contrast, brightness)

return transformations

def decompress(transformations, source\_size, destination\_size, step, nb\_iter=8):

factor = source\_size // destination\_size

height = len(transformations) \* destination\_size

width = len(transformations[0]) \* destination\_size

iterations = [np.random.randint(0, 256, (height, width))]

cur\_img = np.zeros((height, width))

for i\_iter in range(nb\_iter):

#print(i\_iter)

for i in range(len(transformations)):

for j in range(len(transformations[i])):

# Apply transform

k, l, flip, angle, contrast, brightness = transformations[i][j]

S = reduce(iterations[-1][k\*step:k\*step+source\_size,l\*step:l\*step+source\_size], factor)

D = apply\_transformation(S, flip, angle, contrast, brightness)

cur\_img[i\*destination\_size:(i+1)\*destination\_size,j\*destination\_size:(j+1)\*destination\_size] = D

iterations.append(cur\_img)

cur\_img = np.zeros((height, width))

return iterations

# Compression for color images

def reduce\_rgb(img, factor):

img\_r, img\_g, img\_b = extract\_rgb(img)

img\_r = reduce(img\_r, factor)

img\_g = reduce(img\_g, factor)

img\_b = reduce(img\_b, factor)

return assemble\_rbg(img\_r, img\_g, img\_b)

def compress\_rgb(img, source\_size, destination\_size, step):

img\_r, img\_g, img\_b = extract\_rgb(img)

return [compress(img\_r, source\_size, destination\_size, step), \

compress(img\_g, source\_size, destination\_size, step), \

compress(img\_b, source\_size, destination\_size, step)]

def decompress\_rgb(transformations, source\_size, destination\_size, step, nb\_iter=8):

img\_r = decompress(transformations[0], source\_size, destination\_size, step, nb\_iter)[-1]

img\_g = decompress(transformations[1], source\_size, destination\_size, step, nb\_iter)[-1]

img\_b = decompress(transformations[2], source\_size, destination\_size, step, nb\_iter)[-1]

return assemble\_rbg(img\_r, img\_g, img\_b)

# Plot

def plot\_iterations(iterations, target=None):

# Configure plot

plt.figure()

nb\_row = math.ceil(np.sqrt(len(iterations)))

nb\_cols = nb\_row

# Plot

for i, img in enumerate(iterations):

plt.subplot(nb\_row, nb\_cols, i+1)

plt.imshow(img, cmap='gray', vmin=0, vmax=255, interpolation='none')

if target is None:

plt.title(str(i))

else:

# Display the RMSE

plt.title(str(i) + ' (' + '{0:.2f}'.format(np.sqrt(np.mean(np.square(target - img)))) + ')')

frame = plt.gca()

frame.axes.get\_xaxis().set\_visible(False)

frame.axes.get\_yaxis().set\_visible(False)

plt.tight\_layout()

# Parameters

directions = [1, -1]

angles = [0, 90, 180, 270]

candidates = [[direction, angle] for direction in directions for angle in angles]

# Tests

def test\_greyscale(img):

#img = reduce(img, 4)

#plt.figure()

#plt.imshow(img, cmap='gray', interpolation='none')

transformations = compress(img, 8, 4, 8)

iterations = decompress(transformations, 8, 4, 8)

#plot\_iterations(iterations, img)

#plt.show()

return transformations, iterations

def test\_rgb(img):

img = reduce\_rgb(img, 8)

transformations = compress\_rgb(img, 64, 4, 8)

retrieved\_img = decompress\_rgb(transformations, 64, 4, 8)

plt.figure()

plt.subplot(121)

plt.imshow(np.array(img).astype(np.uint8), interpolation='none')

plt.subplot(122)

plt.imshow(retrieved\_img.astype(np.uint8), interpolation='none')

plt.show()

return retrieved\_img

if \_\_name\_\_ == '\_\_main\_\_':

test\_greyscale()

#test\_rgb()