Department of Electronic Systems

TTT4135 - Multimedia Signal Processing

Assignment 1

Authors:
Vegard Iversen, Sondre Olimb, Sebastian Skogen Raa

Feb, 2022

Table of Contents

1 Still image compression

a

Real part:

Table 1: DCT of datablock

1063	6	-2	1	0	-1	0	1
-102	4	2	1	0	0	-1	0
37	1	1	0	-1	-2	0	0
-5	2	-1	0	1	0	0	0
-3	0	-1	0	0	-2	-1	0
5	0	0	0	1	0	1	0
3	5	2	0	0	0	0	0
-3	0	-1	0	0	0	0	0

b

Table 2: Quantized data block

66	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

When encoding/quantizing we distribute the bits more efficiently.

c

Table 3: Decoded data block

122	121	121	120	119	119	118	118
120	120	120	119	118	117	117	117
120	120	119	118	118	117	116	116
123	123	122	121	121	120	119	119
130	130	129	129	128	127	127	126
141	141	140	140	139	138	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155

The PSNR is found to be 40.62dB.

2 Prediction

2a

$$R_x(0) = \sigma_x, \quad R_x(1) = a, \quad R_x(k) = 0, \quad \text{for } |k| > 1 \tag{1}$$

For a second order predictor we have:

$$\hat{x}(n) = \sum_{i=1} b_i x(n-1) = b_1 x(n-1) + b_2 x(n-2) \tag{2}$$

The error between the predictor and the signal is given by

$$e(n) = x(n) - \hat{x}(n) = x(n) - b_1 x(n-1) - b_2 x(n-2) \tag{3}$$

Applying Wiener-Hopf equation (Ohm 2004, p.101).

$$\begin{bmatrix} r_x(1) \\ r_x(2) \end{bmatrix} = \begin{bmatrix} r_x(0) & r_x(1) \\ r_x(1) & r_x(0) \end{bmatrix} \cdot \begin{bmatrix} b(1) \\ b(2) \end{bmatrix} \tag{4}$$

Solving first equation from Equation 4:

$$\begin{aligned} R_x(1) &= R_x(0)b(1) + R_x(1)b(2) \\ R_x(0)b(1) &= R_x(1) - R_x(1)b(2) \\ b(1) &= \frac{R_x(1) - R_x(1)b(2)}{R_x(0)} \\ b(1) &= \frac{a - ab(2)}{\sigma_x} \end{aligned} \tag{5}$$

Solving second equation from Equation 4:

$$\begin{aligned} R_x(2) &= R_x(1)b(1) + R_x(0)b(2) \\ R_x(0)b(2) &= R_x(2) - R_x(1)b(1) \\ b(2) &= \frac{R_x(2) - R_x(1)b(1)}{R_x(0)} \\ b(2) &= \frac{R_x(2) - R_x(1)\frac{a-ab(2)}{\sigma_x}}{R_x(0)} \\ b(2) &= \frac{0 - a\frac{a-ab(2)}{\sigma_x}}{\sigma_x} \\ b(2) &= -\frac{a^2 - a^2 b(2)}{\sigma_x^2} \\ b(2) &= \frac{a^2}{a^2 - \sigma_x^2} \end{aligned} \tag{6}$$

Now using the value found for $b(2)$ in Equation 5:

$$\begin{aligned}
b(1) &= \frac{a - a\frac{a^2}{a^2-\sigma_x^2}}{\sigma_x} \\
b(1) &= \frac{a}{\sigma_x}(1 - \frac{a^2}{a^2-\sigma_x^2}) \\
b(1) &= \frac{a}{\sigma_x \cdot (a^2-\sigma_x^2)}(a^2 - \sigma_x^2 - a^2) \\
b(1) &= -\frac{a\sigma_x}{(a^2-\sigma_x^2)} \\
b(1) &= -\frac{a\sigma_x}{(a^2-\sigma_x^2)} \\
b(1) &= \frac{a\sigma_x}{\sigma_x^2 - a^2}
\end{aligned} \tag{7}$$

3 Compression basics

a

Q:
What are the fundamental steps for lossless and lossy multimedia compression, respectively? A: Lossless compression will perfectly reconstruct data that are compressed, but lossy compression will only allow approximation of the original data. Usually the lossy compression have an improved compression rate and there for the size of the data is much less than with lossless compression. The steps for lossless compression are:

1. Generate a statistical model of the input data.
2. Use the model to map input data to bit sequences in such a way that ”probable” data will produce shorter output than ”improbable” data. (Often used Huffmann coding here).

The steps for lossy compression are:

1. Quantization of the data. (Here we ”remove” things we can not see, hear etc.)
2. Transform coding.

b

Lossless:

1. Find the distribution of data so we can apply the next step.
2. store the most frequent data in small ”bit strings” and the less frequent in larger bit strings. For example if we have ”hi hi hi hi hi, yo”, ”hi” could be stored as 0 and yo as 1 (Note that this is just a simple example not how it actally works).

Lossy

1. Remove things that are not necessary, to simplfy and compress the data.
2. This is the process of creating a quantized group of block of consecutive samples from a source input and converting it into vectors [https://bitmovin.com/lossy-compression-algorithms/].

Figure 1: Caption

c

Lossy: for step 2

1. Transform Encryption
2. Discrete Cosine Transform
3. Discrete Wavelet Transform
4. fractal encoding

Lossless: for step 2

1. Run Length Encoding
2. Lempel-Ziv-Welch
3. Huffman Coding
4. EBCOT
5. Arithmetic encoding

d

Mean square error is one possibility of measuring the quality of compression. hear you measured the MSE between the original and decompressed image, which is easy to compute and therefor use by the algorithm designers. However this measurement of quality however thus not correlate with the end users perceived quality, mining that an image with a low MSE can be perceived as whore quality by the end user compared to one withe an higher MSE.

4 JPEG and JPEG2000

a)

Q:
Give a short overview of the most important differences between JPEG and JPEG 2000. Discuss this in two categories: (5 points) A: JPEG only offers

b

c

JPEG2000 offers scalability in the form of only

	JPEG	JPEG2000
Functionalities	Only offers lossy compression	Offers Lossy and lossless compression in one bitstream.
	At low bit rates JPEG achieves unacceptable quality	offers better compression at low bitrates
	Poor performance on non natural images	offers better performance for compound images and graphics (i.e computer generated images)
	Can produce blocking artifacts	Can produce ringing artifacts
	No built in scalability	The different sub-bands enables streaming of increasingly improving quality and dyadic transform offers multi-resolution
	No suppoert for ROI	Can set a region of interest with lossless compression
	Vulnerable to errors in bit stream	JPEG2000 offer better error resilience and performance better under error prone conditions
Coding techniques	Uses DCT	Uses DWT
	Support bit depth up to 16 bits	Support bit depth up to 38 bits
	Divides the image in to blocks	Divides the image in to sub- bands of high and low frequency
	Entropy: Uses run-length- encoding followed by Huffman encoding but also allows for arithmetic encoding instead of Huffman.	Entropy: Uses Block coding paradigm as in embedded block coding with optimized truncation (EBCOT). Each sub band is encoded independently for better error resilience. It also uses arithmetic MQ encoder
	Uses transform on 8x8 blocks	Transform used on entire image

Figure 2: Jpeg vs Jpeg 2000

Comparing the the two algorithm we see that jpeg produces blocking artifacts while jpeg 2000 produces ringing artifacts. Furthermore, we see that the jpeg algorithm shows contouring artifacts while jpeg 2000 manages to preserve smooth low frequency areas. These properties can make JPEG look somewhat sharper in high contrast areas, however jpeg 2000 looks better overall at high compression rates.

d

After calculating the corresponding BPP of the different JPEG quality levels we used these to compress with JP2. Comparison of the PSNRs can be seen in ??

Table 4: PSNR comparison on bike image.

Quality	JPEG	JP2
5 / 0.2bpp	24.67dB	28.44dB
20 / 0.4bpp	29.63dB	32.98dB
40 / 0.7bpp	32.177dB	35.62dB
60 / 0.9bpp	33.925dB	37.95dB
80 / 1.4bpp	36.92dB	40.95dB

Table 5: PSNR comparison on cafe image.

Quality	JPEG	JP2
5 / 0.3bpp	29.5dB	30.3dB
20 / 0.7bpp	31.2dB	31.8dB
40 / 1.1bpp	32dB	33.6dB
60 / 1.4bpp	33.1dB	35.3dB
80 / 2.1bpp	35.1dB	39.7dB

Table 6: PSNR comparison on woman image.

Quality	JPEG	JP2
5 / 0.2bpp	24.9dB	28.3dB
20 / 0.4bpp	29.6dB	32.5db
40 / 0.7bpp	32.1dB	35.3dB
60 / 0.9bpp	33.9dB	37.6dB
80 / 1.4bpp	37.0dB	41.1dB

Appendix

```
#!pip install ipympl
```

Setup

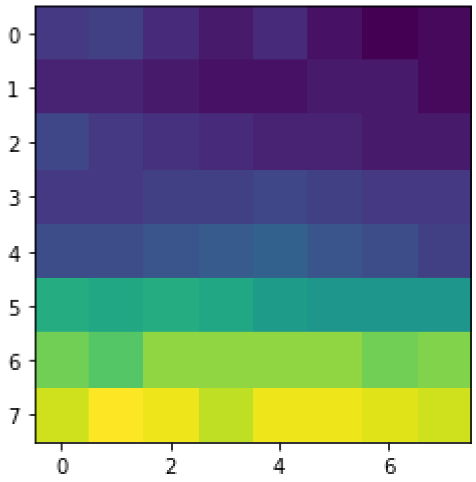
```
import numpy as np
import scipy.fftpack as fftpack
import matplotlib.pyplot as plt
import os
from google.colab import output
output.enable_custom_widget_manager()
```

```
#%matplotlib widget
```

```
def dct2(block):
    return fftpack.dct(fftpack.dct(block.T, norm="ortho").T, norm="ortho")

# implement 2D IDCT
def idct2(a):
    return fftpack.idct(fftpack.idct(a.T, norm="ortho").T, norm="ortho")

def compute_psnr(original: np.ndarray, compressed: np.ndarray):
    mse = np.mean((original.astype(np.float64) - compressed.astype(np.float64)) ** 2)
    if mse == 0:  # MSE is zero means no noise is present in the signal .
        # Therefore PSNR have no importance.
        return 100
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel) - 10 * np.log10(mse)
    return psnr
```

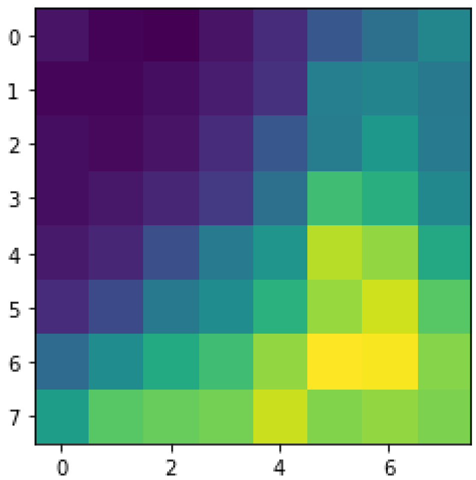
```
def JPEG_BPP(width, height, bits):
  return bits/(width* height)

```

```
data_block = np.array([
    [124,121,126,124,127,143,150,156],
    [125,121,124,124,127,142,148,159],
    [122,120,123,125,128,143,152,158],
    [120,119,122,125,129,142,152,155],
    [122,119,121,126,130,140,152,158],
    [119,120,121,125,128,139,152,158],
    [117,120,120,124,127,139,150,157],
    [118,118,120,124,125,139,151,156]
    ]).T

plt.imshow(data_block)
plt.show()
```



Quantization tabel

```
q_table = np.array([
        [16,12,14,14,18,24,49,72],
        [11,12,13,17,22,35,64,92],
        [10,14,16,22,37,55,78,95],
        [16,19,24,29,56,64,87,98],
        [24,26,40,51,68,81,103,112],
        [40,58,57,87,109,104,121,100],
        [51,60,69,80,103,113,120,103],
        [61,55,56,62,77,92,101,99]
]).T

plt.imshow(q_table)
plt.show()
```



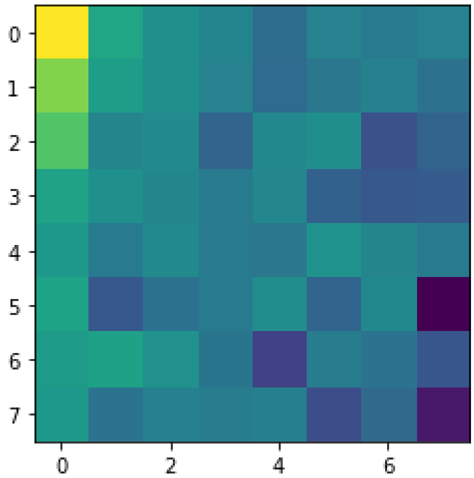
1 Still image compression

1a)

```
data_block_DCT2 = dct2(data_block)
```

```
print(data_block_DCT2)
np.savetxt(
    "data_block_DCT2.csv", (data_block_DCT2), fmt="%d"
)  # used for tabular in latex with generator

plt.imshow(np.log(np.abs(data_block_DCT2)))
plt.show()

```

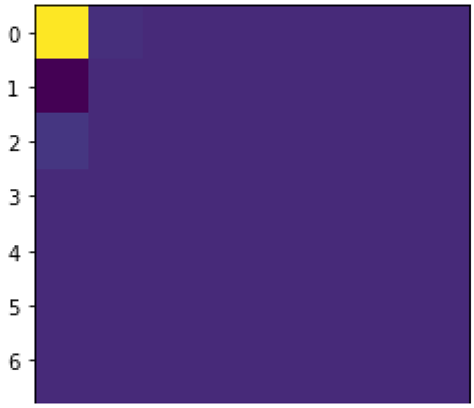
```
[[ 1.06387500e+03  6.56530796e+00 -2.24204932e+00  1.22031790e+00
  -3.75000000e-01 -1.08739329e+00  7.93388210e-01  1.13473851e+00]
 [-1.02438801e+02  4.56749149e+00  2.26372588e+00  1.12061110e+00
   3.58148693e-01 -6.33633466e-01 -1.05302724e+00 -4.80185917e-01]
 [ 3.77706166e+01  1.31440734e+00  1.77404852e+00  2.58329695e-01
  -1.50951150e+00 -2.21817548e+00 -1.00951481e-01  2.32881560e-01]
 [-5.67404043e+00  2.24214694e+00 -1.32600744e+00 -8.13211172e-01
   1.41728756e+00  2.21194688e-01 -1.39308310e-01  1.70278151e-01]
 [-3.37500000e+00 -7.45091955e-01 -1.75689591e+00  7.76371149e-01
  -6.25000000e-01 -2.65967417e+00 -1.30175526e+00  7.62049287e-01]
 [ 5.98943278e+00 -1.39948066e-01 -4.59461377e-01 -7.78805312e-01
   1.99935482e+00 -2.65215953e-01  1.46434767e+00  4.71007726e-03]
 [ 3.97325697e+00  5.52797698e+00  2.39904852e+00 -5.58772548e-01
  -5.12349883e-02 -8.47568736e-01 -5.24048519e-01 -1.30130006e-01]
 [-3.43314493e+00  5.19814083e-01 -1.07206540e+00  8.71070333e-01
   9.63382468e-01  9.02810104e-02  3.30504406e-01  1.09356359e-02]]
```



1b)

```
DCT2_quantized = np.floor(data_block_DCT2 / q_table + 0.5)
print(DCT2_quantized)
np.savetxt(
    "DCT2_quantized.csv", (DCT2_quantized), fmt="%d"
)  # used for tabular in latex with generator

plt.imshow(DCT2_quantized)
plt.show()
```

```
[[66.  1.  0.  0.  0.  0.  0.  0.]
 [-9.  0.  0.  0.  0.  0.  0.  0.]
 [ 3.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]
```



1c)

```
dequantized = q_table * DCT2_quantized

data_block_IDCT2 = idct2(dequantized)

print(data_block_IDCT2)
plt.imshow(data_block_IDCT2)
plt.show()

np.savetxt(
    "data_block_IDCT2.csv", (data_block_IDCT2), fmt="%d"
)  # used for tabular in latex with generator

PSNR = compute_psnr(data_block, data_block_IDCT2)
print("psnr: ", PSNR)

```

```
[[122.04159744 121.75124661 121.21474822 120.5137793  119.75505601
  119.05408709 118.5175887  118.22723787]
 [120.87413868 120.58378785 120.04728946 119.34632054 118.58759725
  117.88662833 117.35012994 117.05977911]
 [120.45901832 120.16866749 119.6321691  118.93120018 118.17247689
  117.47150797 116.93500958 116.64465875]
 [123.3230826  123.03273177 122.49623338 121.79526446 121.03654117
  120.33557225 119.79907386 119.50872303]
 [130.77236584 130.48201501 129.94551662 129.2445477  128.48582441
  127.78485549 127.2483571  126.95800627]
 [141.6727822  141.38243137 140.84593298 140.14496406 139.38624077
  138.68527184 138.14877346 137.85842263]
 [152.62277995 152.33242911 151.79593073 151.0949618  150.33623851
  149.63526959 149.09877121 148.80842037]
 [159.49167326 159.20132243 158.66482405 157.96385512 157.20513183
```

4 JPEG and JPEG2000

4d)


```
im_list_bike_pgm_jp2 = ['bike_jp2_005bpp.pgm','bike_jp2_02bpp.pgm','bike_jp2_04bpp.pg
im_list_bike_pgm_jp2_decoded_bbp = ['bike_jp2_0.1976bpp.pgm','bike_jp2_0.4543bpp.pgm'

im_list_cafe_pgm_jp2 = ['cafe_jp2_005bpp.pgm','cafe_jp2_02bpp.pgm','cafe_jp2_04bpp.pg
im_list_cafe_pgm_jp2_decoded_bbp = ['cafe_jp2_0296bpp.pgm','cafe_jp2_073bpp.pgm','caf

im_list_woman_pgm_jp2_decoded_bbp = ['woman_jp2_02bpp.pgm','woman_jp2_04bpp.pgm','wom

im_list_bike_pgm_jpeg_decoded = ['bike_jpeg_decoded_N5.pgm','bike_jpeg_decoded_N20.pg
im_list_cafe_pgm_jpeg_decoded = ['cafe_jpeg_decoded_N5.pgm','cafe_jpeg_decoded_N20.pg
im_list_woman_pgm_jpeg_decoded = ['woman_jpeg_decoded_N5.pgm','woman_jpeg_decoded_N20

im_list_bike_jp2 = ['bike_005bpp.jp2','bike_02bpp.jp2','bike_04bpp.jp2','bike_06bpp.j
im_list_cafe_jp2 = ['cafe_005bpp.jp2','cafe_02bpp.jp2','cafe_04bpp.jp2','cafe_06bpp.j

im_list_bike_jpg = ['bike_N5.jpg','bike_N20.jpg','bike_N40.jpg','bike_N60.jpg','bike_
im_list_cafe_jpg = ['cafe_N5.jpg','cafe_N20.jpg','cafe_N40.jpg','cafe_N60.jpg','cafe_
im_list_woman_jpg = ['woman_N5.jpg','woman_N20.jpg','woman_N40.jpg','woman_N60.jpg','
```

```
def showImage(im_list):
  for im in im_list:
    title = im
    f = open(im,'rb')
    im = plt.imread(f)
    plt.title(title)
    plt.imshow(im, cmap='gray')
    plt.show()
```

```
if False:
  showImage(im_list_bike_pgm_jp2)
  showImage(im_list_cafe_pgm_jp2)
```

```
  showImage(im_list_bike_pgm_jpeg_decoded)
  showImage(im_list_cafe_pgm_jpeg_decoded)
  showImage(im_list_bike_jp2)
  showImage(im_list_cafe_jp2)
  showImage(im_list_bike_jpg)
  showImage(im_list_cafe_jpg)
```

Find BPP

```
def get_list_of_BPP(paths):
  BPPs = []
  for path in paths:
    size_in_bytes = os.path.getsize(path)
    print(path, size_in_bytes*0.0009765625, "KB")
    size_in_bits = size_in_bytes*8
    BPPs.append(JPEG_BPP(2048, 2560, size_in_bits))

  return BPPs
```

```
cafe_BPPs = get_list_of_BPP(im_list_cafe_jpg)
bike_BPPs = get_list_of_BPP(im_list_bike_jpg)
woman_BPPs = get_list_of_BPP(im_list_woman_jpg)
```

```
cafe_N5.jpg 189.21875 KB
cafe_N20.jpg 466.6064453125 KB
cafe_N40.jpg 701.7587890625 KB
cafe_N60.jpg 908.095703125 KB
cafe_N80.jpg 1327.3720703125 KB
bike_N5.jpg 126.466796875 KB
bike_N20.jpg 290.7587890625 KB
bike_N40.jpg 452.517578125 KB
bike_N60.jpg 603.1982421875 KB
bike_N80.jpg 919.1376953125 KB
woman_N5.jpg 106.578125 KB
woman_N20.jpg 263.9462890625 KB
woman_N40.jpg 416.9189453125 KB
woman_N60.jpg 570.2421875 KB
woman_N80.jpg 896.3173828125 KB
```

```
print(cafe_BPPs)
print(bike_BPPs)
print(woman_BPPs)
```

```
[0.295654296875, 0.7290725708007812, 1.0964981079101563, 1.4188995361328125, 2.07401
[0.1976043701171875, 0.45431060791015626, 0.7070587158203125, 0.9424972534179688, 1.
[0.1665283203125, 0.41241607666015623, 0.6514358520507812, 0.89100341796875, 1.40049
```

```
def path_to_imlist(paths):
  image_collection = []
  for path in paths:
    f = open(path,'rb')
    im = plt.imread(f)
```

```
    image_collection.append(im)

  return image_collection
```

```
jpeg_bike_imgs = path_to_imlist(im_list_bike_pgm_jpeg_decoded)
jp2_bike_imgs = path_to_imlist(im_list_bike_pgm_jp2_decoded_bbp)

jpeg_cafe_imgs = path_to_imlist(im_list_cafe_pgm_jpeg_decoded)
jp2_cafe_imgs = path_to_imlist(im_list_cafe_pgm_jp2_decoded_bbp)

jpeg_woman_imgs = path_to_imlist(im_list_woman_pgm_jpeg_decoded)
jp2_woman_imgs = path_to_imlist(im_list_woman_pgm_jp2_decoded_bbp)

im_cafe_original = path_to_imlist(["cafe.pgm"])[0]
im_bike_original = path_to_imlist(["bike.pgm"])[0]
im_woman_original = path_to_imlist(["woman.pgm"])[0]
```

```
jpeg_quality = ["5", "20", "40", "60", "80"]

def compare_psnr(original, images):
  for i, quality in enumerate(jpeg_quality):
    psnr = compute_psnr(original, images[i])
    print('psnr for jpeg quality level', quality, psnr)
```

Bike

```
print("JPEG")
compare_psnr(im_bike_original, jpeg_bike_imgs)
print("JP2")
compare_psnr(im_bike_original, jp2_bike_imgs)
```

```
JPEG
psnr for jpeg quality level 5 24.6708302665451
psnr for jpeg quality level 20 29.637224110797867
psnr for jpeg quality level 40 32.17728153025021
psnr for jpeg quality level 60 33.95135473067994
psnr for jpeg quality level 80 36.925046349052984
JP2
psnr for jpeg quality level 5 28.446817305964487
psnr for jpeg quality level 20 32.980091981958395
psnr for jpeg quality level 40 35.621720410663656
psnr for jpeg quality level 60 37.71226822502957
psnr for jpeg quality level 80 40.95358376205617
```

Cafe

```
print("JPEG")
compare_psnr(im_cafe_original, jpeg_cafe_imgs)
print("JP2")
compare_psnr(im_cafe_original, jp2_cafe_imgs)
```

```
JPEG
psnr for jpeg quality level 5 21.691378202078237
psnr for jpeg quality level 20 26.503611241802368
psnr for jpeg quality level 40 29.213705946527384
psnr for jpeg quality level 60 31.199640170698554
psnr for jpeg quality level 80 34.672451157529295
JP2
psnr for jpeg quality level 5 23.983529928713452
psnr for jpeg quality level 20 29.36882671042407
psnr for jpeg quality level 40 32.88008973232255
psnr for jpeg quality level 60 35.12816362089012
psnr for jpeg quality level 80 39.68858463340644
```

Woman

```
print("JPEG")
compare_psnr(im_woman_original, jpeg_woman_imgs)
print("JP2")
compare_psnr(im_woman_original, jp2_woman_imgs)
```

```
JPEG
psnr for jpeg quality level 5 24.906864454111698
psnr for jpeg quality level 20 29.629634033350957
psnr for jpeg quality level 40 32.08678947082698
psnr for jpeg quality level 60 33.89549193425248
psnr for jpeg quality level 80 37.007761356958454
JP2
psnr for jpeg quality level 5 28.335104539134253
psnr for jpeg quality level 20 32.48822793480885
psnr for jpeg quality level 40 35.25514518827637
psnr for jpeg quality level 60 37.6386427595168
psnr for jpeg quality level 80 41.06731818707129
```

✓ 0s completed at 3:52 PM