ECE-6123 Nikita Makarov HW3

October 25, 2024

0.0.1 ECE-GY 6123

0.0.2 Nikita Makarov, Fall 2024

0.0.3 Homework 3

Problem 1a:

[2]:
```python
import numpy as np
import cv2
%matplotlib inline
from matplotlib import pyplot as plt

# Create Gaussian Pyramid function
def gaussian_pyramid(im_data: np.ndarray, J: int) -> list[np.ndarray]:
    '''
    Compute and return the J-level Gaussian pyramid decomposition of a␣
↪grayscale image.

    Parameters:
    im_data: 2D numpy array of the image data in grayscale
    J: int of the Gaussian pyramid level

    Returns:
    G: list of size J filled with 2D numpy arrays of the Gaussian pyramid, with␣
↪the first element being level 0
    '''

    # Create Gaussian pyramid structure
    G = [im_data] # Create list with last element as level J

    # Iterate over all remaining levels
    for _ in range(0,J):
        width = int(im_data.shape[1]/2) # Calculate new width
        height = int(im_data.shape[0]/2) # Calculate new height
        im_data = cv2.resize(im_data,(width,height),interpolation=cv2.
↪INTER_LINEAR) # Get next layer of image data
        G.insert(0, im_data) # Add new layer to front of list
```

```
    return G

# Create Laplacian Pyramid function
def laplacian_pyramid(im_data: np.ndarray, J: int) -> list[np.ndarray]:
    '''
    Compute and return the J-level Laplacian pyramid decomposition of a␣
 ↪grayscale image.

    Parameters:
    im_data: 2D numpy array of the image data in grayscale
    J: int of the Laplacian pyramid level

    Returns:
    L: list of size J filled with 2D numpy arrays of the Laplacian pyramid,␣
 ↪with the first element being level 0
    '''

    # Calculate Gaussian pyramid structure
    G = gaussian_pyramid(im_data, J)

    # Create Laplacian pyramid structure
    L = [G[0]] # Create list with first element as level 0

    # Iterate over all remaining levels
    for i in range(0,J):
        im_data = G[i+1] # Get first element in Gaussian pyramid
        (height,width) = im_data.shape # Get new width and height
        U = cv2.resize(G[i],(width,height),interpolation=cv2.INTER_CUBIC) #␣
 ↪Perform upscaling for next layer
        L.append(im_data - U) # Add new layer to end of list

    return L
```

Problem 1b:

[3]:
```
# Create Laplacian reconstruction function
def reconstruct_laplacian(L: list[np.ndarray]) -> list[np.ndarray]:
    '''
    Reconstruct and return a grayscale image from a Laplacian pyramid structure.

    Parameters:
    L: list of size J filled with 2D numpy arrays of the Laplacian pyramid,␣
 ↪with the first element being element 0

    Returns:
    R: list of size J filled with 2D numpy array of the reconstructed image in␣
 ↪grayscale
```

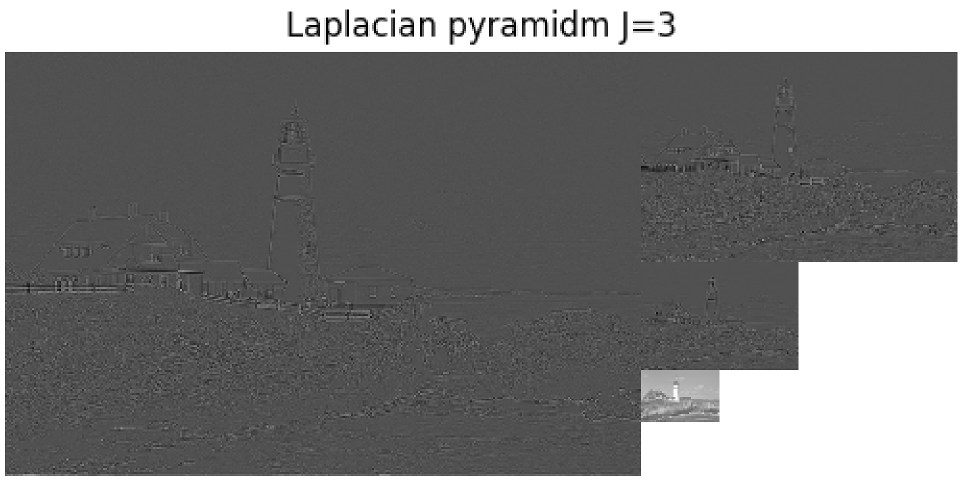
```python
    '''

    # Create reconstructed image structure
    R = [L[0]]

    # Iterate over levels of the laplacian pyramid
    for i in range(0,len(L)-1):
        (height,width) = L[i+1].shape # Get new width and height
        U = cv2.resize(R[i],(width,height),interpolation=cv2.INTER_CUBIC) #
 ↪Perform upscaling for next layer
        R.append(L[i+1] + U) # Calculate new reconstructed layer

    return R

# Create function for composite arrays for image display
def composite_image(P: list[np.ndarray]) -> np.ndarray:
    '''
    Create a composite array from a Gaussian or Laplacian pyramid structure.

    Parameters:
    P: list filled with 2D numpy arrays of the Gaussian or Laplacian pyramids,
 ↪with the first element being element 0

    Returns:
    C: 2D numpy array of the composite image
    '''

    # If pyramid has one element, just return that level
    if len(P) == 1:
        return P[0]

    # Create initial composite array
    height = max(np.shape(P[-1])[0],sum([np.shape(x)[0] for x in P[:-1]])) #
 ↪Total height of composite image
    width = np.shape(P[-1])[1] + np.shape(P[-2])[1] # Total width of composite
 ↪image
    C = np.full((height, width), np.nan) # Composite image array full of nan

    # Populate composite array
    height, width = np.shape(P[-1])
    C[:height, :width] = P[-1]
    height = 0
    P.reverse()
    for lvl in P[1:]:
        rows, cols = np.shape(lvl)
        C[height:height+rows, width:width+cols] = lvl
        height += rows
```

```python
    return C

# Test reconstruction with a sample image
image = np.asarray(cv2.imread('lighthouse.png', cv2.
 ↪IMREAD_GRAYSCALE),dtype=float)
J = 3
G = gaussian_pyramid(image, J) # Gaussian pyramid
L = laplacian_pyramid(image, J) # Laplacian pyramid
R = reconstruct_laplacian(L) # Reconstructed image

# Create figure for original image
plt.figure()
plt.imshow(image, interpolation='none', cmap='gray')
plt.title('Original image')
plt.axis('off')

# Create figure for composite Gaussian image
C = composite_image(G)
plt.figure()
plt.imshow(C, interpolation='none', cmap='gray')
plt.title(f'Gaussian pyramid, J={J}')
plt.axis('off')

# Create figure for composite Laplacian image
C = composite_image(L)
plt.figure()
plt.imshow(C, interpolation='none', cmap='gray')
plt.title(f'Laplacian pyramidm J={J}')
plt.axis('off')

# Create figure for reconstructed image
C = composite_image(R)
plt.figure()
plt.imshow(C, interpolation='none', cmap='gray')
plt.title('Reconstructed image')
plt.axis('off')
```

[3]:
```
(np.float64(-0.5), np.float64(1151.5), np.float64(511.5), np.float64(-0.5))
```

Original image

Gaussian pyramid, J=3



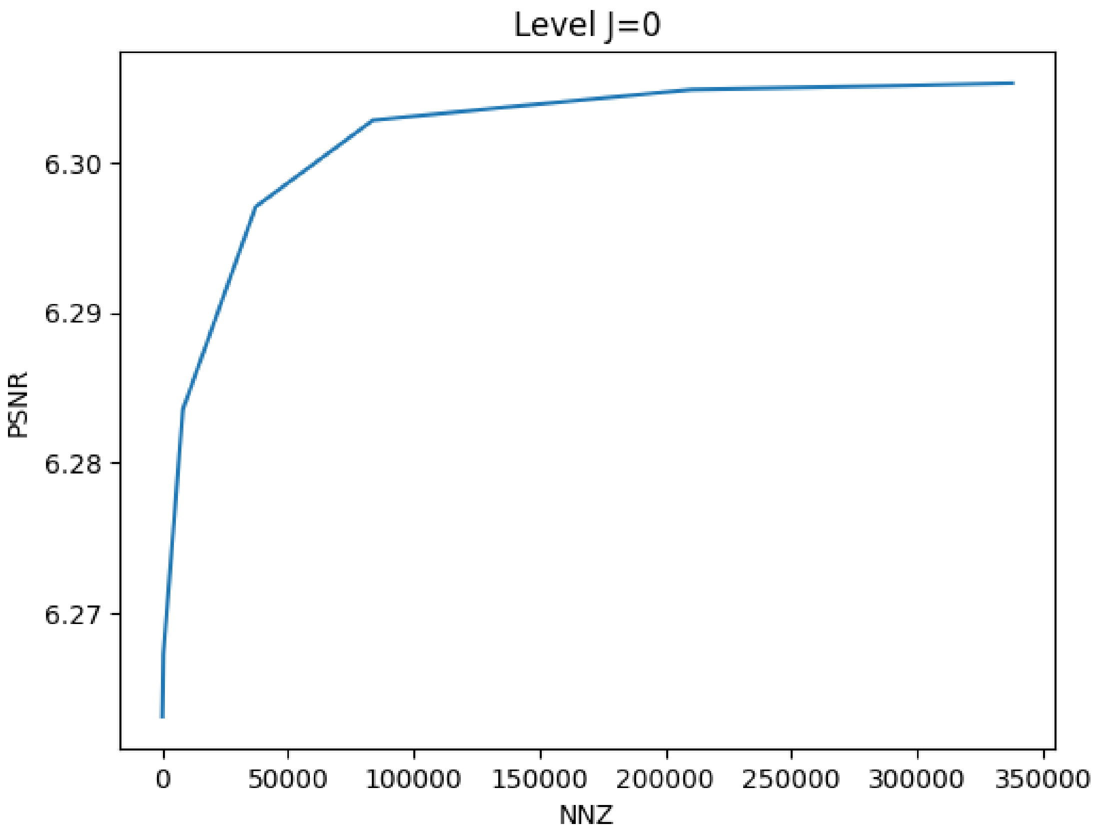
Problem 1c:

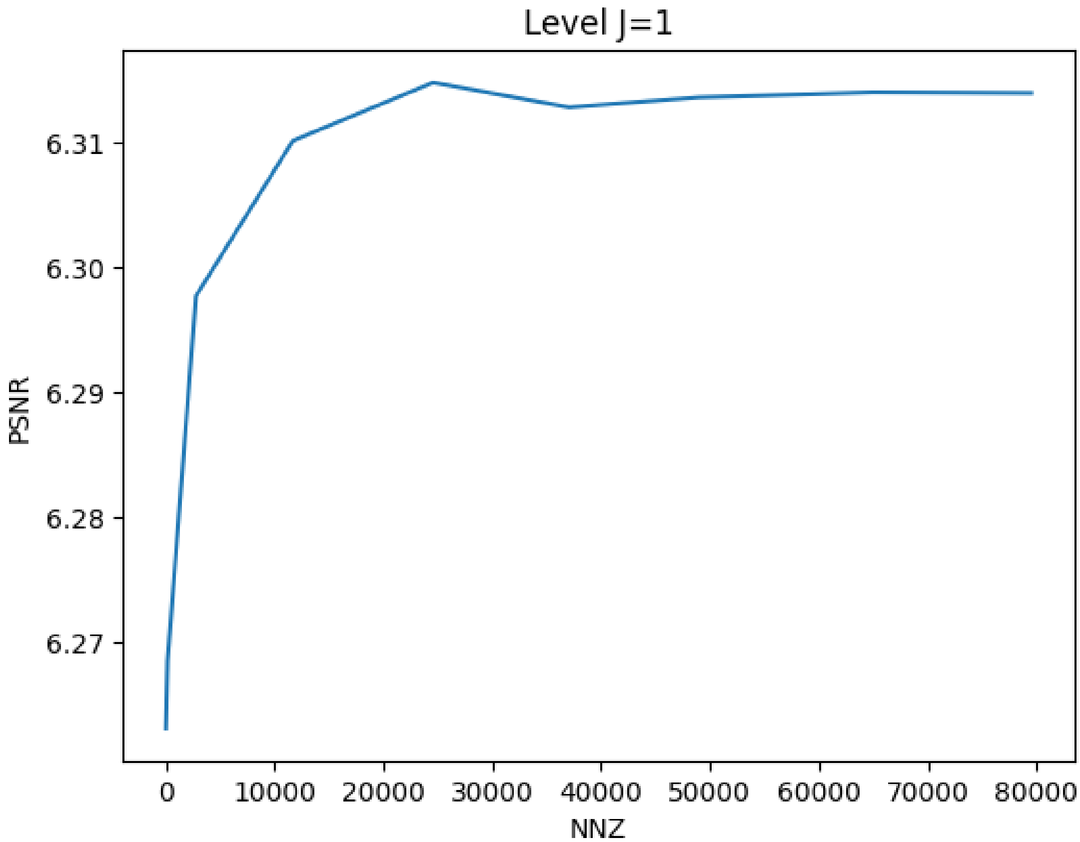
[4]:
```
# Create function for quantize pyramid
def quantize_pyramid(P: list[np.ndarray], q: float, ep: float) -> list[np.
↪ndarray]:
    '''
    Quantize the coefficients of the given pyramid with given step size and␣
↪given mean.

    Parameters:
    P: list filled with 2D numpy arrays of the Gaussian or Laplacian pyramids,␣
↪with the first element being element 0
    q: float of the quantization step size
```

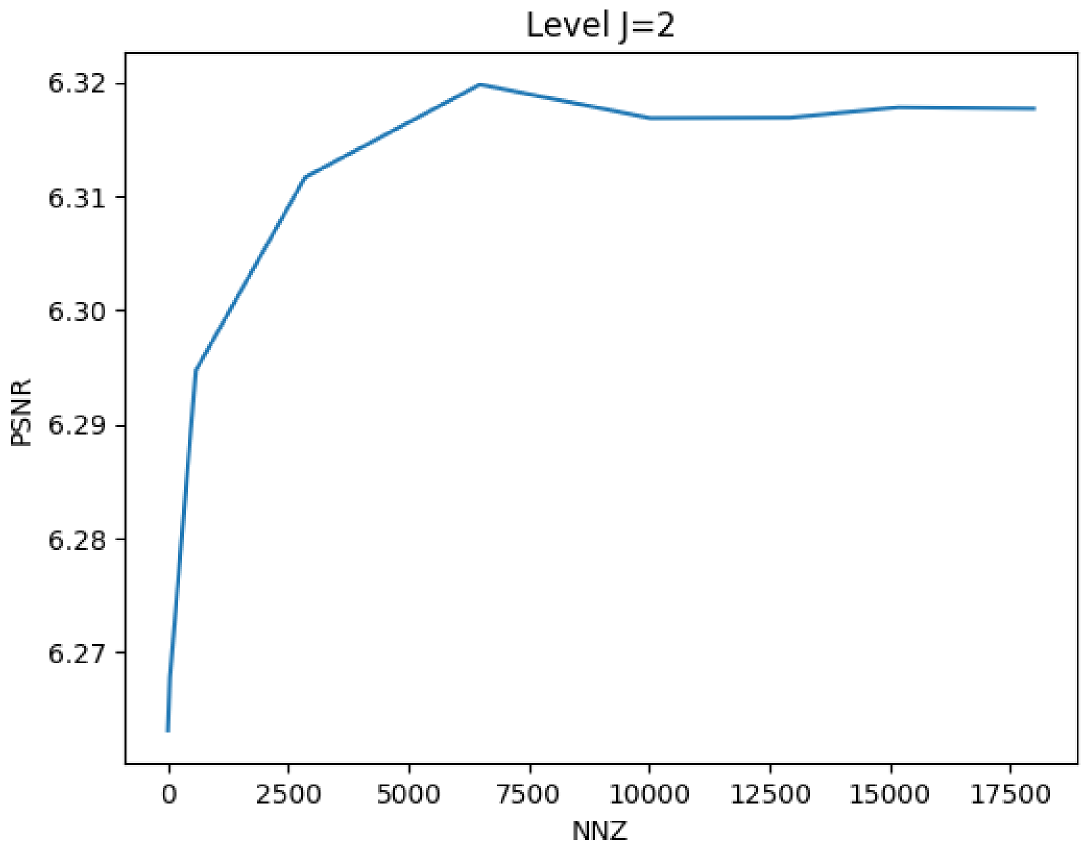
```
    ep: float of the mean of the coefficient map

    Returns:
    Q: list filled with 2D numpy arrays of the quantized pyramid, with the␣
↪first element being element 0
    '''

    # Create initial quantized pyramid
    Q = []

    # Iterate over given pyramid and fill quantize pyramid
    for lvl in P:
        Q.append(q*np.floor((lvl-ep)/q + 1/2) + ep)

    return Q
```

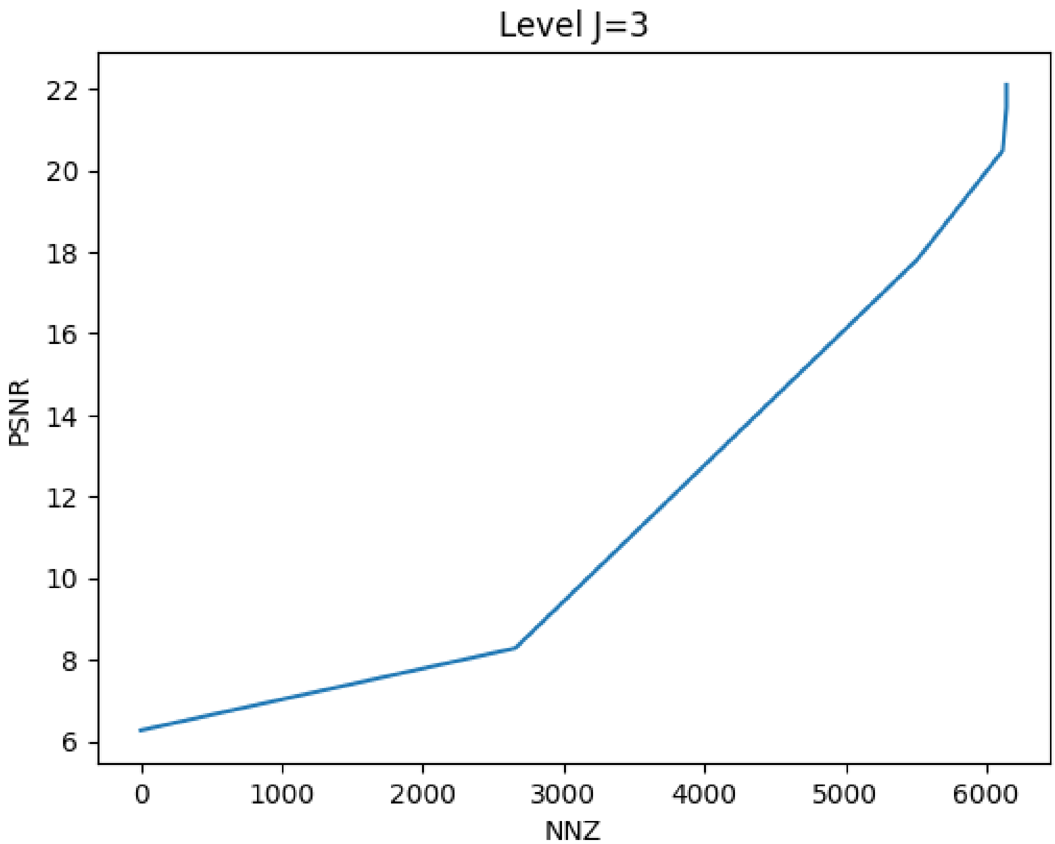
Problem 1d:

```
[5]: # Create initial PSNR and NNZ arrays
q = 9
PSNR = np.zeros((q+1,J+1))
NNZ = np.zeros((q+1,J+1))
[rows,cols] = np.shape(image)

# Calculate quantized pyramids
for n in range(q+1):
    Q = quantize_pyramid(L, 2**n, 0)
    R = reconstruct_laplacian(Q)

    # Calculate PSNR and NNZ for given levels
    for i in range(J+1):
        R_U = cv2.resize(R[i], (cols,rows))
        MSE = np.mean((image - R_U)**2) # Mean squared error
        PSNR[n,i] = 10*np.log10((255**2)/MSE) # Peak signal to noise ratio, for␣
↪image in range [0,255]
        NNZ[n,i] = np.sum(Q[i] != 0)

# Plot PSNR vs NNZ
for i in range(J+1):
    plt.figure()
    plt.plot(NNZ[:,i], PSNR[:,i])
    plt.xlabel('NNZ')
    plt.ylabel('PSNR')
    plt.title(f'Level J={i}')
```


Level J=0
PSNR
6.30
6.29
6.28
6.27
NNZ
0
50000
100000
150000
200000
250000
300000
350000


Level J=1
6.31
6.30
6.29
6.28
6.27
PSNR
0
10000
20000
30000
40000
50000
60000
70000
80000
NNZ


Level J=2
6.32
6.31
6.30
6.29
6.28
6.27
PSNR
0
2500
5000
7500
10000
12500
15000
17500
NNZ



As the pyramid level increases, the image size gets smaller, since level 0 is the original image. At a certain point, specifically J=3, the PSNR for the given pyramid level increases at an exponential rate. This is somewhat expected, since as the pyramid level increases, we loose more and more of the data from the original image, so the potential error also increases.

Problem 1e:

For J=3, the curve shape of the PSNR vs NNZ plot has flipped. Instead of the PSNR reaching an equilibrium with the number of non-zero elements, it starts increasing exponentially. It appears that at around 6000 number of non-zeros the PSNR starts increasing faster than the NNZ.

Problem 2a:

[6]:
```
import pywt
from copy import copy

# Create wavelet transform function
def wavelet_transform(im_data: np.ndarray, J: int) -> list:
    '''
    Compute and return the J-level wavelet transform of a grayscale image.
```

```
    Parameters:
    im_data: 2D numpy array of the image data in grayscale
    J: int of the levels

    Returns:
    coeffs: list of the approximation and coefficients for each level
    '''

    coeffs = []

    # Calculate coefficients
    for _ in range(J):
        (im_data, (cH, cV, cD)) = pywt.dwt2(im_data, 'haar')
        coeffs.insert(0, (cH, cV, cD))

    coeffs.insert(0, im_data)

    return coeffs

# Reconstruct image from wavelet transform
def reconstruct_wavelet(coeffs: list) -> np.ndarray:
    '''
    Reconstruct an image from the J-level wavelet approximation and␣
 ↪coefficients.

    Parameters:
    coeffs: list of the approximation and coefficients for each level

    Returns:
    R: 2D numpy array of the reconstructed image data
    '''

    R = coeffs[0] # Use last level approximation to start

    # Reconstruct the image
    for i in range(1,len(coeffs)):
        (cH, cV, cD) = coeffs[i]
        R = pywt.idwt2((R, (cH, cV, cD)), 'haar')

    return R
```

Problem 2b:

[21]:
```
# Create initial PSNR and NNZ arrays
q = 9
PSNR = np.zeros((q+1,J+1))
NNZ = np.zeros((q+1,J+1))
```

```python
# Create figure for original image
plt.figure()
plt.imshow(image, interpolation='none', cmap='gray')
plt.title('Original image')
plt.axis('off')

# Create figure for composite wavelet image
Coeffs = wavelet_transform(image, J)
image_full = reconstruct_wavelet(Coeffs)
plt.figure()
plt.imshow(image_full, interpolation='none', cmap='gray')
plt.title(f'Reconstructed image full')
plt.axis('off')

# Calculate quantized wavelets
for n in range(q):
    Coeffs_temp = copy(Coeffs)
    for i in range(1,4):
        (cH, cV, cD) = Coeffs_temp[i]
        [cH, cV, cD] = quantize_pyramid([cH, cV, cD], 2**n, 0)
        Coeffs_temp[i] = (cH, cV, cD)
        NNZ[n,0] += (np.sum(cH != 0) + np.sum(cV != 0) + np.sum(cD != 0))
    image_full = reconstruct_wavelet(Coeffs_temp)
    MSE = np.mean((image - image_full)**2) # Mean squared error
    PSNR[n,0] = 10*np.log10((255**2)/MSE) # Peak signal to noise ratio

# Plot PSNR and NNZ
plt.figure()
plt.plot(NNZ[:,0], PSNR[:,0])
plt.xlabel('NNZ')
plt.ylabel('PSNR')
plt.title('Level J=0')

# Create figure for level 1 wavelet image
Coeffs[-1] = tuple([np.zeros_like(v) for v in Coeffs[-1]])
image_1 = reconstruct_wavelet(Coeffs)
plt.figure()
plt.imshow(image_1, interpolation='none', cmap='gray')
plt.title(f'Reconstructed image 1')
plt.axis('off')

# Calculate quantized wavelets
for n in range(q):
    Coeffs_temp = copy(Coeffs)
    for i in range(1,4):
        (cH, cV, cD) = Coeffs_temp[i]
```

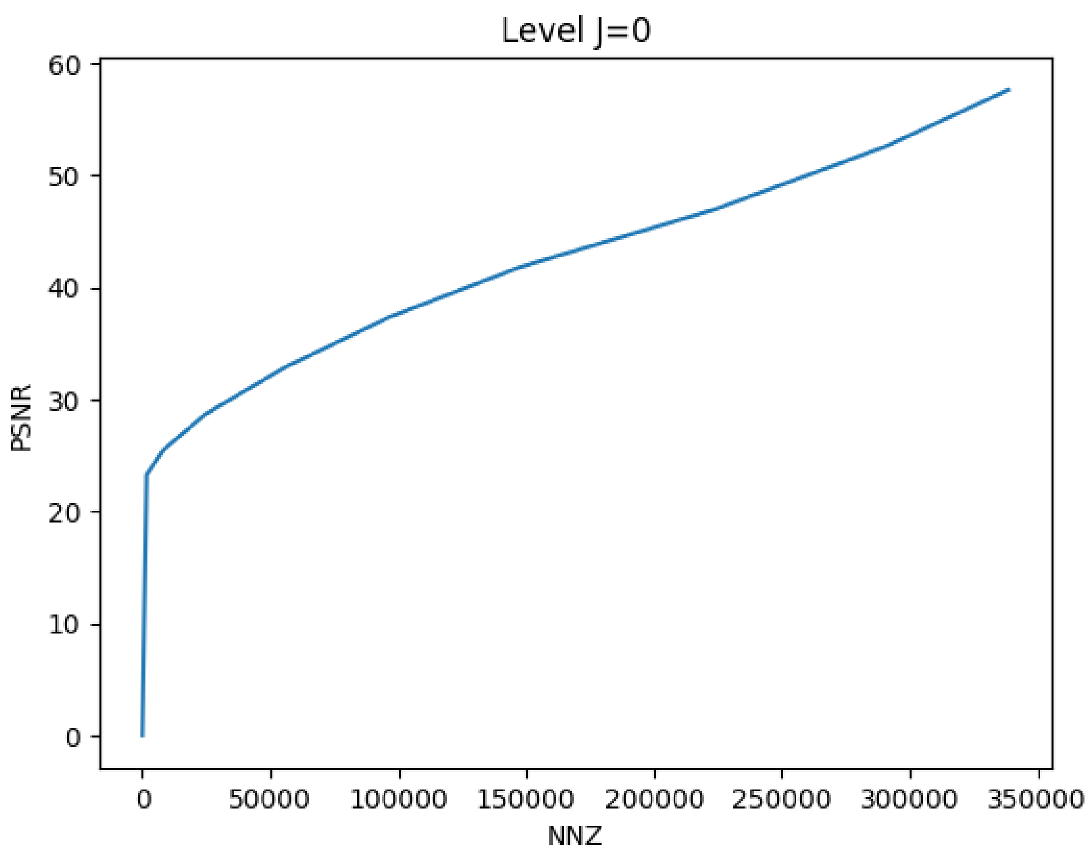
```
        [cH, cV, cD] = quantize_pyramid([cH, cV, cD], 2**n, 0)
        Coeffs_temp[i] = (cH, cV, cD)
        NNZ[n,1] += (np.sum(cH != 0) + np.sum(cV != 0) + np.sum(cD != 0))
    image_full = reconstruct_wavelet(Coeffs_temp)
    MSE = np.mean((image - image_full)**2) # Mean squared error
    PSNR[n,1] = 10*np.log10((255**2)/MSE) # Peak signal to noise ratio

# Plot PSNR and NNZ
plt.figure()
plt.plot(NNZ[:,1], PSNR[:,1])
plt.xlabel('NNZ')
plt.ylabel('PSNR')
plt.title('Level J=1')

# Create figure for level 2 wavelet image
Coeffs[-2] = tuple([np.zeros_like(v) for v in Coeffs[-2]])
image_2 = reconstruct_wavelet(Coeffs)
plt.figure()
plt.imshow(image_2, interpolation='none', cmap='gray')
plt.title(f'Reconstructed image 2')
plt.axis('off')

# Calculate quantized wavelets
for n in range(q):
    Coeffs_temp = copy(Coeffs)
    for i in range(1,4):
        (cH, cV, cD) = Coeffs_temp[i]
        [cH, cV, cD] = quantize_pyramid([cH, cV, cD], 2**n, 0)
        Coeffs_temp[i] = (cH, cV, cD)
        NNZ[n,2] += (np.sum(cH != 0) + np.sum(cV != 0) + np.sum(cD != 0))
    image_full = reconstruct_wavelet(Coeffs_temp)
    MSE = np.mean((image - image_full)**2) # Mean squared error
    PSNR[n,2] = 10*np.log10((255**2)/MSE) # Peak signal to noise ratio

# Plot PSNR and NNZ
plt.figure()
plt.plot(NNZ[:,2], PSNR[:,2])
plt.xlabel('NNZ')
plt.ylabel('PSNR')
plt.title('Level J=2')

# Create figure for level 3 wavelet image
Coeffs[-3] = tuple([np.zeros_like(v) for v in Coeffs[-3]])
image_3 = reconstruct_wavelet(Coeffs)
plt.figure()
plt.imshow(image_3, interpolation='none', cmap='gray')
plt.title(f'Reconstructed image 3')
```

```python
plt.axis('off')

# Calculate quantized wavelets
for n in range(q):
    Coeffs_temp = copy(Coeffs)
    for i in range(1,4):
        (cH, cV, cD) = Coeffs_temp[i]
        [cH, cV, cD] = quantize_pyramid([cH, cV, cD], 2**n, 0)
        Coeffs_temp[i] = (cH, cV, cD)
        NNZ[n,3] += (np.sum(cH != 0) + np.sum(cV != 0) + np.sum(cD != 0))
    image_full = reconstruct_wavelet(Coeffs_temp)
    MSE = np.mean((image - image_full)**2) # Mean squared error
    PSNR[n,3] = 10*np.log10((255**2)/MSE) # Peak signal to noise ratio

# Plot PSNR and NNZ
plt.figure()
plt.plot(NNZ[:,3], PSNR[:,3])
plt.xlabel('NNZ')
plt.ylabel('PSNR')
plt.title('Level J=3')
```

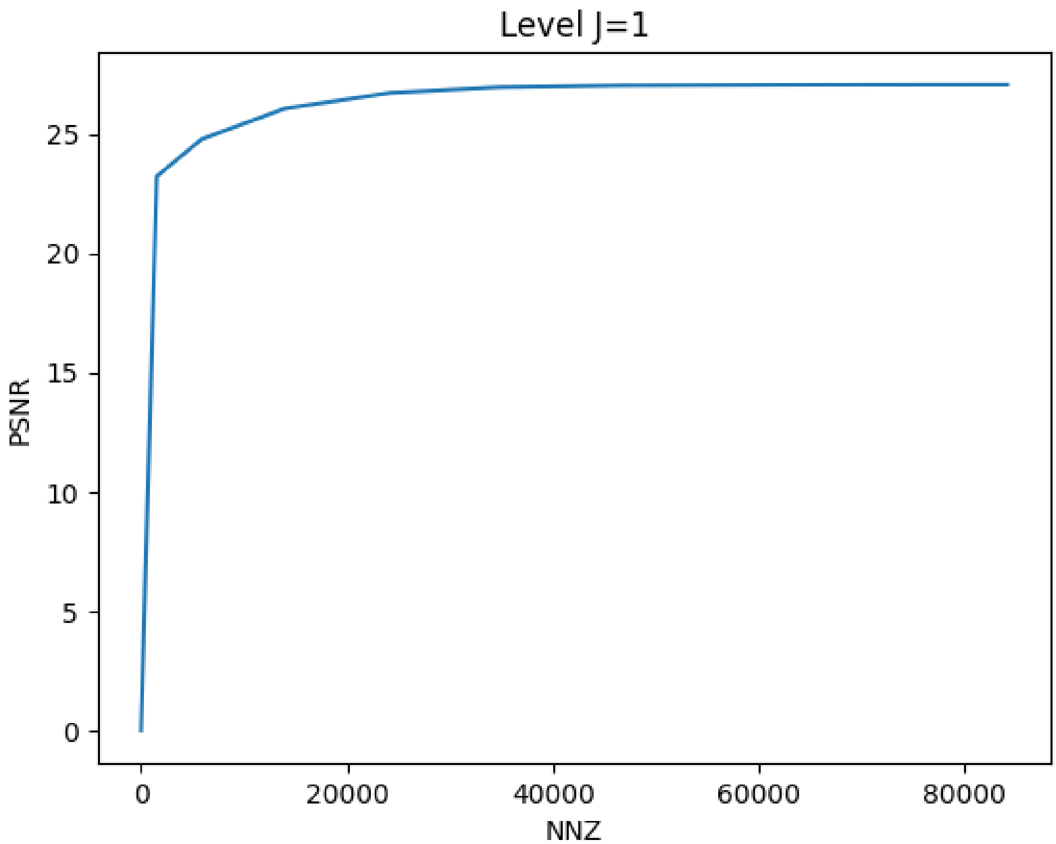
[21]:

```
Text(0.5, 1.0, 'Level J=3')
```

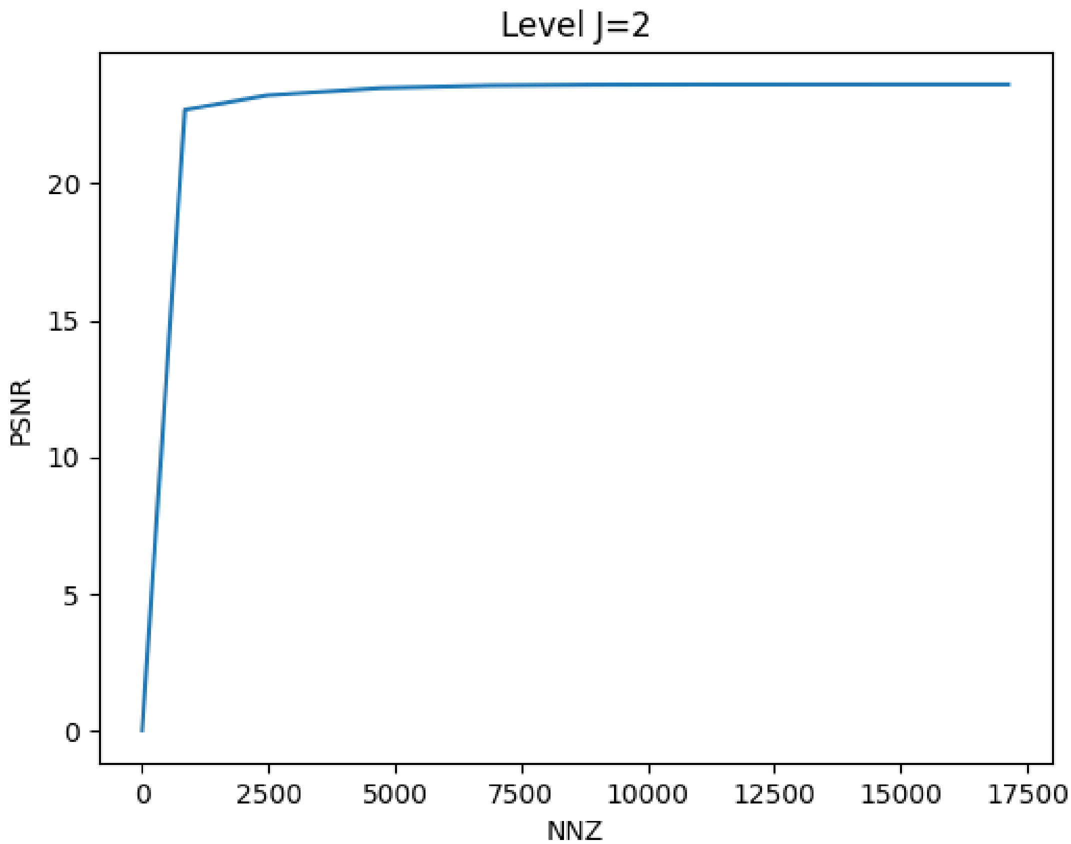

Reconstructed image full


Level J=0
PSNR
NNZ
0
10
20
30
40
50
60
0
50000
100000
150000
200000
250000
300000
350000

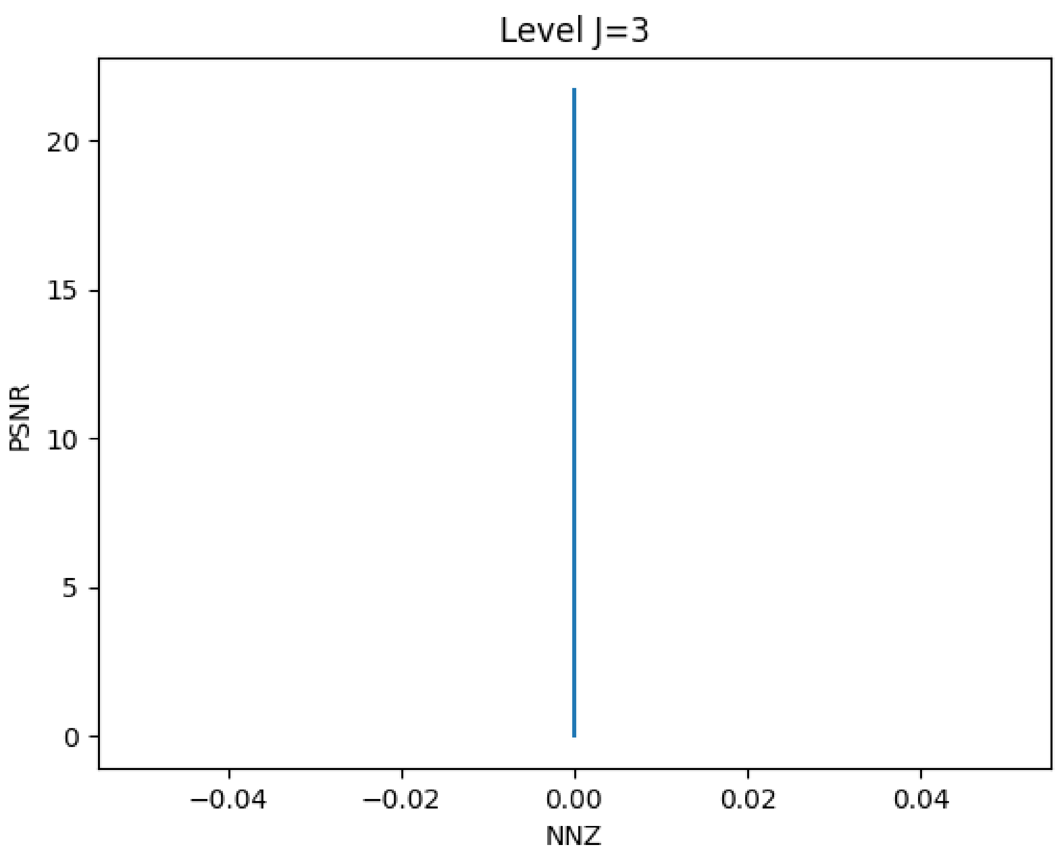
Reconstructed image 1



Reconstructed image 2


Level J=2
PSNR
NNZ
0
5
10
15
20
0
2500
5000
7500
10000
12500
15000
17500


Reconstructed image 3


Level J=3
PSNR
NNZ
0
5
10
15
20
−0.04
−0.02
0.00
0.02
0.04