Homework8_DS-GA1013

April 20, 2021

Name: Zhuoyuan Xu

NetID: zx1137

Problem 1

(a) The given definition of discrete Haar scaling function $\varphi_{2^k,p}$ is an orthonormal basis for V_k.

First, $\varphi_{2^k,p}$ satisfies the definition of $V_k = \{x \in R^{2^n} : x[i] = x[j] \text{ if } \lfloor i/2^k \rfloor = \lfloor j/2^k \rfloor\}$.

For any indices $i, j \in \{p2^k, p2^k + 1, ..., (p+1)2^k - 1\}$,

$$\lfloor i/2^k \rfloor = \lfloor j/2^k \rfloor = p$$

and

$$\varphi_{2^k,p}[i] = \varphi_{2^k,p}[j] = \frac{1}{\sqrt{2^k}}$$

and the other 0 elements also satisfies the definition.

Second, $\varphi_{2^k,p}$ is orthonormal. Since the norm

$$||\varphi_{2^k,p}|| = (\sum_{i=p2^k}^{(p+1)2^k-1} (\frac{1}{\sqrt{2^k}})^2) = (2^k)(\frac{1}{2^k}) = 1$$

$\varphi_{2^k,p}$ has an unit length. Meanwhile, because $\varphi_{2^k,p}$ is non-zero only when $j \in \{p2^k, p2^k + 1, ..., (p+1)2^k - 1\}$, 2 vectors of the form $\varphi_{2^k,p}$ with different p would have a dot product equals 0. Suppose with have 2 different positions p and q, then

$$\begin{aligned} < \varphi_{2^k,p}, \varphi_{2^k,q} > &= \sum_{i=p2^k}^{(p+1)2^k-1} (\varphi_{2^k,p}[i]\varphi_{2^k,q}[i]) + \sum_{j=q2^k}^{(q+1)2^k-1} (\varphi_{2^k,p}[j]\varphi_{2^k,q}[j]) \\ &= \sum_{i=p2^k}^{(p+1)2^k-1} (\varphi_{2^k,p}[i]0) + \sum_{j=q2^k}^{(q+1)2^k-1} (0\varphi_{2^k,q}[j]) \\ &= 0 \end{aligned}$$

Then, since $\varphi_{2^k,p}$ is orthonormal, we know it is linearly independent which shows it spans V_k. Therefore, $\varphi_{2^k,p}$ is an orthonormal basis of V_k.

Also, the dimension of $\varphi_{2^k,p}$ would be 2^{n-k}, since we have 2^{n-k} segments of 2^k components that are constant.

(b) Given that $V_k = \{x \in R^{2^n} : x[i] = x[j] \text{ if } \lfloor i/2^k \rfloor = \lfloor j/2^k \rfloor\}$, V_k keeps information on the segment of indices $[p2^k, (p+1)2^k)$ where p is a chosen position. The segment contains 2^k components and all of them have the same value of $\frac{1}{\sqrt{2^k}}$.

If we set an arbitrary x which we want to project onto V_k, and $\varphi_{2^k,p} \in V_k$,

$$proj_{V_k} x = \sum_{p=0}^{2^{n-k}-1} < x, \varphi_{2^k,p} > \varphi_{2^k,p}$$

Since we know that $\varphi_{2^k,p}$ has non-zero constant value $\frac{1}{\sqrt{2^k}}$ on $[p2^k, (p+1)2^k)$, and the basis of V_k has non-overlapping support, the result of this projection will be $\frac{x[j]}{2^k}$ on $j \in [p2^k, (p+1)2^k)$. This is equivalent to averaging the values on the segment $[p2^k, (p+1)2^k)$.

$$proj_{V_k} x = \sum_{p=0}^{2^{n-k}-1} < x, \varphi_{2^k,p} > \varphi_{2^k,p} = \sum_{j=q2^k}^{(p+1)2^k-1} \frac{1}{\sqrt{2^k}} x[j] \frac{1}{\sqrt{2^k}} = \sum_{j=q2^k}^{(p+1)2^k-1} \frac{1}{2^k} x[j]$$

This process can also be understood as minimizing the mean squared error of an arbitrary x to an element in V_k, and by intuition the result is consistent.

(c) For V_{k+1}, the j-th element of the orthonormal basis can be written as

$$\begin{aligned} \varphi_{2^{k+1},p}[j] &= \begin{cases} 1/\sqrt{2^{k+1}} & \text{if } j \in \{p2^{k+1}, p2^{k+1}+1, ..., (p+1)2^{k+1}-1\} \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} 1/\sqrt{2^{k+1}} & \text{if } j \in \{2p2^k, 2p2^k+1, ..., 2(p+1)2^k-1\} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

So the i-th and j-th elements of $\varphi_{2^{k+1},p}$ that are non-zero also have $\lfloor i/2^k \rfloor = \lfloor j/2^k \rfloor = 2p$ with equal value $\frac{1}{\sqrt{2^{k+1}}}$, and other zero elements also satisfy the definition of V_k. This implies if $\varphi_{2^{k+1},p} \in V_{k+1}$, it also $\varphi_{2^{k+1},p} \in V_k$.

Therefore, $V_{k+1} \subset V_k$

(d) An orthonormal basis of $W_{k+1} = \{x \in V_k :< x, y >= 0 \text{ for all } y \in V_{k+1}\}$ can be $\mu_{2^{k+1},p}$.

First, $\mu_{2^{k+1},p}$ satisfies the definition of $W_{k+1} = \{x \in V_k :< x, y >= 0 \text{ for all } y \in V_{k+1}\}$. $\mu_{2^{k+1},p} \in V_k$ because it has constant values on segments of size 2^k given the definition.

Also, for an arbitrary $y = \varphi_{2^{k+1},p} \in V_{k+1}$ and $x = \mu_{2^{k+1},p} \in V_k$,

$$< \varphi_{2^{k+1},p}, \mu_{2^{k+1},p} >= \sum_{p2^{k+1}}^{p2^{k+1}+2^k-1} (\frac{-1}{\sqrt{2^{k+1}}} \frac{1}{\sqrt{2^{k+1}}}) + \sum_{p2^{k+1}+2^k}^{(p+1)2^{k+1}-1} (\frac{1}{\sqrt{2^{k+1}}} \frac{1}{\sqrt{2^{k+1}}}) = 0$$

The negative and positive terms cancel out.

Second, $\mu_{2^{k+1},p}$ is orthonormal. Since the norm

$$||\mu_{2^{k+1},p}|| = \sum_{p2^{k+1}}^{p2^{k+1}+2^k-1} (\frac{-1}{\sqrt{2^{k+1}}})^2 + \sum_{p2^{k+1}+2^k}^{(p+1)2^{k+1}-1} (\frac{1}{\sqrt{2^{k+1}}})^2 = \frac{1}{2}(2^{k+1})(\frac{1}{2^{k+1}}) + \frac{1}{2}(2^{k+1})(\frac{1}{2^{k+1}}) = 1$$

$\mu_{2^{k+1},p}$ has an unit length. Meanwhile, because $\mu_{2^{k+1},p}$ is non-zero only when $j \in \{p2^{k+1}, p2^{k+1}+1, ..., (p+1)2^{k+1}-1\}$, 2 vectors of the form $\mu_{2^{k+1},p}$ with different p would have a dot product equals 0.

$$\begin{aligned} < \mu_{2^{k+1},p}, \mu_{2^{k+1},q} > &= \sum_{i=p2^{k+1}}^{(p+1)2^{k+1}-1} (\mu_{2^{k+1},p}[i]\mu_{2^{k+1},q}[i]) + \sum_{j=q2^{k+1}}^{(q+1)2^{k+1}-1} (\mu_{2^{k+1},p}[j]\mu_{2^{k+1},q}[j]) \\ &= \sum_{i=p2^{k+1}}^{(p+1)2^{k+1}-1} (\mu_{2^{k+1},p}[i]0) + \sum_{j=q2^{k+1}}^{(q+1)2^{k+1}-1} (0\mu_{2^{k+1},q}[j]) \\ &= 0 \end{aligned}$$

Then, since $\mu_{2^{k+1},p}$ is orthonormal, we know it is linearly independent which shows it spans W_{k+1}. Therefore, $\mu_{2^{k+1},p}$ is an orthonormal basis of W_{k+1}.

(e) We can write R^{2^n} as direct sum of subspaces

$$R^{2^n} = V_k \oplus W_{\leq k}$$

and $W_{\leq k}$ as a direct sum of subspaces

$$W_{\leq k} = V_k \oplus W_k \oplus ... \oplus W_2 \oplus W_1$$

From the previous question, W_k is spanned by $\mu_{2^k,p}$. If we define $B_{W_{\leq k}}$ to be the basis of $W_{\leq k}$, the basis of a direct sum can be $B_{W_{\leq k}} = \bigcup_{i=1}^{k} B_{W_j} = \bigcup_{i=1}^{k} \mu_{2^i,p}$.

From the previous question, B_{W_i} has 2^{n-i} orthonormal vectors. For 2 arbitrary number $n, m \in [1, k]$, and $m < n$, set $x \in B_{W_m}$ and $y \in B_{W_n}$. We know that since $y \in W_n$, $y \in W_{n-1}$ as well. Thus if $m = n-1$, $y \in W_m$ and x and y will be orthogonal. Similarly, since we have $V_n \subset V_{n-1} \subset V_{n-2} \subset ... \subset V_m$, $y \in W_m$ and x and y should be orthogonal. This shows the $B_{W_{\leq k}} = \bigcup_{i=1}^{k} B_{W_j}$ are orthogonal and unit length, i.e. orthonormal.

Problem 2

(a)

[90]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

"""
Implements the Haar wavelet function (psi) at scale 2**s, position p.
Arguments:
s: the scale of the wavelet is 2**s
p: the position of the wavelet
n: the length of the returned vector is 2**n
Returns:
A numpy array of length 2**n containing the Haar wavelet with
scale 2**s at position p.
"""
```

```
def wavelet(s,p,n) :
    element = 1/np.sqrt(2**s)
    mu = np.zeros(2**n)

    mu[p*(2**s):p*(2**s)+2**(s-1)] = -element
    mu[p*(2**s)+2**(s-1):(p+1)*(2**s)] = element

    return mu

"""
Implements the Haar scaling function (phi) at scale 2**s, position p.
Arguments:
s: the scale of the scaling function is 2**s
p: the position of the scaling function
n: the length of the returned vector is 2**n
Returns:
A numpy array of length 2**n containing the Haar scaling function with
scale 2**s at position p.
"""
def scaling(s,p,n) :
    element = 1/np.sqrt(2**s)
    phi = np.zeros(2**n)

    phi[p*(2**s):(p+1)*(2**s)] = element

    return phi
```

(b)

[91]:
```
"""
Orthogonally projects the vector x onto the subspace V_k of vectors
that are constant on patches of length 2**k.
Arguments:
x: numpy array of data (of length 2**n for some n)
k: 0 <= k <= n
n: x has length 2**n
Returns:
A numpy array of length 2**n containing the orthogonal projection of x onto V_k.
"""
def projectV(x,k,n) :
    res = np.zeros(2**n)

    for i in range(0, 2**n, 2**k):
        segment = x[i:i+2**k]
        res[i:i+2**k] = np.sum(segment)/len(segment)
    return res
```

(c)

[116]:
```
"""
Orthogonally projects the vector x onto the subspace W_k of Haar wavelets of␣
 ↪scale 2**k.
Arguments:
x: numpy array of data (of length 2**n for some n)
k: 1 <= k <= n
n: x has length 2**n
Returns:
A numpy array of length 2**n containing the orthogonal projection of x onto W_k.
"""
def projectW(x,k,n) :
    res = np.zeros(2**n)

    v_part = projectV(x, k-1, n)
    for p in range(2**(n-k)):
        s = (v_part[p*(2**k)+2**(k-1)] - v_part[p*(2**k)])/2
        res[p*(2**k):p*(2**k)+2**(k-1)] = -s
        res[p*(2**k)+2**(k-1):(p+1)*(2**k)] = s
    return res
```

(d)

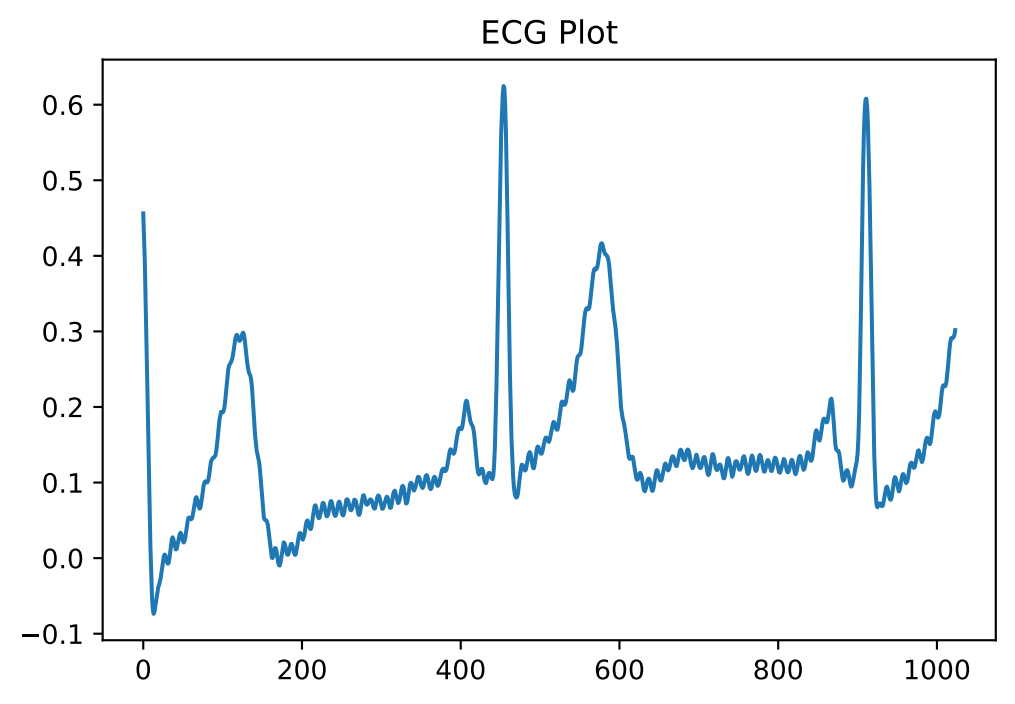
[122]:
```
"""
Computes the wavelet coefficients of x at scale 2**s.
Arguments:
x: numpy array of data (of length 2**n for some n)
s: 1 <= s <= n
n: x has length 2**n
Returns:
A numpy array of length 2**(n-s) containing the wavelet coefficents
at scale 2**s in positions p=0,1,...
"""
def wavelet_coeffs(x,s,n) :
    res = np.zeros(2**(n-s))
    element = 1/np.sqrt(2**s)

    for p in range(0, 2**(n-s)):
        res[p] = element*(np.sum(x[p*(2**s):(p+1)*(2**s)]))

    return res
```

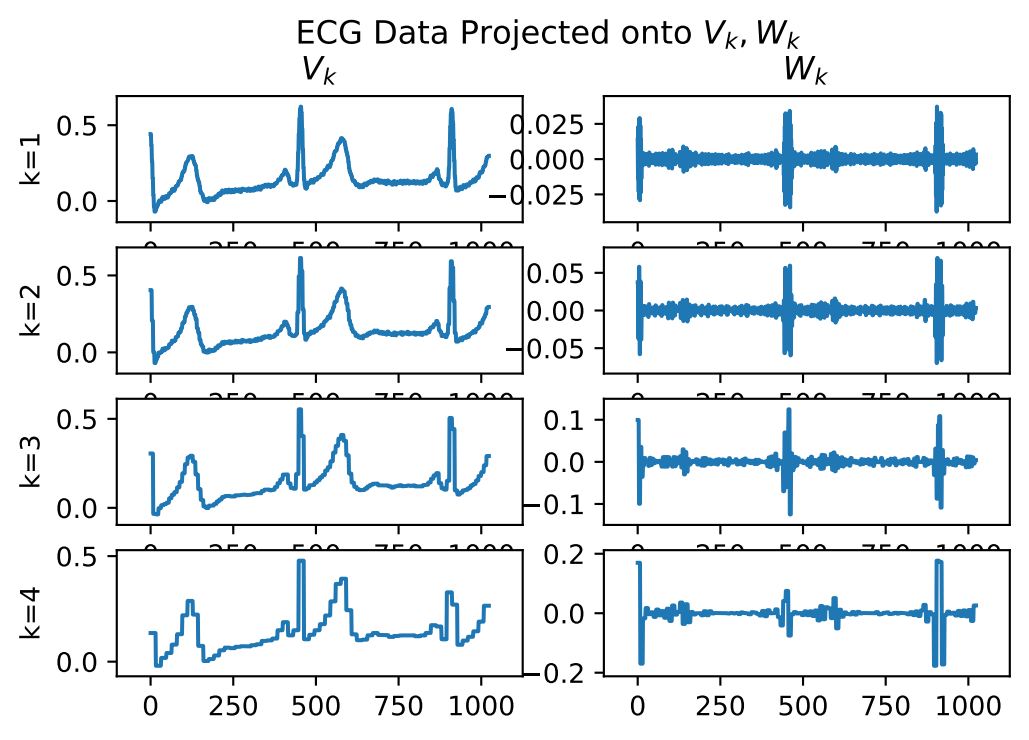
(e)

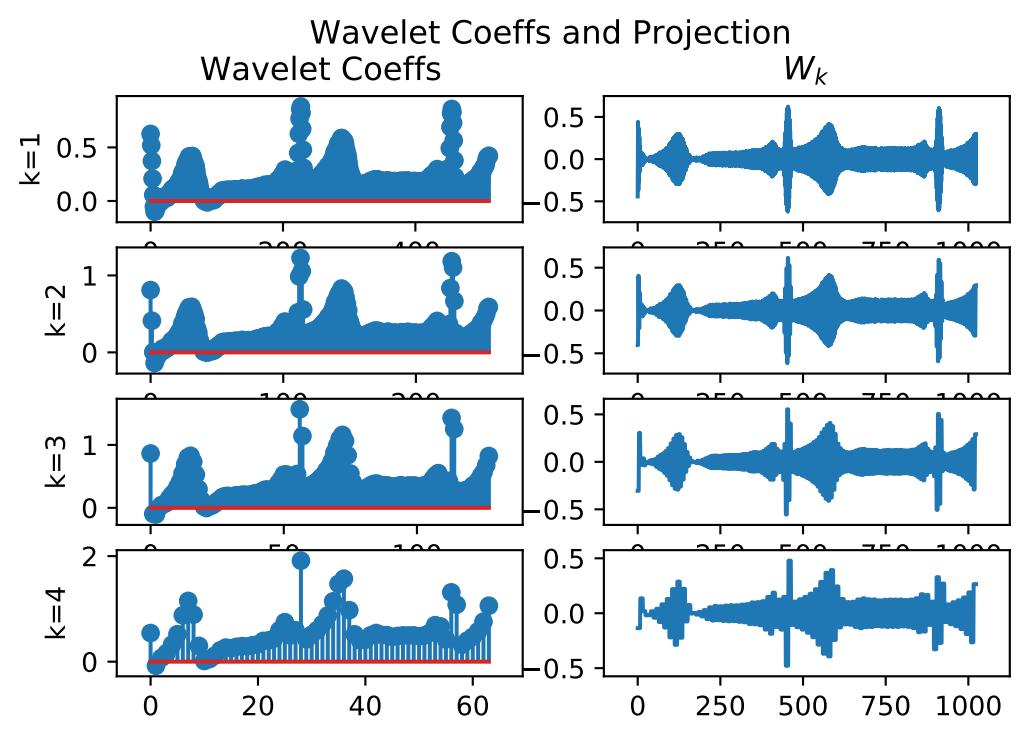
[123]:
```
def projectWavelet(w,s,n) :
    ws = []
    for p in range(len(w)):
        ws.append(wavelet(s,p,n))
    return np.dot(np.array(ws).T,w)
```

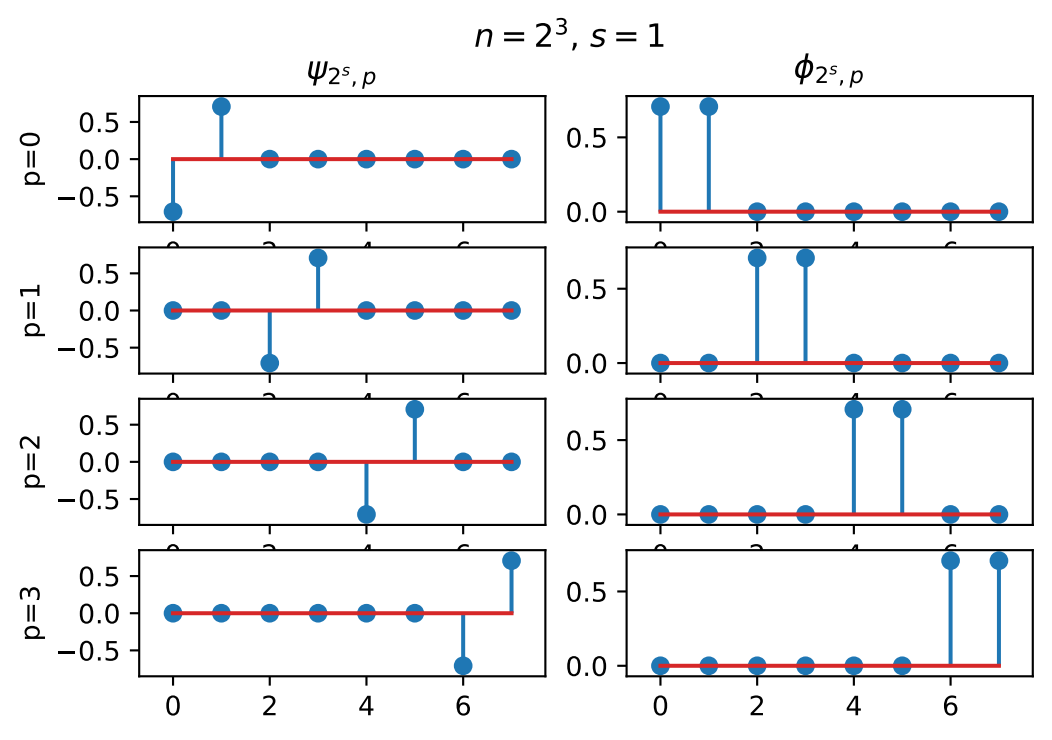
```
def plot_psiphi(s,n) :
    fig,axes = plt.subplots(2**(n-s),2)
    for p in range(len(axes)) :
        wax,sax = axes[p,0],axes[p,1]
        wax.stem(range(2**n),wavelet(s,p,n), use_line_collection = True)
        wax.margins(.1,.1)
        sax.stem(range(2**n),scaling(s,p,n), use_line_collection = True)
        sax.margins(.1,.1)
        wax.set_ylabel('p=%d'%p)
    axes[0,0].set_title('$\psi_{2^s,p}$')
    axes[0,1].set_title('$\phi_{2^s,p}$')
    fig.suptitle('$n=2^%d$, $s=%d$'%(n,s))
    plt.savefig('psiphiplot_%d_%d.pdf'%(s,n),bbox_inches='tight')
    plt.close()

def plot_downsampling(x,m,n) :
    fig,axes = plt.subplots(m-1,2)
    for k in range(1,m) :
        vax = axes[k-1,0]
        wax = axes[k-1,1]
        vax.plot(projectV(x.copy(),k,n))
        vax.margins(.1,.1)
        vax.set_ylabel('k=%d'%k)
        wax.plot(projectW(x.copy(),k,n))
        wax.margins(.1,.1)
    axes[0,0].set_title('$V_k$')
    axes[0,1].set_title('$W_k$')
    fig.suptitle('ECG Data Projected onto $V_k,W_k$')
    plt.savefig('ecg_project.pdf',bbox_inches='tight')
    plt.close()

def plot_wavelet(x,m,n) :
    fig,axes = plt.subplots(m-1,2)
    for k in range(1,m) :
        ax = axes[k-1,0]
        wax = axes[k-1,1]
        w = wavelet_coeffs(x.copy(),k,n)
        ax.stem(w, use_line_collection = True)
        ax.margins(.1,.1)
        ax.set_ylabel('k=%d'%k)
        wax.plot(projectWavelet(w,k,n))
        wax.margins(.1,.1)
    axes[0,0].set_title('Wavelet Coeffs')
    axes[0,1].set_title('$W_k$')
    fig.suptitle('Wavelet Coeffs and Projection')
```

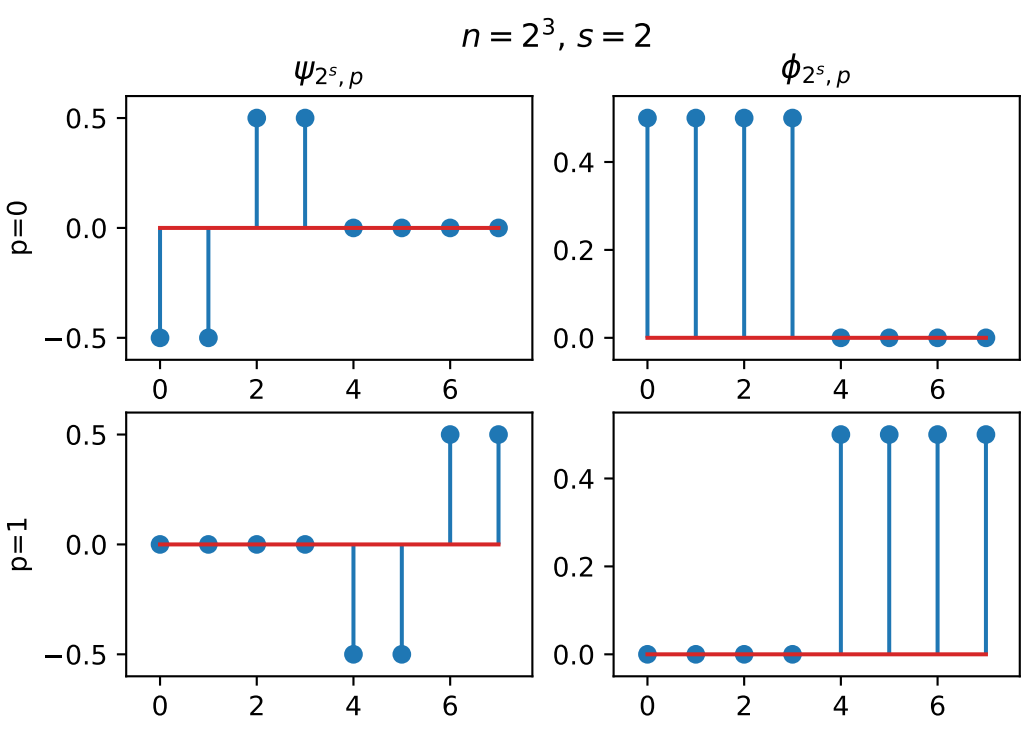
```
    plt.savefig('ecg_wavelet.pdf',bbox_inches='tight')
    plt.close()

def main() :
    plot_psiphi(1,3)
    plot_psiphi(2,3)
    n = 10
    ecg = scipy.io.loadmat('ecg.mat')['ecg'][0:2**n,0]
    plt.plot(ecg)
    plt.title('ECG Plot')
    plt.savefig('ecg.pdf',bbox_inches='tight')
    plt.close()
    m = 5
    plot_downsampling(ecg,m,n)
    plot_wavelet(ecg,m,n)


if __name__ == "__main__" :
    main()
```


ECG Plot
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0
200
400
600
800
1000


ECG Data Projected onto V_k, W_k
V_k
W_k
k=1
k=2
k=3
k=4
0.5
0.0
0.025
0.000
−0.025
0.05
0.00
−0.05
0.1
0.0
−0.1
0.2
0.0
−0.2
0
250
500
750
1000


Wavelet Coeffs and Projection
Wavelet Coeffs
W_k
k=1
k=2
k=3
k=4
0.5
0.0
1
0
2
0
20
40
60
0.5
0.0
−0.5
0
250
500
750
1000


$n = 2^3$, $s = 1$
$\psi_{2^s,p}$
$\phi_{2^s,p}$
p=0
p=1
p=2
p=3
0.5
0.0
−0.5
0
2
4
6


$n = 2^3$, $s = 2$
$\psi_{2^s,p}$
$\phi_{2^s,p}$
p=0
p=1
0.5
0.0
−0.5
0.4
0.2
0
2
4
6

Problem 3

[65]:
```
import os
import glob

import numpy as np

import matplotlib.pyplot as plt
import IPython.display as ipd
import IPython
import matplotlib.colors as colors

from scipy import signal
from copy import deepcopy

import utils
```

[66]:
```
dataurl = 'http://www.openslr.org/resources/1/waves_yesno.tar.gz'
utils.download_and_extract_data(dataurl)
path_to_sound_data = './waves_yesno/'
```

[67]:
```
### STFT hyperparameters

nperseg = 512
window_size = 5
```

[68]:
```
noise_std_array = [1e-1];  ## all noise levels to consider. feel free to play␣
↪around
```

[69]:
```
### for plotting

ini = 34500-1
end = 37500+1

ini2 = 34875
end2 = 35025
```

[70]:
```
train_dataset, train_fs_array, train_max_array, val_dataset, val_fs_array,␣
↪val_max_array = utils.get_data(path_to_sound_data)
```

[71]:
```
max( x.max() for x in train_dataset )
```

[71]:
```
1.0
```

[72]:
```
def get_noise(data, noise_std = 0.1):
    noise = np.random.randn(*data.shape);
    noise = noise * noise_std;
    return noise
```

[73]:
```
def plot_stft(stft_array, save_str=''):

    """ utility function to plot stft """
    f, t, Zxx = stft_array;
    Zxx_abs = np.abs(Zxx)
    Zxx_abs[Zxx_abs<1e-5] = 1e-5
    plt.figure()
    plt.pcolormesh(t, f,Zxx_abs, norm=colors.LogNorm(vmin=Zxx_abs.min(),␣
↪vmax=Zxx_abs.max()),cmap='jet' )
    tick_size = 18
    label_size = 18
    cb = plt.colorbar()
    plt.tick_params(labelsize=tick_size)
    cb.ax.tick_params(labelsize=tick_size)
    plt.ylabel('Frequency (Hz)',fontsize=label_size)
    plt.xlabel('Time (s)',fontsize=label_size)
    plt.savefig('plots/stft_denoising_'+save_str+'_stft.
↪pdf',bbox_inches="tight")
    plt.show()
```

[74]:
```
def get_block_L2_norm(mat, window_size):

    """ mat: an nxn matrix
        window_size: postivie integer (assume odd)
        return: nxn matrix where the (i,j)th entry is the L2 norm of a␣
↪window_size x window_size
                neighbourhood centered at (i, j)
```

```
        to obtain an nxn output, assume that edges are zero padded.

        hint: implement using a convolution

        sample output for get_block_L2_norm(np.ones([5, 5]), 3):

      [[2.        , 2.44948974, 2.44948974, 2.44948974, 2.        ],
       [2.44948974, 3.        , 3.        , 3.        , 2.44948974],
       [2.44948974, 3.        , 3.        , 3.        , 2.44948974],
       [2.44948974, 3.        , 3.        , 3.        , 2.44948974],
       [2.        , 2.44948974, 2.44948974, 2.44948974, 2.        ]]
    """

    res = signal.convolve2d(np.power(mat, 2), np.ones([window_size,␣
↪window_size]), mode='same')
    res = np.power(res, 0.5)

    return res
```

[83]:
```
def stft_denoising(source, noise_std, fs, nperseg, thresh,
                   window_size, block_thresh = None, plot_res = True, ind=0):

    """
    implements hard and block thresholding.

    thresh - threshold for hard thresholding. Thresholding is implemented per␣
↪coefficient
    block_thresh - threshold for block thresholding. Thresholding is␣
↪implemented based on L2 norm of coeff.

    """

    noisy = source + get_noise(source, noise_std);

    if block_thresh is None:
        block_thresh = thresh;

    source_stft = signal.stft(source, fs = fs, nperseg=nperseg);
    noisy_stft = signal.stft(noisy, fs = fs, nperseg=nperseg);

    ## FILL YOUR CODE
    #HARD THRESHOLDED NOISY STFT COEFFICIENTS
    denoised_stft = np.copy(noisy_stft)
    denoised_stft[2] = np.where(np.greater(np.abs(noisy_stft[2]), thresh),␣
↪noisy_stft[2], 0)
```

```
    #BLOCK THRESHOLDED NOISY STFT COEFFICIENTS
    block_denoised_stft = np.copy(noisy_stft)
    block_denoised_stft[2] = np.where(np.
↪greater(get_block_L2_norm(noisy_stft[2], window_size), block_thresh),␣
↪noisy_stft[2], 0)

    if plot_res:
        plot_stft(source_stft,save_str='_clean_'+str(ind));
        plot_stft(noisy_stft,save_str='_noisy_'+str(ind));
        plot_stft(denoised_stft,save_str='_denoised_'+str(ind));
        plot_stft(block_denoised_stft,save_str='_block_denoised_'+str(ind));

    _, source_istft =  signal.istft(source_stft[2], fs=fs, nperseg=nperseg);
    _, noisy_istft =   signal.istft(noisy_stft[2], fs=fs, nperseg=nperseg);
    _, denoised_istft =   signal.istft(denoised_stft[2], fs=fs,␣
↪nperseg=nperseg);
    _, block_denoised_istft =   signal.istft(block_denoised_stft[2], fs=fs,␣
↪nperseg=nperseg);


    return np.real(source_istft), np.real(noisy_istft), np.
↪real(denoised_istft), np.real(block_denoised_istft)
```

[84]:
```
np.sqrt(1e-2)
```

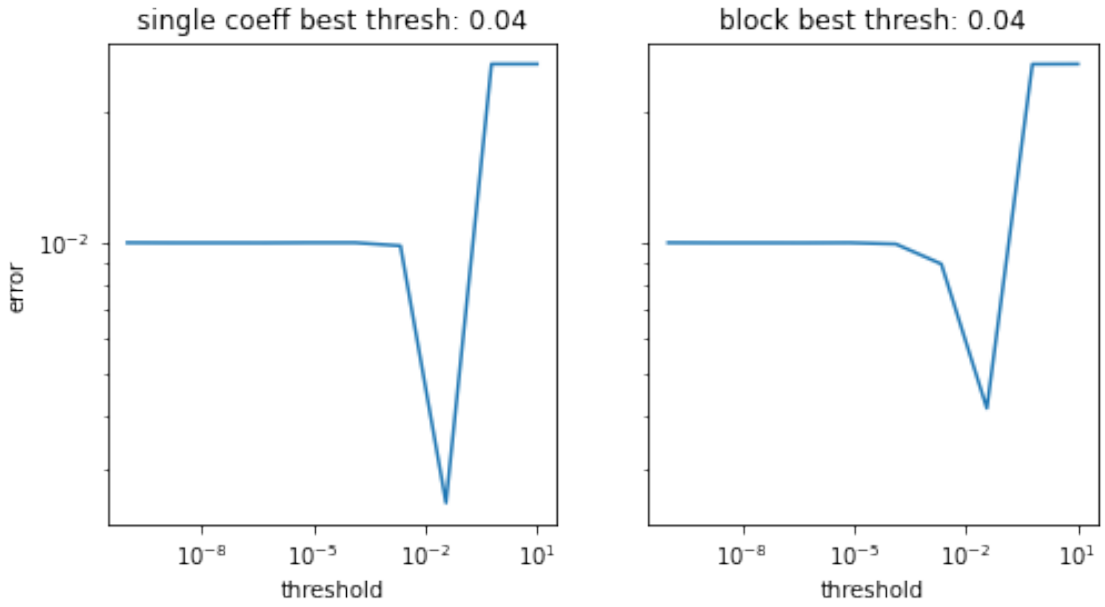
[84]: 0.1

[85]:
```
# threshold_array = [1e-10, 1e-8, 1e-6, 1e-4, 1e-2, 1e0, 1e2]

threshold_array = np.logspace(-10, 1, num = 10)
```

[86]:
```
error_dict = {}
block_error_dict = {}
best_threshold_dict = {}
block_best_threshold_dict = {}
for noise_std in noise_std_array:

    error_dict[noise_std] = np.zeros_like(threshold_array);
    block_error_dict[noise_std] = np.zeros_like(threshold_array);

    for thresh_i, thresh in enumerate(threshold_array):
        total_error = 0.0;
        block_total_error = 0.0;
        total_length = 0.0;
        for i, x in enumerate(train_dataset):
```

```
            rec_source, rec_noisy, rec_denoised, rec_denoised_block =␣
↪stft_denoising(x, noise_std, train_fs_array[i], nperseg, thresh,␣
↪window_size,plot_res=False);
            total_error += np.linalg.norm( rec_denoised - rec_source)**2;
            block_total_error += np.linalg.norm( rec_denoised_block -␣
↪rec_source)**2;
            total_length += len(rec_source);

        error_dict[noise_std][thresh_i] = (total_error/total_length);
        block_error_dict[noise_std][thresh_i] = (block_total_error/
↪total_length);

    print('noise std: ', noise_std);
    fig, axes = plt.subplots(1, 2, sharex=True, sharey=True, figsize = (8, 4));

    axes[0].loglog(threshold_array, error_dict[noise_std])
    axes[0].set_xlabel('threshold')
    axes[0].set_ylabel('error')

    best_threshold_dict[noise_std] = threshold_array[np.
↪argmin(error_dict[noise_std])]

    axes[0].set_title('single coeff best thresh: '+str(␣
↪round(best_threshold_dict[noise_std], 2)) )

    axes[1].loglog(threshold_array, block_error_dict[noise_std])
    axes[1].set_xlabel('threshold')
#     axes[1].set_ylabel('error')

    block_best_threshold_dict[noise_std] = threshold_array[np.
↪argmin(block_error_dict[noise_std])]

    axes[1].set_title('block best thresh: '+str(␣
↪round(block_best_threshold_dict[noise_std], 2)) )

    plt.show()

    print('='*50+'\n')
```

```
noise std:  0.1
```



==========

[87]:
```
val_idx = 2

source = val_dataset[val_idx]
fs = val_fs_array[val_idx]
max_val = val_max_array[val_idx]
```

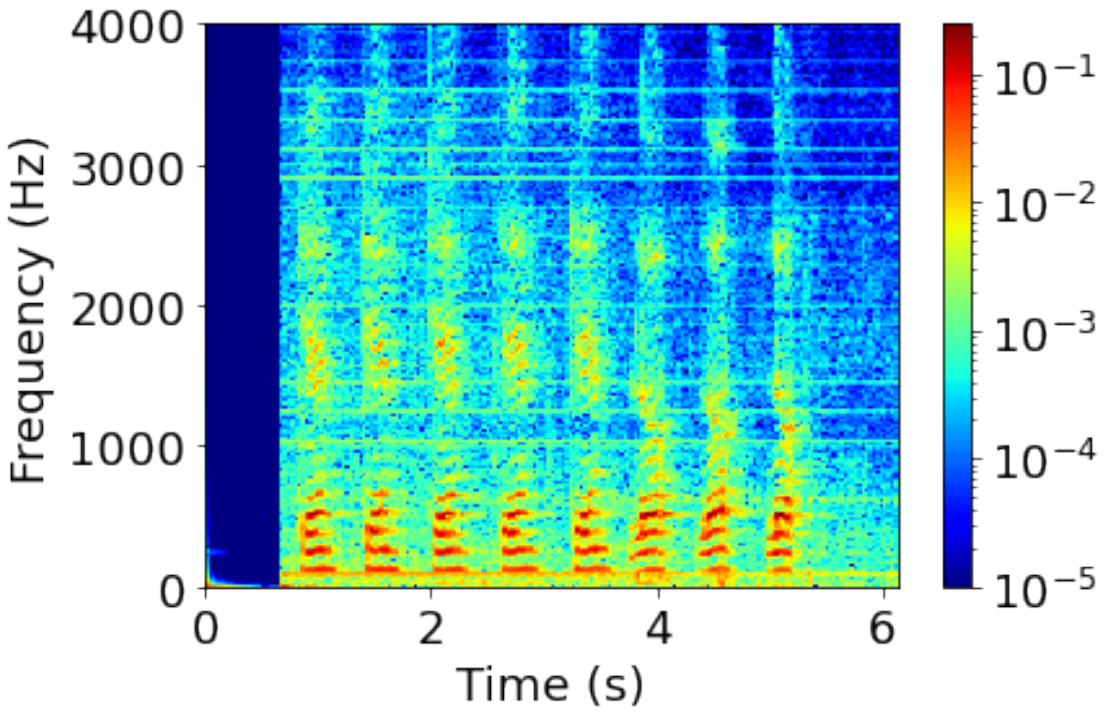
[88]:
```
fig_size=(20,6)
```

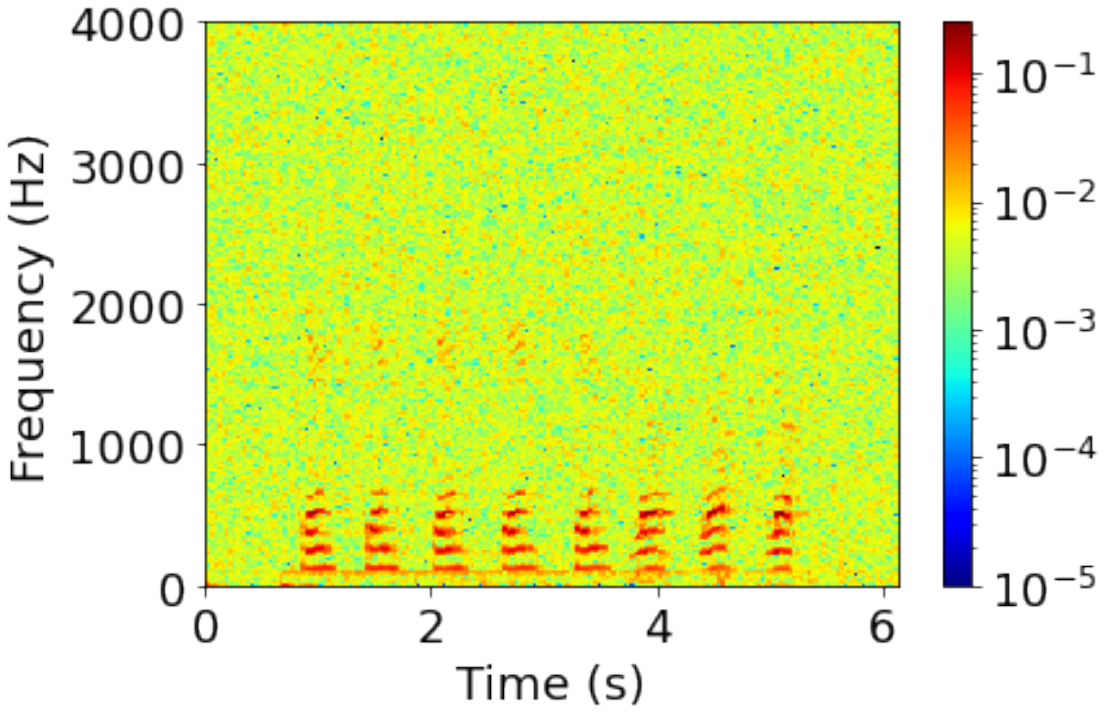
[89]:
```
for ind_fig, noise_std in enumerate(noise_std_array):

    print('Noise Std: ', noise_std)

    noise_sample = get_noise(source, noise_std = noise_std);


    istft_source, istft_noisy, istft_denoised, istft_denoised_block =␣
↪stft_denoising(source, noise_std, fs, nperseg,
                                                                thresh =␣
↪best_threshold_dict[noise_std],
                                                                window_size =␣
↪window_size,
                                                                block_thresh =␣
↪block_best_threshold_dict[noise_std])
    istft_denoised_block *= max_val
```

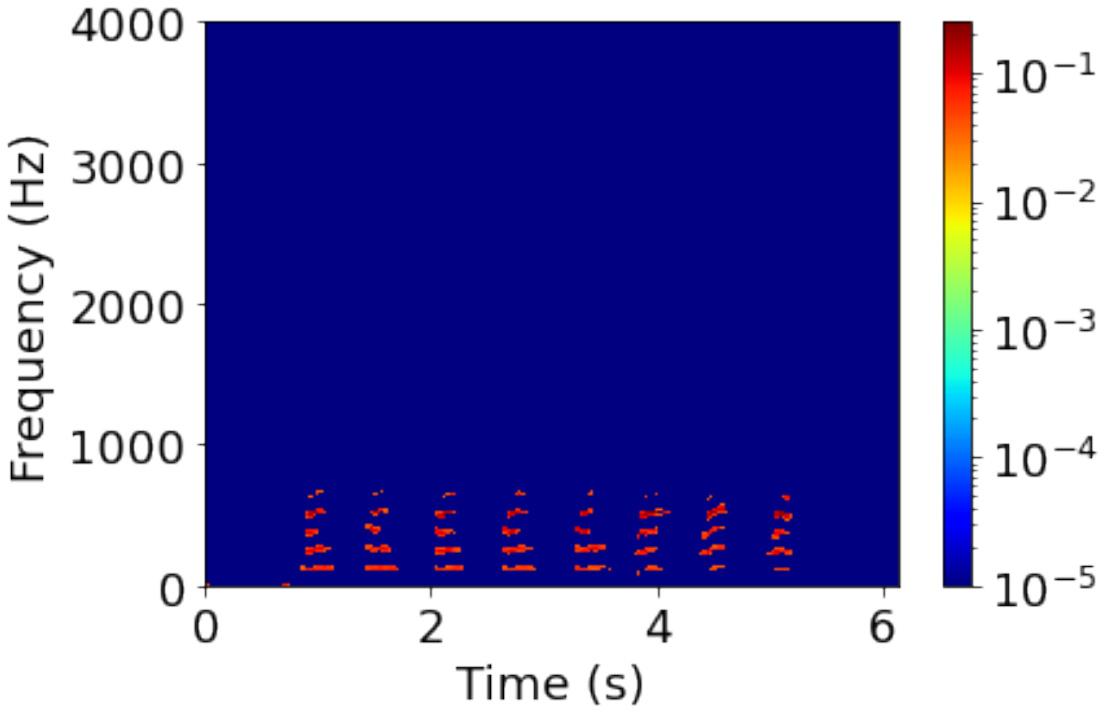
```
    istft_denoised *= max_val;
    istft_noisy *= max_val;
    istft_source *= max_val;

    print('STFT Denoised: ')
    IPython.display.display(ipd.Audio(istft_denoised, rate=fs))

    print('Block STFT Denoised: ')
    IPython.display.display(ipd.Audio(istft_denoised_block, rate=fs))

    label_size = 18
    font_size = 18

    t_indices = np.arange(len(istft_source))/fs

    plt.figure(figsize = fig_size)
    plt.plot(np.real(istft_source))
    plt.xlabel('Time (s)',fontsize=font_size)
    plt.tick_params(labelsize=label_size)

    plt.figure(figsize = fig_size)
    plt.plot(t_indices[ini:end],np.real(istft_denoised[ini:end]))
    plt.xlabel('Time (s)',fontsize=font_size)
    plt.tick_params(labelsize=label_size)
    plt.savefig('plots/stft_stft_denoised_' + str(ind_fig) + '.
↪pdf',bbox_inches="tight")

    plt.figure(figsize = fig_size)
    plt.plot(t_indices[ini:end],np.real(istft_denoised_block[ini:end]))
    plt.xlabel('Time (s)',fontsize=font_size)
    plt.tick_params(labelsize=label_size)
    plt.savefig('plots/stft_block_denoised_' + str(ind_fig) + '.
↪pdf',bbox_inches="tight")



    plt.figure(figsize = fig_size)
    plt.plot(t_indices[ini2:end2],np.real(istft_source[ini2:
↪end2]),'--o',markersize=6,lw=1.5,color='deepskyblue', label='Signal')
    plt.plot(t_indices[ini2:end2],np.real(istft_denoised[ini2:
↪end2]),'x',color='tomato',markersize=8,mew=3, label='STFT thresholding')
    plt.plot(t_indices[ini2:end2],np.real(istft_noisy[ini2:end2]),'.
↪',color='darkgreen',markersize=10,label='Noisy data')
    plt.xlabel('Time (s)',fontsize=font_size)
    plt.tick_params(labelsize=label_size)
    plt.legend(fontsize=font_size)
```

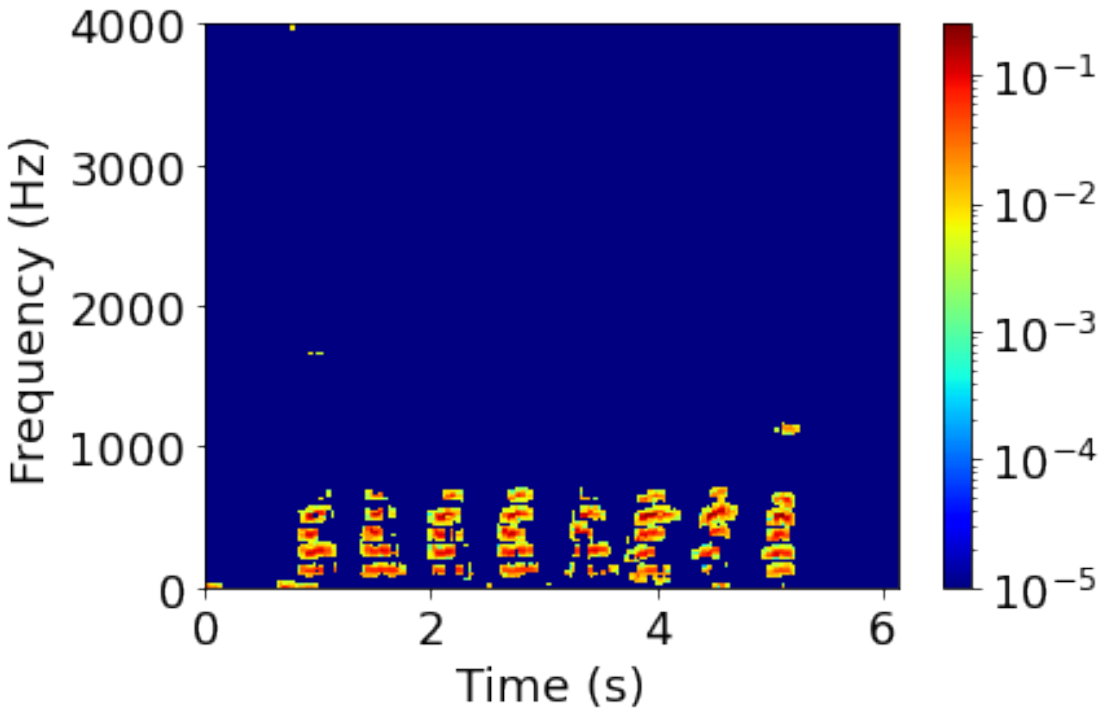
```
   plt.savefig('plots/stft_stft_denoised_' + str(ind_fig) + '_zoom.
↪pdf',bbox_inches="tight")

   plt.figure(figsize = fig_size)
   plt.plot(t_indices[ini2:end2],np.real(istft_source[ini2:
↪end2]),'--o',markersize=6,lw=1.5,color='deepskyblue', label='Signal')
   plt.plot(t_indices[ini2:end2],np.real(istft_denoised_block[ini2:
↪end2]),'x',color='tomato',markersize=8,mew=3, label='STFT block␣
↪thresholding')
   plt.plot(t_indices[ini2:end2],np.real(istft_noisy[ini2:end2]),'.
↪',color='darkgreen',markersize=10,label='Noisy data')
   plt.xlabel('Time (s)',fontsize=font_size)
   plt.tick_params(labelsize=label_size)
   plt.legend(fontsize=font_size)
   plt.savefig('plots/stft_block_denoised_' + str(ind_fig) + '_zoom.
↪pdf',bbox_inches="tight")

   plt.show()


   print('='*50 + '\n')
```

```
Noise Std:  0.1
```




4000
3000
2000
1000
0
Frequency (Hz)
0
2
4
6
Time (s)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}


4000
3000
2000
1000
0
Frequency (Hz)
0
2
4
6
Time (s)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}

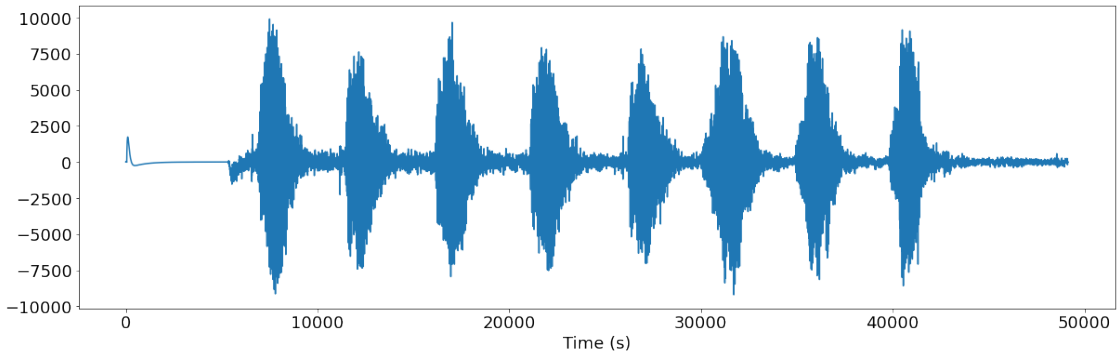


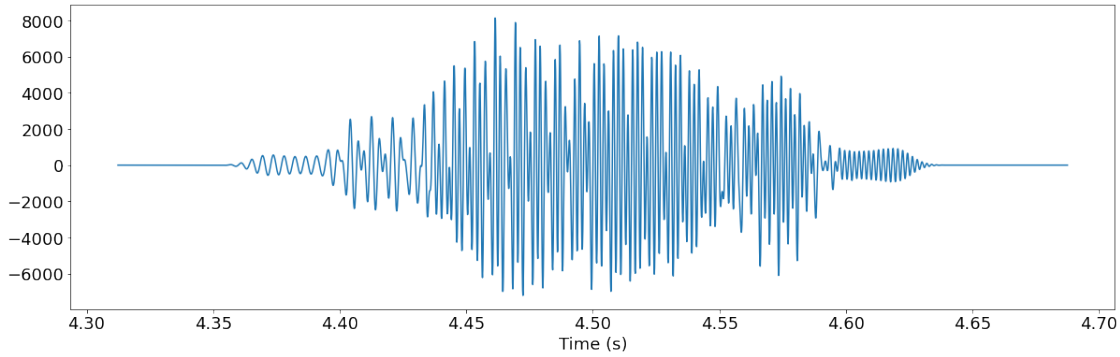
STFT Denoised:

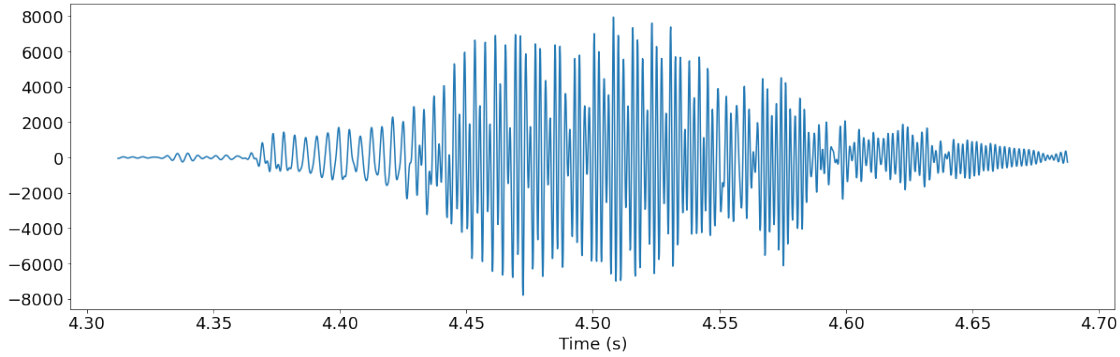
<IPython.lib.display.Audio object>

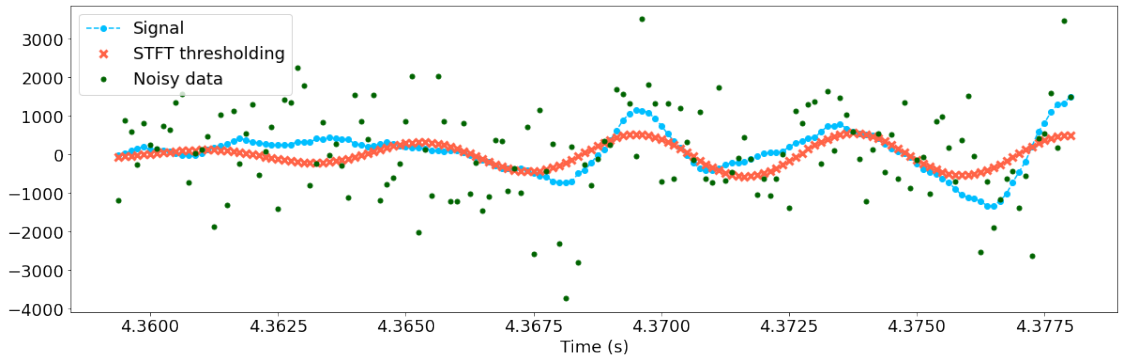
Block STFT Denoised:

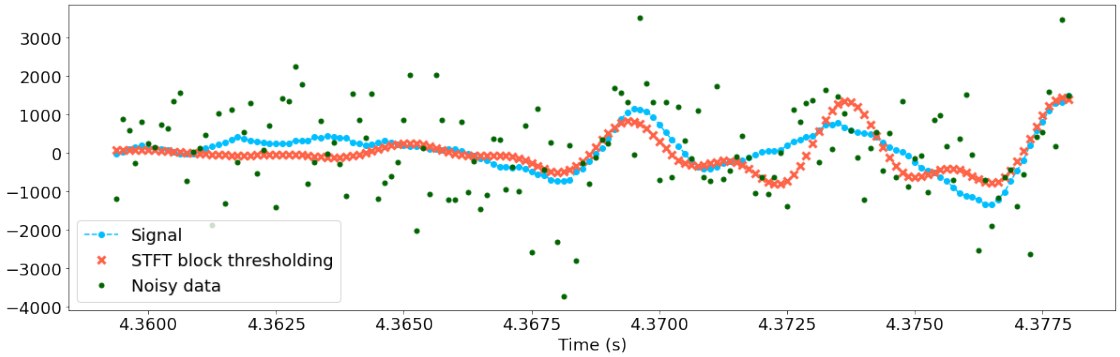
<IPython.lib.display.Audio object>




8000
6000
4000
2000
0
−2000
−4000
−6000
4.30
4.35
4.40
4.45
4.50
4.55
4.60
4.65
4.70
Time (s)


8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
4.30
4.35
4.40
4.45
4.50
4.55
4.60
4.65
4.70
Time (s)


Signal
STFT thresholding
Noisy data
3000
2000
1000
0
−1000
−2000
−3000
−4000
4.3600
4.3625
4.3650
4.3675
4.3700
4.3725
4.3750
4.3775
Time (s)



==========

Problem 4

(a)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter
import imageio as io
import matplotlib.colors as colors
import pywt
from skimage import metrics
```

[2]:
```
import requests
from PIL import Image
from io import BytesIO

img_urls = ['https://cims.nyu.edu/~cfgranda/foto.jpg']

img_array = []
for url in img_urls:
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    img = img.convert('L')
    img = img.resize((512, 512))
    img = np.array(img)
    img_array.append(img)
```

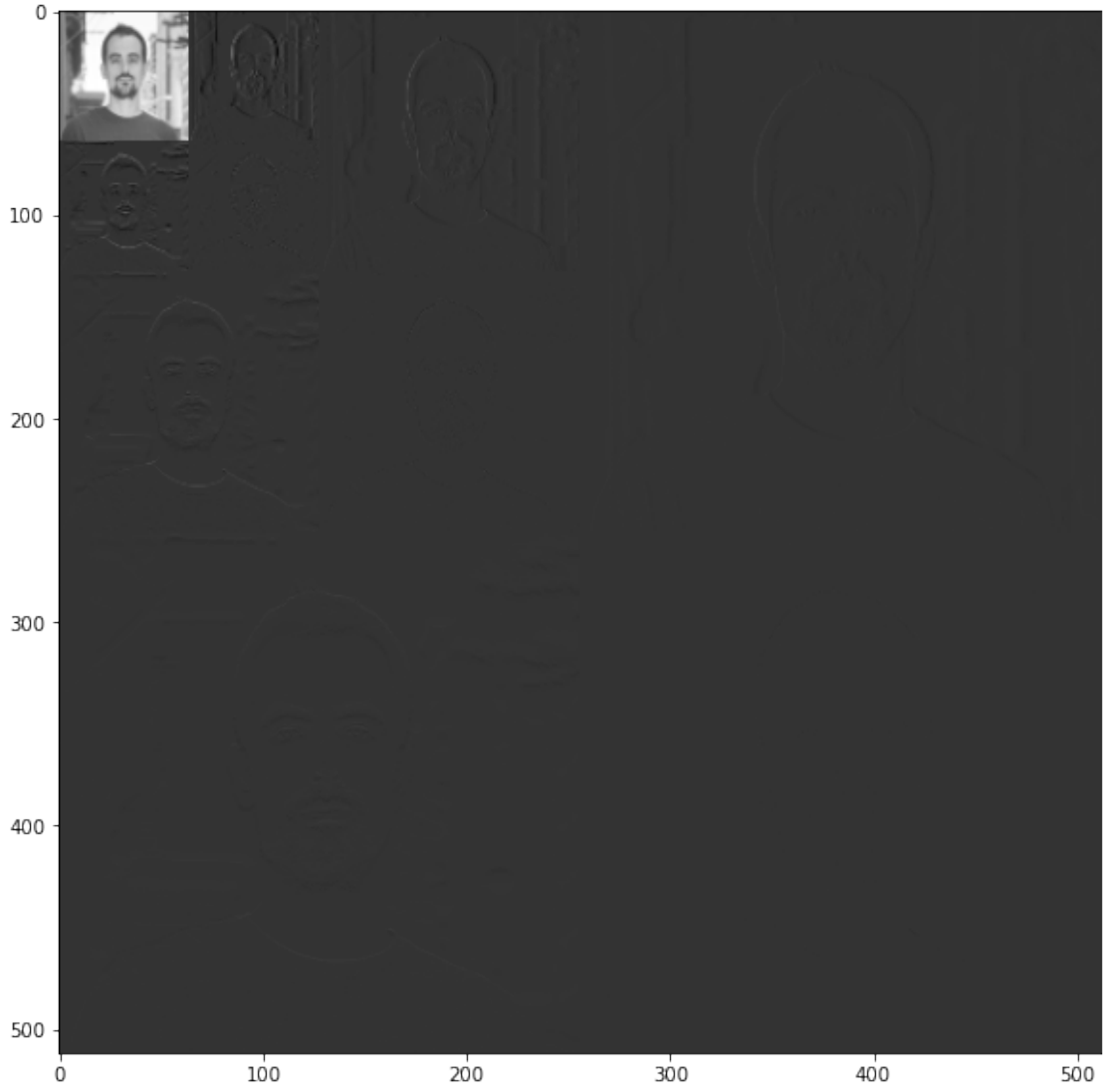
[11]:
```
def visualize_image_and_recon(image, recon_image, n_level, threshold, figsize =␣
↪(10, 5)):

    fig, axes = plt.subplots(nrows=1, ncols=2,  figsize = figsize)
    axes[0].imshow(image, cmap='gray')
```

```
    axes[0].set_title('Original Image')

    axes[1].imshow(recon_image, cmap='gray')
    axes[1].set_title(f"Reconstructed. PSNR: {metrics.
↪peak_signal_noise_ratio(image, recon_image, data_range = 255.): .3f},␣
↪n_level={n_level}, threshold={threshold}")

    plt.show()
```

[4]:
```
image = np.array(img_array[0])
```

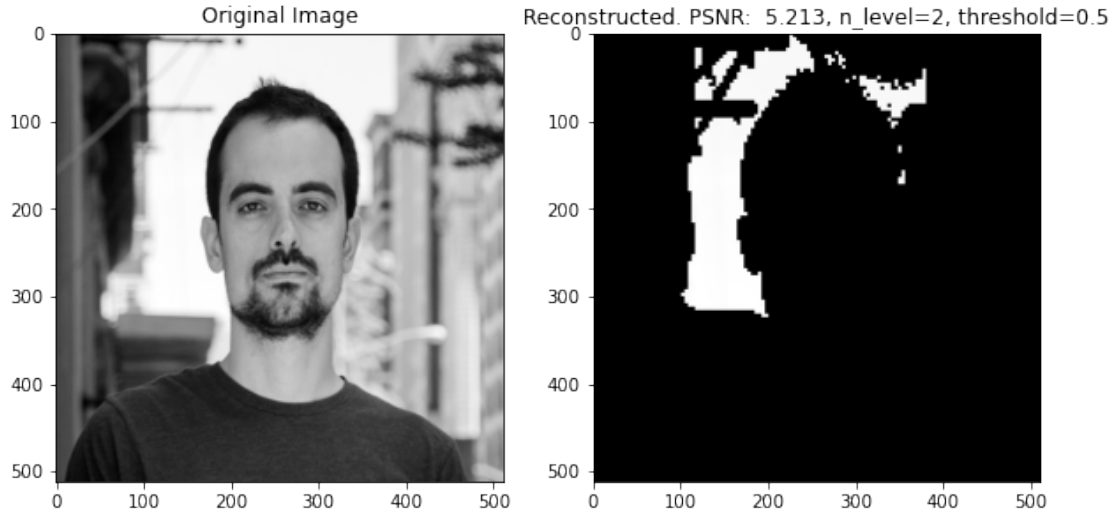
[5]:
```
n_levels = 3
wav_type = 'haar'
x_w = pywt.wavedec2(image, wav_type, level=n_levels)
coeff_array, coeff_slices = pywt.coeffs_to_array(x_w)
plt.figure(figsize = (10, 10))
plt.imshow(coeff_array, cmap='gray')
```

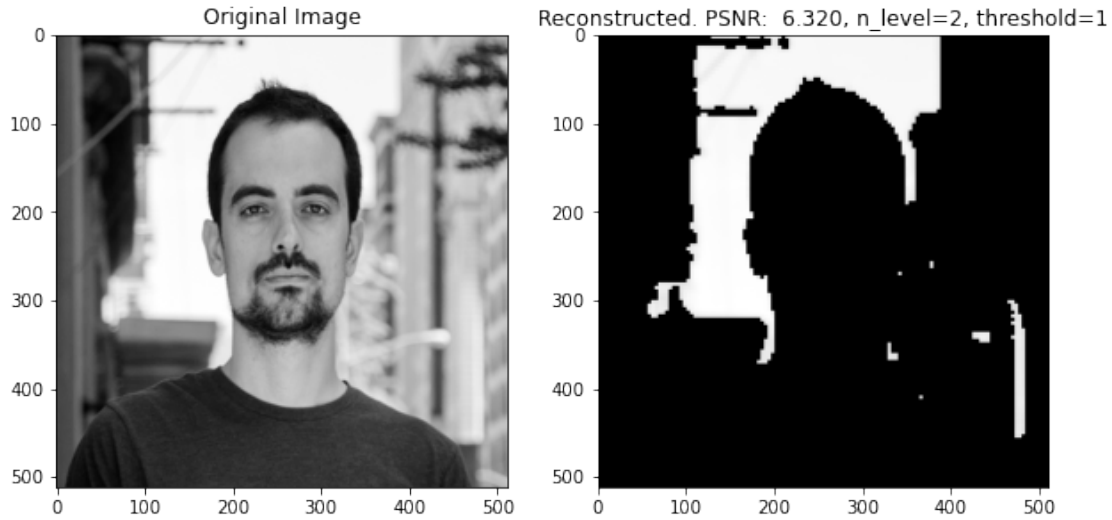
[5]:
```
<matplotlib.image.AxesImage at 0x1657b03a2e0>
```

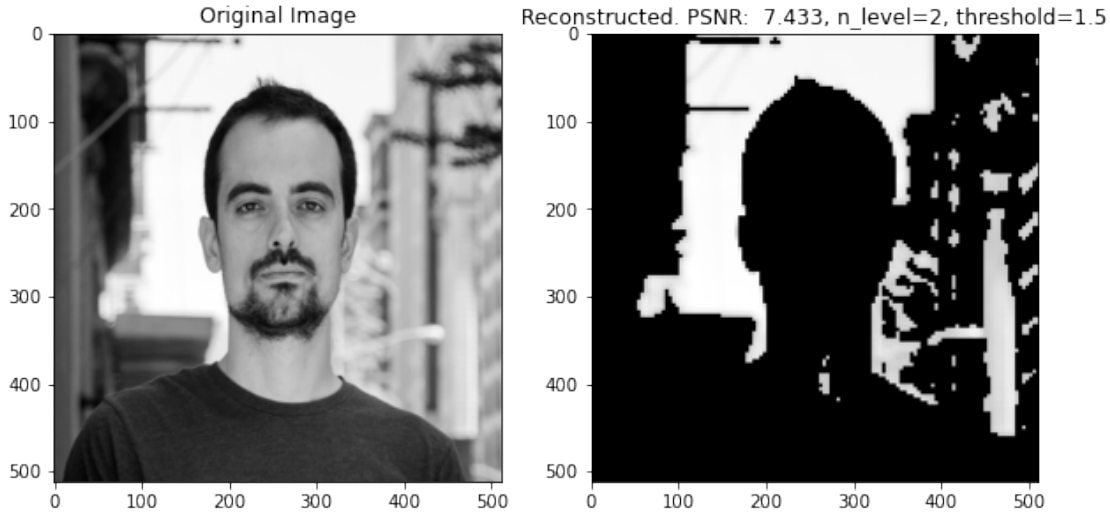


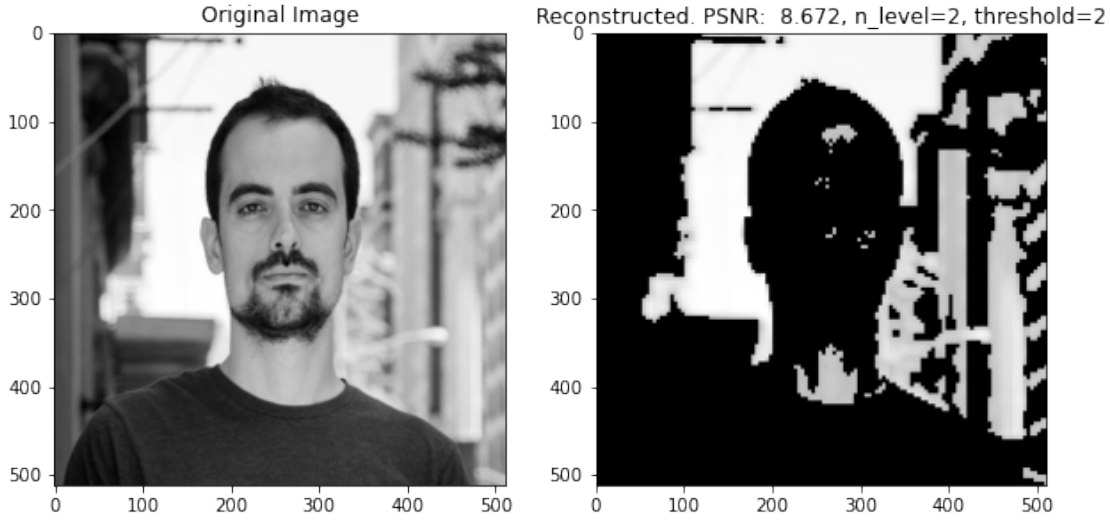
[12]:
```
n_levels = [2, 3, 4, 5]
pct = [0.5, 1, 1.5, 2, 3, 4, 5, 10, 25, 50, 75]
wav_type = 'haar'
PSNR = []

for i in range(len(n_levels)):
    temp = []
    for j in range(len(pct)):
        # wavelet decomposition
        x_w = pywt.wavedec2(image, wav_type, level=n_levels[i])
        coeff_array, coeff_slices = pywt.coeffs_to_array(x_w)

        # top x%
```

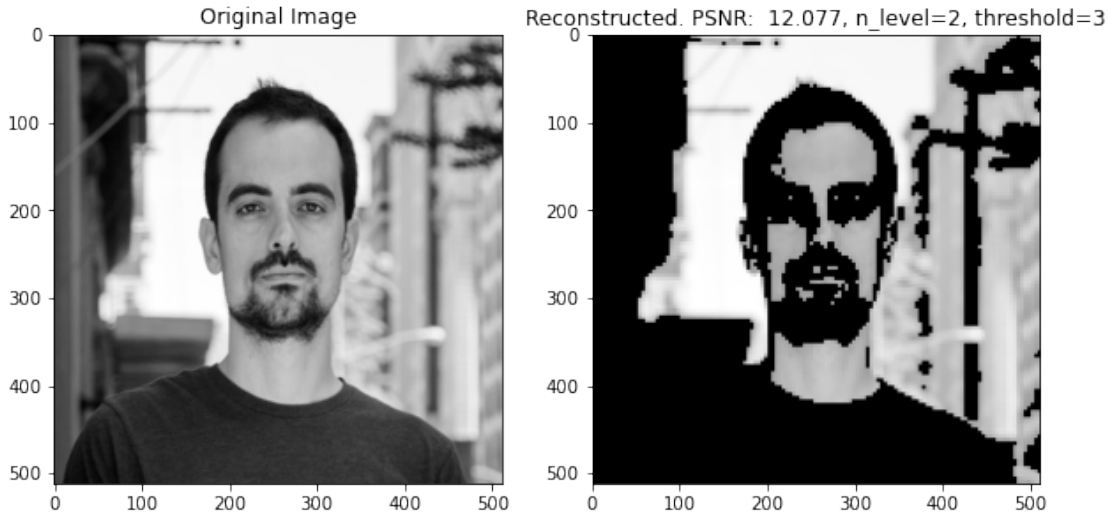
```
        x = sorted(np.abs(coeff_array.flatten()), reverse=True)
        threshold = int(np.ceil(len(x)*(pct[j]/100)))
        for r in range(len(coeff_array)):
            for s in range(len(coeff_array[0])):
                if np.abs(coeff_array[r][s]) < x[threshold]:
                    coeff_array[r][s] = 0

        coeff_correctformat = pywt.array_to_coeffs(coeff_array, coeff_slices,␣
↪output_format='wavedec2')
        recon_image = pywt.waverec2(coeff_correctformat, wavelet=wav_type)
        visualize_image_and_recon(image, recon_image, n_levels[i], pct[j])
        temp.append(metrics.peak_signal_noise_ratio(image, recon_image,␣
↪data_range = 255.))
    PSNR.append(temp)
```

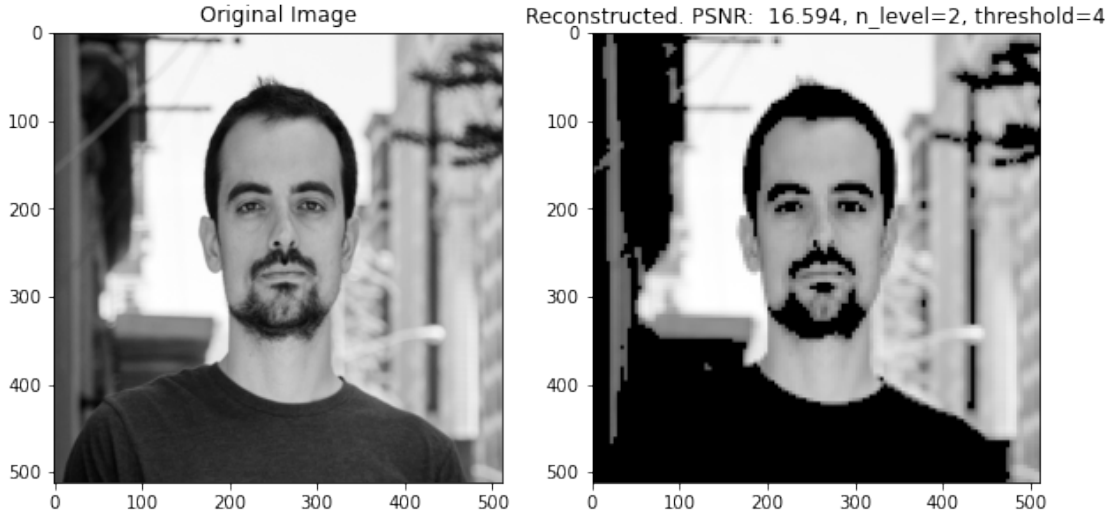


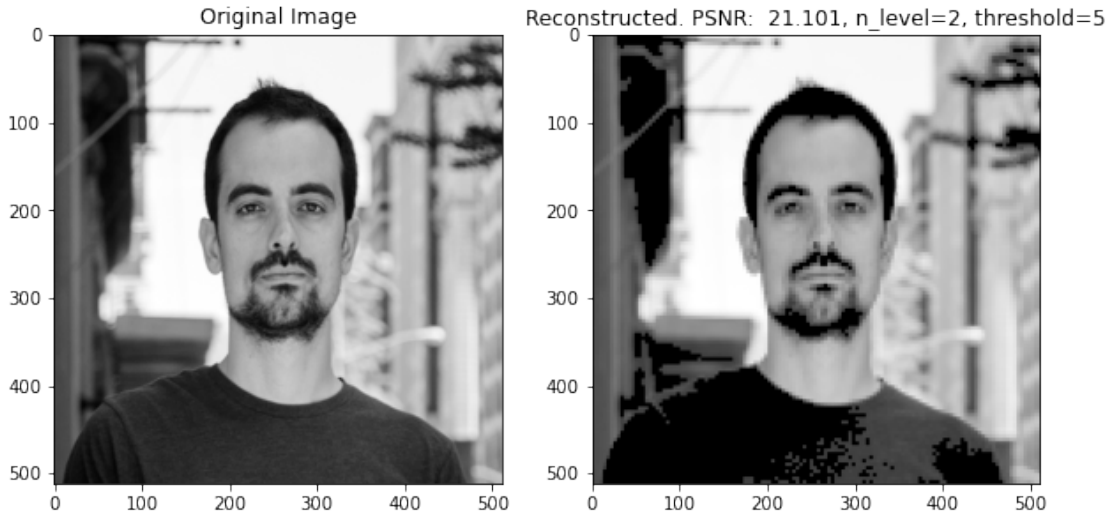


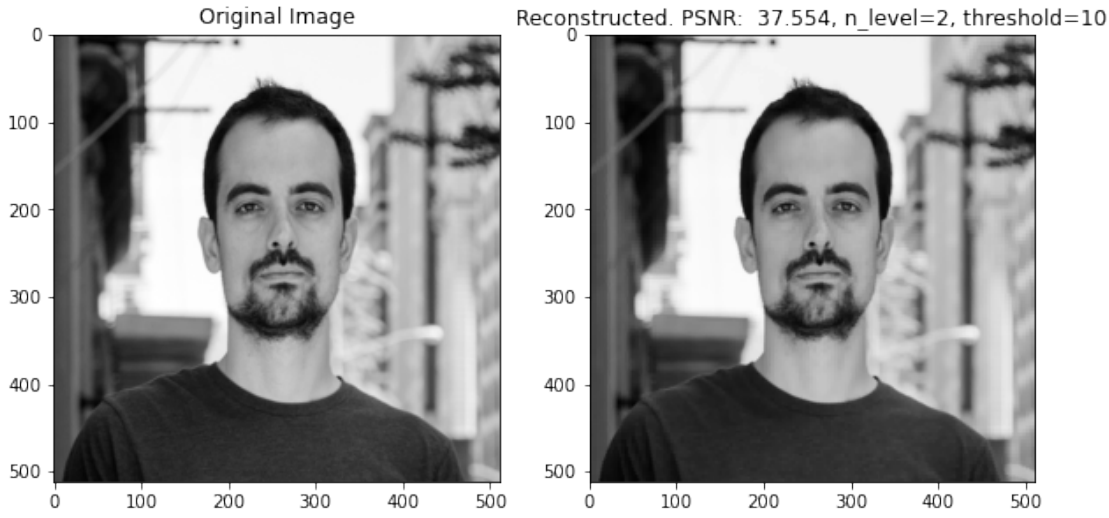

Original Image
Reconstructed. PSNR: 7.433, n_level=2, threshold=1.5
0
100
200
300
400
500

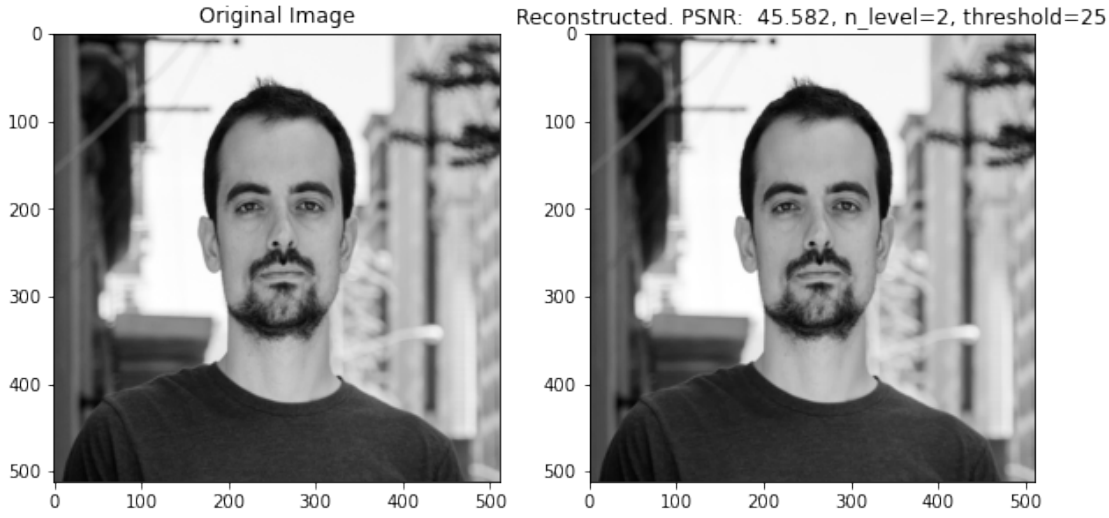

Original Image
Reconstructed. PSNR: 8.672, n_level=2, threshold=2
0
100
200
300
400
500

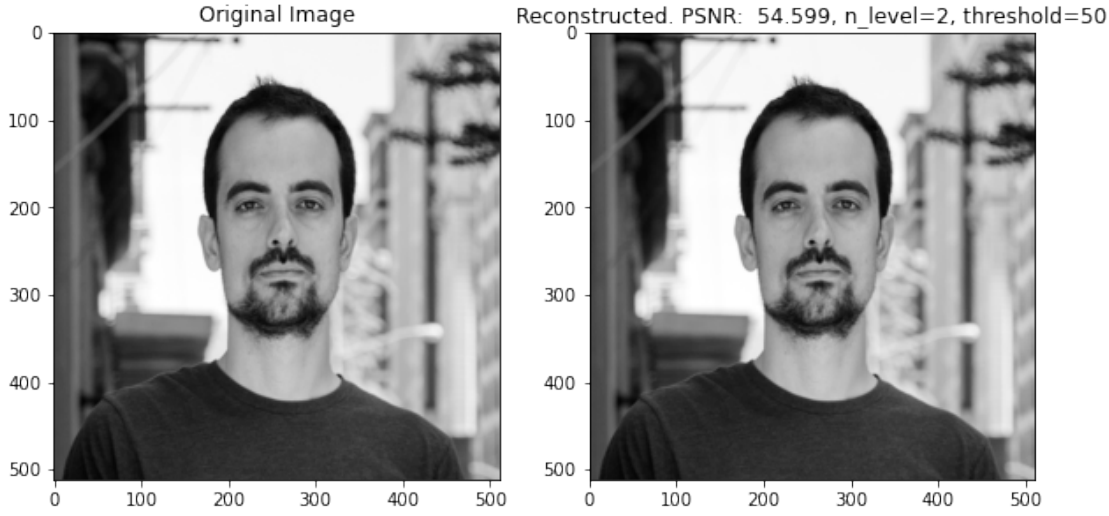

Original Image
Reconstructed. PSNR: 12.077, n_level=2, threshold=3

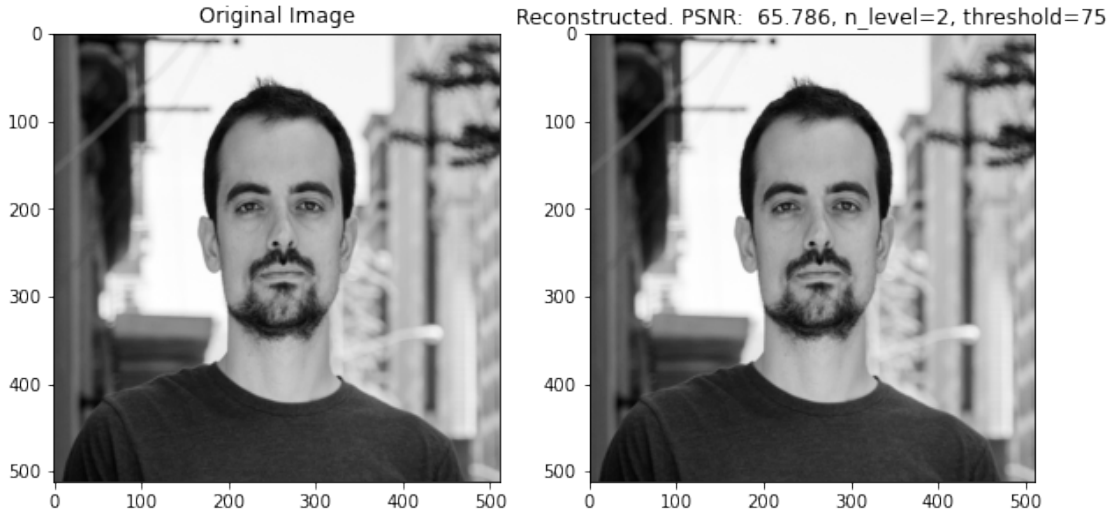

Original Image
Reconstructed. PSNR: 16.594, n_level=2, threshold=4

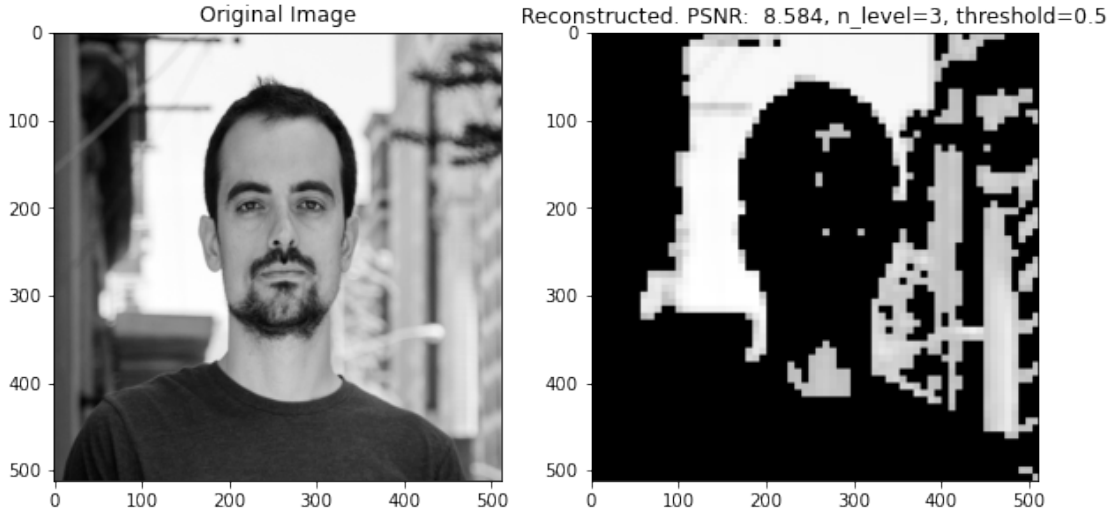

Original Image
Reconstructed. PSNR: 21.101, n_level=2, threshold=5
0
100
200
300
400
500

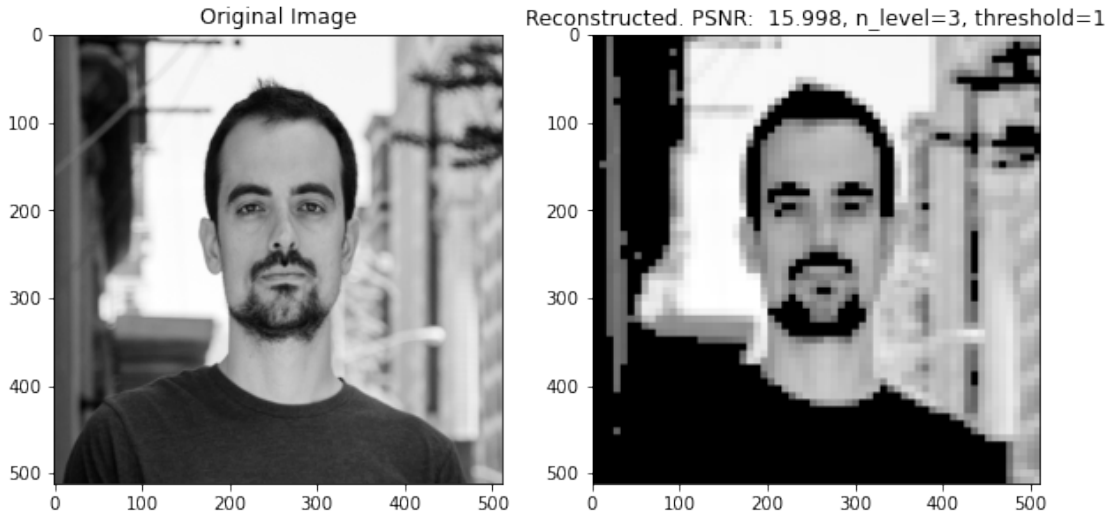

Original Image
Reconstructed. PSNR: 37.554, n_level=2, threshold=10
0
100
200
300
400
500

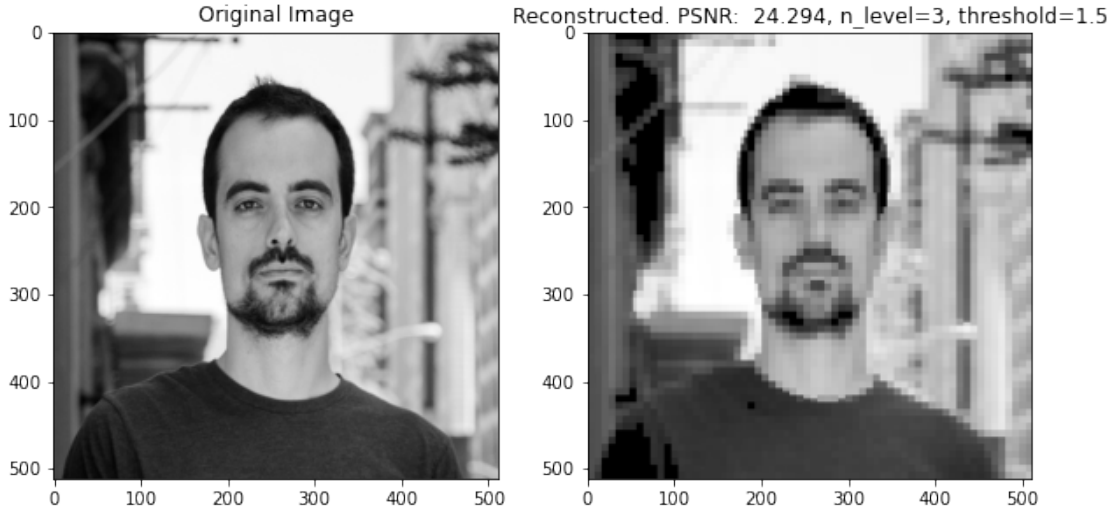

Original Image
Reconstructed. PSNR: 45.582, n_level=2, threshold=25
0
100
200
300
400
500

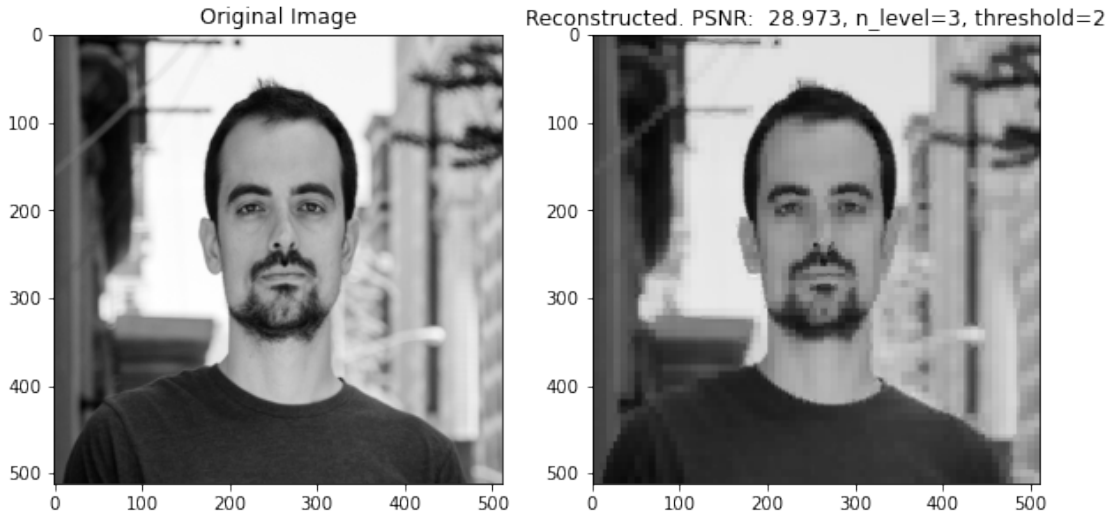

Original Image
Reconstructed. PSNR: 54.599, n_level=2, threshold=50
0
100
200
300
400
500

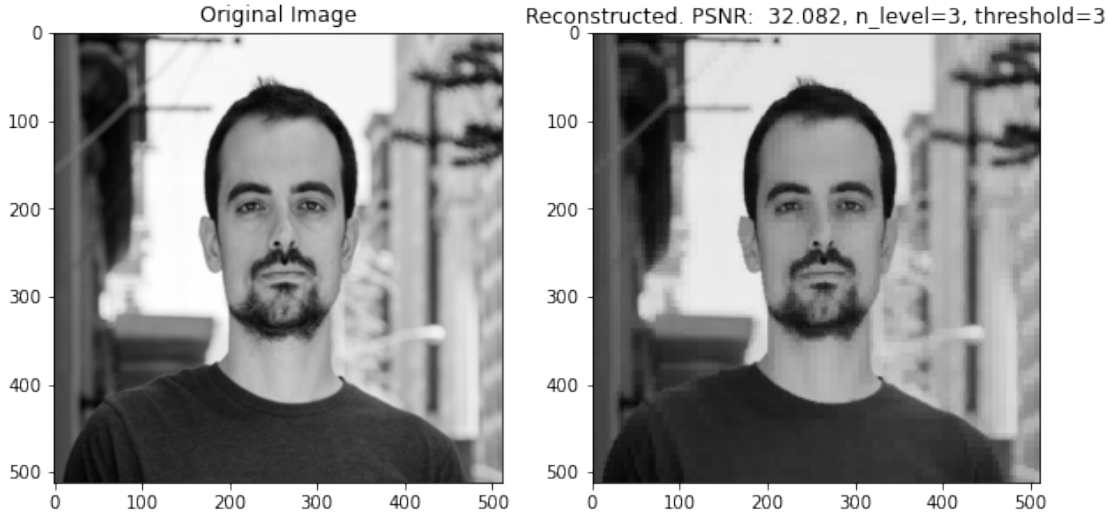

Original Image
Reconstructed. PSNR: 65.786, n_level=2, threshold=75

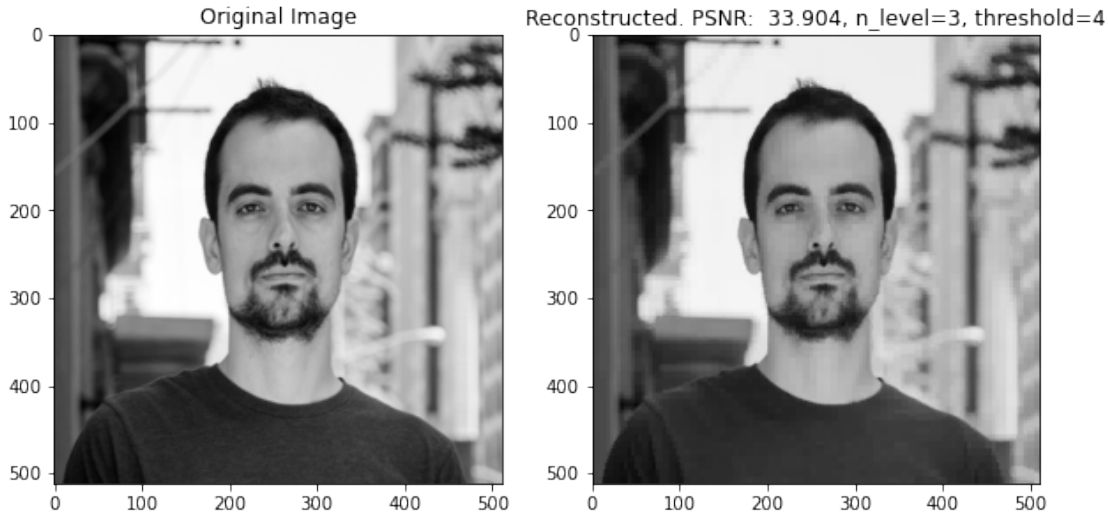

Original Image
Reconstructed. PSNR: 8.584, n_level=3, threshold=0.5

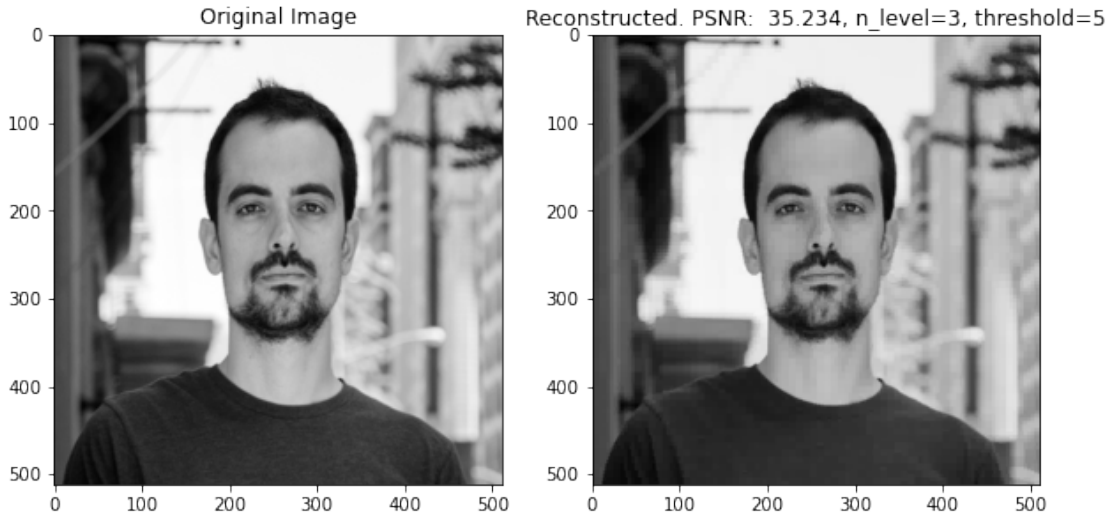

Original Image
Reconstructed. PSNR: 15.998, n_level=3, threshold=1
0
100
200
300
400
500

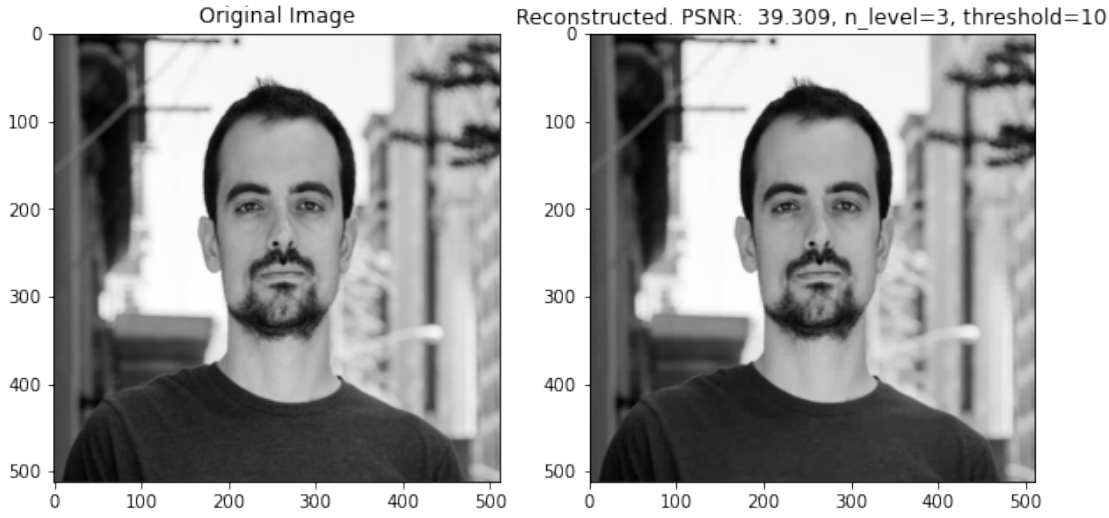

Original Image
Reconstructed. PSNR: 24.294, n_level=3, threshold=1.5
0
100
200
300
400
500

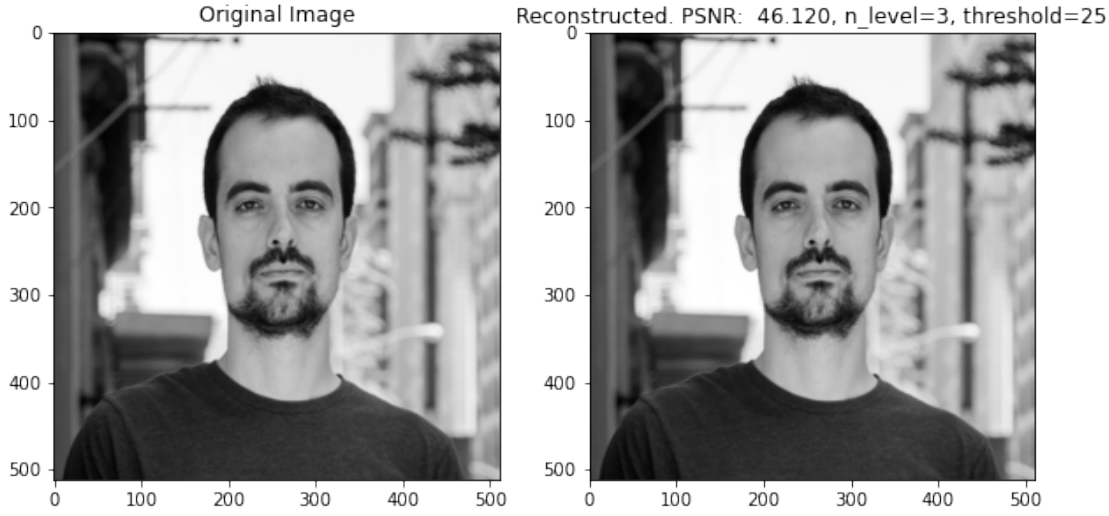

Original Image
Reconstructed. PSNR: 28.973, n_level=3, threshold=2
0
100
200
300
400
500

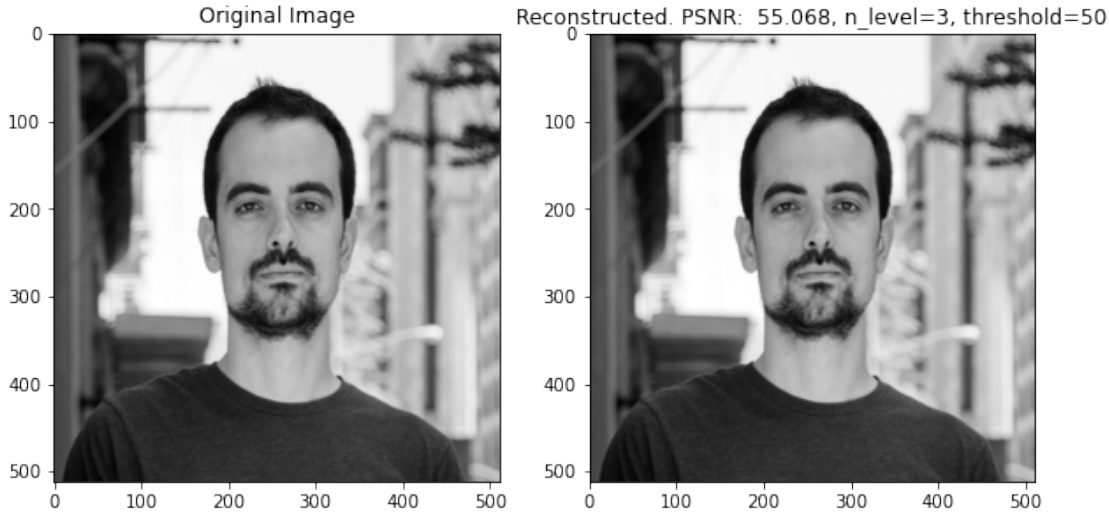

Original Image
Reconstructed. PSNR: 32.082, n_level=3, threshold=3
0
100
200
300
400
500

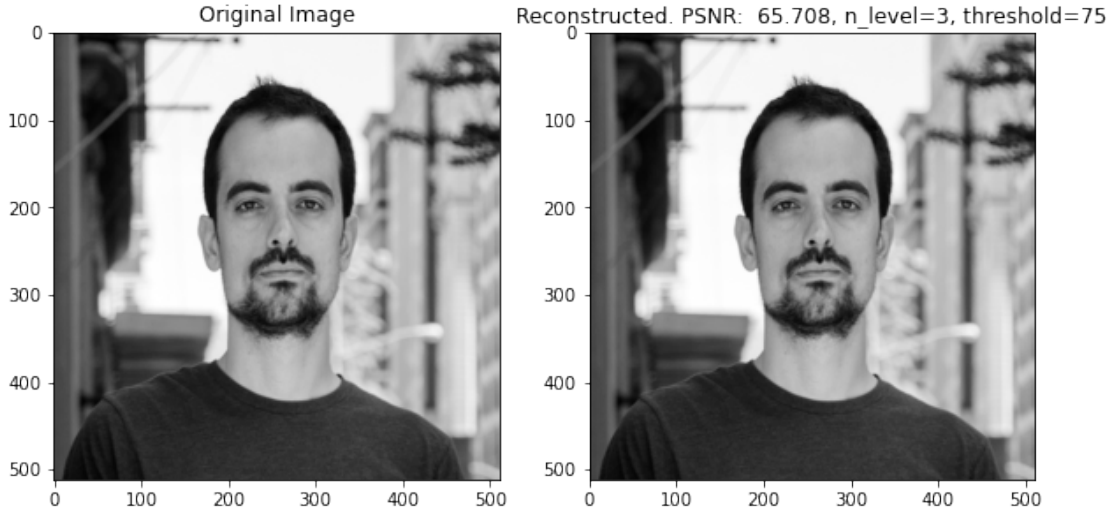

Original Image
Reconstructed. PSNR: 33.904, n_level=3, threshold=4
0
100
200
300
400
500

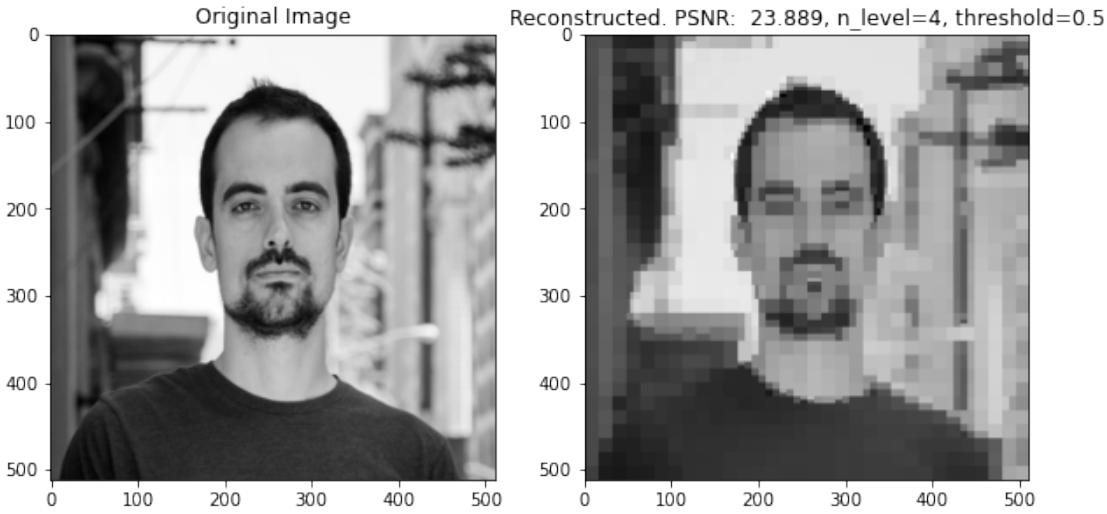

Original Image
Reconstructed. PSNR: 35.234, n_level=3, threshold=5
0
100
200
300
400
500

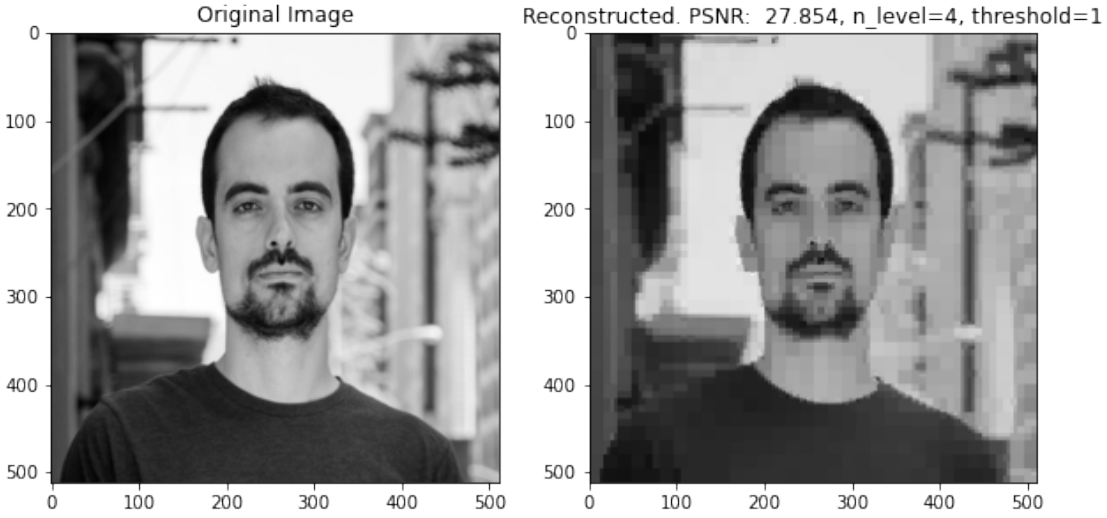

Original Image
Reconstructed. PSNR: 39.309, n_level=3, threshold=10
0
100
200
300
400
500

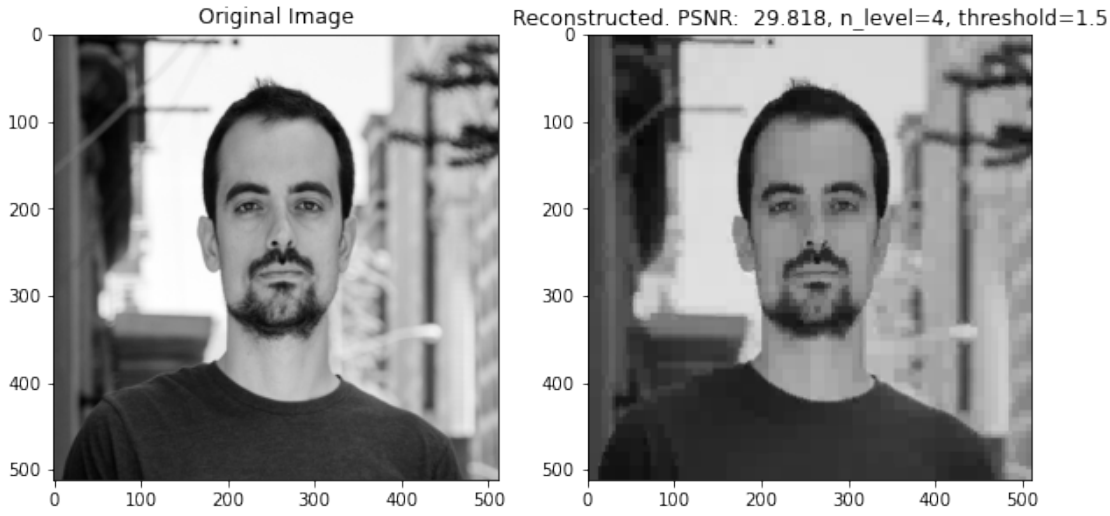

Original Image
Reconstructed. PSNR: 46.120, n_level=3, threshold=25
0
100
200
300
400
500

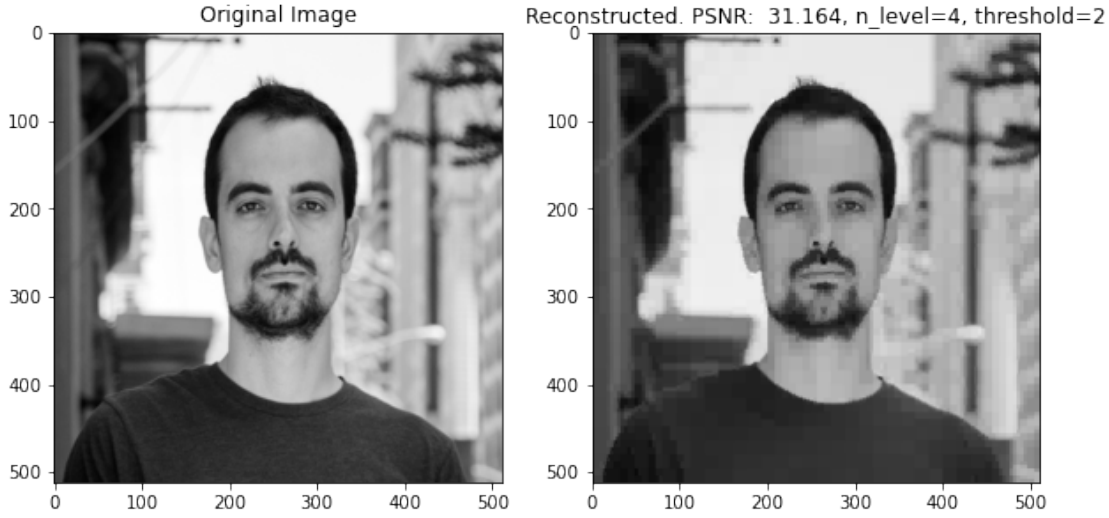

Original Image
Reconstructed. PSNR: 55.068, n_level=3, threshold=50
0
100
200
300
400
500

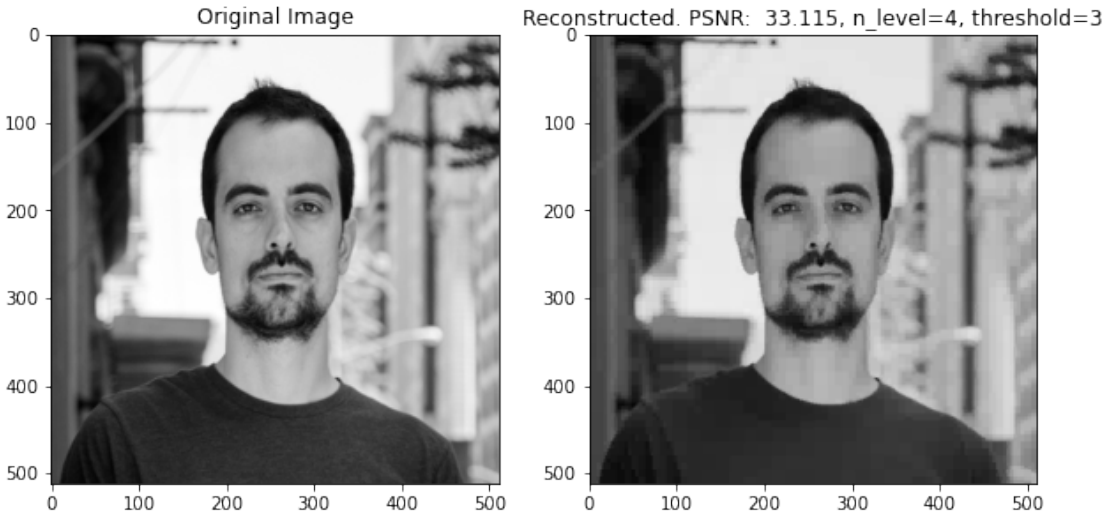

Original Image
Reconstructed. PSNR: 65.708, n_level=3, threshold=75
0
100
200
300
400
500

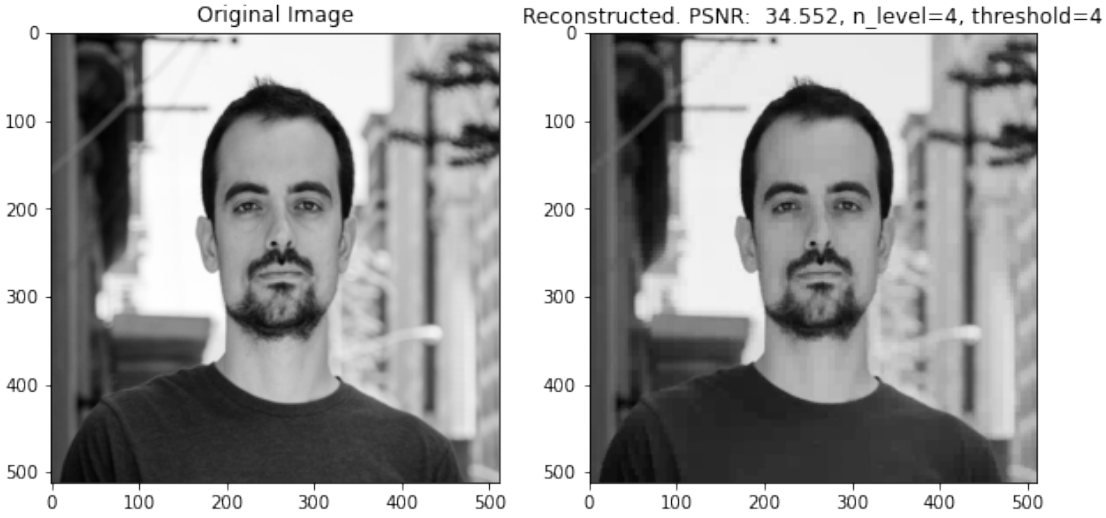

Original Image
Reconstructed. PSNR: 23.889, n_level=4, threshold=0.5
0
100
200
300
400
500

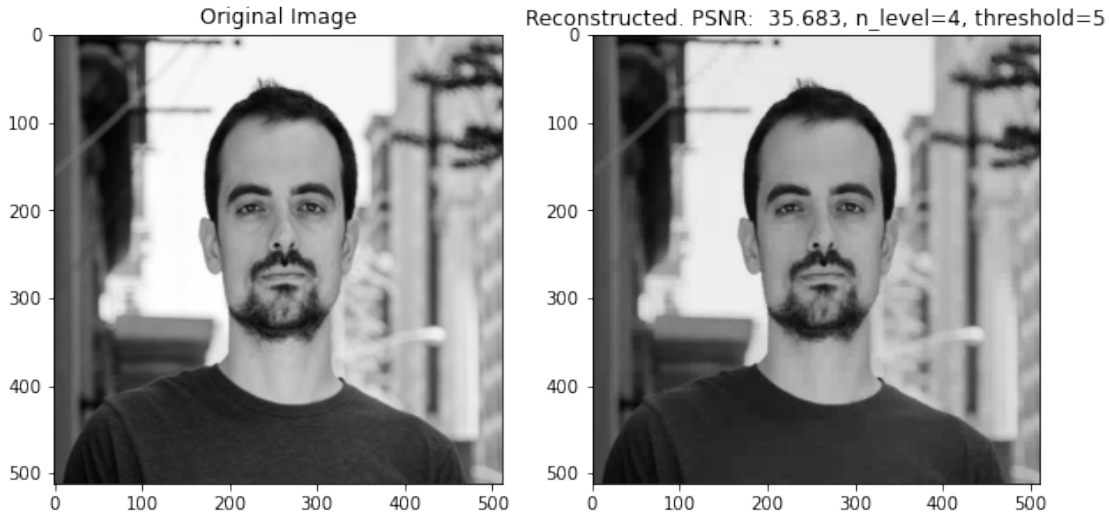

Original Image
Reconstructed. PSNR: 27.854, n_level=4, threshold=1
0
100
200
300
400
500

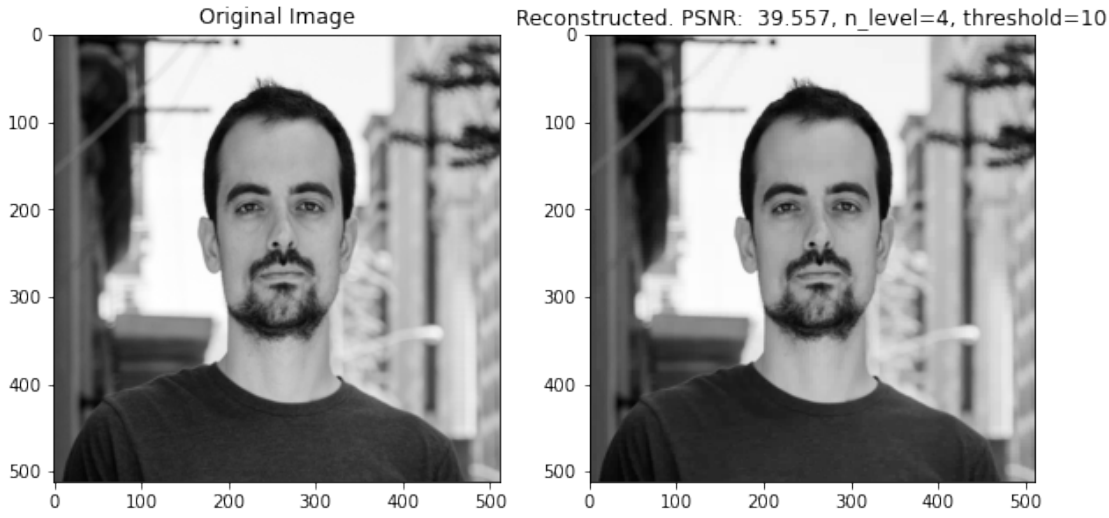

Original Image
Reconstructed. PSNR: 29.818, n_level=4, threshold=1.5
0
100
200
300
400
500

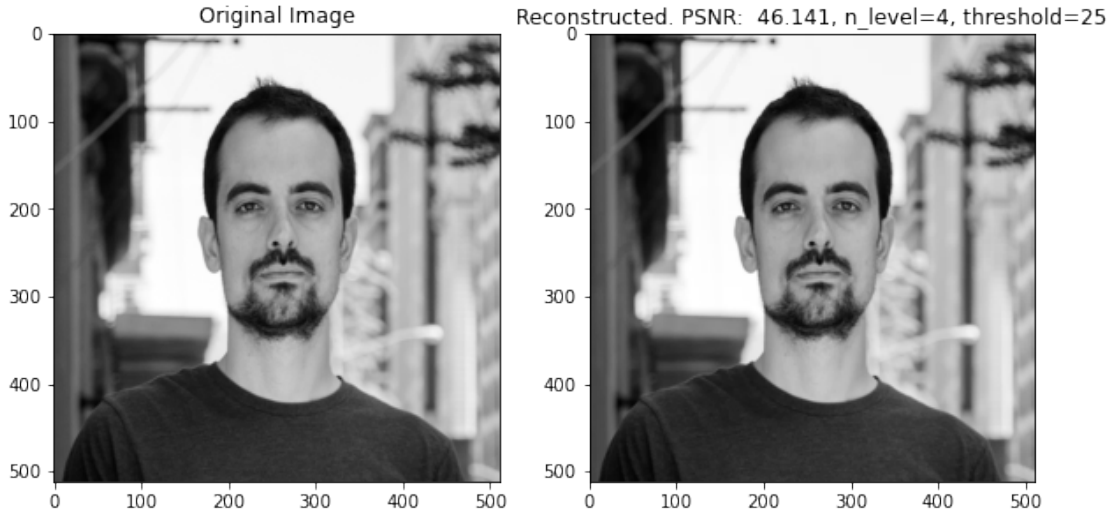

Original Image
Reconstructed. PSNR: 31.164, n_level=4, threshold=2
0
100
200
300
400
500

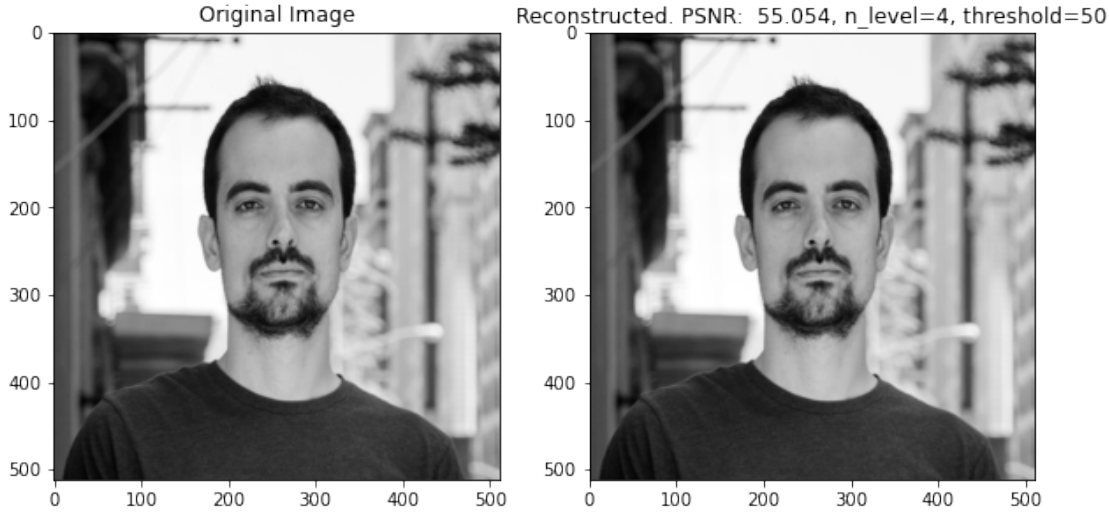

Original Image
Reconstructed. PSNR: 33.115, n_level=4, threshold=3
0
100
200
300
400
500

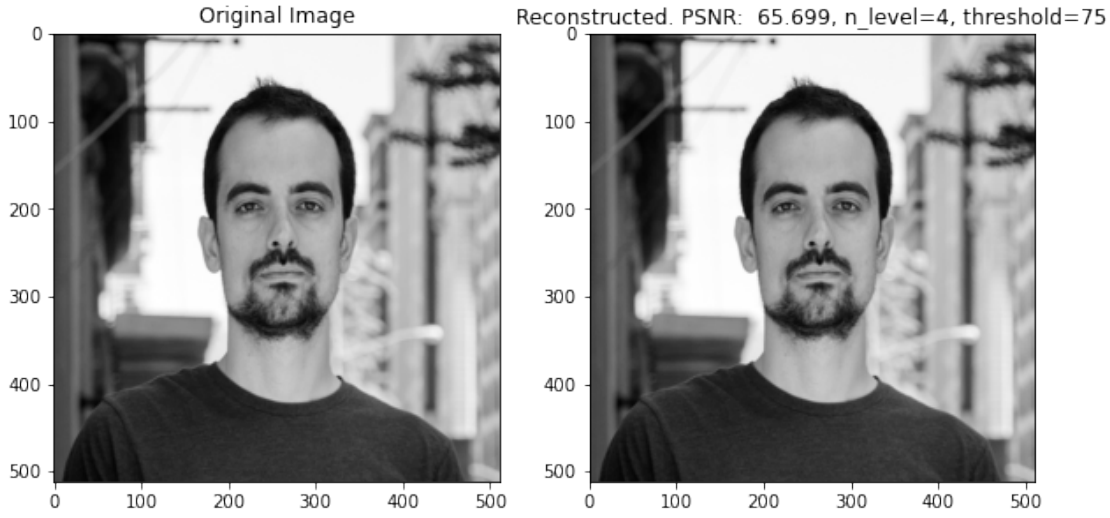

Original Image
Reconstructed. PSNR: 34.552, n_level=4, threshold=4
0
100
200
300
400
500

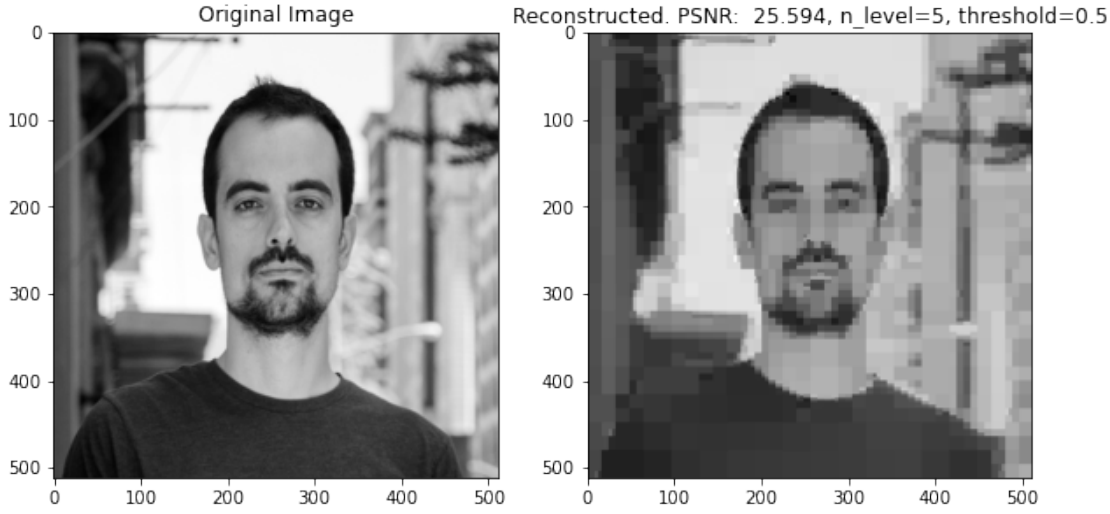

Original Image
Reconstructed. PSNR: 35.683, n_level=4, threshold=5
0
100
200
300
400
500

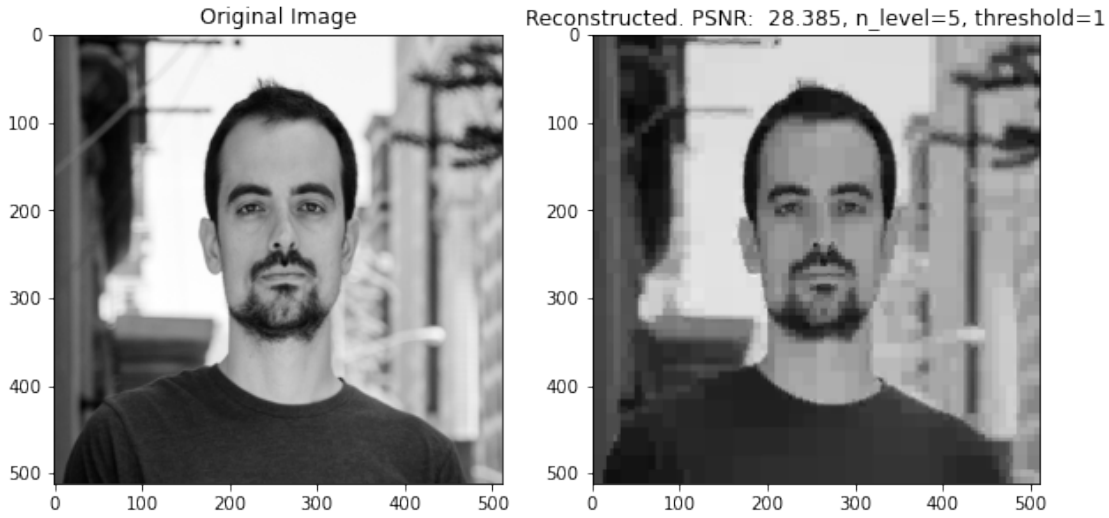

Original Image
Reconstructed. PSNR: 39.557, n_level=4, threshold=10
0
100
200
300
400
500

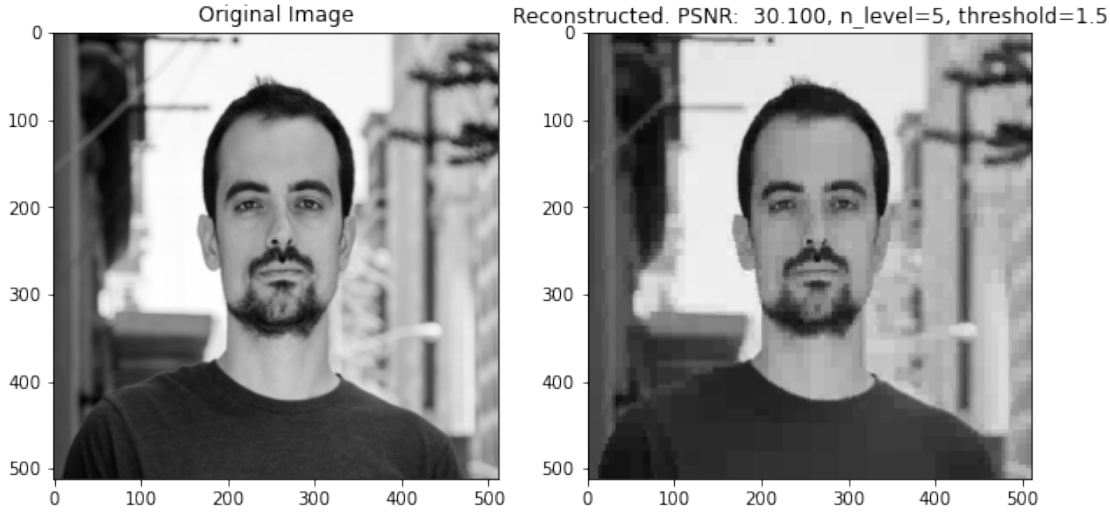

Original Image
Reconstructed. PSNR: 46.141, n_level=4, threshold=25
0
100
200
300
400
500

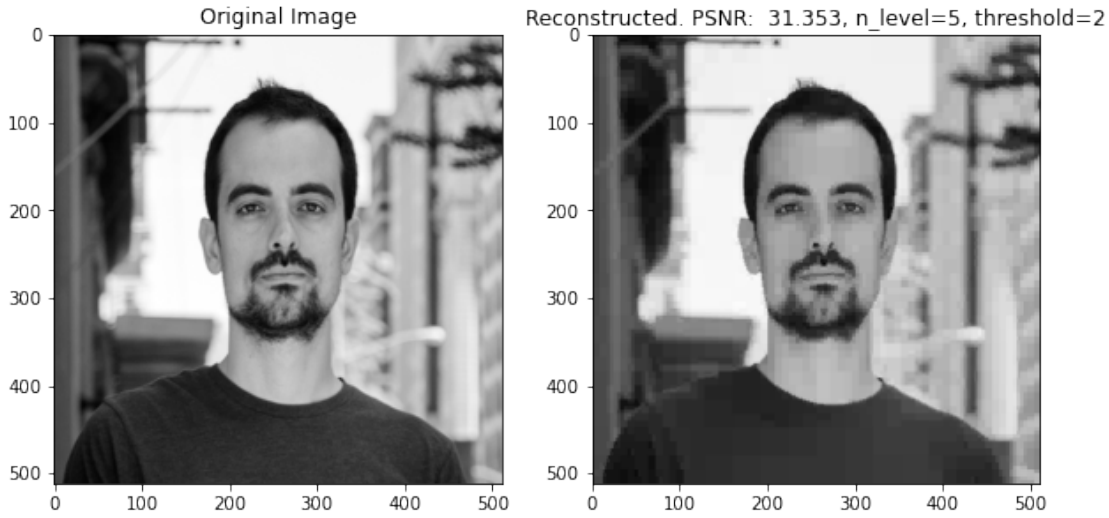

Original Image
Reconstructed. PSNR: 55.054, n_level=4, threshold=50
0
100
200
300
400
500

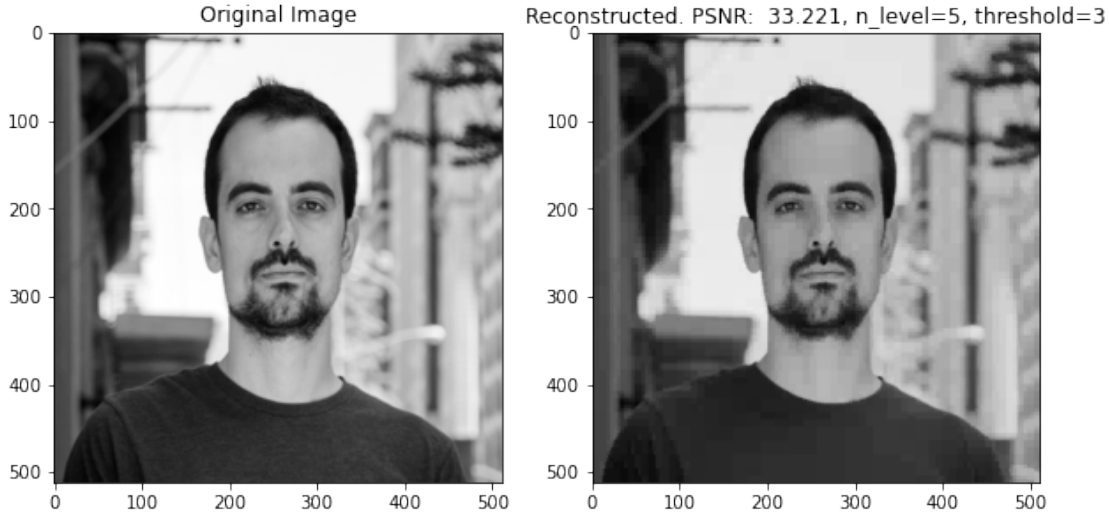

Original Image
Reconstructed. PSNR: 65.699, n_level=4, threshold=75
0
100
200
300
400
500

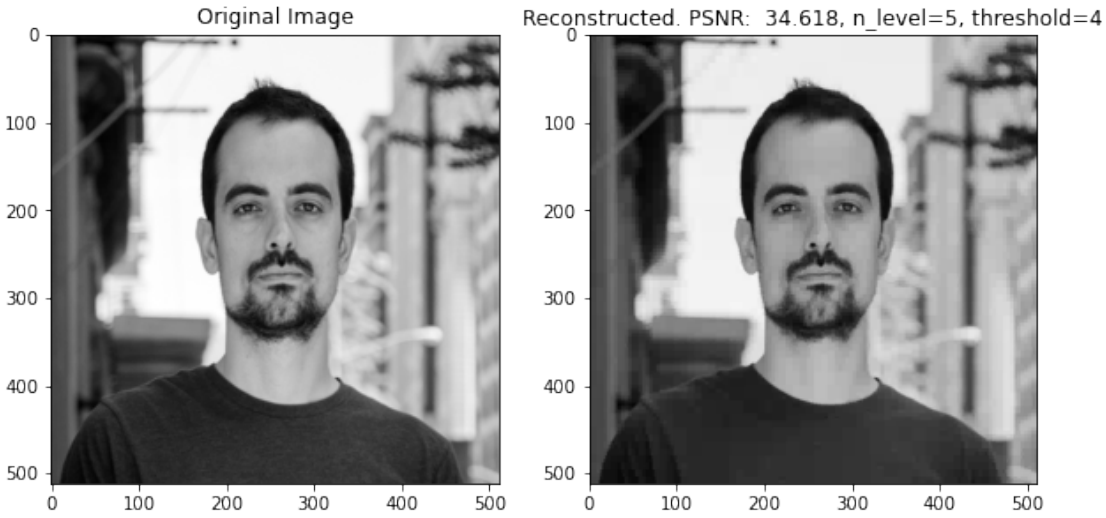

Original Image
Reconstructed. PSNR: 25.594, n_level=5, threshold=0.5
0
100
200
300
400
500

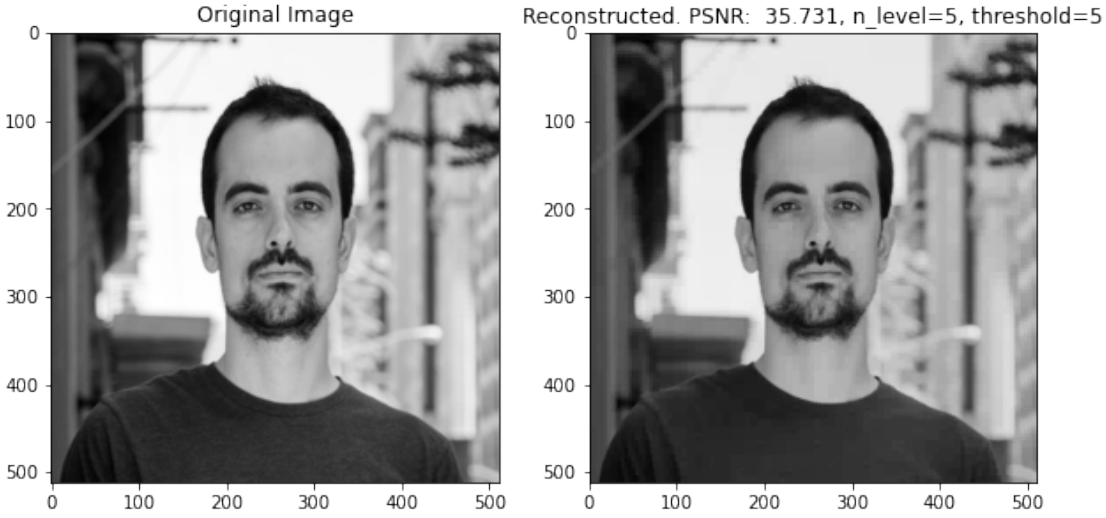

Original Image
Reconstructed. PSNR: 28.385, n_level=5, threshold=1
0
100
200
300
400
500

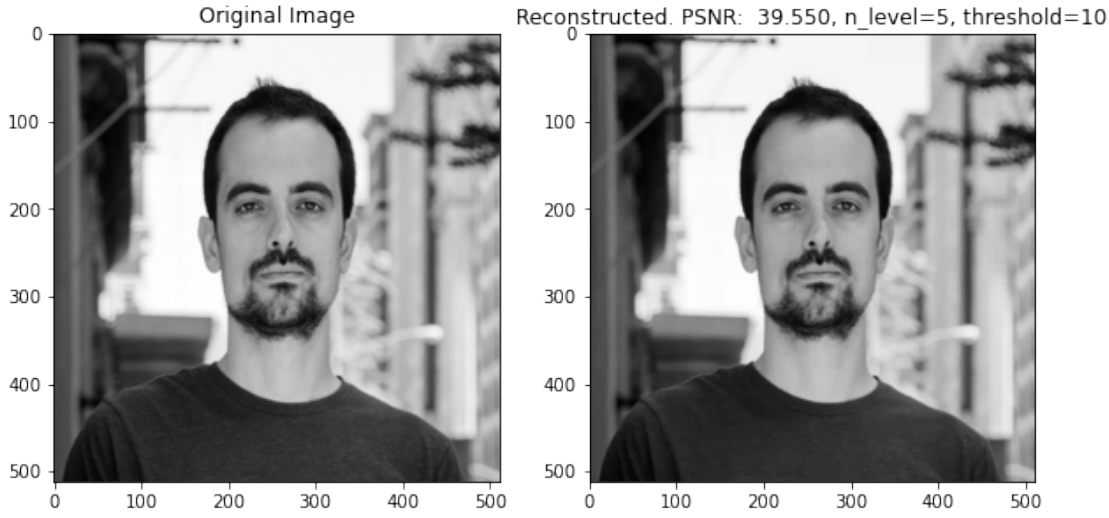

Original Image
Reconstructed. PSNR: 30.100, n_level=5, threshold=1.5
0
100
200
300
400
500

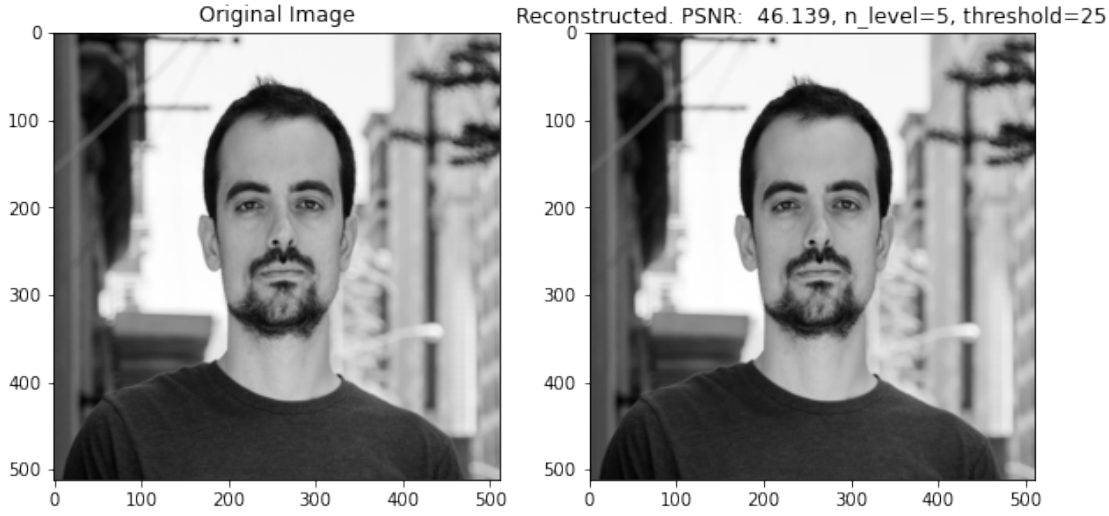

Original Image
Reconstructed. PSNR: 31.353, n_level=5, threshold=2
0
100
200
300
400
500

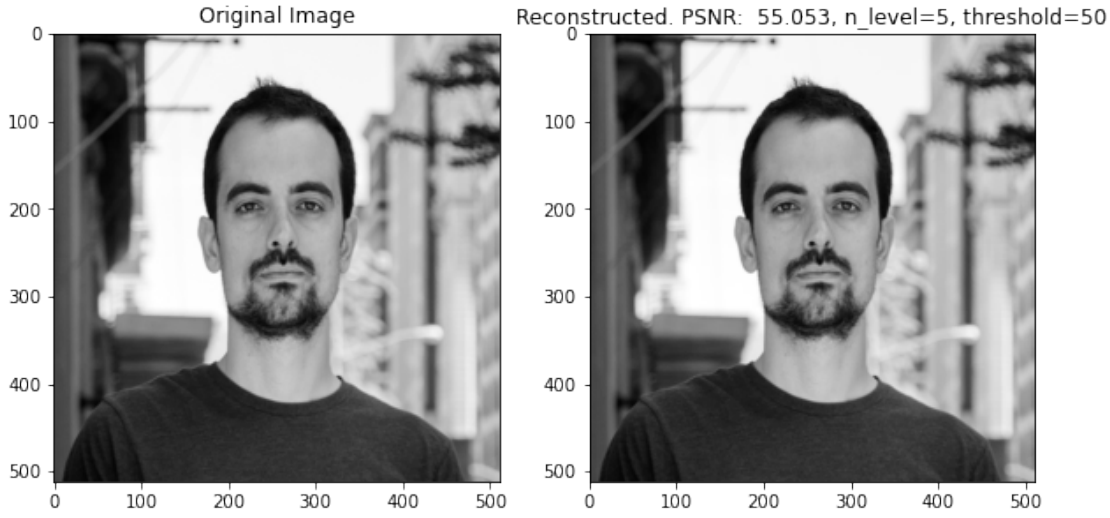

Original Image
Reconstructed. PSNR: 33.221, n_level=5, threshold=3
0
100
200
300
400
500

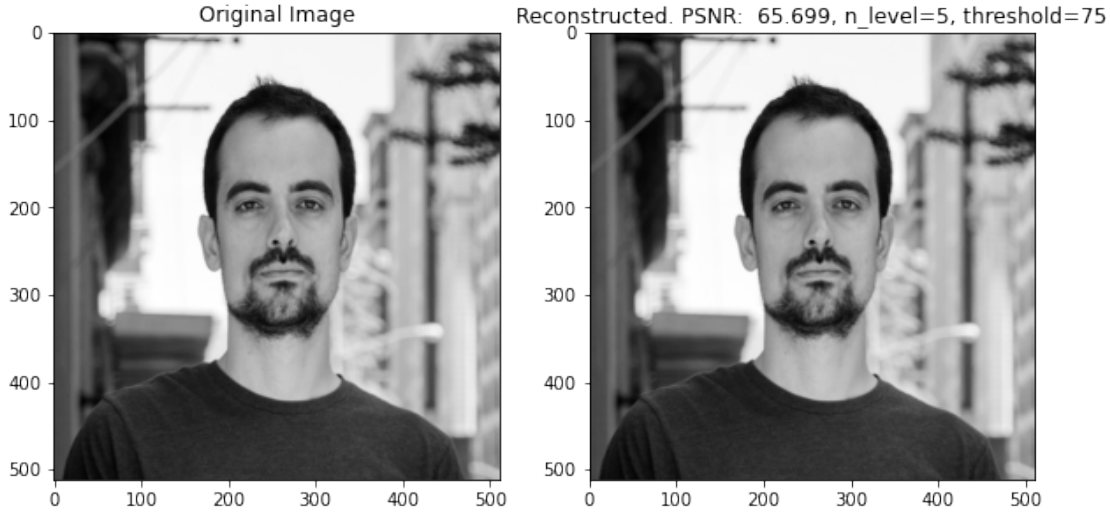

Original Image
Reconstructed. PSNR: 34.618, n_level=5, threshold=4
0
100
200
300
400
500


Original Image
Reconstructed. PSNR: 35.731, n_level=5, threshold=5
0
100
200
300
400
500


Original Image
Reconstructed. PSNR: 39.550, n_level=5, threshold=10
0
100
200
300
400
500


Original Image
Reconstructed. PSNR: 46.139, n_level=5, threshold=25
0
100
200
300
400
500

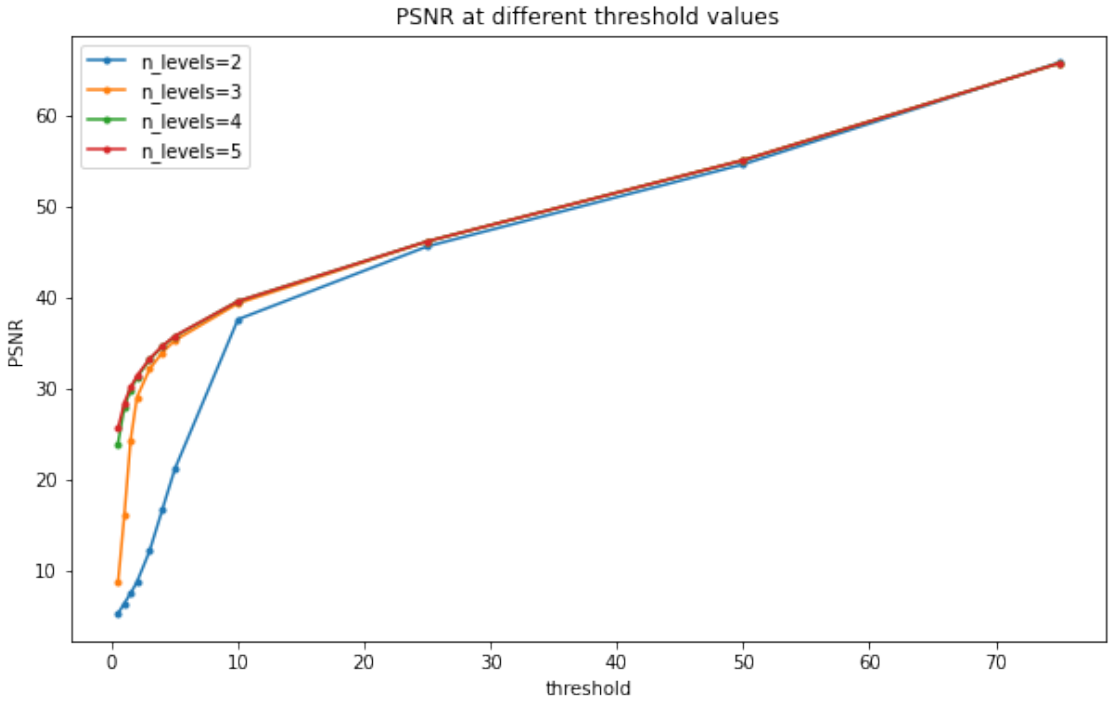




(b)

[9]:
```
fig, ax = plt.subplots(figsize=(10, 6))
for i in range(len(PSNR)):
    ax.plot(pct, PSNR[i], '.-', label='n_levels={}'.format(n_levels[i]))
ax.legend()
ax.set_xlabel('threshold')
ax.set_ylabel('PSNR')
ax.set_title('PSNR at different threshold values')
```

[9]: Text(0.5, 1.0, 'PSNR at different threshold values')


PSNR at different threshold values
n_levels=2
n_levels=3
n_levels=4
n_levels=5
PSNR
threshold
10
20
30
40
50
60
0
10
20
30
40
50
60
70