MATH-GA 2840 HW#8

Yifei(Fahy) Gao yg1753

1. (Haar wavelet) Define the discrete Haar wavelet $\mu_{2^s,p} \in \mathbb{R}^{2^n}$ at scale 2^s and position p by

$$\mu_{2^s,p}[j] := \begin{cases} -1/\sqrt{2^s} & \text{if } j \in \left\{p \cdot 2^s, p \cdot 2^s + 1, \ldots, p \cdot 2^s + 2^{s-1} - 1\right\} \\ 1/\sqrt{2^s} & \text{if } j \in \left\{p \cdot 2^s + 2^{s-1}, p \cdot 2^s + 2^{s-1} + 1, \ldots, (p+1) \cdot 2^s - 1\right\} \\ 0 & \text{otherwise} \end{cases}$$

where $0 < s \leq n$ and $0 \leq p \leq 2^{n-s} - 1$. Define the discrete Haar scaling function $\varphi_{2^s,p} \in \mathbb{R}^{2^n}$ at scale 2^s and position p by

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

where $0 < s \leq n$ and $0 \leq p \leq 2^{n-s} - 1$.

(a) Define $V_0 := \mathbb{R}^{2^n}$. For $k > 0$, let $V_k \subset \mathbb{R}^{2^n}$ denote the subspace of all vectors that are constant on segments of size 2^k. That is

$$V_k := \left\{x \in \mathbb{R}^{2^n} : x[i] = x[j] \text{ if } \left\lfloor i/2^k \right\rfloor = \left\lfloor j/2^k \right\rfloor\right\}$$

Give an orthonormal basis for V_k (proving that it is indeed orthonormal and a basis). What is the dimension of V_k?

Since $V_k \subset \mathbb{R}^{2^n}$, and it contains the segments of size 2^k where $n > k$, then there will be another 2^{n-k} segments. Now we can define V_k with 2^{n-k} orthogonal vectors that are constant on 2^k but 0 otherwise. Then the dimension of V_k will be 2^{n-k}.

By the lecture notes, we can define the the basis of V_k is:

$$B := \left\{\varphi_{2^k,p} : 0 \leq p \leq 2^{n-k} - 1\right\}$$

Then

$$\begin{aligned} \left\langle \varphi_{2^k,p}, \varphi_{2^k,p} \right\rangle &= \sum_{j=0}^{2^n-1} \varphi_{2^k,p}[j]^2 = \sum_{j=2^k p}^{(p+1)2^k-1} 1/2^k = \frac{(p+1)2^k - 1 - 2^k p + 1}{2^k} \\ &= \frac{2^k}{2^k} \\ &= 1 \end{aligned}$$

Then the vectors $\varphi_{2^k,p}$ are unit length vectors.

Also, assume there are p and q: $0 \leq p \leq q \leq 2^{n-k} - 1$, then

$$\left\langle \varphi_{2^k,p}, \varphi_{2^k,q} \right\rangle = \sum_{j=0}^{2^n-1} \varphi_{2^k,p}[j]\varphi_{2^k,q}[j] = \sum_{j=2^k p}^{(p+1)2^k-1} \varphi_{2^k,p}[j]\varphi_{2^k,q}[j] + \sum_{j=2^k q}^{(q+1)2^k-1} \varphi_{2^k,p}[j]\varphi_{2^k,q}[j]$$
$$= 0 + 0$$
$$= 0$$

Thus B forms an orthnormal basis of V_k.

(b) Show that one can project onto V_k by averaging (explain what needs to be averaged).

Let $proj_{V_k} u$ be the projection of u onto V_k, then:

$$proj_{V_k} u = \sum_{p=0}^{2^{n-k}-1} \left\langle u, \varphi_{2^k,p} \right\rangle \varphi_{2^k,p}$$

And since basis of V_k only contains non-overlapping support, then

$$proj_{V_k} u[j] = \left\langle u, \varphi_{2^k,q} \right\rangle \varphi_{2^k,q}[j]$$

Then based on the question condition,

$$= 1/\sqrt{2^k} \sum_{j=2^k q}^{(q+1)2^k-1} u[j](1/\sqrt{2^k}) = (1/2^k) \sum_{j=2^k q}^{(q+1)2^k-1} u[j]$$

Thus we now can say $proj_{V_k} u$ is the average of the elements $u[j]$ for $2^k q \le j \le (q+1)2^k - 1$, which means that elements of $proj_{V_k} u[j]$ are the average value of u on the same 2^k length segment.

(c) Show that $V_{k+1} \subset V_k$

If we want to show V_{k+1} is the subset of V_k then we can try to prove the basis of V_{k+1} is also nested inside the basis of V_k.

We know:

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

and A

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

Then,

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

Which is exactly the function definiton of $\varphi_{2^{k+1},p}[j]$. Also $|B_{V_{k+1}}| = 2^{n-k-1} < |B_{V_k}|$. Then, $V_{k+1} \subset V_k$.

(d) Fix $0 \leq k < n$. Consider the set

$$W_{k+1} = \{x \in V_k : \langle x, y\rangle = 0 \text{ for all } y \in V_{k+1}\}$$

the orthogonal complement of V_{k+1} in V_k, so that $V_k = V_{k+1} \oplus W_{k+1}$. Give an orthonormal basis for W_{k+1} (proving that it is indeed orthonormal and a basis).

Since V_k and V_{k+1} have the dimensions 2^{n-k} and 2^{n-k-1}, and we need to locate the condition of W_{k+1} that $2^{n-k} - 2^{n-k-1} = 2^{n-k-1}$. Then let us assume:

$$B_W := \left\{\psi_{2^{k+1},p} : 0 \leq p \leq 2^{n-k-1}\right\}$$

Then

$$\begin{aligned} \left\langle \varphi_{2^{k+1},p}, \psi_{2^{k+1},p} \right\rangle &= \sum_{j=0}^{2^n-1} \psi_{2^{k+1},p}[j]\varphi_{2^{k+1},p}[j] \\ &= -\sum_{j=2^{k+1}p}^{p2^{k+1}+2^k-1} \frac{1}{\sqrt{2^{k+1}}} + \sum_{j=2^{k+1}p+2^k}^{(p+1)2^{k+1}-1} \frac{1}{\sqrt{2^{k+1}}} \\ &= \frac{2^k}{\sqrt{2^{k+1}}}(-1+1) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \left\langle \varphi_{2^{k+1},p}, \psi_{2^{k+1},q} \right\rangle &= \sum_{j=0}^{2^n-1} \psi_{2^{k+1},p}[j]\varphi_{2^{k+1},q}[j] \\ &= 0 \end{aligned}$$

So both are results of non-overlapping support.

Then, for $\psi_{2^{k+1},p}$ in B_W

$$\begin{aligned} \left\langle \psi_{2^{k+1},p}, \psi_{2^{k+1},p} \right\rangle &= \sum_{j=0}^{2^n-1} \psi_{2^{k+1},p}[j]^2 \\ &= \sum_{j=p^{k+1}}^{p2^{k+1}+2^k-1} \psi_{2^{k+1},p}[j]^2 + \sum_{j=p2^{k+1}+2^k}^{(p+1)2^{k+1}-1} \psi_{2^{k+1},p}[j]^2 \\ &= \sum_{j=p2^{k+1}}^{p2^{k+1}+2^k-1} \frac{1}{2^{k+1}} + \sum_{j=p2^{k+1}+2^k}^{(p+1)2^{k+1}-1} \frac{1}{2^{k+1}} \\ &= \frac{(p+1)2^{k+1} - 1 - p2^{k+1} + 1}{2^{k+1}} \\ &= 1 \end{aligned}$$

And, for $\psi_{2^{k+1},p}, \psi_{2^{k+1},q}$ in B_W

$$\begin{aligned}\left\langle \psi_{2^{k+1},p}, \psi 2^{k+1}, q \right\rangle &= \sum_{j=0}^{2^n-1} \psi_{2^{k+1},p}[j]\psi_{2^{k+1},q}[j] \\ &= \sum_{j=p2^{k+1}}^{(p+1)2^{k+1}-1} \psi_{2^{k+1},p}[j]\psi_{2^{k+1},q}[j] + \sum_{j=q2^{k+1}}^{(q+1)2^{k+1}-1} \psi_{2^{k+1},p}[j]\psi 2^{k+1}, q[j] \\ &= 0\end{aligned}$$

Therefore B_W is an orthnormal basis.

(e) For $1 \leq k \leq n$ give an orthonormal basis for the set

$$W_{\leq k} = \left\{ x \in \mathbb{R}^{2^n} : \langle x, y \rangle = 0 \text{ for all } y \in V_k \right\}$$

the orthogonal complement of V_k in $\mathbb{R}^{2^n}$, so that $\mathbb{R}^{2^n} = V_k \oplus W_{\leq k}$. Give an orthonormal basis for $W_{\leq k}$ (proving that it is indeed orthonormal and a basis).

We can define $W_{\leq k}$ to be $\oplus_{j=1}^{k} W_j = W_k \oplus W_{k-1} \oplus W_{k-2} \oplus W_1$, then:

Basis of $B_{W_{\leq k}}$ will be: $\bigcup_{j=1}^{k} B_{W_j}$,

and B_W from d part we know it has 2^{n-j} orthonormal vectors. Then we can assume that there are $u \in V_m$ and $v \in V_n$. If m=n-1, then from part d

$$u \in V_n \text{and } \langle u, v \rangle = 0$$

Else based on part c, $V_{n-1} \subset V_m$ and $\langle u, v \rangle = 0$ again.

Thus, $B_{W_{\leq k}}$ has all orthonormal vectors.

And for the dimension of $B_{W_{\leq k}}$:

$$|B_{W_{\leq k}}| = \sum_{j=1}^{k} |B_{W_j}| = \sum_{j=1}^{k} 2^{n-j} = 2^n - 2^{n-k}$$

Thus,

$$B_{W_{\leq k}}$$

has $2^n - 2^{n-k}$ orthonormal vectors.

2. (Implementation of Haar wavelets) The code for this exercise is in the haar.py file. Include all generated plots in your submission.

(a) Complete the wavelet and scaling functions in haar.py that implement μ and φ above, respectively. See the comments for more details.

In [2]:
```
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) :
    """
    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.
    """
    idx = np.arange(2**n)
    negative_basis = (np.greater_equal(idx, p *(2**s))*np.less(idx,p*(2**s)
    positive_basis = (np.greater_equal(idx, p *(2**s)+2**(s-1))*np.less(idx
    result = (1/np.sqrt(2**s))*(-negative_basis.astype(float)+positive_basi
    return result #Your code here

def scaling(s,p,n) :
    """
    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
    """
    idx = np.arange(2**n)
    basis = np.greater_equal(idx, p *(2**s))*np.less(idx,(p+1)*(2**s))
    result = (1/np.sqrt(2**s))*basis.astype(float)

    return result #Your code here
```

(b) Complete the projectV function that orthogonally projects a given vector onto V_k.

In [3]:
```
def projectV(x,k,n) :
    """
    Orthogonally projects the vector x onto the subspace W_k of Haar wavele
    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
    """
    project = np.zeros(2**n)
    for i in range(2**(n-k)):
        mean = np.mean(x[i*(2**k):(i+1)*(2**k)])
        project[i*(2**k):(i+1)*(2**k)] = mean
    return project#Your code here
```

(c) Complete the projectW function that orthogonally projects a given vector onto W_k.

In [4]:
```
def projectW(x,k,n) :
    """
    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,...
    """
    project = np.zeros(2**n)

    v = projectV(x,k-1,n)
    for p in range(2**(n-k)):
        mean = 0.5*(v[p*(2**k)+2**(k-1)]-v[p*(2**k)])
        project[p*(2**k):p*(2**k)+2**(k-1)] = - mean
        project[p*(2**k)+2**(k-1):(p+1)*(2**k)] =  mean
    return project   #Your code here
```

(d) Complete the function wavelet_coeffs which computes all of the (non-overlapping) wavelet coefficients of a given data vector at a given scale. See the comments for more details.

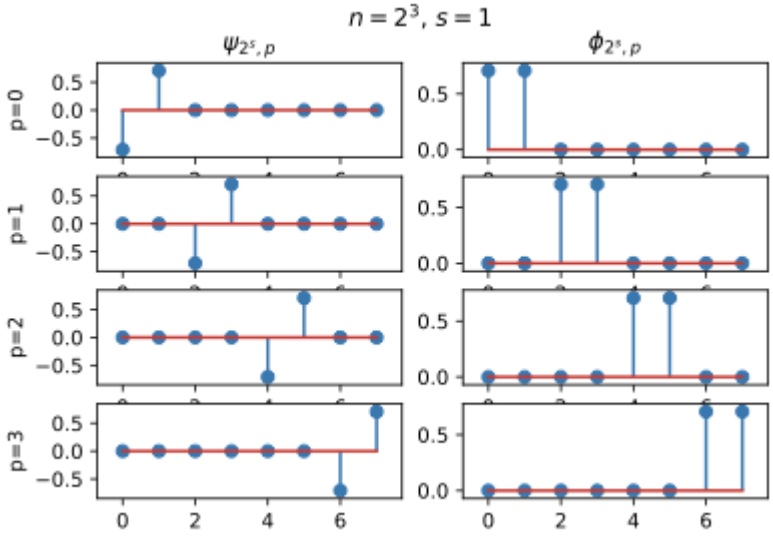
In [5]:
```
def wavelet_coeffs(x,s,n) :
    results = np.zeros(2**(n-s))
    for p in range(2**(n-s)):
        results[p] = np.sqrt(2**s)*x[p*(2**s):(p+1)*(2**s)].mean()

    return results #Your code here
```

(e) Report the plots generated by the code, which apply your wavelet transform to some electrocardiogram data.

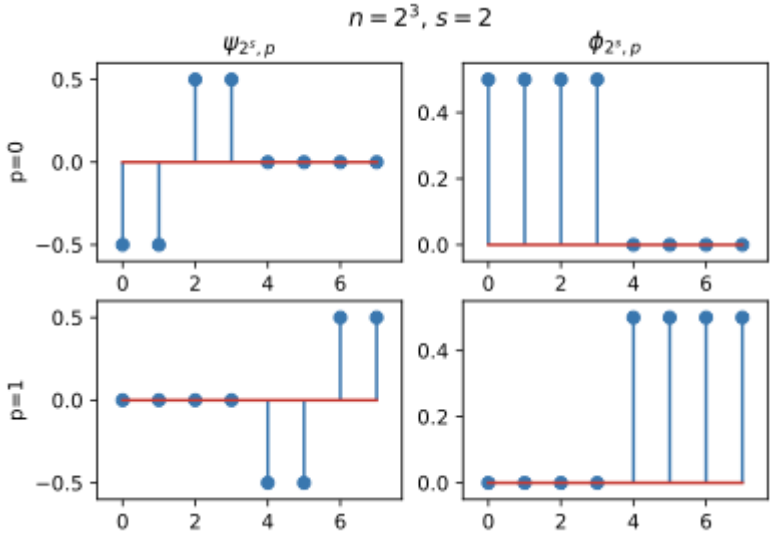
In [6]:
```
def projectWavelet(w,s,n) :
    ws = []
    for p in range(len(w)):
        ws.append(wavelet(s,p,n))
    return np.dot(np.array(ws).T,w)


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))
        wax.margins(.1,.1)
        sax.stem(range(2**n),scaling(s,p,n))
        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)
        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()
```

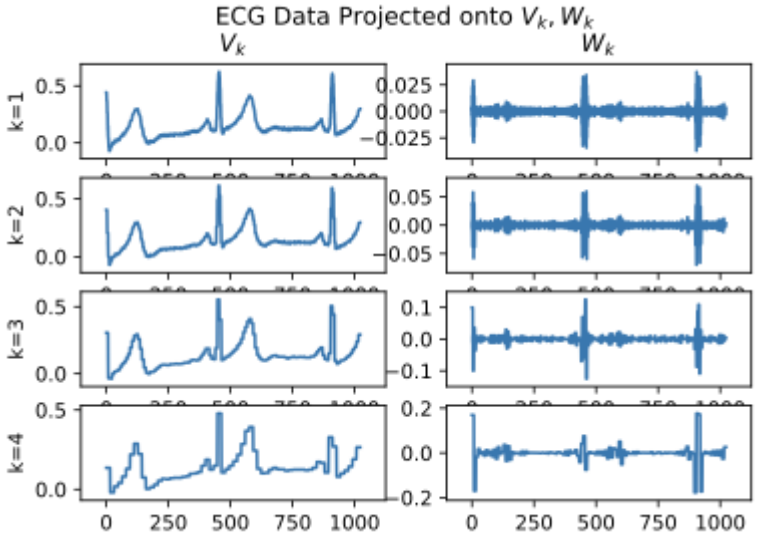
psiphiplot_1_3.pdf



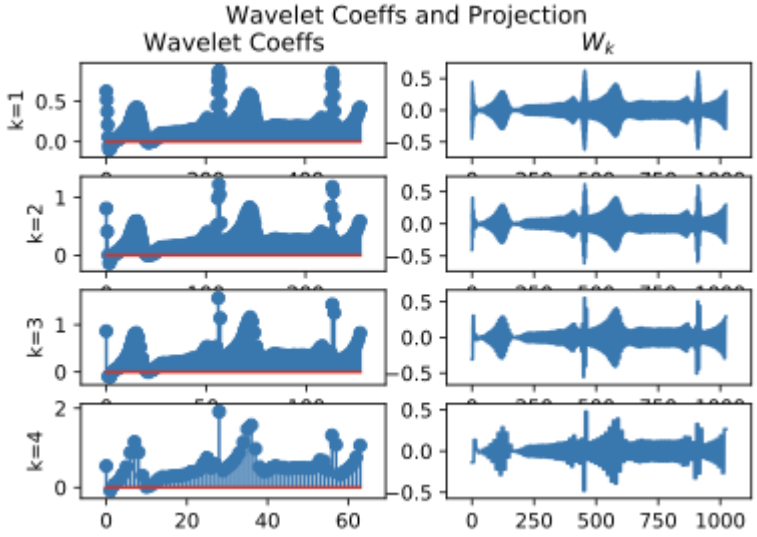
psiphiplot_2_3.pdf



ecg_project.pdf



ecg_wavelet.pdf

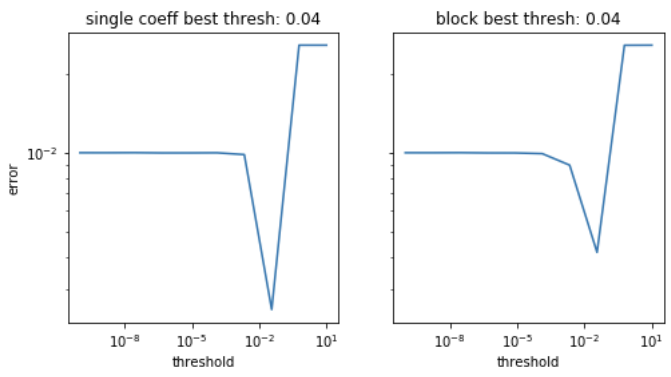


3. (Denoising with the STFT) In the lecture, we saw that STFT often yields sparse repre- sentation for a signal but dense representation for noise. Building on this, we derived hard thresholding and block thresholding to denoise signals. In this question, we will denoise audio signals. audio denoising.ipynb contains skeleton code for the task. The notebook will download required dataset and contains other utility functions for loading data, plot- ting and playing the audio signals. You have to fill in the functions get block L2 norm() and stft denoising(). Report all the plots generated by the script.

In [6]:
```
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 wind
                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.        ]]
    """
    pad = (window_size-1)//2
    mat = np.sqrt(signal.convolve2d(mat*mat,np.ones((window_size,window_siz


    return mat
```

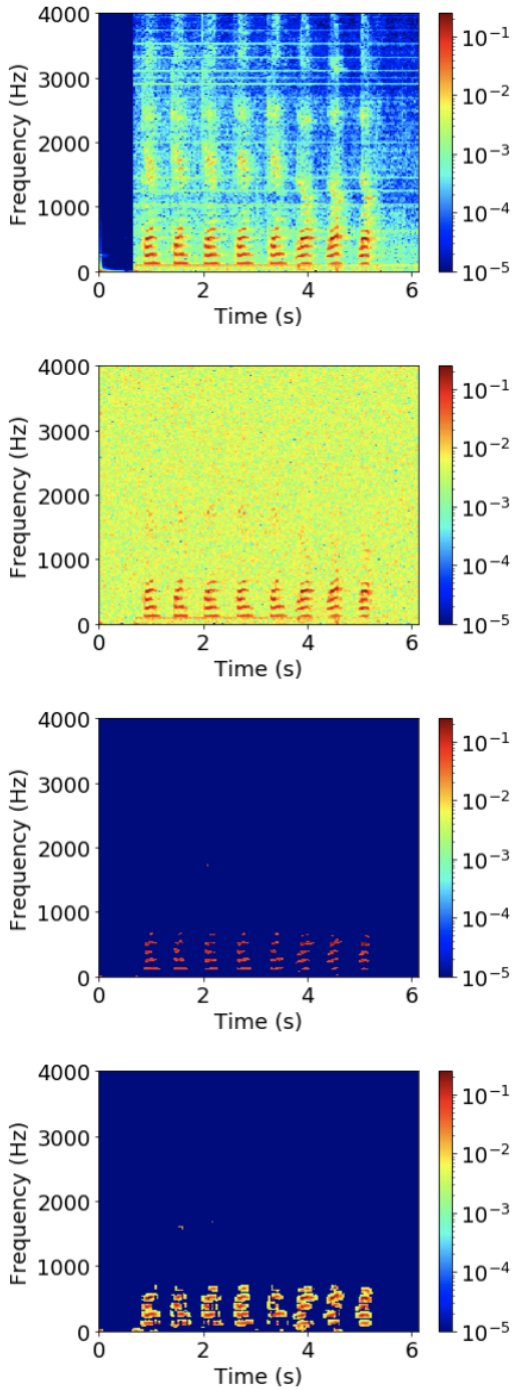
In [7]:
```
def stft_denoising(source, noise_std, fs, nperseg, thresh,
                   window_size, block_thresh = None, plot_res = True, ind=0

    """
    implements hard and block thresholding.

    thresh - threshold for hard thresholding. Thresholding is implemented p
    block_thresh - threshold for block thresholding. Thresholding is implem

    """

    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
    coeffs = noisy_stft[2]
    hardmask = np.greater(np.abs(coeffs),thresh)
    block12s= get_block_L2_norm(coeffs, window_size)
    blockmask= np.greater(block12s,block_thresh)

    denoised_stft = ( noisy_stft[0], noisy_stft[1], hardmask*coeffs)#HARD T
    block_denoised_stft =  ( noisy_stft[0], noisy_stft[1], blockmask*coeffs

    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=npe
    _, block_denoised_istft =   signal.istft(block_denoised_stft[2], fs=fs,


    return np.real(source_istft), np.real(noisy_istft), np.real(denoised_is
```

```
noise std:  0.1
```



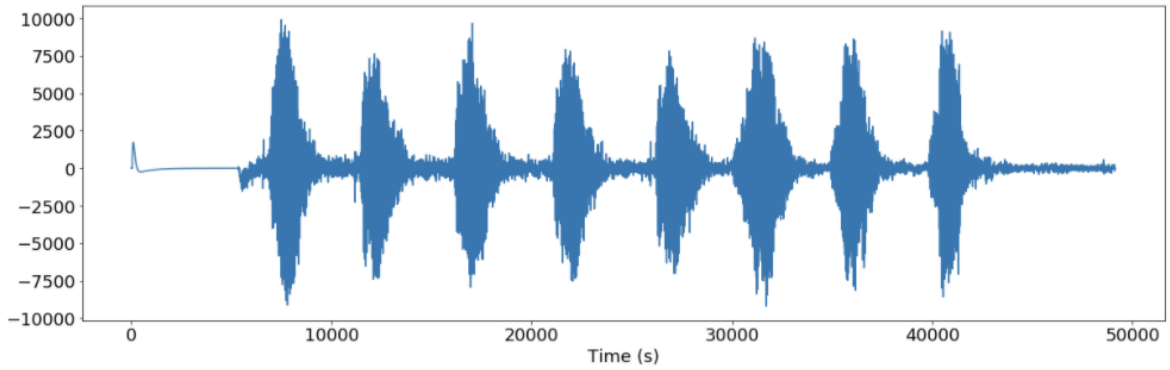

```
==================================================
```

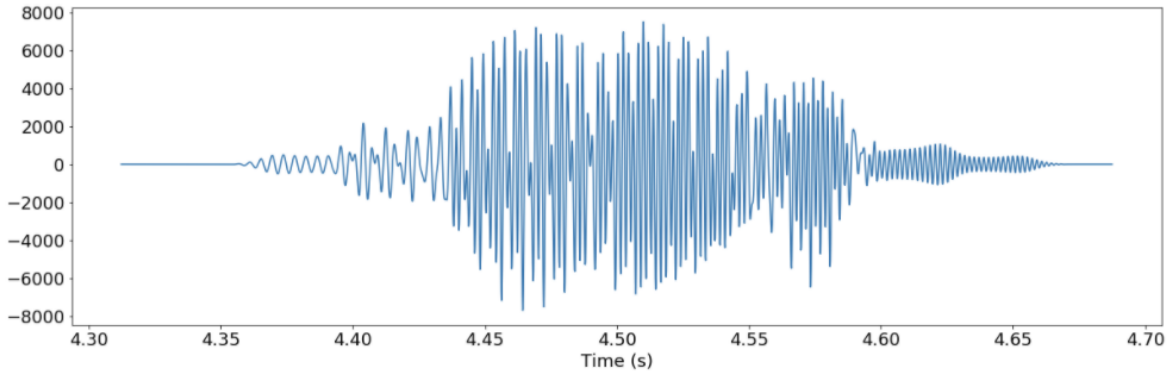
```
Noise Std:  0.1
```

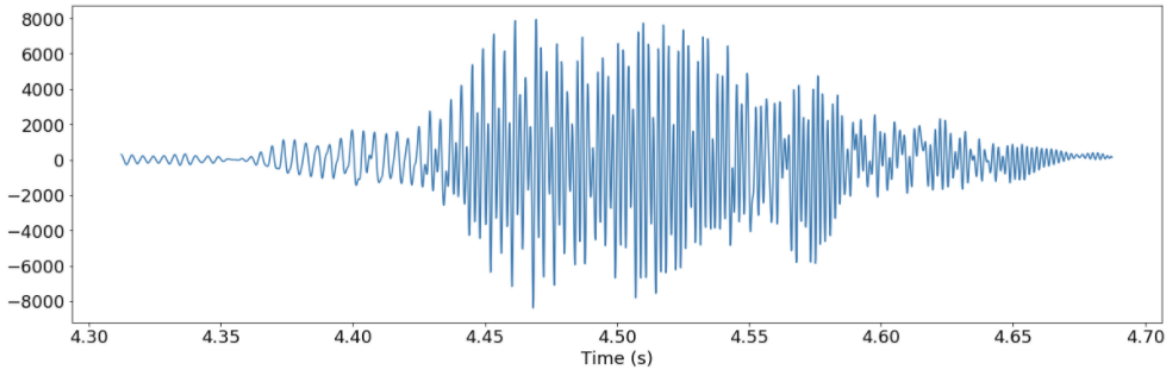

Frequency (Hz)
Time (s)
4000
3000
2000
1000
0
0
2
4
6
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}

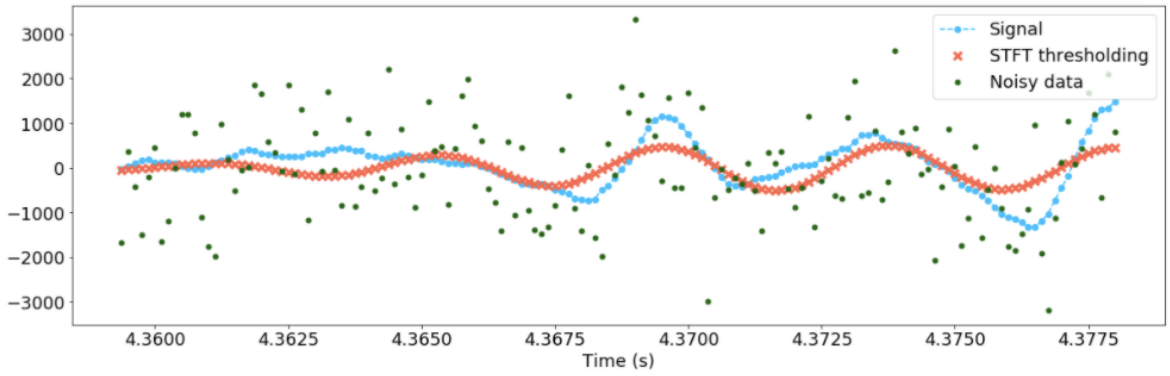
STFT Denoised:

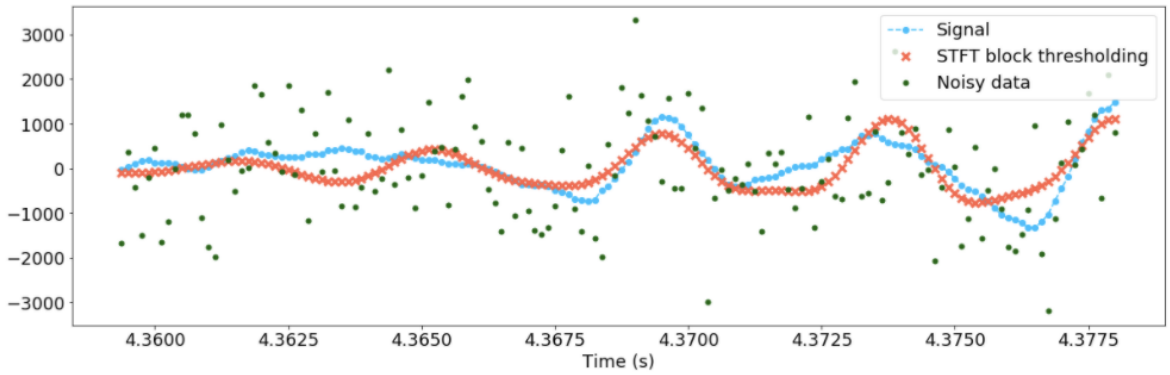
Block STFT Denoised:











==========

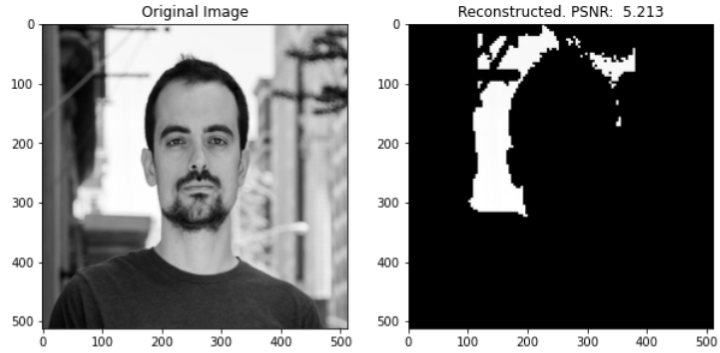
4. (Compression with wavelets) Similar to the STFT representation, a wavelet decompo- sition of an image is often sparse. In this question, we will exploit this sparsity to compress images. We will retain only the top x% of coefficients in the wavelet domain and set everything else to zero to perform compression. The support code provided in image wavelet.ipynb will load an image, perform its wavelet decomposition, recon- struct the image back from the wavelet domain and compute the Peak signal-to-noise ratio (PSNR) between the reconstructed image and the original image.

(a) Use haar wavelets as provided in the support code, but perform the wavelet de- composition for atleast [2, 3, 4, 5] levels. For each of these levels retain the top [0.5, 1, 1.5, 2, 3, 4, 5, 10, 25, 50, 75] percentage of coefficients and perform the reconstruction. You don't have to perform the thresholding in each sub-band separately - you can use the entire set of wavelet coefficients to determine the top x%. Plot the original image and reconstructed image side by side for each of the level and threshold using visualize image and recon()

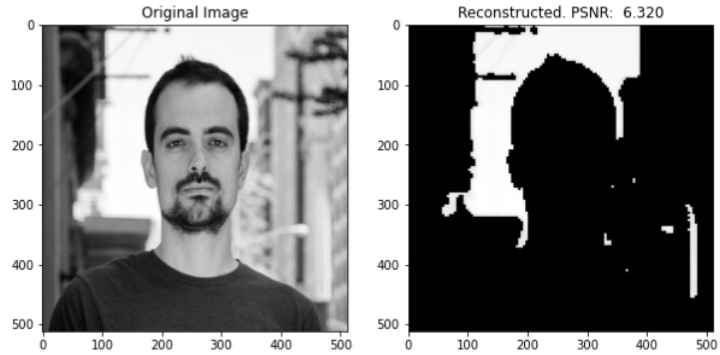
fill codes:

Visualizing wavelet coeffs

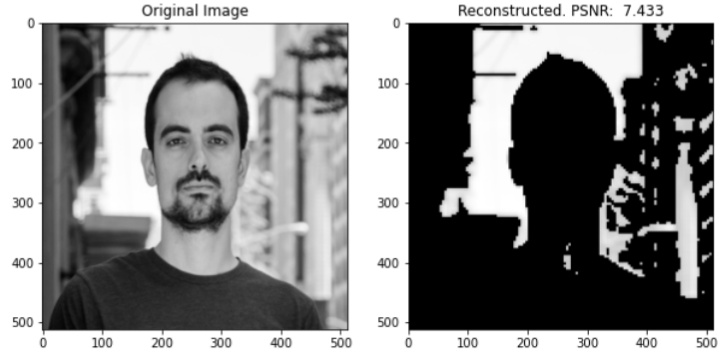
In [8]:
```
n_levels = [2, 3, 4, 5]
pct = [0.5, 1, 1.5, 2, 3, 4, 5, 10, 25, 50, 75]
titles=[]
wav_type = 'haar'
PSNR = []
n=0
for i in range(len(n_levels)):
    temp = []
    for j in range(len(pct)):
        x_w = pywt.wavedec2(image, wav_type, level=n_levels[i])
        coeff_array, coeff_slices = pywt.coeffs_to_array(x_w)
        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)
        temp.append(metrics.peak_signal_noise_ratio(image, recon_image,data_range = 255.))
        titles.append('percent kept:' +str(pct[j]/100)+' Number of levels:'+str(n_levels[i]))

        PSNR.append(temp)

        print(titles[n])
        n+=1
```



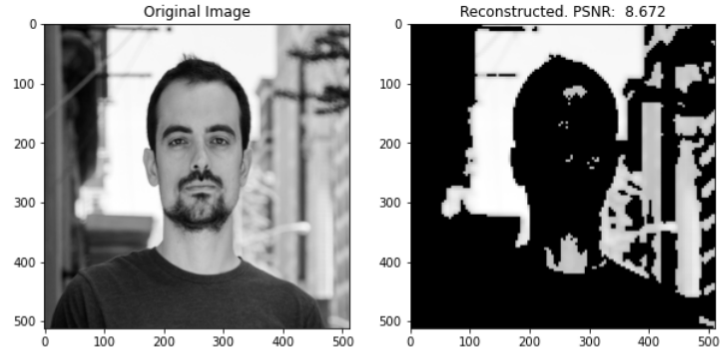

```
percent kept:0.005 Number of levels:2
```



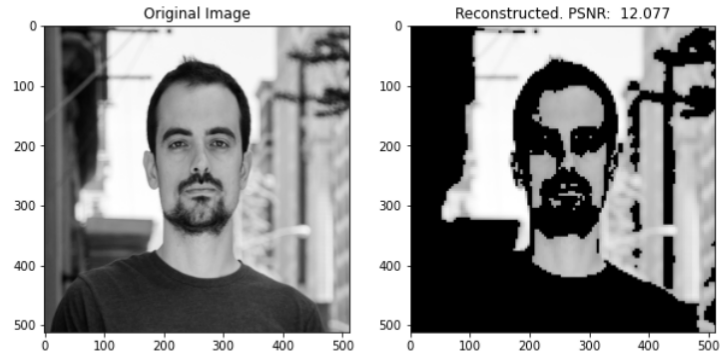

```
percent kept:0.01 Number of levels:2
```



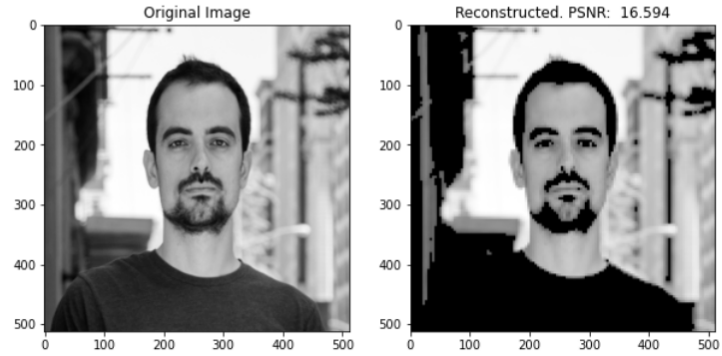

```
percent kept:0.015 Number of levels:2
```



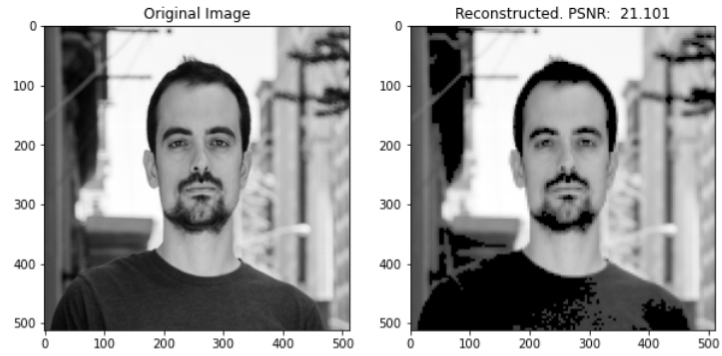

```
percent kept:0.02 Number of levels:2
```



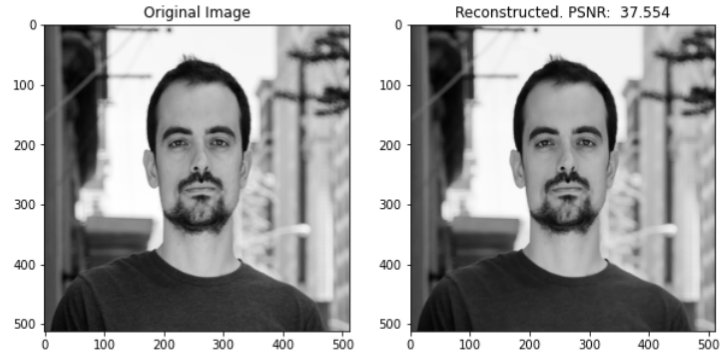

```
percent kept:0.03 Number of levels:2
```



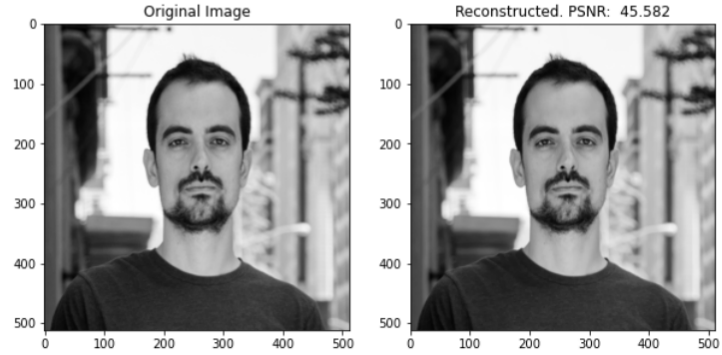

```
percent kept:0.04 Number of levels:2
```




```
percent kept:0.05 Number of levels:2
```

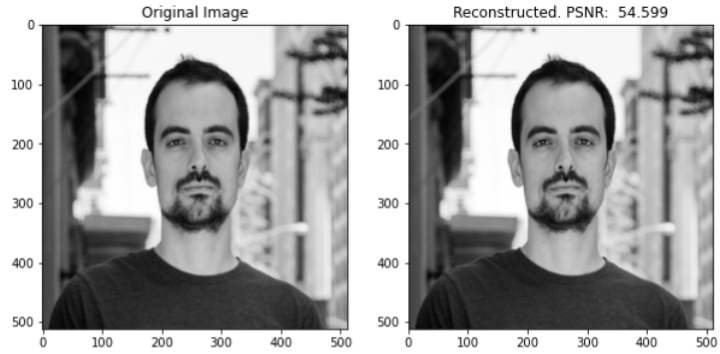



```
percent kept:0.1 Number of levels:2
```

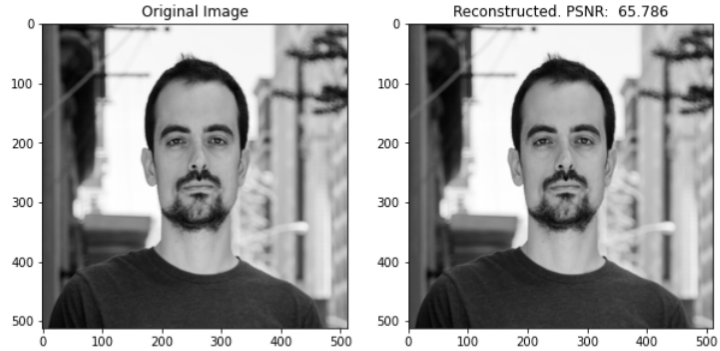



```
percent kept:0.25 Number of levels:2
```

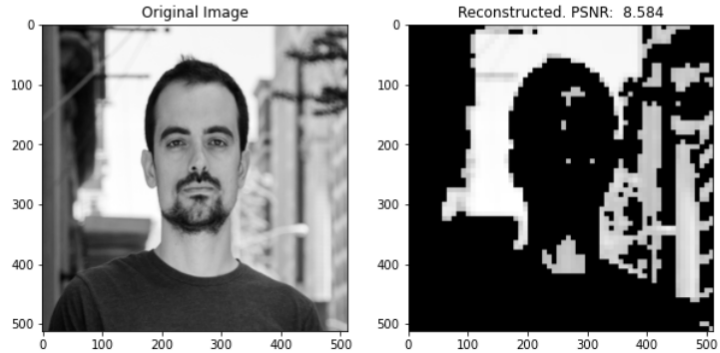
```
percent kept:0.25 Number of levels:2
```



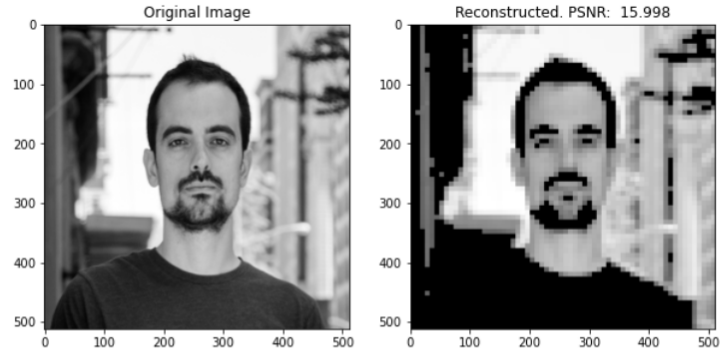

```
percent kept:0.5 Number of levels:2
```



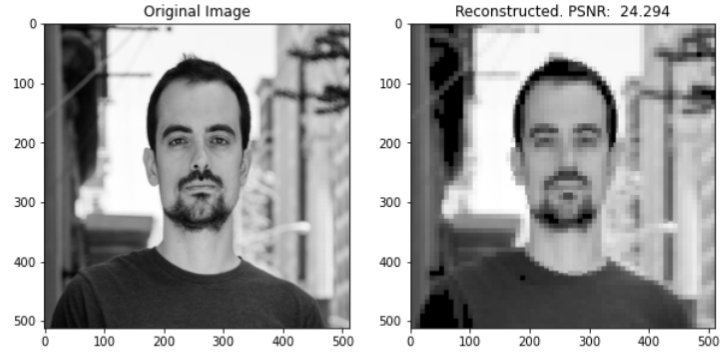

```
percent kept:0.75 Number of levels:2
```



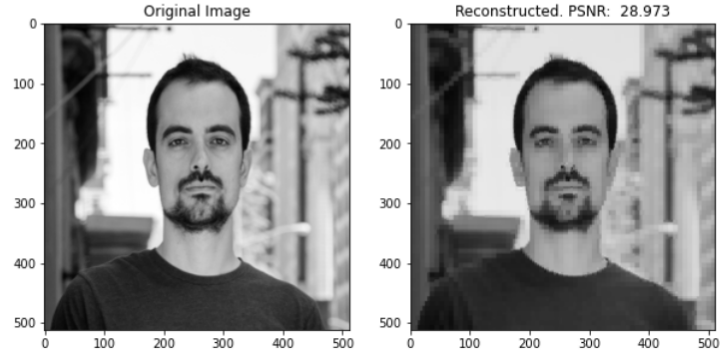
```
percent kept:0.005 Number of levels:3
```



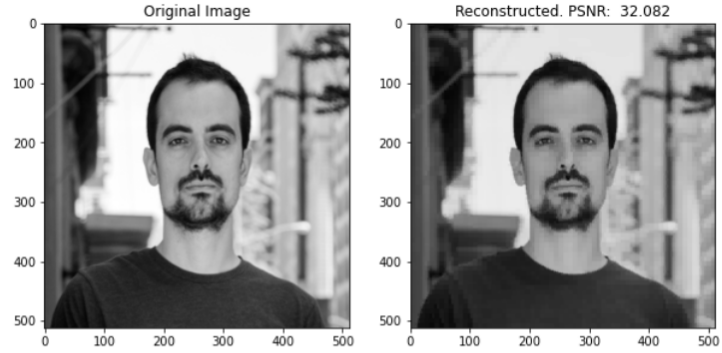

```
percent kept:0.01 Number of levels:3
```



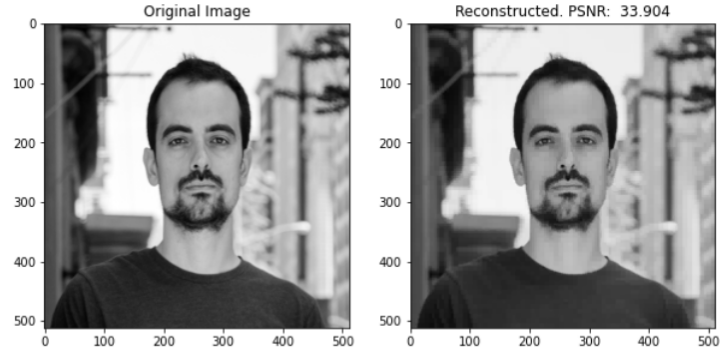

```
percent kept:0.015 Number of levels:3
```



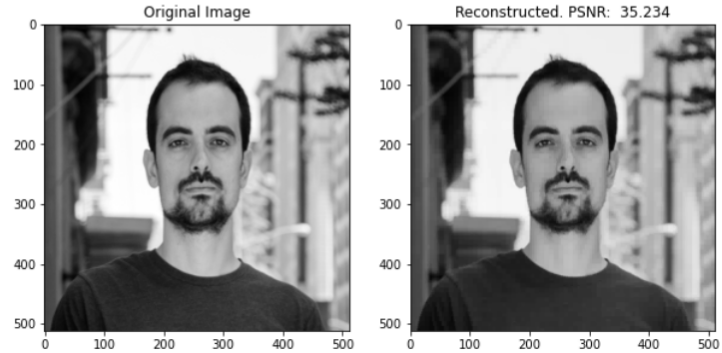
```
percent kept:0.02 Number of levels:3
```



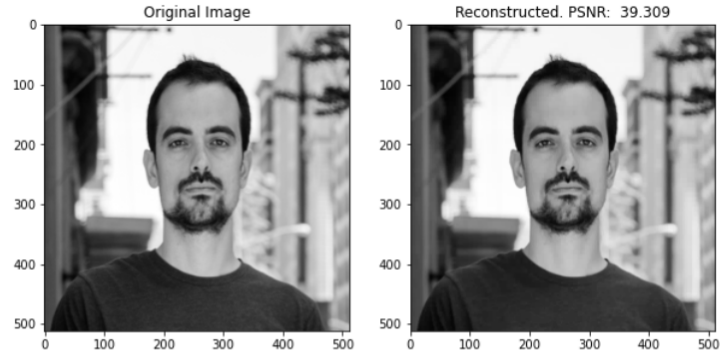

```
percent kept:0.03 Number of levels:3
```



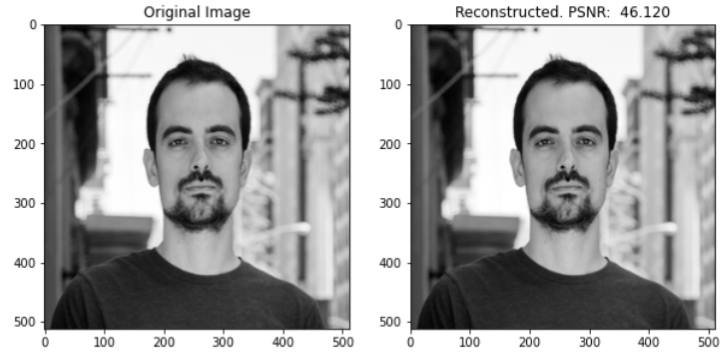

```
percent kept:0.04 Number of levels:3
```



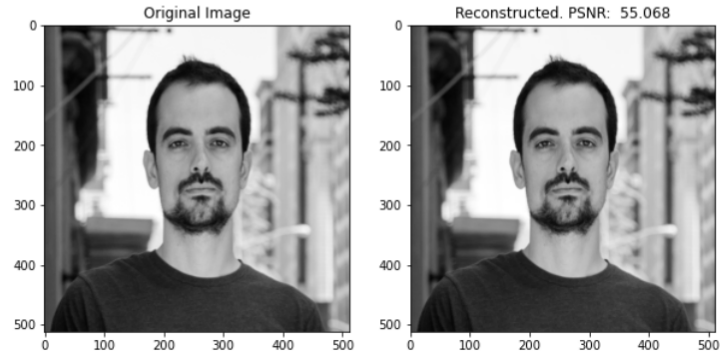
```
percent kept:0.05 Number of levels:3
```



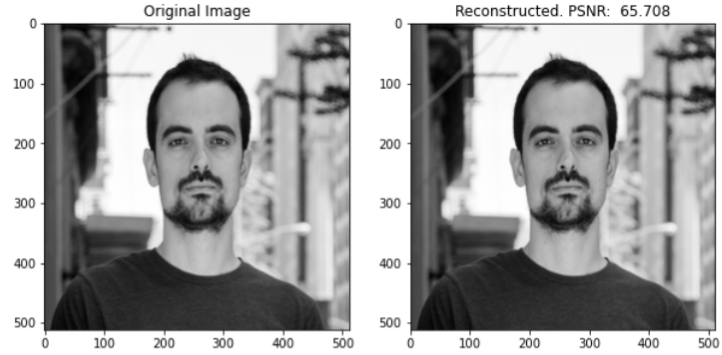

```
percent kept:0.1 Number of levels:3
```



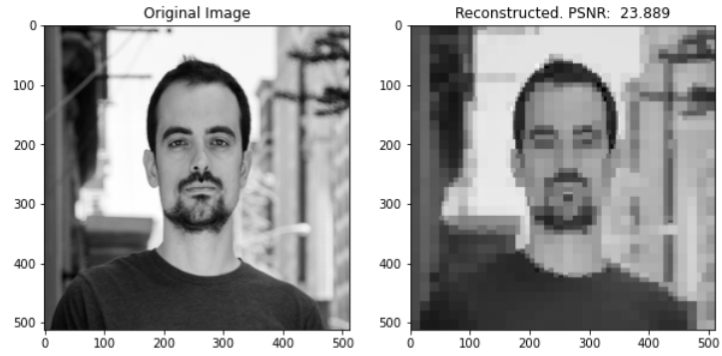

```
percent kept:0.25 Number of levels:3
```



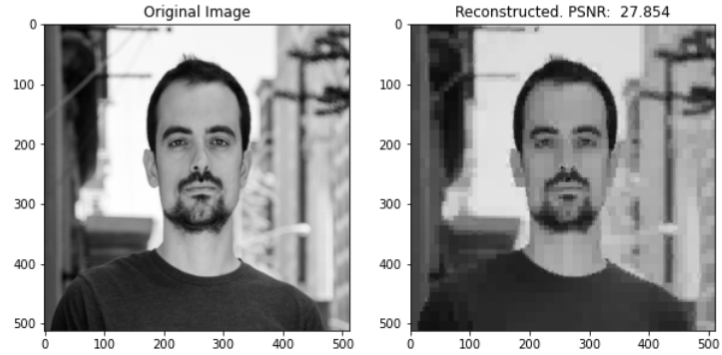
```
percent kept:0.5 Number of levels:3
```



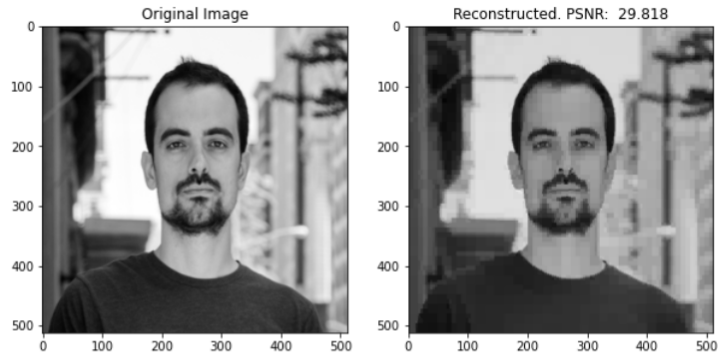

```
percent kept:0.75 Number of levels:3
```



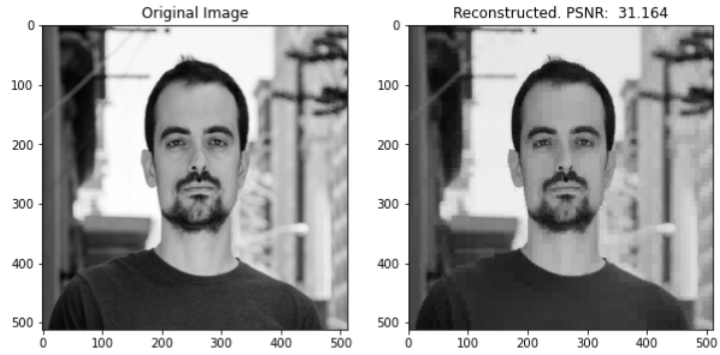

```
percent kept:0.005 Number of levels:4
```



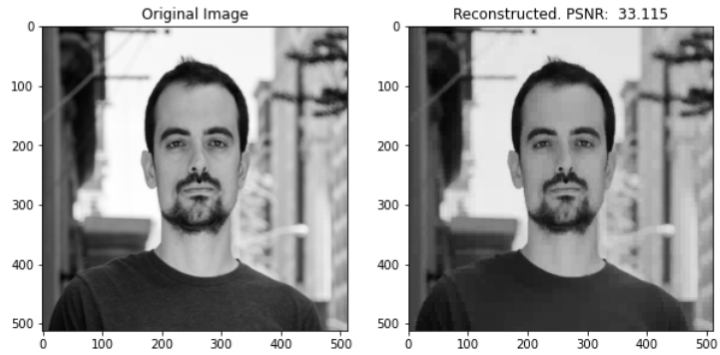
```
percent kept:0.01 Number of levels:4
```



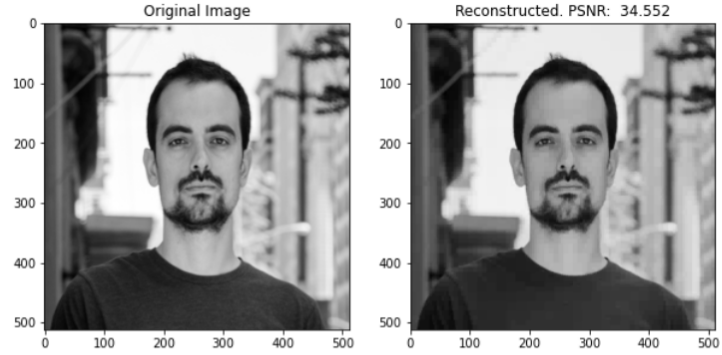

```
percent kept:0.015 Number of levels:4
```



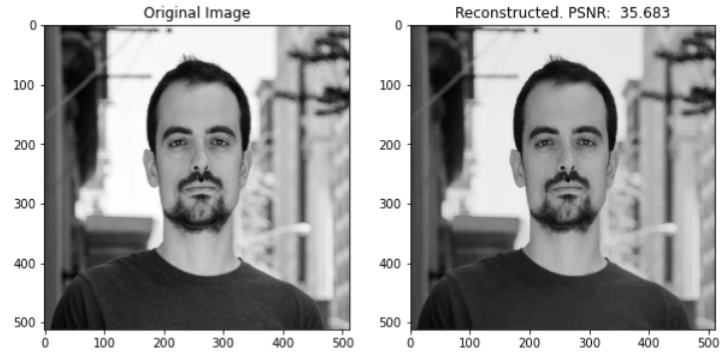

```
percent kept:0.02 Number of levels:4
```



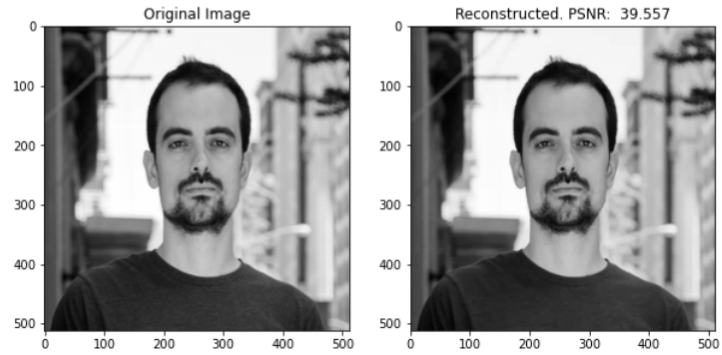
```
percent kept:0.03 Number of levels:4
```



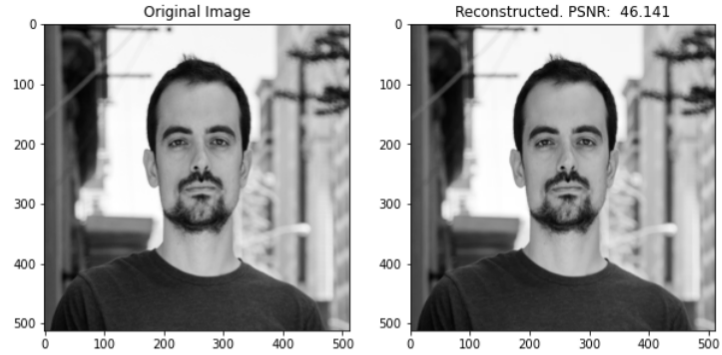

```
percent kept:0.04 Number of levels:4
```



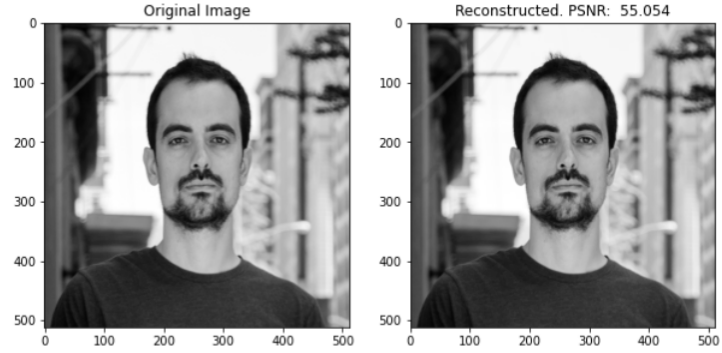

```
percent kept:0.05 Number of levels:4
```



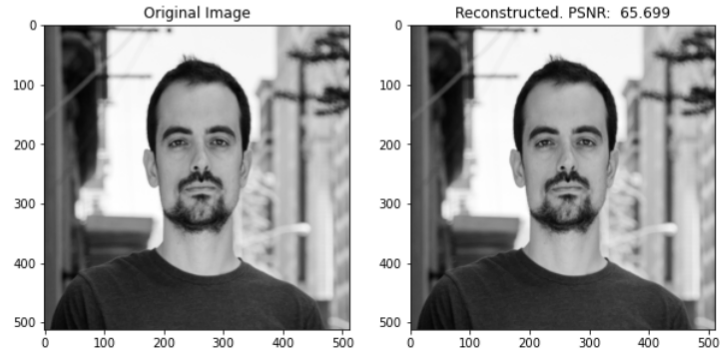
```
percent kept:0.1 Number of levels:4
```



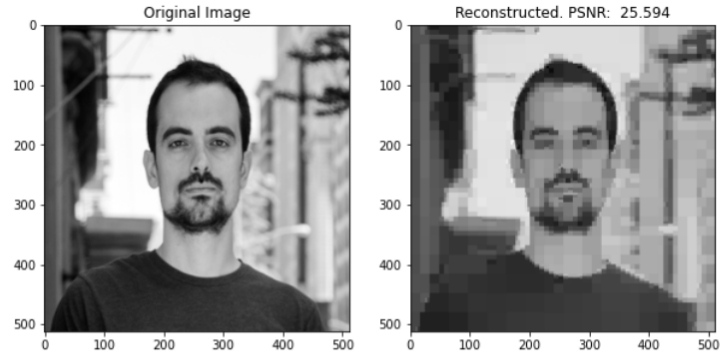

```
percent kept:0.25 Number of levels:4
```



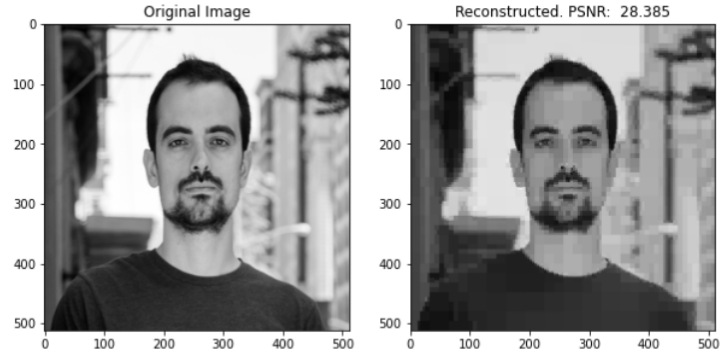

```
percent kept:0.5 Number of levels:4
```



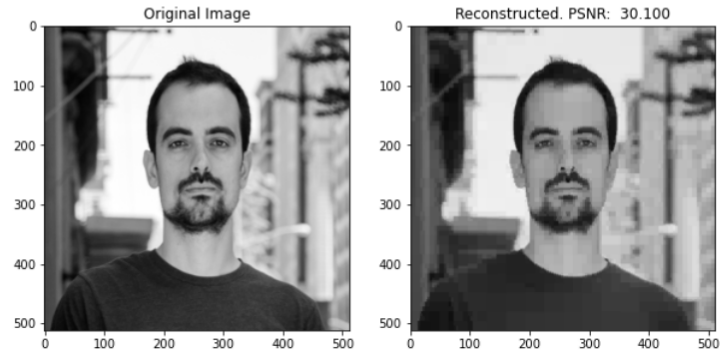
```
percent kept:0.75 Number of levels:4
```



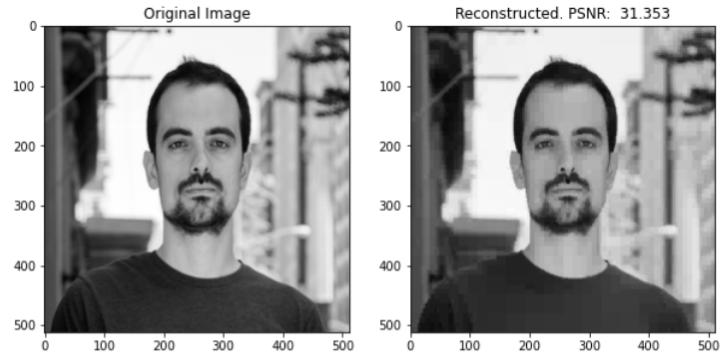

```
percent kept:0.005 Number of levels:5
```



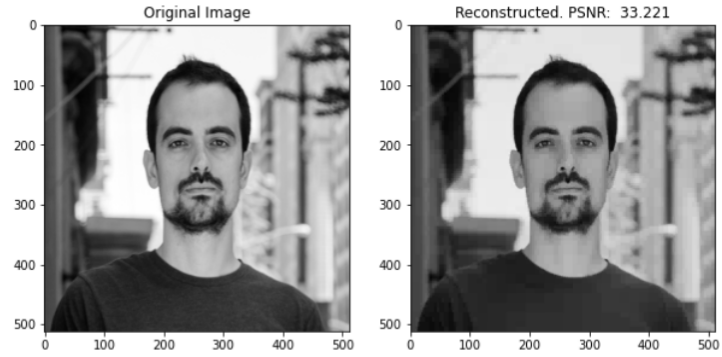

```
percent kept:0.01 Number of levels:5
```



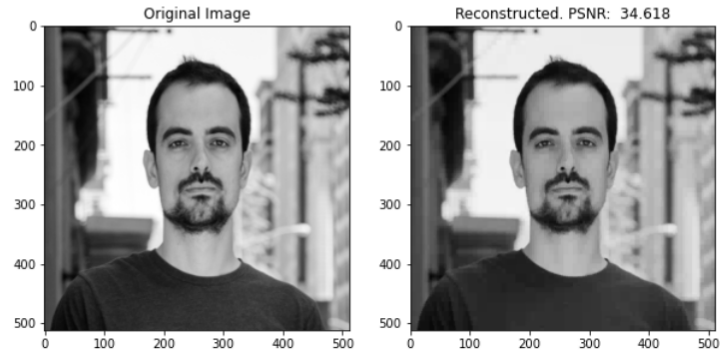
```
percent kept:0.015 Number of levels:5
```



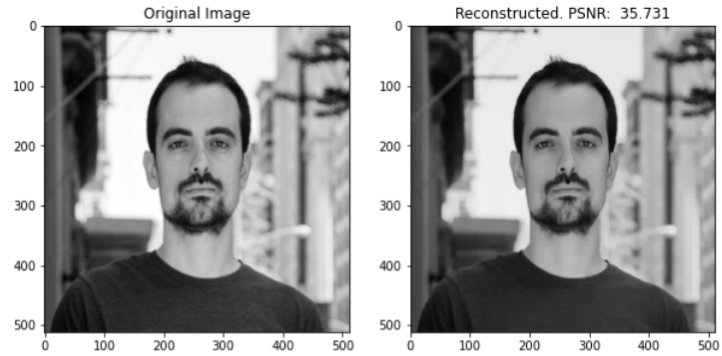

```
percent kept:0.02 Number of levels:5
```



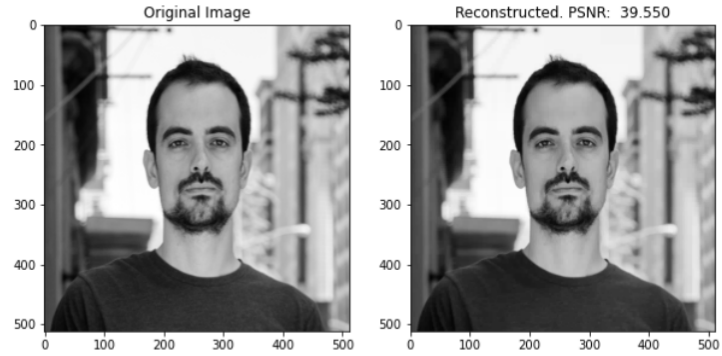

```
percent kept:0.03 Number of levels:5
```



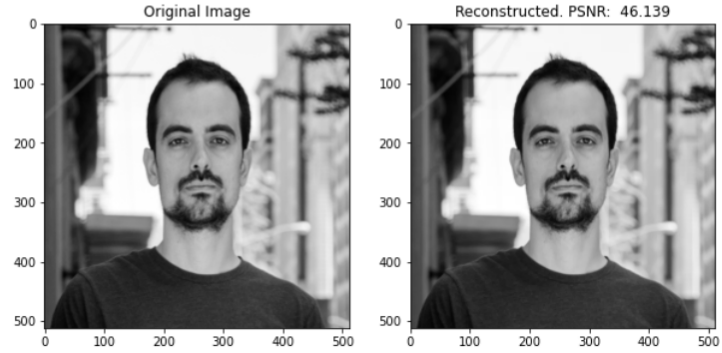
```
percent kept:0.04 Number of levels:5
```



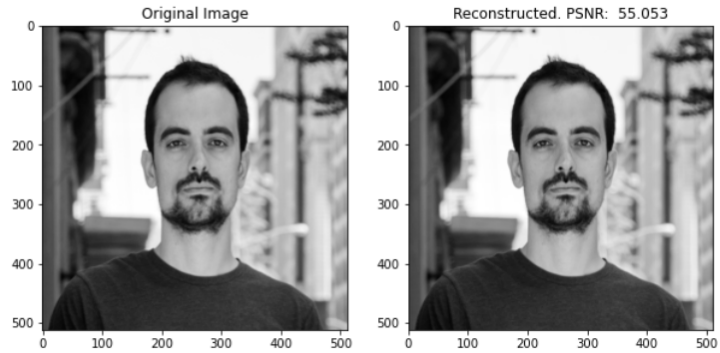

```
percent kept:0.05 Number of levels:5
```



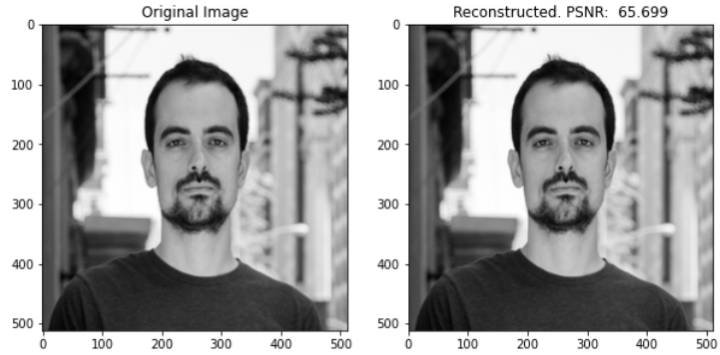

```
percent kept:0.1 Number of levels:5
```



```
percent kept:0.25 Number of levels:5
```




```
percent kept:0.5 Number of levels:5
```




```
percent kept:0.75 Number of levels:5
```

(b) Plot the PSNR of reconstructed image with respect to the clean image for different threshold values. Include different levels of decomposition as different curve in the same graph.

Screen Shot 2021-04-21 at 02.51.09.png

```
In [10]: len(PSNR)
Out[10]: 44
```

```
In [63]: fig, ax = plt.subplots(figsize=(10, 6))
for i in range(len(PSNR)):
    if i in range(11):
        ax.plot(pct, PSNR[i], '.r-',label="Level 2"if i == 0 else "")
    elif i in range(11,22):
        ax.plot(pct, PSNR[i], '.b-',label="Level 3"if i == 11 else "")
    elif i in range(22,33):
        ax.plot(pct, PSNR[i], '.g-',label="Level 4"if i == 22 else "")
    elif i in range(33,44):
        ax.plot(pct, PSNR[i], '.c-',label="Level 5"if i == 33 else "")

ax.legend()
```

```
Out[63]: <matplotlib.legend.Legend at 0x7fc38dcca4f0>
```