```python
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.image as im
import copy

pi = math.pi

img = im.imread('grey_noised_imgs/cameraman_original.png')
noisy_img = im.imread('grey_noised_imgs/cameraman_025.png')


def create_onb_DCT(N):
    d = np.zeros((N, N))
    for k in range(N):
        for n in range(N):
            d[n, k] = (math.cos(pi/N * (n+0.5) * k))
        # normalize each column
        d[:, k] = d[:, k]/np.linalg.norm(d[:, k])

    return d


def calc_mse(im1, im2):
    return round(np.square(im1-im2).mean(axis=None), 6)


def add_noise(image, var):
    return image + np.random.normal(0, math.sqrt(var), image.
shape)


def est_thresh(N):
    return math.sqrt(2 * math.log10(N))


def hard_threshold(image, t):
    image[abs(image) < t] = 0
    return image


def soft_threshold(image, t):
    image[abs(image) < t] = 0
    image[image > 0] = image[image > 0] - t
    image[image < 0] = image[image < 0] + t
    return image


def fDCT(image):
    onb1 = np.linalg.inv(create_onb_DCT(image.shape[0]))
```

```
    onb2 = np.linalg.inv(create_onb_DCT(image.shape[1]).T)
    return np.matmul(onb1, np.matmul(image, onb2))


def fDCT_block(image, onb):
    return np.matmul(onb, np.matmul(image, onb.T))


def iDCT(image):
    onb1 = create_onb_DCT(image.shape[0])
    onb2 = create_onb_DCT(image.shape[1]).T
    return np.matmul(np.matmul(onb1, image), onb2)


def iDCT_block(image, onb):
    return np.matmul(np.matmul(onb, image), onb.T)


def compress_image_DCT_hard(image, thresh):
    out = np.zeros(image.shape)
    # RGB
    if len(image.shape) == 3:
        for n in range(image.shape[2]):
            im_c = iDCT(image[:, :, n])
            image_median_value = np.mean(im_c)
            deviation = im_c - image_median_value
            abs_deviation = np.absolute(deviation)
            abs_deviation_median_value = np.median(
abs_deviation)
            dev = abs_deviation_median_value / 0.6745
            out[:, :, n] = hard_threshold(im_c, math.sqrt(dev
) * est_thresh(image.shape[0] * image.shape[1]))

    # B+W
    else:
        im_c = iDCT(image[:, :])
        image_median_value = np.median(im_c)
        deviation = im_c - image_median_value
        abs_deviation = np.absolute(deviation)
        abs_deviation_median_value = np.median(abs_deviation)
        dev = abs_deviation_median_value / 0.6745
        out[:, :] = hard_threshold(im_c, math.sqrt(dev) *
est_thresh(image.shape[0] * image.shape[1]))

    return out


def compress_image_DCT_soft(image, thresh):
    out = np.zeros(image.shape)
    # Rgb
```

```
    if len(image.shape) == 3:
        for n in range(image.shape[2]):
            im_c = iDCT(image[:, :, n])
            image_median_value = np.median(im_c)
            deviation = im_c - image_median_value
            abs_deviation = np.absolute(deviation)
            abs_deviation_median_value = np.median(
abs_deviation)
            dev = abs_deviation_median_value / 0.6745
            out[:, :, n] = soft_threshold(im_c, math.sqrt(dev
) * est_thresh(image.shape[0] * image.shape[1]))
    # B+W
    else:
        im_c = iDCT(image[:, :])
        image_median_value = np.median(im_c)
        deviation = im_c - image_median_value
        abs_deviation = np.absolute(deviation)
        abs_deviation_median_value = np.median(abs_deviation)
        dev = abs_deviation_median_value / 0.6745
        out[:, :] = soft_threshold(im_c, math.sqrt(dev) *
est_thresh(image.shape[0] * image.shape[1]))

    return out


def decompress_image_DCT(compressed_image):
    out = np.zeros(compressed_image.shape)
    # for RGB
    if len(compressed_image.shape) == 3:
        for n in range(compressed_image.shape[2]):
            out[:, :, n] = fDCT(compressed_image[:, :, n])
    # for B+W
    else:
        out[:, :] = fDCT(compressed_image[:, :])
    return out


def block_compress_soft(image, thresh, blk_size):
    out = np.zeros(image.shape)
    # RGB
    if len(image.shape) == 3:
        for i in range(0,  image.shape[0] - blk_size, blk_size
):
            for j in range(0, image.shape[1] - blk_size,
blk_size):
                out[i:i+blk_size, j:j+blk_size, :] =
compress_image_DCT_soft(image[i:i+blk_size, j:j+blk_size, :],
thresh)
    # B+W
    else:
```

```
        for i in range(0,  image.shape[0] - blk_size, blk_size
):
            for j in range(0, image.shape[1] - blk_size,
blk_size):
                out[i:i+blk_size, j:j+blk_size] =
compress_image_DCT_soft(image[i:i+blk_size, j:j+blk_size],
thresh)

    return out


def block_compress_hard(image, thresh, blk_size):
    out = np.zeros(image.shape)
    # RGB
    if len(image.shape) == 3:
        for i in range(0,  image.shape[0] - blk_size, blk_size
):
            for j in range(0,image.shape[1] - blk_size,
blk_size):
                out[i:i+blk_size, j:j+blk_size, :] =
compress_image_DCT_hard(image[i:i+blk_size, j:j+blk_size, :],
thresh)
    # B + W
    else:
        for i in range(0,  image.shape[0] - blk_size, blk_size
):
            for j in range(0,image.shape[1] - blk_size,
blk_size):
                out[i:i+blk_size, j:j+blk_size] =
compress_image_DCT_hard(image[i:i+blk_size, j:j+blk_size],
thresh)

    return out


def block_decompress(image, blk_size):
    out = np.zeros(image.shape)
    # RGB
    if len(image.shape) == 3:
        for i in range(0, image.shape[0] - blk_size, blk_size
):
            for j in range(0, image.shape[1] - blk_size,
blk_size):
                out[i:i + blk_size, j:j + blk_size, :] =
decompress_image_DCT(image[i:i + blk_size, j:j + blk_size, :])
    # B+W
    else:
        for i in range(0, image.shape[0] - blk_size, blk_size
):
            for j in range(0, image.shape[1] - blk_size,
```

```
blk_size):
                out[i:i + blk_size, j:j + blk_size] =
decompress_image_DCT(image[i:i + blk_size, j:j + blk_size])

    return out


def show_compression_hard(image, t):
    onb = create_onb_DCT(image.shape[0])
    x = compress_image_DCT_hard(image, t)
    y = decompress_image_DCT(x)
    plt.subplot(121)
    plt.title("Original image")
    plt.imshow(image, interpolation='nearest', cmap='gray')
    plt.subplot(122)
    plt.title("Image after compression with threshold = " +
str(t))
    plt.imshow(y, interpolation='nearest', cmap='gray')
    plt.show()


def do_compression_soft(image, t):
    x = compress_image_DCT_soft(image, t)
    return decompress_image_DCT(x)


def do_compression_hard(image, t):
    x = compress_image_DCT_hard(image, t)
    return decompress_image_DCT(x)


def do_block_compression_soft(image, t, blk_sz):
    x = block_compress_soft(image, t, blk_sz)
    return block_decompress(x, blk_sz)


def do_block_compression_hard(image, t, blk_sz):
    x = block_compress_hard(image, t, blk_sz)
    return block_decompress(x, blk_sz)


def do_block_compression_averaged_soft(image, t, blk_sz,
step_sz):
    tot_img = np.zeros(image.shape)
    for i in range(0, blk_sz, step_sz):
        im_cpy = copy.copy(image)
        c_img = do_block_compression_soft(image[i:, i:], t,
blk_sz)
        im_cpy[i:, i:] = c_img
        tot_img += im_cpy
```

```
    return tot_img/blk_sz


def do_block_compression_averaged_hard(image, t, blk_sz,
step_sz):
    tot_img = np.zeros(image.shape)
    for i in range(0, blk_sz, step_sz):
        im_cpy = copy.copy(image)
        c_img = do_block_compression_hard(image[i:, i:], t,
blk_sz)
        im_cpy[i:, i:] = c_img
        tot_img += im_cpy
    return tot_img/blk_sz


def show_all_compressions(image, blk_sz):

    var = 0.025
    fig = plt.figure()
    k = 1

    # add noise
    # noisy_image = add_noise(image, var)
    noisy_image = noisy_img
    # thresholding
    # t = math.sqrt(np.var(noisy_image)) * est_thresh(image.
shape[0] * image.shape[1])
    t = math.sqrt(var) * k * est_thresh(image.shape[0] * image
.shape[1])
    # bt = t
    # bt = math.sqrt(np.var(noisy_image)) * est_thresh(blk_sz
 ** 2)
    bt = math.sqrt(var) * k * est_thresh(blk_sz ** 2)
    # t = 3 * math.sqrt(var)
    # bt = 0.003 * math.sqrt(var)

    # soft
    # s = do_compression_soft(noisy_image, t)
    # # # hard
    # h = do_compression_hard(noisy_image, t)
    # # soft block
    # sb = do_block_compression_soft(noisy_image, bt, blk_sz)
    s = do_block_compression_soft(noisy_image, bt, blk_sz)
    sb = do_block_compression_averaged_soft(noisy_image, bt,
blk_sz, 1)
    hb = do_block_compression_averaged_hard(noisy_image, bt,
blk_sz, 1)
    # hb = do_block_compression_averaged_soft(noisy_image, bt
, blk_sz, 4)
    # hard block
```

```
    h = do_block_compression_hard(noisy_image, bt, blk_sz)
    # ax1 = fig.add_subplot(321)
    # plt.title("Original image")
    # plt.imshow(image, interpolation='nearest', cmap='gray')
    # ax1.xaxis.set_visible(False)
    # ax1.yaxis.set_visible(False)
    # ax2 = fig.add_subplot(322)
    # plt.title("Image after noise added (var = " + str(var
) + ")")
    # plt.imshow(noisy_image, interpolation='nearest', cmap='
gray')
    # ax2.xaxis.set_visible(False)
    # ax2.yaxis.set_visible(False)

    ax3 = fig.add_subplot(323)
    plt.title("Soft  block threshold \n(mse = " + str(calc_mse
(image, s)) + ")")
    plt.imshow(s, interpolation='nearest', cmap='gray')
    ax3.xaxis.set_visible(False)
    ax3.yaxis.set_visible(False)

    ax4 = fig.add_subplot(324)
    plt.title("Hard threshold \n(mse = " + str(calc_mse(image
, h)) + ")")
    plt.imshow(h, interpolation='nearest', cmap='gray')
    ax4.xaxis.set_visible(False)
    ax4.yaxis.set_visible(False)

    ax5 = fig.add_subplot(325)
    plt.title("averaged\n(mse = " + str(calc_mse(image, sb
)) + ")", fontsize=6)

    plt.imshow(sb, interpolation='nearest', cmap='gray')
    ax5.xaxis.set_visible(False)
    ax5.yaxis.set_visible(False)

    ax6 = fig.add_subplot(326)
    plt.title("(mse = " + str(calc_mse(image, hb)) + ")",
fontsize=6)
    plt.imshow(hb, interpolation='nearest', cmap='gray')
    ax6.xaxis.set_visible(False)
    ax6.yaxis.set_visible(False)

    plt.show()


show_all_compressions(img, 8)
```