1 Python correction

1.1 Pyramidal decomposition and reconstruction

1.1.1 Decomposition

The following function makes the decomposition of the Laplacian and Gaussian pyramids at the same time. The Laplacian pyramid can be reconstructed without any additional information. This is illustrated in Fig. 1.

```python
from skimage.transform import rescale, resize

def LaplacianPyramidDecomposition(Image, levels):
    """
    Laplacian / Gaussian Pyramid
    The last image of the laplacian pyramid allows a full reconstruction of the original image.
    Image: original image, float32
    levels: number of levels of decomposition

    returns: pyrL, pyrG: Laplacian and Gaussian pyramids, respectively, as a list of arrays
    """

    pyrL = [];
    pyrG = [];

    sigma = 3.;
    for l in range(levels):
        prevImage = Image.copy();
        g = ndimage.gaussian_filter(Image, sigma);
        print(g.dtype)

        Image = rescale(g, .5);
        primeImage= resize(Image, prevImage.shape);

        pyrL.append(prevImage - primeImage);
        pyrG.append(prevImage);

    pyrL.append(Image);
    pyrG.append(Image);
    return pyrL, pyrG;
```

Informations

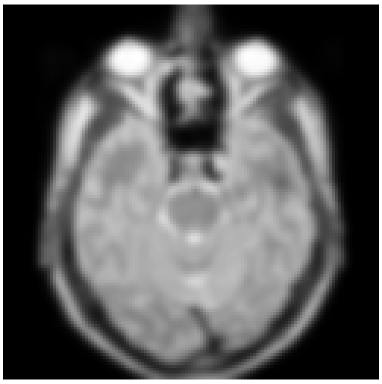
rescale and resize functions have the same goal but use a scale or a shape as a parameter.

Figure 1: Gaussian and Laplacian pyramids, for 3 levels of decomposition. The Laplacian pyramid in addition to the last level of the Gaussian pyramid is required to exactly reconstruct the original image.

(a) Original image.

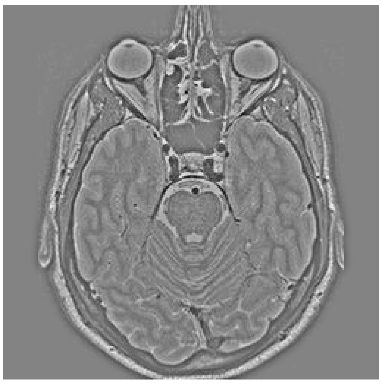
(b) Gaussian pyramid level 1.

(c) Level 2.

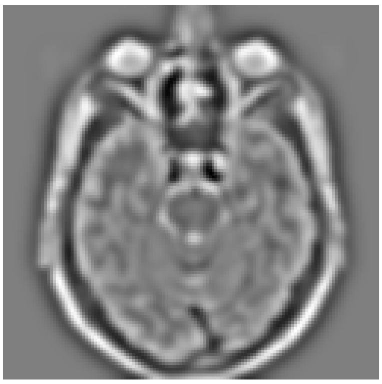


(d) Level 3.

(e) Laplacian pyramid level 1.



(f) Level 2.



(g) Level 3.

1.1.2 Reconstruction

The reconstruction is straighforward and exact because of the construction of the residue. The details can be filtered (removed for example), thus giving the following result Fig. 2.

```python
def LaplacianPyramidReconstruction(pyr, interp='bilinear'):
    """
    Reconstruction of the Laplacian pyramid, starting from the last image
    pyr: pyramid of images (list of arrays)
    interp: interpolation mode, for upsizing the image
    returns: Image, reconstructed image
    """

    Image = pyr[-1];
    for i in range(len(pyr)-2, -1, -1):
        Image = pyr[i] + resize(Image, pyr[i].shape);

    return Image;
```

Figure 2: Reconstruction of the Laplacian pyramid.

(a) Reconstruction of the pyramid without any detail.

(b) Reconstruction of the pyramid with all the details.

1.2 Scale-space decomposition and multiscale filtering

The pyramid of erosions and dilations is illustrated in Fig.3.

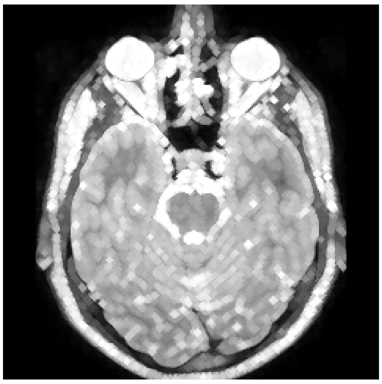
```python
def morphoMultiscale(I, levels):
    """
    Morphological multiscale decomposition
    I: original image, float32
    levels: number of levels, int

    returns: pyrD, pyrE: pyramid of Dilations/Erosions, respectively
    """
    pyrD=[];
    pyrE=[];
    for r in np.arange(1,levels):
        se = morphology.disk(r);
        pyrD.append( morphology.dilation(I, selem=se));
        pyrE.append( morphology.erosion(I, selem=se));
    return pyrD, pyrE;
```

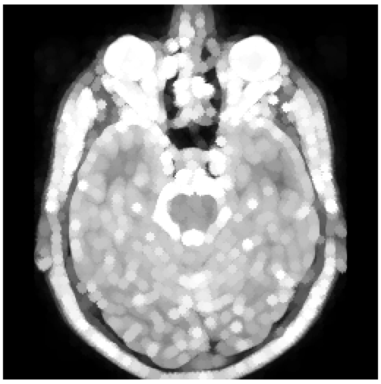
Figure 3: Morphological multiscale decomposition by dilation and erosion.

(a) Dilation scale 1.

(b) Dilation scale 2.

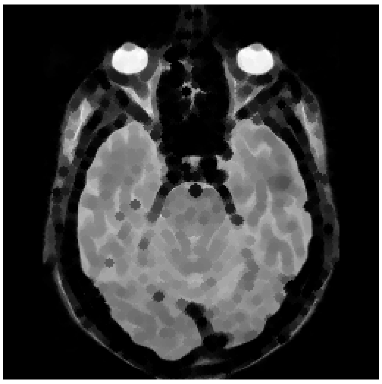


(c) Dilation scale 3.

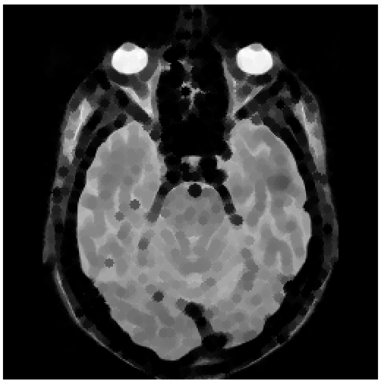


(d) Erosion scale 1.

(e) Erosion scale 2.



(f) Erosion scale 3.



1.3 Kramer and Bruckner multiscale decomposition

The results are illustrated in Fig.4.

Figure 4: Kramer and Bruckner multiscale decomposition, with $r = 5$.

(a) MK_B^1. (b) MK_B^2. (c) MK_B^3.

```python
def kb(I, r):
    """
    Elementary Kramer/Bruckner filter. Also called toggle filter.
    I: image
    r: radius of structuring element (disk), for max/min evaluation
    """
    se = morphology.disk(r);
    D=morphology.dilation(I, selem=se);
    E=morphology.erosion(I, selem=se);
    difbool = D-I < I-E;
    k = D*difbool + E * (~difbool);
    return k;
```

```python
def KBmultiscale(I, levels, r=1):
    """
    Kramer and Bruckner multiscale decomposition

    I: original image, float32
    pyrD: pyramid of Dilations
    pyrE: pyramid of Erosions

    returns: MKB: Kramer/Bruckner filters
    """
    MKB = [];
    MKB.append(I);
    for i in range(levels):
        MKB.append(kb(MKB[i-1], r));
    return MKB
```