lfepydocs

Documentation for lfepy

Table of Contents

Getting Started

Requirements

python>=3.0

numpy>=1.26.4

scipy>=1.13.0

scikit-image>=0.23.2

Installation

To install lfepy, use the following command:

```
pip install lfepy
```

lfepy.Descriptor.BPPC(image, **kwargs)

Description:
Compute Binary Phase Pattern Concatenation (BPPC) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing BPPC extraction:
  - **mode (str):** Mode for histogram computation. Options are:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **BPPC_hist (numpy.ndarray):** Histogram(s) of BPPC descriptors.
  - **imgDesc (list):** A list of dictionaries containing BPPC descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option (`'nh'` or `'h'`).

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = BPPC(image, mode='nh')

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
S. Shojaeilangari, W.-Y. Yau, J. Li, and E.-K. Teoh,
Feature extraction through binary pattern of phase congruency for facial expression recognition,
In Control Automation Robotics & Vision (ICARCV), 2012 12th International Conference on, IEEE, 2012, pp. 166-170.

lfepy.Descriptor.GDP(image, **kwargs)

Description:
Compute Gradient Directional Pattern (GDP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing GDP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **mask (str):** Mask type for gradient computation. Options:
    - `'sobel'` (default).
    - `'prewitt'`.
  - **t (float):** Threshold value for gradient angle difference. Default is 22.5.

Returns:

- **tuple:**
  A tuple containing:
  - **GDP_hist (numpy.ndarray):** Histogram(s) of GDP descriptors.
  - **imgDesc (numpy.ndarray):** GDP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` or `mask` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = GDP(image, mode='nh', mask='sobel', t=22.5)

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:

F. Ahmed,
Gradient directional pattern: a robust feature descriptor for facial expression recognition,
Electronics Letters, vol. 48, no. 23, pp. 1203-1204, 2012.

W. Chu,
Facial expression recognition based on local binary pattern and gradient directional pattern,
In Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things
(iThings/CPSCom), IEEE, 2013, pp. 1458-1462.

lfepy.Descriptor.GDP2(image, **kwargs)

Description:
Compute Gradient Direction Pattern (GDP2) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- **kwargs (dict):**
  Additional keyword arguments for customizing GDP2 extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **GDP2_hist (numpy.ndarray):** Histogram(s) of GDP2 descriptors.
  - **imgDesc (numpy.ndarray):** GDP2 descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option (`'nh'` or `'h'`).

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = GDP2(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
M.S. Islam,
Gender Classification using Gradient Direction Pattern,
Science International, vol. 25, 2013.

lfepy.Descriptor.GLTP(image, **kwargs)

Description:
Compute Gradient-based Local Ternary Pattern (GLTP) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing GLTP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **t (int):** Threshold value for ternary pattern computation. Default is 10.
  - **DGLP (int):** Flag to include Directional Gradient-based Local Pattern (DGLP).
    - If set to `1`, includes DGLP. Default is `0`.

Returns:

- **tuple:**
  A tuple containing:
  - **GLTP_hist (numpy.ndarray):** Histogram(s) of GLTP descriptors.
  - **imgDesc (list):** A list of dictionaries containing GLTP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` or `DGLP` in `kwargs` are not valid options.

Example:

```
python
Copy code
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = GLTP(image, mode='nh', t=10, DGLP=1)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:

M. Valstar, and M. Pantic,
Fully automatic facial action unit detection and temporal analysis,
In Computer Vision and Pattern Recognition Workshop, IEEE, 2006.

F. Ahmed, and E. Hossain,
Automated facial expression recognition using gradient-based ternary texture patterns,
Chinese Journal of Engineering, vol. 2013, 2013.

lfepy.Descriptor.IWBC(image, **kwargs)

Description:
Compute Improved Weber Contrast (IWBC) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing IWBC extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **scale (int):** Scale factor for IWBC computation. Default is 1.

Returns:

- **tuple:**
  A tuple containing:
  - **IWBC_hist (numpy.ndarray):** Histogram(s) of IWBC descriptors.
  - **imgDesc (list):** A list of dictionaries containing IWBC descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = IWBC(image, mode='nh', scale=1)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
B.-Q. Yang, T. Zhang, C.-C. Gu, K.-J. Wu, and X.-P. Guan,
A novel face recognition method based on IWLD and IWBC,
Multimedia Tools and Applications, vol. 75, pp. 6979, 2016.

lfepy.Descriptor.LAP(image, **kwargs)

Description:
Compute Local Arc Pattern (LAP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LAP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LAP_hist (numpy.ndarray):** Histogram(s) of LAP descriptors.
  - **imgDesc (list):** A list of dictionaries containing LAP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LAP(image, mode='nh')

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
M.S. Islam, and S. Auwatanamongkol,
Facial Expression Recognition using Local Arc Pattern,
Trends in Applied Sciences Research, vol. 9, pp. 113, 2014.

lfepy.Descriptor.LBP(image, **kwargs)

Description:
Compute Local Binary Patterns (LBP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LBP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **radius (int):** Radius for LBP computation. Default is 1.
  - **mappingType (str):** Type of mapping for LBP computation. Options:
    - `'full'` (default).
    - `'ri'`: Rotation invariant.
    - `'u2'`: Uniform LBP with 2 transitions.
    - `'riu2'`: Rotation invariant uniform LBP with 2 transitions.

Returns:

- **tuple:**
  A tuple containing:
  - **LBP_hist (numpy.ndarray):** Histogram(s) of LBP descriptors.
  - **imgDesc (numpy.ndarray):** LBP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` or `mappingType` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LBP(image, mode='nh', radius=1, mappingType='full')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
T. Ojala, M. Pietikainen, and T. Maenpaa,
Multi-resolution gray-scale and rotation invariant texture classification with local binary patterns,
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, pp. 971-987, 2002.

lfepy.Descriptor.LDiP(image, **kwargs)

Description:
Compute Local Directional Pattern (LDiP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LDiP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LDiP_hist (numpy.ndarray):** Histogram(s) of LDiP descriptors.
  - **imgDesc (numpy.ndarray):** LDiP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LDiP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
T. Jabid, M.H. Kabir, and O. Chae,
Local Directional Pattern (LDP) – A Robust Image Descriptor for Object Recognition,
Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on, IEEE, 2010, pp. 482-487.

lfepy.Descriptor.LDiPv(image, **kwargs)

Description:
Compute Local Directional Pattern Variance (LDiPv) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LDiPv extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LDiPv_hist (numpy.ndarray):** Histogram(s) of LDiPv descriptors.
  - **imgDesc (numpy.ndarray):** LDiPv descriptors themselves.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LDiPv(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
M.H. Kabir, T. Jabid, and O. Chae,
A Local Directional Pattern Variance (LDPv) Based Face Descriptor for Human Facial Expression Recognition,
Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on, IEEE, 2010, pp. 526-532.

lfepy.Descriptor.LDN(image, **kwargs)

Description:
Compute Local Directional Number (LDN) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LDN extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **mask (str):** Mask type for LDN computation. Options:
    - `'gaussian'`: Gaussian mask.
    - `'kirsch'`: Kirsch mask (default).
    - `'sobel'`: Sobel mask.
    - `'prewitt'`: Prewitt mask.
  - **msize (int):** Mask size if `'mask'` is set to `'kirsch'`. Default is 3.
  - **start (float):** Starting sigma value if `'mask'` is set to `'gaussian'`. Default is 0.5.

Returns:

- **tuple:**
  A tuple containing:
  - **LDN_hist (numpy.ndarray):** Histogram(s) of LDN descriptors.
  - **imgDesc (list):** A list of dictionaries containing LDN descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` or `mask` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LDN(image, mode='nh', mask='kirsch', msize=3, start=0.5)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
A.R. Rivera, J.R. Castillo, and O.O. Chae,

Local Directional Number Pattern for Face Analysis: Face and Expression Recognition,
IEEE Transactions on Image Processing, vol. 22, 2013, pp. 1740-1752.

lfepy.Descriptor.LDTP(image, **kwargs)

Description:
Compute Local Directional Texture Pattern (LDTP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LDTP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **epsi (int):** Threshold value for texture difference. Default is 15.

Returns:

- **tuple:**
  A tuple containing:
  - **LDTP_hist (numpy.ndarray):** Histogram(s) of LDTP descriptors.
  - **imgDesc (numpy.ndarray):** LDTP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LDTP(image, mode='nh', epsi=15)

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
A.R. Rivera, J.R. Castillo, and O. Chae,
Local Directional Texture Pattern Image Descriptor,
Pattern Recognition Letters, vol. 51, 2015, pp. 94-100.

lfepy.Descriptor.LFD(image, **kwargs)

Description:
Compute Local Frequency Descriptor (LFD) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LFD extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LFD_hist (numpy.ndarray):** Histogram(s) of LFD descriptors.
  - **imgDesc (list):** List of dictionaries containing LFD descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LFD(image, mode='nh')

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
Z. Lei, T. Ahonen, M. Pietikäinen, and S.Z. Li,
Local Frequency Descriptor for Low-Resolution Face Recognition,
Automatic Face & Gesture Recognition and Workshops (FG 2011), IEEE, 2011, pp. 161-166.

lfepy.Descriptor.LGBPHS(image, **kwargs)

Description:
Compute Local Gabor Binary Pattern Histogram Sequence (LGBPHS) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LGBPHS extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **uniformLBP (int):** Flag to use uniform LBP. Default is 1 (use uniform LBP).
  - **scaleNum (int):** Number of scales for Gabor filters. Default is 5.
  - **orienNum (int):** Number of orientations for Gabor filters. Default is 8.

Returns:

- **tuple:**
  A tuple containing:
  - **LGBPHS_hist (numpy.ndarray):** Histogram(s) of LGBPHS descriptors.
  - **imgDesc (list):** List of dictionaries containing LGBPHS descriptors for each scale and orientation.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LGBPHS(image, mode='nh', uniformLBP=1, scaleNum=5,
orienNum=8)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
W. Zhang, S. Shan, W. Gao, X. Chen, and H. Zhang,
Local Gabor Binary Pattern Histogram Sequence (LGBPHS): A Novel Non-Statistical Model for Face

Representation and Recognition,
ICCV 2005: Tenth IEEE International Conference on Computer Vision, IEEE, 2005, pp. 786-791.

lfepy.Descriptor.LGDiP(image, **kwargs)

Description:
Compute Local Gabor Directional Pattern (LGDiP) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LGDiP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LGDiP_hist (numpy.ndarray):** Histogram(s) of LGDiP descriptors.
  - **imgDesc (list):** List of dictionaries containing LGDiP descriptors for each scale.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LGDiP(image, mode='nh')

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
S.Z. Ishraque, A.H. Banna, and O. Chae,
Local Gabor Directional Pattern for Facial Expression Recognition,
ICCIT 2012: 15th International Conference on Computer and Information Technology, IEEE, 2012, pp. 164-167.

lfepy.Descriptor.LGIP(image, **kwargs)

Description:
Compute Local Gradient Increasing Pattern (LGIP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LGIP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LGIP_hist (numpy.ndarray):** Histogram(s) of LGIP descriptors.
  - **imgDesc (numpy.ndarray):** LGIP descriptors themselves.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LGIP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
Z. Lubing, and W. Han,
Local Gradient Increasing Pattern for Facial Expression Recognition,
Image Processing (ICIP), 2012 19th IEEE International Conference on, IEEE, 2012, pp. 2601-2604.

lfepy.Descriptor.LGP(image, **kwargs)

Description:
Compute Local Gradient Pattern (LGP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LGP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LGP_hist (numpy.ndarray):** Histogram(s) of LGP descriptors.
  - **imgDesc (numpy.ndarray):** LGP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LGP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
M.S. Islam,
Local Gradient Pattern-A Novel Feature Representation for Facial Expression Recognition,
Journal of AI and Data Mining 2, (2014), pp. 33-38.

lfepy.Descriptor.LGTrP(image, **kwargs)

Description:
Compute Local Gabor Transitional Pattern (LGTrP) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LGTrP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LGTrP_hist (numpy.ndarray):** Histogram(s) of LGTrP descriptors.
  - **imgDesc (numpy.ndarray):** LGTrP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LGTrP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
M.S. Islam,
Local Gradient Pattern-A Novel Feature Representation for Facial Expression Recognition,
Journal of AI and Data Mining 2, (2014), pp. 33-38.

lfepy.Descriptor.LMP(image, **kwargs)

Description:
Compute Local Monotonic Pattern (LMP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LMP extraction:
    - **mode (str):** Mode for histogram computation. Options:
        - `'nh'`: Normalized histogram (default).
        - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
    - **LMP_hist (numpy.ndarray):** Histogram(s) of LMP descriptors.
    - **imgDesc (numpy.ndarray):** LMP descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LMP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
T. Mohammad, and M.L. Ali,
Robust Facial Expression Recognition Based on Local Monotonic Pattern (LMP),
Computer and Information Technology (ICCIT), 2011 14th International Conference on, IEEE, 2011, pp. 572-576.

lfepy.Descriptor.LPQ(image, **kwargs)

Description:
Compute Local Phase Quantization (LPQ) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LPQ extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **windowSize (int):** Size of the sliding window for LPQ. Default is 5.

Returns:

- **tuple:**
  A tuple containing:
  - **LPQ_hist (numpy.ndarray):** Histogram of LPQ descriptors.
  - **imgDesc (numpy.ndarray):** LPQ descriptors.

Raises:

- **TypeError:** If the `image` is not a valid `numpy.ndarray`.
- **ValueError:** If the `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LPQ(image, mode='nh', windowSize=5)

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
V. Ojansivu, and J. Heikkilä,
Blur Insensitive Texture Classification Using Local Phase Quantization,
International Conference on Image and Signal Processing, Springer, 2008, pp. 236-243.

A. Dhall, A. Asthana, R. Goecke, and T. Gedeon,
Emotion Recognition Using PHOG and LPQ Features,
Automatic Face & Gesture Recognition and Workshops (FG 2011), IEEE, 2011, pp. 878-883.

lfepy.Descriptor.LTeP(image, **kwargs)

Description:
Compute Local Ternary Pattern (LTeP) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LTeP extraction:
  - **t (int):** Threshold value for ternary pattern computation. Default is 2.
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LTeP_hist (numpy.ndarray):** Histogram(s) of LTeP descriptors.
  - **imgDesc (list of dicts):** List of dictionaries containing LTeP descriptors.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LTeP(image, mode='nh', t=2)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
F. Bashar, A. Khan, F. Ahmed, and M.H. Kabir,
Robust Facial Expression Recognition Based on Median Ternary Pattern (MTP),
Electrical Information and Communication Technology (EICT), IEEE, 2014, pp. 1-5.

lfepy.Descriptor.LTrP(image, **kwargs)

Description:
Compute Local Transitional Pattern (LTrP) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing LTrP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LTrP_hist (numpy.ndarray):** Histogram(s) of LTrP descriptors.
  - **imgDesc (numpy.ndarray):** LTrP descriptors.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = LTrP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
T. Jabid, and O. Chae,
Local Transitional Pattern: A Robust Facial Image Descriptor for Automatic Facial Expression Recognition,
Proc. International Conference on Computer Convergence Technology, Seoul, Korea, 2011, pp. 333-44.

T. Jabid, and O. Chae,
Facial Expression Recognition Based on Local Transitional Pattern,
International Information Institute (Tokyo), Information, 15 (2012) 2007.

lfepy.Descriptor.MBC(image, **kwargs)

Description:
Compute Monogenic Binary Coding (MBC) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing MBC extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **mbcMode (str):** Mode for MBC computation. Options:
    - `'A'`: Amplitude (default).
    - `'O'`: Orientation.
    - `'P'`: Phase.

Returns:

- **tuple:**
  A tuple containing:
  - **MBC_hist (numpy.ndarray):** Histogram of MBC descriptors.
  - **imgDesc (list):** List of dictionaries containing MBC descriptors.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` or `mbcMode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = MBC(image, mode='nh', mbcMode='A')

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
M. Yang, L. Zhang, S.C.-K. Shiu, and D. Zhang,
Monogenic binary coding: An efficient local feature extraction approach to face recognition,
IEEE Transactions on Information Forensics and Security, 7 (2012) 1738-1751.

X.X. Xia, Z.L. Ying, and W.J. Chu,
Facial Expression Recognition Based on Monogenic Binary Coding,
Applied Mechanics and Materials, Trans Tech Publ, 2014, pp. 437-440.

lfepy.Descriptor.MBP(image, **kwargs)

Description:
Compute Median Binary Pattern (MBP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing MBP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **MBP_hist (numpy.ndarray):** Histogram(s) of MBP descriptors.
  - **imgDesc (numpy.ndarray):** MBP descriptors.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = MBP(image, mode='nh')

plt.imshow(imgDesc, cmap='gray')
plt.axis('off')
plt.show()
```

References:
F. Bashar, A. Khan, F. Ahmed, and M.H. Kabir,
Robust facial expression recognition based on median ternary pattern (MTP),
Electrical Information and Communication Technology (EICT), 2013 International Conference on, IEEE, 2014, pp. 1-5.

lfepy.Descriptor.MRELBP(image, **kwargs)

Description:
Compute the Median Robust Extended Local Binary Pattern (MRELBP) descriptors and histogram from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing MRELBP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **MRELBP_hist (numpy.ndarray):** Histogram(s) of MRELBP descriptors.
  - **imgDesc (list of dicts):** List of dictionaries where each dictionary contains the LBP descriptors for different radii. Each dictionary has:
    - **'fea':** Features extracted for the specific radius, including:
      - **'CImg':** Processed image data after median filtering and LBP transformation.
      - **'NILBPImage':** Histogram of the No-Interpolation LBP image.
      - **'RDLBPImage':** Histogram of the Refined Descriptors LBP image.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = MRELBP(image, mode='nh')

plt.imshow(imgDesc[0]['fea']['NILBPImage'], cmap='gray')
plt.axis('off')
plt.show()
plt.imshow(imgDesc[0]['fea']['RDLBPImage'], cmap='gray')
plt.axis('off')
plt.show()
```

References:

L. Liu, S. Lao, P.W. Fieguth, Y. Guo, X. Wang, and M. Pietikäinen,
Median robust extended local binary pattern for texture classification,
IEEE Transactions on Image Processing, vol. 25, no. 3, pp. 1368-1381, 2016.

lfepy.Descriptor.MTP(image, **kwargs)

Description:
Compute Median Ternary Pattern (MTP) descriptors and histograms from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing MTP extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **t (float):** Threshold value for MTP computation. Default is 10.

Returns:

- **tuple:**
  A tuple containing:
  - **MTP_hist (numpy.ndarray):** Histogram(s) of MTP descriptors.
  - **imgDesc (list of dicts):** List of dictionaries containing MTP descriptors for positive and negative thresholds.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = MTP(image, mode='nh', t=10)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
F. Bashar, A. Khan, F. Ahmed, and M.H. Kabir,
Robust facial expression recognition based on median ternary pattern (MTP),
Electrical Information and Communication Technology (EICT), 2013 International Conference on, IEEE, 2014, pp. 1-5.

lfepy.Descriptor.PHOG(image, **kwargs)

Description:
Compute Pyramid Histogram of Oriented Gradients (PHOG) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing PHOG extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **bin (int):** Number of bins for the histogram. Default is 8.
  - **angle (int):** Range of gradient angles in degrees. Default is 360.
  - **L (int):** Number of pyramid levels. Default is 2.

Returns:

- **tuple:**
  A tuple containing:
  - **PHOG_hist (numpy.ndarray):** Histogram of PHOG descriptors.
  - **imgDesc (list of dicts):** List of dictionaries containing PHOG descriptors for each pyramid level.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = PHOG(image, mode='nh', bin=8, angle=360, L=2)

plt.imshow(imgDesc[0]['fea'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
A. Bosch, A. Zisserman, and X. Munoz,
Representing shape with a spatial pyramid kernel,
Proceedings of the 6th ACM international conference on Image and video retrieval, ACM, 2007, pp. 401-408.

lfepy.Descriptor.WLD(image, **kwargs)

Description:
Compute Weber Local Descriptor (WLD) histograms and descriptors from an input image.

Parameters:

- **image (numpy.ndarray):**
  Input image, preferably in NumPy array format.
- ****kwargs (dict):**
  Additional keyword arguments for customizing WLD extraction:
  - **mode (str):** Mode for histogram computation. Options:
    - `'nh'`: Normalized histogram (default).
    - `'h'`: Histogram.
  - **T (int):** Number of bins for gradient orientation. Default is 8.
  - **N (int):** Number of bins for differential excitation. Default is 4.
  - **scaleTop (int):** Number of scales to consider for WLD computation. Default is 1.

Returns:

- **tuple:**
  A tuple containing:
  - **WLD_hist (numpy.ndarray):** Histogram of WLD descriptors.
  - **imgDesc (list of dicts):** List of dictionaries containing WLD descriptors for each scale.

Raises:

- **TypeError:** If `image` is not a valid `numpy.ndarray`.
- **ValueError:** If `mode` in `kwargs` is not a valid option.

Example:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

image = imread("Path")
histogram, imgDesc = WLD(image, mode='nh', T=8, N=4, scaleTop=1)

plt.imshow(imgDesc[0]['fea']['GO'], cmap='gray')
plt.axis('off')
plt.show()
plt.imshow(imgDesc[1]['fea']['DE'], cmap='gray')
plt.axis('off')
plt.show()
```

References:
S. Li, D. Gong, and Y. Yuan,
Face recognition using Weber local descriptors.,
Neurocomputing, 122 (2013) 272-283.

S. Liu, Y. Zhang, and K. Liu,
Facial expression recognition under partial occlusion based on Weber Local Descriptor histogram and decision fusion,
Control Conference (CCC), 2014 33rd Chinese, IEEE, 2014, pp. 4664-4668.

lfepy.Helper.bin_matrix(A, E, G, angle, bin)

Description:
Compute the bin matrix for a given angle map and gradient magnitude.

Parameters:

- **A (numpy.ndarray):**
  Angle map of the gradient directions.
- **E (numpy.ndarray):**
  Binary edge map where edges are marked.
- **G (numpy.ndarray):**
  Gradient magnitude map.
- **angle (float):**
  Total range of angles in degrees (e.g., 360 for a full circle).
- **bin (int):**
  Number of bins to divide the angle range into.

Returns:

- **tuple:**
  A tuple containing:
  - **bm (numpy.ndarray):** Bin matrix with assigned bins for each pixel based on the angle map.
  - **bv (numpy.ndarray):** Gradient magnitude values corresponding to the bin matrix.

Example:

```
import numpy as np
A = np.array([[0, 45], [90, 135]])
E = np.array([[1, 1], [1, 1]])
G = np.array([[1, 2], [3, 4]])
angle = 360
bin = 8
bm, bv = bin_matrix(A, E, G, angle, bin)
print(bm)
Output: [[1 2]
         [3 4]]
print(bv)
Output: [[1. 2.]
         [3. 4.]]
```

lfepy.Helper.cirInterpSingleRadius_ct(img, lbpPoints, lbpRadius)

Description:
Perform circular interpolation for a single radius in the LBP (Local Binary Pattern) computation. This function computes pixel values in a circular neighborhood around each pixel and interpolates them to approximate values that don't fall on exact pixel coordinates.

Parameters:

- **img (numpy.ndarray):**
  2D grayscale image.
- **lbpPoints (int):**
  Number of points used in the LBP pattern.
- **lbpRadius (int):**
  Radius of the circular neighborhood for computing LBP.

Returns:

- **tuple:**
  A tuple containing:
  - **blocks (numpy.ndarray):** Array of size (lbpPoints, imgNewH * imgNewW) containing the interpolated pixel values for each LBP point.
  - **dx (int):** Width of the output blocks (adjusted image width after accounting for the radius).
  - **dy (int):** Height of the output blocks (adjusted image height after accounting for the radius).

Raises:

- **ValueError:**
  If the input image is smaller than the required size of `(2*radius + 1) x (2*radius + 1).`

Example:

```
import numpy as np
from skimage import data

img = data.camera()
lbpPoints = 8
lbpRadius = 1
blocks, dx, dy = cirInterpSingleRadius_ct(img, lbpPoints, lbpRadius)
print(blocks.shape)
Output: (8, 9216)
```

lfepy.Helper.cirInterpSingleRadiusNew(img, lbpPoints, lbpRadius)

Description:
Extract circularly interpolated image blocks around a specified radius and number of points. This method computes interpolated pixel values in a circular neighborhood around each pixel for Local Binary Pattern (LBP) analysis.

Parameters:

- **img (numpy.ndarray):**
  The input 2D grayscale image.
- **lbpPoints (int):**
  The number of points used in the LBP pattern.
- **lbpRadius (int):**
  The radius of the circular neighborhood used for computing LBP.

Returns:

- **tuple:**
  A tuple containing:
  - **blocks (numpy.ndarray):** A 2D array where each row represents a circularly interpolated block, corresponding to each LBP point.
  - **dx (int):** The width of the output blocks (adjusted image width after considering the radius).
  - **dy (int):** The height of the output blocks (adjusted image height after considering the radius).

Raises:

- **ValueError:**
  If the input image is too small. The image should have dimensions of at least `(2*radius + 1) x (2*radius + 1)` to perform the interpolation.

Example:

```
import numpy as np
from skimage import data

img = data.camera()
lbpPoints = 8
lbpRadius = 1

blocks, dx, dy = cirInterpSingleRadiusNew(img, lbpPoints, lbpRadius)
print(blocks.shape)
```

lfepy.Helper.construct_Gabor_filters(num_of_orient, num_of_scales, size1, fmax=0.25, ni=1.4142135623730951, gamma=1.4142135623730951, separation=1.4142135623730951)

Description:
Constructs a bank of Gabor filters with specified orientations, scales, and filter sizes. The filters are designed to capture frequency and orientation information from images, commonly used for texture analysis and feature extraction.

Parameters:

- **num_of_orient (int):**
  The number of orientations (angles) for the Gabor filters.
- **num_of_scales (int):**
  The number of scales (frequencies) for the Gabor filters.
- **size1 (int or tuple):**
  The size of the filters. Can be an integer for square filters or a tuple of two integers for rectangular filters (height, width).
- **fmax (float, optional):**
  Maximum frequency. Default is `0.25`.
- **ni (float, optional):**
  Bandwidth parameter, controlling the filter's bandwidth. Default is `sqrt(2)`.
- **gamma (float, optional):**
  Aspect ratio of the Gabor filter, influencing the elongation of the filter. Default is `sqrt(2)`.
- **separation (float, optional):**
  Frequency separation factor between different scales. Default is `sqrt(2)`.

Returns:

- **dict:**
  A dictionary containing the Gabor filters in both spatial and frequency domains. The dictionary has the following keys:
  - **'spatial' (numpy.ndarray):** A 2D array where each element is a 2D array representing the spatial domain Gabor filter.
  - **'freq' (numpy.ndarray):** A 2D array where each element is a 2D array representing the frequency domain Gabor filter.
  - **'scales' (int):** The number of scales used in the filter bank.
  - **'orient' (int):** The number of orientations used in the filter bank.

Raises:

- **ValueError:**
  If `size1` is neither an integer nor a tuple of length 2 (for rectangular filters).

Example:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
num_of_orient = 8
num_of_scales = 5
filter_size = 31

gabor_filters = construct_Gabor_filters(num_of_orient, num_of_scales, filter_size)

fig, axes = plt.subplots(num_of_scales, num_of_orient, figsize=(20, 10))
for u in range(num_of_scales):
    for v in range(num_of_orient):
        ax = axes[u, v]
        ax.imshow(np.real(gabor_filters['spatial'][u, v]), cmap='gray')
        ax.axis('off')
plt.show()
```

lfepy.Helper.descriptor_LBP(image, radius, neighbors, mapping, mode)

Description:
Computes the Local Binary Pattern (LBP) of a grayscale image using the specified radius, number of neighbors, and optional mapping and mode settings. The function returns either the LBP histogram or the LBP code image.

Parameters:

- **image (numpy.ndarray):**
  The input image as a 2D NumPy array representing a grayscale image.
- **radius (int, optional):**
  The radius of the LBP, which determines the distance of the sampling points from the center pixel.
- **neighbors (int, optional):**
  The number of sampling points to be used in the LBP computation. More neighbors result in a finer granularity of the pattern.
- **mapping (dict or None, optional):**
  The mapping for LBP codes. Should contain the keys `'samples'` (number of sampling points) and `'table'` (lookup table for LBP codes). If `None`, no mapping is applied, and the standard LBP is computed.
- **mode (str, optional):**
  The mode for LBP calculation. Options include:
  - `'h'` or `'hist'`: Returns the LBP histogram.
  - `'nh'`: Returns the normalized LBP histogram (default).

Returns:

- **tuple:**
  - **result (numpy.ndarray):**
    The LBP histogram or LBP image, depending on the specified `mode`.
  - **codeImage (numpy.ndarray):**
    The LBP code image, where each pixel value represents the LBP code computed for that pixel.

Raises:

- **ValueError:**
  - If the number of input arguments is incorrect or if the provided `mapping` is incompatible with the number of `neighbors`.
  - If the input image is too small for the given `radius`.
  - If the dimensions of the `spoints` (sampling points) are invalid.

Example:

```
import numpy as np
from skimage import data
from descriptor_LBP import descriptor_LBP
```

```
image = data.camera()

result, codeImage = descriptor_LBP(image, radius=1, neighbors=8, mapping=None,
mode='nh')

print("LBP Histogram:", result)
print("LBP Code Image:", codeImage)
```

lfepy.Helper.descriptor_LDN(image, **kwargs)

Description:
This function computes a local descriptor for an input image using specified masks, such as Kirsch masks of varying sizes or a Gaussian gradient mask. The user can adjust the mask size using the `msize` parameter and control the standard deviation for the Gaussian mask using the `sigma` parameter.

Parameters:

- **image (numpy.ndarray):**
  The input image for which the local descriptor is computed. The image should be provided as a 2D NumPy array representing a grayscale image.
- ****kwargs:**
  Additional optional parameters for customizing the mask:
  - **mask (str, optional):**
    The type of mask to use for descriptor computation. Available options are `'kirsch'` (default) for Kirsch masks or `'gaussian'` for a Gaussian gradient mask.
  - **msize (int, optional):**
    The size of the Kirsch mask. Valid sizes are 3, 5, 7, 9, or 11. The default size is 3.
  - **sigma (float, optional):**
    The standard deviation of the Gaussian mask. This parameter is only relevant when the `'gaussian'` mask is selected. The default value is 0.5.

Returns:

- **numpy.ndarray:**
  The computed local descriptor matrix based on the selected mask and options.

Raises:

- **ValueError:**
  If an invalid mask type or size is provided.

Example:

```
import numpy as np
from skimage.data import camera
from descriptor_LDN import descriptor_LDN

image = camera()

descriptor = descriptor_LDN(image, mask='kirsch', msize=5)

descriptor_gaussian = descriptor_LDN(image, mask='gaussian', sigma=1.0)
```

lfepy.Helper.descriptor_LPQ(image, winSize=3, decorr=1, freqestim=1, mode='im')

Description:
This function computes the Local Phase Quantization (LPQ) descriptor, which is used to capture local texture information by analyzing the phase of the frequency components in a grayscale image. The LPQ descriptor can be computed using various frequency estimation methods and can be returned as either an image or a histogram based on the specified mode.

Parameters:

- **image (numpy.ndarray):**
  A 2D NumPy array representing the grayscale input image.
- **winSize (int, optional):**
  The size of the window used for LPQ calculation. This must be an odd number greater than or equal to 3. The default value is 3.
- **decorr (int, optional):**
  A flag indicating whether decorrelation should be applied. Use `0` for no decorrelation and `1` for decorrelation. The default value is 1.
- **freqestim (int, optional):**
  The method used for frequency estimation. Options are:
  - `1` for STFT with a uniform window (default),
  - `2` for STFT with a Gaussian window,
  - `3` for Gaussian derivative quadrature filter pair.
- **mode (str, optional):**
  Specifies the format of the output. Options are:
  - `'im'` for an image-like output (default),
  - `'nh'` for a normalized histogram,
  - `'h'` for a histogram.

Returns:

- **tuple:**
  A tuple containing:
  - **LPQdesc (numpy.ndarray):**
    The LPQ descriptor of the image. Depending on the `mode` parameter, this could be an image or a histogram.
  - **freqRespAll (numpy.ndarray):**
    The frequency responses for all filter pairs, represented as a 3D array.

Raises:

- **ValueError:**
  Raised if:
  - The `image` is not a 2D array.
  - The `winSize` is not an odd number or is less than 3.
  - The `decorr` parameter is not `0` or `1`.

- The `freqestim` parameter is not `1`, `2`, or `3`.
- The `mode` parameter is not one of `'nh'`, `'h'`, or `'im'`.

Example:

```
import numpy as np
from scipy import ndimage
from descriptor_LPQ import descriptor_LPQ

image = np.random.rand(100, 100)
desc, freq_resp = descriptor_LPQ(image, winSize=5, decorr=1, freqestim=2,
mode='h')

print(desc.shape)
Output: (256,)
print(freq_resp.shape)
Output: (100, 100, 8)
```

lfepy.Helper.descriptor_PHOG(image, bin=8, angle=360, L=2, roi=None)

Description:
This function computes the Pyramid Histogram of Oriented Gradients (PHOG) descriptor for a 2D image. The PHOG descriptor captures gradient information at multiple scales and orientations, providing a detailed description of shapes and textures within the image. The descriptor is computed across different levels of a pyramid and is useful for tasks such as object recognition and image analysis.

Parameters:

- **image (numpy.ndarray):**
  The input image, which can be either a 2D grayscale image or a 3D RGB image.
- **bin (int, optional):**
  The number of orientation bins used for the histogram computation. The default value is 8.
- **angle (int, optional):**
  The range of angles used for orientation. Valid options are `180` or `360` degrees. The default value is 360.
- **L (int, optional):**
  The number of pyramid levels for which the PHOG descriptor is computed. The default value is 2.
- **roi (list or None, optional):**
  Specifies the Region of Interest (ROI) within the image as a list of coordinates in the format `[y_min, y_max, x_min, x_max]`. If `None`, the entire image is used. The default value is `None`.

Returns:

- **tuple:**
  A tuple containing:
    - **p_hist (list):**
      A list of histograms representing the PHOG descriptor at each pyramid level.
    - **bh_roi (numpy.ndarray):**
      A gradient magnitude matrix for the specified Region of Interest (ROI).
    - **bv_roi (numpy.ndarray):**
      A gradient orientation matrix for the specified Region of Interest (ROI).

Raises:

- **ValueError:**
  Raised if:
    - The `image` is not a 2D array (grayscale) or a 3D array with a third dimension of size 3 (RGB).
    - The `angle` parameter is not `180` or `360`.
    - The `roi` parameter is not a list or `None`.

Example:

```
import numpy as np
```

```
from skimage import data
from descriptor_PHOG import descriptor_PHOG

image = data.camera()
p_hist, bh_roi, bv_roi = descriptor_PHOG(image, bin=8, angle=360, L=2)

print(len(p_hist))
Output: 2

print(bh_roi.shape)
Output: (480, 640)
print(bv_roi.shape)
Output: (480, 640)
```

lfepy.Helper.dgauss(x, sigma)

Description:
This function calculates the derivative of the Gaussian function, often referred to as the Gaussian derivative. The derivative is useful in various image processing applications, such as edge detection and feature extraction. The derivative is computed with respect to the standard deviation (`sigma`) of the Gaussian distribution, which controls the spread of the Gaussian curve.

Parameters:

- **x (float or numpy.ndarray):**
  The point or array of points at which to evaluate the derivative of the Gaussian function.
- **sigma (float):**
  The standard deviation of the Gaussian distribution, controlling the width of the Gaussian curve.

Returns:

- **float or numpy.ndarray:**
  The derivative of the Gaussian function at the specified point(s). The output type will match the input `x` type.

Example:

```
dgauss(0, 1)
-0.0

import numpy as np
dgauss(np.array([0, 1, 2]), 1)
Output: array([-0., -0.24197072, -0.10798193])
```

lfepy.Helper.filter_image_with_Gabor_bank(image, filter_bank, down_sampling_factor=64)

Description:
This function applies a bank of Gabor filters to an input grayscale image. It performs down-sampling and returns the concatenated features extracted from the filtered image. Gabor filters are widely used for texture analysis and feature extraction in image processing due to their ability to capture spatial frequency content.

Parameters:

- **image (np.ndarray):**
  The input image to be filtered, represented as a 2D numpy array (grayscale).
- **filter_bank (dict):**
  A dictionary containing the Gabor filter bank with the following keys:
  - **'spatial':** A list of 2D arrays representing the spatial domain Gabor filters.
  - **'freq':** A list of 2D arrays representing the frequency domain Gabor filters.
  - **'orient':** The number of orientations in the filter bank.
  - **'scales':** The number of scales in the filter bank.
- **down_sampling_factor (int, optional):**
  The factor by which to down-sample the filtered images. Default value is `64`.

Returns:

- **np.ndarray:**
  A 1D array containing the concatenated filtered features from the Gabor filter bank.

Raises:

- **ValueError:**
  Raised if input parameters are incorrect, dimensions do not match, or required fields are missing in the filter bank.

Example:

```
import numpy as np
from skimage.data import camera
image = camera()
filter_bank = construct_Gabor_filters(num_of_orient=8, num_of_scales=5, size1=31)
features = filter_image_with_Gabor_bank(image, filter_bank)
print(features.shape)
```

lfepy.Helper.gabor_filter(image, orienNum, scaleNum)

Description:
Applies a Gabor filter bank to a 2D grayscale image and organizes the resulting magnitude responses into a multidimensional array. The filter bank consists of various orientations and scales, providing detailed texture information.

Parameters:

- **image (numpy.ndarray):**
  The input image to be filtered. Must be a 2D numpy array (grayscale).
- **orienNum (int):**
  The number of orientation filters in the Gabor filter bank.
- **scaleNum (int):**
  The number of scale filters in the Gabor filter bank.

Returns:

- **numpy.ndarray:**
  A multidimensional array containing the Gabor magnitude responses. The shape of the array is `(height, width, orienNum, scaleNum)`.

Example:

```
import numpy as np
from skimage.data import camera
image = camera()
gabor_magnitudes = gabor_filter(image, orienNum=8, scaleNum=5)
print(gabor_magnitudes.shape)
(512, 512, 8, 5)
```

lfepy.Helper.gauss(x, sigma)

Description:
Calculates the value of the Gaussian (normal) distribution at specified points based on the given standard deviation. The Gaussian function is often used in various image processing and statistical applications.

Parameters:

- **x (float or numpy.ndarray):**
  The point or points at which to evaluate the Gaussian function. Can be a single float or a numpy array.
- **sigma (float):**
  The standard deviation of the Gaussian distribution. It determines the width of the Gaussian bell curve.

Returns:

- **float or numpy.ndarray:**
  The value(s) of the Gaussian function at the given point(s). The type matches the input `x` type.

Example:

```
gauss(0, 1)
Output: 0.3989422804014327
gauss(np.array([0, 1, 2]), 1)
Output: array([0.39894228, 0.24197072, 0.05399097])
```

lfepy.Helper.gauss_gradient(sigma)

Description:
Generates a set of 2-D Gaussian derivative kernels for computing gradients at multiple orientations. These kernels can be used for gradient-based edge detection and feature extraction in image processing.

Parameters:

- **sigma (float):**
  The standard deviation of the Gaussian distribution. It controls the width of the Gaussian and influences the gradient kernels' sensitivity.

Returns:

- **numpy.ndarray:**
  A 3D array where each 2D slice represents a Gaussian derivative kernel oriented at a specific angle. The shape of the array is (height, width, number_of_orientations).

Example:

```
import matplotlib.pyplot as plt
sigma = 1.0
kernels = gauss_gradient(sigma)
fig, axes = plt.subplots(1, 8, figsize=(20, 5))
for i in range(8):
     axes[i].imshow(kernels[:, :, i], cmap='gray')
     axes[i].set_title(f'{i*45} degrees')
     axes[i].axis('off')
plt.tight_layout()
plt.show()
```

lfepy.Helper.get_mapping(samples, mappingtype)

Description:
Generates a mapping table for Local Binary Patterns (LBP) codes based on the number of sampling points and the type of mapping. The mapping table can be used to convert raw LBP codes into more meaningful representations like uniform patterns or rotation-invariant codes.

Parameters:

- **samples (int):**
  The number of sampling points used in the LBP pattern. Determines the dimensionality of the LBP code.
- **mappingtype (str):**
  The type of LBP mapping. Options are:
  - `'u2'` (Uniform 2): Maps LBP codes to a smaller number of uniform patterns.
  - `'ri'` (Rotation Invariant): Maps LBP codes to rotation-invariant patterns.
  - `'riu2'` (Uniform and Rotation Invariant): Maps LBP codes to uniform and rotation-invariant patterns.

Returns:

- **dict:**
  A dictionary with the following keys:
  - `'table' (numpy.ndarray)`: The mapping table, which translates raw LBP codes into the specified format.
  - `'samples' (int)`: The number of sampling points used.
  - `'num' (int)`: The number of patterns in the resulting LBP code.

Raises:

- **ValueError:**
  If an unsupported mapping type is provided.

Example:

```
get_mapping(8, 'u2')
Output: {'table': array([...]), 'samples': 8, 'num': 59}
```

lfepy.Helper.get_mapping_info_ct(lbp_radius, lbp_points, lbp_method)

Description:
Retrieves or generates a mapping for circular (center-symmetric) Local Binary Patterns (LBP). The function supports various LBP methods and returns a mapping table used to convert raw LBP codes into specific patterns.

Parameters:

- **lbp_radius (int):**
  The radius of the LBP, defining the distance of the sampling points from the center pixel.
- **lbp_points (int):**
  The number of sampling points used in the LBP pattern.
- **lbp_method (str):**
  The method for LBP mapping. Options include:
  - `'LBPriu2'` (Uniform 2): Mapping for uniform LBP patterns.
  - `'MELBPVary'` (Modified Enhanced LBP Vary): A variant of LBP mapping with enhanced features.
  - `'AELBPVary'` (Advanced Enhanced LBP Vary): An advanced variant of LBP mapping with more flexibility.

Returns:

- **dict:**
  A dictionary containing the mapping information with the following keys:
  - `'table' (numpy.ndarray)`: The mapping table, which translates raw LBP codes into the specified format.
  - `'samples' (int)`: The number of sampling points used.
  - `'num' (int)`: The number of patterns in the resulting LBP code.

Example:

```
get_mapping_info_ct(1, 24, 'LBPriu2')
Output: {'table': array([...]), 'samples': 24, 'num': 26}
```

lfepy.Helper.get_mapping_mrelbp(samples, mappingtype)

Description:
Generates a mapping table for Modified Rotation and Uniform Local Binary Patterns (MRELBP) codes. The function supports a range of LBP mapping types, including uniform, rotation invariant, and various modified patterns.

Parameters:

- **samples (int):**
  The number of sampling points used in the LBP pattern.
- **mappingtype (str):**
  The type of LBP mapping. Supports various options including:
  - `'u2'` (Uniform 2)
  - `'LBPu2'` (Uniform LBP)
  - `'LBPVu2GMPD2'` (Uniform LBP with specific modifications)
  - `'ri'` (Rotation Invariant)
  - `'riu2'` (Rotation Invariant and Uniform)
  - `'MELBPVary'` (Modified Enhanced LBP with variations)
  - `'AELBPVary'` (Advanced Enhanced LBP with variations)
  - `'GELBPEight'` (Gabor Enhanced LBP with 8 directions)
  - `'CLBPEight'` (Circular LBP with 8 directions)
  - `'ELBPEight'` (Enhanced LBP with 8 directions)
  - `'LBPriu2Eight'` (Rotation Invariant and Uniform LBP with 8 directions)
  - `'MELBPEight'` (Modified Enhanced LBP with 8 directions)
  - Variants like `'MELBPEightSch1'` to `'MELBPEightSch11'` (Various schemes for 8-directional LBP)

Returns:

- **dict:**
  A dictionary containing the mapping information with the following keys:
  - `'table' (numpy.ndarray)`: The mapping table used to translate raw LBP codes into the specified format.
  - `'samples' (int)`: The number of sampling points used.
  - `'num' (int)`: The number of patterns in the resulting LBP code.

Example:

```
get_mapping_mrelbp(8, 'u2')
Output: {'table': array([...]), 'samples': 8, 'num': 59}
```

lfepy.Helper.low_pass_filter(size, cutoff, n)

Description:
Creates a low-pass Butterworth filter for use in the frequency domain. The filter attenuates frequencies higher than the specified cutoff frequency, providing smooth transitions between the passband and stopband.

Parameters:

- **size (tuple of int or int):**
  The size of the filter. If a single integer is provided, the filter will be square with that size.
- **cutoff (float):**
  The cutoff frequency for the filter, specifying the boundary between the passband and stopband. Must be between 0 and 0.5, where 0 represents DC and 0.5 represents the Nyquist frequency.
- **n (int):**
  The order of the Butterworth filter, which determines the smoothness of the filter's transition. Must be an integer greater than or equal to 1.

Returns:

- **np.ndarray:**
  The low-pass Butterworth filter in the frequency domain. The filter will have the specified size and will be centered around the origin.

Raises:

- **ValueError:**
  If `cutoff` is not in the range [0, 0.5], or if `n` is not an integer greater than or equal to 1.

Example:

```
filter_size = (256, 256)
cutoff_frequency = 0.1
order = 2
lp_filter = low_pass_filter(filter_size, cutoff_frequency, order)
print(lp_filter.shape)
Output: (256, 256)
```

lfepy.Helper.lxp_phase(image, radius=1, neighbors=8, mapping=None, mode='h')

Description:
Compute the Local X-Y Pattern (LXP) descriptor for a 2D grayscale image based on local phase information. The LXP descriptor captures texture and structural features by analyzing the local phase information of the image.

Parameters:

- **image (numpy.ndarray):**
  2D grayscale image. The input image should be a 2D numpy array.
- **radius (int, optional):**
  Radius of the circular neighborhood for computing the pattern. Default is 1.
- **neighbors (int, optional):**
  Number of directions or neighbors to consider. Default is 8.
- **mapping (numpy.ndarray or None, optional):**
  Coordinates of neighbors relative to each pixel. If `None`, uses a default circular pattern. If a single digit, computes neighbors in a circular pattern based on the digit. Default is `None`.
- **mode (str, optional):**
  Mode for output. Options are:
  - `'h'` or `'hist'`: Returns the histogram of the LXP descriptor.
  - `'nh'`: Returns the normalized histogram of the LXP descriptor. Default is `'h'`.

Returns:

- **numpy.ndarray:**
  The LXP descriptor. The output will be either a histogram or an image, depending on the `mode` parameter.

Raises:

- **ValueError:**
  If the input image is too small for the specified radius or if the coordinates in `mapping` are invalid.

Example:

```
import numpy as np
from skimage import data
image = data.camera()
lxp_desc = lxp_phase(image, radius=1, neighbors=8, mode='nh')
print(lxp_desc.shape)
Output: (256,)
```

lfepy.Helper.monofilt(im, nscale, minWaveLength, mult, sigmaOnf, orientWrap=0, thetaPhase=1)

Description:
Apply a multiscale directional filter bank to a 2D grayscale image using Log-Gabor filters. This function performs filtering at multiple scales and orientations to capture detailed texture and structural information.

Parameters:

- **im (numpy.ndarray):**
  2D grayscale image. The input image should be a 2D numpy array.
- **nscale (int):**
  Number of scales in the filter bank.
- **minWaveLength (float):**
  Minimum wavelength of the filters.
- **mult (float):**
  Scaling factor between consecutive scales.
- **sigmaOnf (float):**
  Bandwidth of the Log-Gabor filter.
- **orientWrap (int, optional):**
  If 1, wrap orientations to the range [0, π]. Default is 0 (no wrapping).
- **thetaPhase (int, optional):**
  If 1, compute phase angles (theta and psi). Default is 1.

Returns:

- **tuple:** A tuple containing:
  - **f (list of numpy.ndarray):** Filter responses in the spatial domain.
  - **h1f (list of numpy.ndarray):** x-direction filter responses in the spatial domain.
  - **h2f (list of numpy.ndarray):** y-direction filter responses in the spatial domain.
  - **A (list of numpy.ndarray):** Amplitude of the filter responses.
  - **theta (list of numpy.ndarray, optional):** Phase angles of the filter responses, if `thetaPhase` is 1.
  - **psi (list of numpy.ndarray, optional):** Orientation angles of the filter responses, if `thetaPhase` is 1.

Raises:

- **ValueError:**
  If the input image is not a 2D array.

Example:

```
import numpy as np
from scipy import ndimage
image = np.random.rand(100, 100)
nscale = 4
```

```
minWaveLength = 6
mult = 2.0
sigmaOnf = 0.55
f, h1f, h2f, A, theta, psi = monofilt(image, nscale, minWaveLength, mult,
sigmaOnf)
print(len(f))
Output: 4
print(f[0].shape)
Output: (100, 100)
```

lfepy.Helper.NewRDLBP_Image(img, imgPre, lbpRadius, lbpRadiusPre, lbpPoints, mapping=None, mode='h')

Description:
Compute the Radial Difference Local Binary Pattern (RDLBP) between two 2D grayscale images. This function calculates the difference in Local Binary Patterns (LBP) between a current image and a reference image, which can be useful for texture comparison and image analysis.

Parameters:

- **img (numpy.ndarray):**
  2D grayscale image for which the LBP is computed.
- **imgPre (numpy.ndarray):**
  2D grayscale image used for comparison.
- **lbpRadius (int):**
  Radius of the circular neighborhood for computing the LBP in the current image.
- **lbpRadiusPre (int):**
  Radius of the circular neighborhood for computing the LBP in the comparison image.
- **lbpPoints (int):**
  Number of points used in the LBP pattern.
- **mapping (dict or None, optional):**
  Mapping dictionary for converting the LBP result to a different bin scheme. If provided, must contain:
    - `'num'`: Number of bins in the new scheme.
    - `'table'`: Mapping from old bin to new bin.
      Default is `None`, meaning no mapping is applied.
- **mode (str, optional):**
  Specifies the output format. Options are:
    - `'h'` or `'hist'`: Returns histogram of the RDLBP.
    - `'nh'`: Returns normalized histogram of the RDLBP. Default is `'h'`.

Returns:

- **numpy.ndarray:**
  RDLBP descriptor, either as a histogram or an image depending on the `mode` parameter.

Raises:

- **ValueError:**
  If `mapping` is provided but does not contain the required keys `'num'` or `'table'`.

Example:

```
import numpy as np
from skimage import data
img = data.camera()
imgPre = data.coins()
lbpRadius = 1
```

```
lbpRadiusPre = 1
lbpPoints = 8
hist = NewRDLBP_Image(img, imgPre, lbpRadius, lbpRadiusPre, lbpPoints, mode='nh')
print(hist.shape)
Output: (256,)
```

lfepy.Helper.NILBP_Image_ct(img, lbpPoints, mapping, mode, lbpRadius)

Description:
Compute the Neighborhood Binary Pattern (NILBP) descriptor for a 2D grayscale image using circular interpolation. The NILBP captures local texture features by analyzing the binary patterns in a circular neighborhood around each pixel.

Parameters:

- **img (numpy.ndarray):**
  2D grayscale image for which the NILBP descriptor is computed.
- **lbpPoints (int):**
  Number of points used in the LBP pattern.
- **mapping (dict or None):**
  Optional dictionary for converting LBP codes to a different bin scheme. If provided, it must contain:
  - `'num'`: Number of bins in the new scheme.
  - `'table'`: Mapping from old bin to new bin.
    If `None`, no mapping is applied.
- **mode (str):**
  Specifies the output format. Options are:
  - `'h'` or `'hist'`: Returns the histogram of the NILBP.
  - `'nh'`: Returns the normalized histogram of the NILBP.
- **lbpRadius (int):**
  Radius of the circular neighborhood for computing the LBP.

Returns:

- **numpy.ndarray:**
  NILBP descriptor, which can be either a histogram or an image depending on the `mode` parameter.

Example:

```
import numpy as np
from skimage import data
img = data.camera()
lbpPoints = 8
lbpRadius = 1
mapping = {'num': 256, 'table': np.arange(256)}
descriptor = NILBP_Image_ct(img, lbpPoints, mapping, mode='nh',
lbpRadius=lbpRadius)
print(descriptor.shape)
Output: (256,)
```

lfepy.Helper.phase_cong3(image, nscale=4, norient=6, minWaveLength=3, mult=2.1, sigmaOnf=0.55, dThetaOnSigma=1.5, k=2.0, cutOff=0.5, g=10)

Description:
Computes the phase congruency of an image using a multiscale, multi-orientation approach. Phase congruency measures the image's local contrast based on the phase information of its frequency components, making it useful for edge detection and texture analysis.

Parameters:

- **image (numpy.ndarray):**
  Input grayscale image as a 2D numpy array.
- **nscale (int, optional):**
  Number of scales to be used in the analysis. Default is 4.
- **norient (int, optional):**
  Number of orientations to be used in the analysis. Default is 6.
- **minWaveLength (float, optional):**
  Minimum wavelength of the log-Gabor filters. Default is 3.
- **mult (float, optional):**
  Scaling factor for the wavelength of the log-Gabor filters. Default is 2.1.
- **sigmaOnf (float, optional):**
  Standard deviation of the Gaussian function used in the log-Gabor filter. Default is 0.55.
- **dThetaOnSigma (float, optional):**
  Angular spread of the Gaussian function relative to the orientation. Default is 1.5.
- **k (float, optional):**
  Constant to adjust the threshold for noise. Default is 2.0.
- **cutOff (float, optional):**
  Cut-off parameter for the weighting function. Default is 0.5.
- **g (float, optional):**
  Gain parameter for the weighting function. Default is 10.

Returns:

- **tuple:**
  A tuple containing:
  - **M (numpy.ndarray):** The measure of local phase congruency.
  - **m (numpy.ndarray):** The measure of local phase concavity.
  - **ori (numpy.ndarray):** Orientation of the phase congruency.
  - **featType (numpy.ndarray):** Complex representation of phase congruency.
  - **PC (list of numpy.ndarray):** List of phase congruency maps for each orientation.
  - **EO (list of numpy.ndarray):** List of complex responses for each scale and orientation.

Raises:

- **ValueError:**
  If the input image is not a 2D numpy array.

Example:

```
import numpy as np
from skimage import data
image = data.camera()
M, m, ori, featType, PC, EO = phase_cong3(image)
```

lfepy.Helper.phogDescriptor_hist(bh, bv, L, bin)

Description:
Computes the histogram of the Pyramid Histogram of Oriented Gradients (PHOG) descriptor. This function calculates histograms for multiple pyramid levels, each representing different spatial resolutions, using the provided bin matrix `bh` and gradient magnitude matrix `bv`.

Parameters:

- **bh (numpy.ndarray):**
  Bin matrix of the image, where each pixel is assigned a bin index.
- **bv (numpy.ndarray):**
  Gradient magnitude matrix corresponding to the bin matrix `bh`.
- **L (int):**
  Number of pyramid levels.
- **bin (int):**
  Number of bins for the histogram.

Returns:

- **numpy.ndarray:**
  Normalized histogram of the PHOG descriptor.

Example:

```
import numpy as np
bh = np.array([[1, 2], [2, 1]])
bv = np.array([[1, 2], [2, 1]])
L = 2
bin = 4
phog_hist = phogDescriptor_hist(bh, bv, L, bin)
print(phog_hist)
Output: [0.1 0.2 0.2 0.1 0.1 0.1 0.1 0.1]
```

lfepy.Helper.RDLBP_Image_SmallestRadiusOnly(imgCenSmooth, img, lbpRadius, lbpPoints, mapping, mode)

Description:
Computes the Radial Difference Local Binary Pattern (RDLBP) for an image with a focus on the smallest radius. This function calculates the RDLBP descriptor by comparing radial differences between the original image and a smoothed version of the image, using a circular neighborhood defined by the specified radius and number of points.

Parameters:

- **imgCenSmooth (numpy.ndarray):**
  Smoothed image from which the radial difference is computed.
- **img (numpy.ndarray):**
  Original image used for extracting circularly interpolated blocks.
- **lbpRadius (int):**
  Radius of the circular neighborhood for Local Binary Pattern (LBP).
- **lbpPoints (int):**
  Number of points used in the LBP pattern.
- **mapping (dict or None):**
  Optional mapping dictionary for converting LBP results to a different bin scheme. Must contain:
  - `'num'` (int): Number of bins.
  - `'table'` (numpy.ndarray): Mapping from old bin to new bin. If None, no mapping is applied.
- **mode (str):**
  Output mode. Options are:
  - `'h'` or `'hist'`: Histogram of the RDLBP.
  - `'nh'`: Normalized histogram.

Returns:

- **numpy.ndarray:**
  RDLBP descriptor, either as a histogram or image depending on the `mode` parameter.

Example:

```
import numpy as np
from skimage import data
img = data.camera()
imgCenSmooth = data.coins()
lbpRadius = 1
lbpPoints = 8
mapping = {'num': 256, 'table': np.arange(256)}
hist = RDLBP_Image_SmallestRadiusOnly(imgCenSmooth, img, lbpRadius, lbpPoints,
mapping, mode='nh')
print(hist.shape)
Output: (256,)
```

lfepy.Helper.roundn(x, n)

Description:
Rounds a number or array of numbers to a specified number of decimal places. If `n` is negative, the function rounds to the left of the decimal point. If `n` is zero, it rounds to the nearest integer.

Parameters:

- **x (float or array-like):**
  The number or array of numbers to be rounded.
- **n (int):**
  The number of decimal places to round to. If `n` is negative, `x` is rounded to the left of the decimal point. If `n` is zero, `x` is rounded to the nearest integer.

Returns:

- **float or array-like:**
  The rounded number or array of numbers.

Examples:

```
roundn(123.456, 2)
Output: 123.46

roundn(123.456, -1)
Output: 120.0

roundn(123.456, 0)
Output: 123.0
```

lfepy.Helper.view_as_windows(arr, window_shape, step=1)

Description:
Generates a view of the input array where each element is a sliding window of a specified shape. The windows are extracted with a given step size.

Parameters:

- **arr (numpy.ndarray):**
  The input array from which windows will be extracted.
- **window_shape (tuple):**
  Shape of the sliding window.
- **step (int or tuple, optional):**
  Step size of the sliding window. If an integer is provided, it is applied uniformly across all dimensions. Default is 1.

Returns:

- **numpy.ndarray:**
  A view of the array with sliding windows.

Raises:

- **ValueError:**
  If any dimension of the window shape is larger than the corresponding dimension of the array.

Examples:

```
import numpy as np
view_as_windows(np.array([1, 2, 3, 4]), window_shape=(2,), step=1)
Output: array([[1, 2], [2, 3], [3, 4]])
```