lfepydocs

Documentation for lfepy

Table of Contents

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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **BPPC_hist**: Histogram(s) of BPPC descriptors.
- **imgDesc**: List of dictionaries containing BPPC descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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, 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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.mask** (`str`): Mask type for gradient computation. Options: `'sobel'`, `'prewitt'`. Default: `'sobel'`.
  - **kwargs.t** (`float`): Threshold value for gradient angle difference. Default: 22.5.

Returns:

- **GDP_hist**: Histogram(s) of GDP descriptors.
- **imgDesc**: GDP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 48 (2012) 1203-1204.

W. Chu, Facial expression recognition based on local binary pattern and gradient directional pattern, Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **GDP2_hist**: Histogram(s) of GDP2 descriptors.
- **imgDesc**: GDP2 descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
    - **kwargs.t** (`int`): Threshold value for ternary pattern computation. Default: 10.
    - **kwargs.DGLP** (`int`): Flag to include Directional Gradient-based Local Pattern. If set to 1, includes DGLP. Default: 0.

Returns:

- **GLTP_hist**: Histogram(s) of GLTP descriptors.
- **imgDesc**: List of dictionaries containing GLTP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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, Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on, IEEE, 2006, pp. 149-149.

F. Ahmed, and E. Hossain, Automated facial expression recognition using gradient-based ternary texture patterns. Chinese Journal of Engineering 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
    - **kwargs.scale** (`int`): Scale factor for IWBC computation. Default: 1.

Returns:

- **IWBC_hist**: Histogram(s) of IWBC descriptors.
- **imgDesc**: List of dictionaries containing IWBC descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 75 (2016) 6979.

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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LAP_hist**: Histogram(s) of LAP descriptors.
- **imgDesc**: List of dictionaries containing LAP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 9 (2014) 113.

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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.radius** (`int`): Radius for LBP computation. Default: 1.
  - **kwargs.mappingType** (`str`): Type of mapping for LBP computation. Options: `'full'`, `'ri'`, `'u2'`, `'riu2'`. Default: `'full'`.

Returns:

- **LBP_hist**: Histogram(s) of LBP descriptors.
- **imgDesc**: LBP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 24 (2002) 971-987.

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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LDiP_hist**: Histogram(s) of LDiP descriptors.
- **imgDesc**: LDiP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LDiPv_hist**: Histogram(s) of LDiPv descriptors.
- **imgDesc**: LDiPv descriptors themselves.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 Difference 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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.mask** (`str`): Mask type for LDN computation. Options: `'gaussian'`, `'kirsch'`, `'sobel'`, or `'prewitt'`. Default: `'kirsch'`.
  - **kwargs.msize** (`int`): Mask size if `'mask'` is set to `'kirsch'`. Default: 3.
  - **kwargs.start** (`float`): Starting sigma value if `'mask'` is set to `'gaussian'`. Default: 0.5.

Returns:

- **LDN_hist**: Histogram(s) of LDN descriptors.
- **imgDesc**: List of dictionaries containing LDN descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 22 (2013) 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
    - **kwargs.epsi** (`int`): Threshold value for texture difference. Default: 15.

Returns:

- **LDTP_hist**: Histogram(s) of LDTP descriptors.
- **imgDesc**: LDTP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 51 (2015) 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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LFD_hist**: Histogram(s) of LFD descriptors.
- **imgDesc**: List of dictionaries containing LFD descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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), 2011 IEEE International Conference on, 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
    - **kwargs.uniformLBP** (`int`): Flag to use uniform LBP. Default: 1 (use uniform LBP).
    - **kwargs.scaleNum** (`int`): Number of scales for Gabor filters. Default: 5.
    - **kwargs.orienNum** (`int`): Number of orientations for Gabor filters. Default: 8.

Returns:

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

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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, Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, 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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LGDiP_hist**: Histogram(s) of LGDiP descriptors.
- **imgDesc**: List of dictionaries containing LGDiP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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, Computer and Information Technology (ICCIT), 2012 15th International Conference on, 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LGIP_hist**: Histogram(s) of LGIP descriptors.
- **imgDesc**: LGIP descriptors themselves.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LGP_hist**: Histogram(s) of LGP descriptors.
- **imgDesc**: LGP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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) 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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LGTrP_hist**: Histogram(s) of LGTrP descriptors.
- **imgDesc**: LGTrP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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) 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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LMP_hist**: Histogram(s) of LMP descriptors.
- **imgDesc**: LMP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.windowSize** (`int`): Size of the sliding window for LPQ. Default: 5.

Returns:

- **LPQ_hist**: Histogram of LPQ descriptors.
- **imgDesc**: LPQ descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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), 2011 IEEE International Conference on, 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.
  - **kwargs.t** (`int`): Threshold value for ternary pattern computation. Default: 2.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LTeP_hist**: Histogram(s) of LTeP descriptors.
- **imgDesc**: List of dictionaries containing LTeP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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), 2013 International Conference on, 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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **LTrP_hist**: Histogram(s) of LTrP descriptors.
- **imgDesc**: LTrP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.mbcMode** (`str`): Mode for MBC computation. Options: `'A'` (amplitude), `'O'` (orientation), `'P'` (phase). Default: `'A'`.

Returns:

- **MBC_hist**: Histogram of MBC descriptors.
- **imgDesc**: List of dictionaries containing MBC descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **MBP_hist**: Histogram(s) of MBP descriptors.
- **imgDesc**: MBP descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.

Returns:

- **MRELBP_hist**: Histogram(s) of MRELBP descriptors.
- **imgDesc**: A 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.

Return Type:

- Tuple of (`numpy.ndarray`, `list of dicts`)

Example:

```
python
Copy code
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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 25 (2016) 1368-1381.

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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.t** (`float`): Threshold value for MTP computation. Default: 10.

Returns:

- **MTP_hist**: Histogram(s) of MTP descriptors.
- **imgDesc**: List of dictionaries containing MTP descriptors for positive and negative thresholds.

Return Type:

- Tuple of (`numpy.ndarray`, `list of dicts`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
    - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
    - **kwargs.bin** (`int`): Number of bins for the histogram. Default: 8.
    - **kwargs.angle** (`int`): Range of gradient angles in degrees. Default: 360.
    - **kwargs.L** (`int`): Number of pyramid levels. Default: 2.

Returns:

- **PHOG_hist**: Histogram of PHOG descriptors.
- **imgDesc**: List of dictionaries containing PHOG descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list of dicts`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.
  - **kwargs.mode** (`str`): Mode for histogram computation. Options: `'nh'` (normalized histogram) or `'h'` (histogram). Default: `'nh'`.
  - **kwargs.T** (`int`): Number of bins for gradient orientation. Default: 8.
  - **kwargs.N** (`int`): Number of bins for differential excitation. Default: 4.
  - **kwargs.scaleTop** (`int`): Number of scales to consider for WLD computation. Default: 1.

Returns:

- **WLD_hist**: Histogram(s) of WLD descriptors.
- **imgDesc**: List of dictionaries containing WLD descriptors.

Return Type:

- Tuple of (`numpy.ndarray`, `list of dicts`)

Example:

```
>>> from PIL import Image
>>> import matplotlib.pyplot as plt
>>> image = Image.open(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.NILBP_Image_ct(img, lbpPoints, mapping, mode, lbpRadius)

Description: Compute the Neighborhood Binary Pattern (NILBP) descriptor for an image using circular interpolation.

Parameters:

- **img** (`numpy.ndarray`): 2D grayscale image.
- **lbpPoints** (`int`): Number of points used in the LBP pattern.
- **mapping** (`dict` or `None`): A dictionary containing `'num'` (number of bins) and `'table'` (mapping table). If `None`, no mapping is applied.
- **mode** (`str`): Mode for output. Options: `'h'` or `'hist'` for histogram of the NILBP, `'nh'` for normalized histogram.
- **lbpRadius** (`int`): Radius of the circular neighborhood for computing LBP.

Returns:

- **NILBP descriptor**: Either as a histogram or image depending on the mode parameter.

Return Type:

- `numpy.ndarray`

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 shape for normalized histogram
(256,)
```

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

Description: Compute the Radial Difference Local Binary Pattern (RDLBP) between two images.

Parameters:

- **img** (`numpy.ndarray`): 2D grayscale image.
- **imgPre** (`numpy.ndarray`): 2D grayscale image for comparison.
- **lbpRadius** (`int`): Radius of the circular neighborhood for the current image.
- **lbpRadiusPre** (`int`): Radius of the circular neighborhood for 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) and `'table'` (mapping from old bin to new bin).
- **mode** (`str`, optional): Mode for output. Options: `'h'` or `'hist'` for histogram of the RDLBP, `'nh'` for normalized histogram. Default: `'h'`.

Returns:

- **RDLBP descriptor**: Either as a histogram or image depending on the mode parameter.

Return Type:

- `numpy.ndarray`

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 shape for normalized histogram
(256,)
```

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

Description: Compute the Radial Difference Local Binary Pattern (RDLBP) for an image with a focus on the smallest radius.

Parameters:

- **imgCenSmooth** (`numpy.ndarray`): Smoothed image from which the radial difference is computed.
- **img** (`numpy.ndarray`): Original image for extracting circularly interpolated blocks.
- **lbpRadius** (`int`): Radius of the circular neighborhood for LBP.
- **lbpPoints** (`int`): Number of points used in the LBP pattern.
- **mapping** (`dict` or `None`): Optional mapping dictionary for converting LBP result to a different bin scheme. Must contain `'num'` (number of bins) and `'table'` (mapping from old bin to new bin).
- **mode** (`str`): Output mode. Options: `'h'` or `'hist'` for histogram of the RDLBP, `'nh'` for normalized histogram.

Returns:

- **RDLBP descriptor**: Either as a histogram or image depending on the mode parameter.

Return Type:

- `numpy.ndarray`

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 shape for normalized histogram
(256,)
```

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 full circle).
- **bin** (`int`): Number of bins to divide the angle range into.

Returns:

- **bm**: Bin matrix with assigned bins for each pixel.
- **bv**: Gradient magnitude values corresponding to the bin matrix.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

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)
[[1 2]
 [3 4]]
>>> print(bv)
[[1. 2.]
 [3. 4.]]
```

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

Description: Extract circularly interpolated image blocks around a specified radius and number of points.

Parameters:

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

Returns:

- **blocks**: A 2D array where each row represents a circularly interpolated block.
- **dx**: The width of the output blocks.
- **dy**: The height of the output blocks.

Return Type:

- Tuple of (`numpy.ndarray`, `int`, `int`)

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)  # Shape of the blocks array
```

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

Description: Perform circular interpolation for a single radius in the LBP (Local Binary Pattern) computation.

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:

- **blocks**: Array of size `(lbpPoints, imgNewH * imgNewW)` containing the interpolated pixel values.
- **dx**: Width of the output blocks.
- **dy**: Height of the output blocks.

Return Type:

- Tuple of (`numpy.ndarray`, `int`, `int`)

Example:

```
>>> import numpy as np
>>> from skimage import data
>>> img = data.camera()  # Example grayscale image
>>> lbpPoints = 8
>>> lbpRadius = 1
>>> blocks, dx, dy = cirInterpSingleRadius_ct(img, lbpPoints, lbpRadius)
>>> print(blocks.shape)  # Shape of the blocks array
(8, 9216)
```

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.

Parameters:

- **num_of_orient** (`int`): Number of orientations.
- **num_of_scales** (`int`): Number of scales.
- **size1** (`int` or `tuple`): Size of the filters. Can be an integer for square filters or a tuple for rectangular filters.
- **fmax** (`float`, optional): Maximum frequency. Default is 0.25.
- **ni** (`float`, optional): Bandwidth parameter. Default is `sqrt(2)`.
- **gamma** (`float`, optional): Aspect ratio. Default is `sqrt(2)`.
- **separation** (`float`, optional): Frequency separation factor. Default is `sqrt(2)`.

Returns:

- **A dictionary** containing the spatial and frequency representations of the Gabor filters.

Return Type:

- `dict`

Example:

```
>>> import matplotlib.pyplot as plt
>>> 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: Compute the Local Binary Pattern (LBP) of an image with various options for radius, neighbors, mapping, and mode.

Parameters:

- **image** (`numpy.ndarray`): The input grayscale image.
- **radius** (`int`): The radius of the LBP.
- **neighbors** (`int`): The number of sampling points in the LBP.
- **mapping** (`dict` or `None`): The mapping information for converting LBP codes. If `None`, no mapping is applied.
- **mode** (`str`): The mode for LBP calculation. Options are `'h'` (histogram), `'hist'` (histogram), or `'nh'` (normalized histogram).

Returns:

- **result**: The LBP histogram or LBP image, depending on the mode.
- **codeImage**: The LBP code image.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Raises:

- **ValueError**: If the number of input arguments is incorrect or other validation fails.

Example:

```
>>> import numpy as np
>>> image = np.random.rand(100, 100)
>>> result, codeImage = descriptor_LBP(image, 1, 8, None, 'nh')
>>> print(result.shape)  # Shape of LBP histogram or image
```

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

Description: Compute the Local Descriptor using Kirsch or Gaussian masks. This function computes a local descriptor for an input image using different masks based on the specified options.

Parameters:

- **image** (`numpy.ndarray`): The input image for which the descriptor is computed.
- **kwargs** (`dict`, optional): Additional optional parameters to customize the mask.
    - **'mask'** (`str`): Type of mask to use. Options are `'kirsch'` (default) or `'gaussian'`.
    - **'msize'** (`int`): Size of the Kirsch mask. Options are 3, 5, 7, 9, or 11 (default is 3).
    - **'sigma'** (`float`): Standard deviation for the Gaussian mask (default is 0.5).

Returns:

- **numpy.ndarray**: The local descriptor matrix computed using the specified mask.

Return Type:

- `numpy.ndarray`

Raises:

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

Example:

```
>>> import numpy as np
>>> from skimage.data import camera
>>> image = camera()
>>> descriptor = descriptor_LDN(image, mask='kirsch', msize=5)
>>> print(descriptor.shape)
```

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

Description: Compute the Local Phase Quantization (LPQ) descriptor for a given grayscale image.

Parameters:

- **image** (`numpy.ndarray`): Grayscale input image.
- **winSize** (`int`): Size of the window used for LPQ calculation. Must be an odd number ≥ 3. Default is 3.
- **decorr** (`int`): Flag to apply decorrelation. 0 for no decorrelation, 1 for decorrelation. Default is 1.
- **freqestim** (`int`): Frequency estimation method. Options are:
    - 1: STFT uniform window
    - 2: STFT Gaussian window
    - 3: Gaussian derivative quadrature filter pair
- **mode** (`str`): Specifies the output format. Options are `'im'` for image-like output, `'nh'` for normalized histogram, `'h'` for histogram. Default is `'im'`.

Returns:

- **LPQdesc**: The LPQ descriptor of the image. Depending on the mode, it could be an image or a histogram.
- **freqRespAll**: The frequency responses for all filter pairs.

Return Type:

- Tuple of (`numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> import numpy as np
>>> from scipy import ndimage
>>> image = np.random.rand(100, 100)
>>> desc, freq_resp = descriptor_LPQ(image, winSize=5, decorr=1, freqestim=2,
mode='h')
>>> print(desc.shape)  # Histogram shape
>>> print(freq_resp.shape)  # Frequency responses shape
```

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

Description: Compute the Pyramid Histogram of Oriented Gradients (PHOG) descriptor for a 2D image.

Parameters:

- **image** (`numpy.ndarray`): Input image, which can be grayscale or RGB.
- **bin** (`int`, optional): Number of orientation bins for the histogram. Default is 8.
- **angle** (`int`, optional): Angle range for orientation. Can be 180 or 360 degrees. Default is 360.
- **L** (`int`, optional): Number of pyramid levels. Default is 2.
- **roi** (`list` or `None`, optional): Region of Interest (ROI) as `[y_min, y_max, x_min, x_max]`. If `None`, the entire image is used.

Returns:

- **p_hist**: List of histograms for each pyramid level.
- **bh_roi**: Gradient magnitude matrix for the ROI.
- **bv_roi**: Gradient orientation matrix for the ROI.

Return Type:

- Tuple of (`list of numpy.ndarray`, `numpy.ndarray`, `numpy.ndarray`)

Example:

```
>>> import numpy as np
>>> from skimage import data
>>> image = data.camera()  # Example grayscale image
>>> p_hist, bh_roi, bv_roi = descriptor_PHOG(image, bin=8, angle=360, L=2)
>>> print(len(p_hist))  # Number of levels in the PHOG descriptor
>>> print(bh_roi.shape)  # Shape of the gradient magnitude matrix for the ROI
>>> print(bv_roi.shape)  # Shape of the gradient orientation matrix for the
ROI
```

lfepy.Helper.dgauss(x, sigma)

Description: Compute the derivative of the Gaussian (normal) distribution with respect to `x`.

Parameters:

- **x** (`float` or `numpy.ndarray`): The point or points at which to evaluate the derivative.
- **sigma** (`float`): The standard deviation of the Gaussian distribution.

Returns:

- The derivative of the Gaussian function at the given point(s).

Return Type:

- `float` or `numpy.ndarray`

Example:

```
>>> dgauss(0, 1)
-0.0
>>> dgauss(np.array([0, 1, 2]), 1)
array([-0., -0.24197072, -0.10798193])
```

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

Description: Apply a Gabor filter bank to an image and return the filtered features.

Parameters:

- **image** (`numpy.ndarray`): Input image to be filtered. Should be a 2D NumPy array.
- **filter_bank** (`dict`): Dictionary containing Gabor filter bank with `'spatial'` and `'freq'` keys.
- **down_sampling_factor** (`int`, optional): Factor for down-sampling the filtered images. Default is 64.

Returns:

- Concatenated filtered features from the Gabor filter bank.

Return Type:

- `numpy.ndarray`

Raises:

- **ValueError**: If the inputs are not as expected or dimensions do not match.

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: Apply a Gabor filter bank to an image and organize the results into a multidimensional array.

Parameters:

- **image** (`numpy.ndarray`): Input image to be filtered. Should be a 2D NumPy array.
- **orienNum** (`int`): Number of orientation filters in the Gabor filter bank.
- **scaleNum** (`int`): Number of scale filters in the Gabor filter bank.

Returns:

- Multidimensional array containing the Gabor magnitude responses. Shape is `(height, width, orienNum, scaleNum)`.

Return Type:

- `numpy.ndarray`

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)  # Shape of Gabor magnitudes
```

lfepy.Helper.gauss(x, sigma)

Description: Calculate the value of the Gaussian (normal) distribution at a given point.

Parameters:

- **x** (`float` or `numpy.ndarray`): The point or points at which to evaluate the Gaussian function.
- **sigma** (`float`): The standard deviation of the Gaussian distribution.

Returns:

- The value(s) of the Gaussian function at the given point(s).

Return Type:

- `float` or `numpy.ndarray`

Example:

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

lfepy.Helper.gauss_gradient(sigma)

Description: Generate a set of 2-D Gaussian derivative kernels for gradient computation at multiple orientations.

Parameters:

- **sigma** (`float`): The standard deviation of the Gaussian distribution.

Returns:

- A 3D array where each 2D slice represents a Gaussian derivative kernel at a specific orientation.

Return Type:

- `numpy.ndarray`

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: Generate a mapping table for Local Binary Patterns (LBP) codes.

Parameters:

- **samples** (`int`): The number of sampling points in the LBP.
- **mappingtype** (`str`): The type of LBP mapping. Options are `'u2'`, `'ri'`, `'riu2'`.

Returns:

- **'table'**: The mapping table.
- **'samples'**: The number of sampling points.
- **'num'**: The number of patterns in the resulting LBP code.

Return Type:

- `dict`

Raises:

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

Example:

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

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

Description: Retrieve or generate a mapping for circular (center-symmetric) LBP.

Parameters:

- **lbp_radius** (`int`): The radius of the LBP.
- **lbp_points** (`int`): The number of sampling points in the LBP.
- **lbp_method** (`str`): The method for LBP mapping.

Returns:

- **'table'**: The mapping table.
- **'samples'**: The number of sampling points.
- **'num'**: The number of patterns in the resulting LBP code.

Return Type:

- `dict`

Example:

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

lfepy.Helper.get_mapping_mrelbp(samples, mappingtype)

Description: Generate a mapping table for Modified Rotation and Uniform Local Binary Patterns (MRELBP) codes.

Parameters:

- **samples** (`int`): The number of sampling points in the LBP.
- **mappingtype** (`str`): The type of LBP mapping. Supports various uniform, rotation invariant, and modified patterns.

Returns:

- **'table'** (`numpy.ndarray`): The mapping table.
- **'samples'** (`int`): The number of sampling points.
- **'num'** (`int`): The number of patterns in the resulting LBP code.

Return Type:

- `dict`

Raises:

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

Example:

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

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

Description: Create a low-pass Butterworth filter.

Parameters:

- **size** (`tuple of 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. Must be between 0 and 0.5.
- **n** (`int`): The order of the Butterworth filter. Must be an integer greater than or equal to 1.

Returns:

- **np.ndarray**: The low-pass Butterworth filter in the frequency domain.

Return Type:

- `numpy.ndarray`

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)
(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.

Parameters:

- **image** (`numpy.ndarray`): 2D grayscale image.
- **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. Default is `None`.
- **mode** (`str`, optional): Mode for output. `'h'` or `'hist'` for histogram of the LXP, `'nh'` for normalized histogram. Default is `'h'`.

Returns:

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

Return Type:

- `numpy.ndarray`

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)
(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.

Parameters:

- **im** (`numpy.ndarray`): 2D grayscale image.
- **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:

- **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 angle of the filter responses, if `thetaPhase` is 1.
- **psi** (`list of numpy.ndarray`, optional): Orientation angle of the filter responses, if `thetaPhase` is 1.

Return Type:

- `tuple`

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))
4
>>> print(f[0].shape)  # Shape of the response for the first scale
(100, 100)
```

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: Compute the phase congruency of an image using a multiscale, multi-orientation approach. Phase congruency is a measure of the image's local contrast, based on the phase information of its frequency components. This method is used 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 weighting function. Default is 0.5.
- **g** (`float`, optional): Gain parameter for the weighting function. Default is 10.

Returns:

- **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`): Feature type (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.

Return Type:

- `tuple of numpy.ndarray and list`

Raises:

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

Example:

```
>>> import numpy as np
>>> image = np.random.rand(256, 256)
>>> M, m, ori, featType, PC, EO = phase_cong3(image)
>>> print(M.shape)
(256, 256)
```

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

Description: Compute the histogram of the Pyramid Histogram of Oriented Gradients (PHOG) descriptor.

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.
- **L** (`int`): Number of pyramid levels.
- **bin** (`int`): Number of bins for the histogram.

Returns:

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

Return Type:

- `numpy.ndarray`

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)
[0.1 0.2 0.2 0.1 0.1 0.1 0.1 0.1]
```

lfepy.Helper.roundn(x, n)

Description: Round a number to a specified number of decimal places.

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:

- The rounded number or array of numbers.

Return Type:

- `float` or `array-like`

Example:

```
>>> roundn(123.456, 2)
123.46
>>> roundn(123.456, -1)
120.0
>>> roundn(123.456, 0)
123.0
```

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

Description: Create a view of an array with sliding windows.

Parameters:

- **arr** (`numpy.ndarray`): The input array.
- **window_shape** (`tuple`): Shape of the sliding window.
- **step** (`int` or `tuple`): Step size of the sliding window.

Returns:

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

Return Type:

- `numpy.ndarray`

Raises:

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

Example:

```
>>> view_as_windows(np.array([1, 2, 3, 4]), window_shape=(2,), step=1)
array([[1, 2],
       [2, 3],
       [3, 4]])
```