# pyradiomics Documentation

*Release v3.0.1.post15+g2791e23*

pyradiomics community

Apr 03, 2022

# Contents

This is an open-source python package for the extraction of Radiomics features from medical imaging. With this package we aim to establish a reference standard for Radiomic Analysis, and provide a tested and maintained open-source platform for easy and reproducible Radiomic Feature extraction. By doing so, we hope to increase awareness of radiomic capabilities and expand the community. The platform supports both the feature extraction in 2D and 3D and can be used to calculate single values per feature for a region of interest ("segment-based") or to generate feature maps ("voxel-based").

**If you publish any work which uses this package, please cite the following publication:** *van Griethuysen, J. J. M., Fedorov, A., Parmar, C., Hosny, A., Aucoin, N., Narayan, V., Beets-Tan, R. G. H., Fillon-Robin, J. C., Pieper, S., Aerts, H. J. W. L. (2017). Computational Radiomics System to Decode the Radiographic Phenotype. Cancer Research, 77(21), e104–e107. 'https://doi.org/10.1158/0008-5472.CAN-17-0339 <https://doi.org/10.1158/0008-5472.CAN-17-0339>'_*

**Note:** This work was supported in part by the US National Cancer Institute grant 5U24CA194354, QUANTITATIVE RADIOMICS SYSTEM DECODING THE TUMOR PHENOTYPE.

**Warning:** Not intended for clinical use.

# CHAPTER 1

# Join the Community!

Join the PyRadiomics community on google groups here.

CHAPTER 2

# Table of Contents

## 2.1 Installation

There are three ways you can use pyradiomics: 1. Install via pip 2. Install from source 3. Use 3D Slicer Radiomics extension 4. Use pyradiomics Docker

### 2.1.1 1. Install via pip

Pre-built binaries are available on PyPi for installation via pip. For the python versions mentioned below, wheels are automatically generated for each release of PyRadiomics, allowing you to install pyradiomics without having to compile anything. For other python versions, a source distribution is also available, but this requires compiling the C extensions.

- Ensure that you have `python` installed on your machine, version 3.5, 3.6 or 3.7 (64-bits).
- Install PyRadiomics:

  ```
  python -m pip install pyradiomics
  ```

### 2.1.2 2. Install via conda

Besides pre-built binaries for PyPi, PyRadiomics is also available on conda cloud. To install PyRadiomics on Conda, run:

```
conda install -c radiomics pyradiomics
```

### 2.1.3 3. Install from source

PyRadiomics can also be installed from source code. This allows for the bleeding edge version, but does require you to have a compiler set up for python, as PyRadiomics comes with C extensions for the calculation of texture matrices and some shape features.

- Ensure you have the version control system `git` installed on your machine.
- Ensure that you have `python` installed on your machine, at least version 3.5 (64-bits).
- Clone the repository:

```
git clone git://github.com/Radiomics/pyradiomics
```

- For unix like systems (MacOSX, linux):

```
cd pyradiomics
python -m pip install -r requirements.txt
python setup.py install
```

  - To use your build for interactive use and development:

  ```
  python setup.py build_ext --inplace
  ```

  - If you don't have sudo/admin rights on your machine, you need to locally install numpy, nose, tqdm, PyWavelets, SimpleITK (specified in requirements.txt). In a bash shell:

  ```
  pip install --user --upgrade pip
  export PATH=$HOME/.local/bin:$PATH
  pip install --user -r requirements.txt
  export PYTHONPATH=$HOME/.local/lib64/python2.7/site-packages
  ```

- For Windows:

```
cd pyradiomics
python -m pip install -r requirements.txt
python setup.py install
```

- If the installation fails, check out the *Frequently Asked Questions*. If your error is not listed there, contact us by creating an issue on the PyRadiomics Github.

### 2.1.4 3. Use 3D Slicer Radiomics extension

3D Slicer is a free open source research platform for medical image computing. Learn more and download 3D Slicer binary for your platform here: http://slicer.org.

Once installed, you can use 3D Slicer ExtensionManager to install Radiomics extension, which provides a graphical user interface to the pyradiomics library. The advantage of using pyradiomics from 3D Slicer is that you can view images and segmentations, you can import existing segmentations and confirm their quality, or you can use the variety of tools in 3D Slicer to automate your segmentation tasks.

More detailed instructions about installing 3D Slicer Radiomics extension are available here: https://github.com/Radiomics/SlicerRadiomics

### 2.1.5 4. Use pyradiomics Docker

This approach may be preferred if you are interested in using pyradiomics from the command line, but have difficulties installing the library on your system.

First, you will need to install Docker on your system, if it is not installed already. You can follow the instructions below to do this.

Once Docker is installed, you can issue `docker pull radiomics/pyradiomics:CLI` command in the shell to download the pyradiomics Docker image. After that you can invoke pyradiomics tool as follows:

```
docker run radiomics/pyradiomics:CLI --help
```

Docker containers cannot directly access the filesystem of the host. In order to pass files as arguments to pyradiomics and to access files that converters create, an extra step is required to specify which directories will be used for file exchange using the -v argument:

```
-v <HOST_DIR>:<CONTAINER_DIR>
```

The argument above will make the `HOST_DIR` path available within the container at `CONTAINER_DIR` location. The files that will be read or written by the converter run from the docker container should be referred to via the `CONTAINER_DIR` path.

### 2.1.6 Setting up Docker

Docker (http://docker.com) is a project that automates deployment of applications inside software containers. Docker application is defined by _images_ that contain all of the components and steps needed to initialize the application instance. A _container_ is a running instance of the image. We provide an image that contains the compiled *pyradiomics* library in the *docker/pyradiomics:CLI* image. By using *pyradiomics* Docker container you can use *pyradiomics* on any operating system that supports Docker without having to compile *pyradiomics*. All you need to do is install Docker on your system, and download the *pyradiomics* Docker image.

You will first need to install Docker on your system following these instructions. Docker is available for Mac, Windows and Linux. For the most part Docker installation is straightforward, but some extra steps need to be taken on Windows as discussed below.

**If you use Docker on Windows . . .**

Note the system requirements:

- you will need to have Windows 10 Pro or above
- you will need to enable Hyper-V package (Docker will prompt you)
- you will need to enable virtualization; read this to learn how to check if it is enabled, and if it is not - here is one guide that may help you do that, but it assumes you can access your BIOS settings

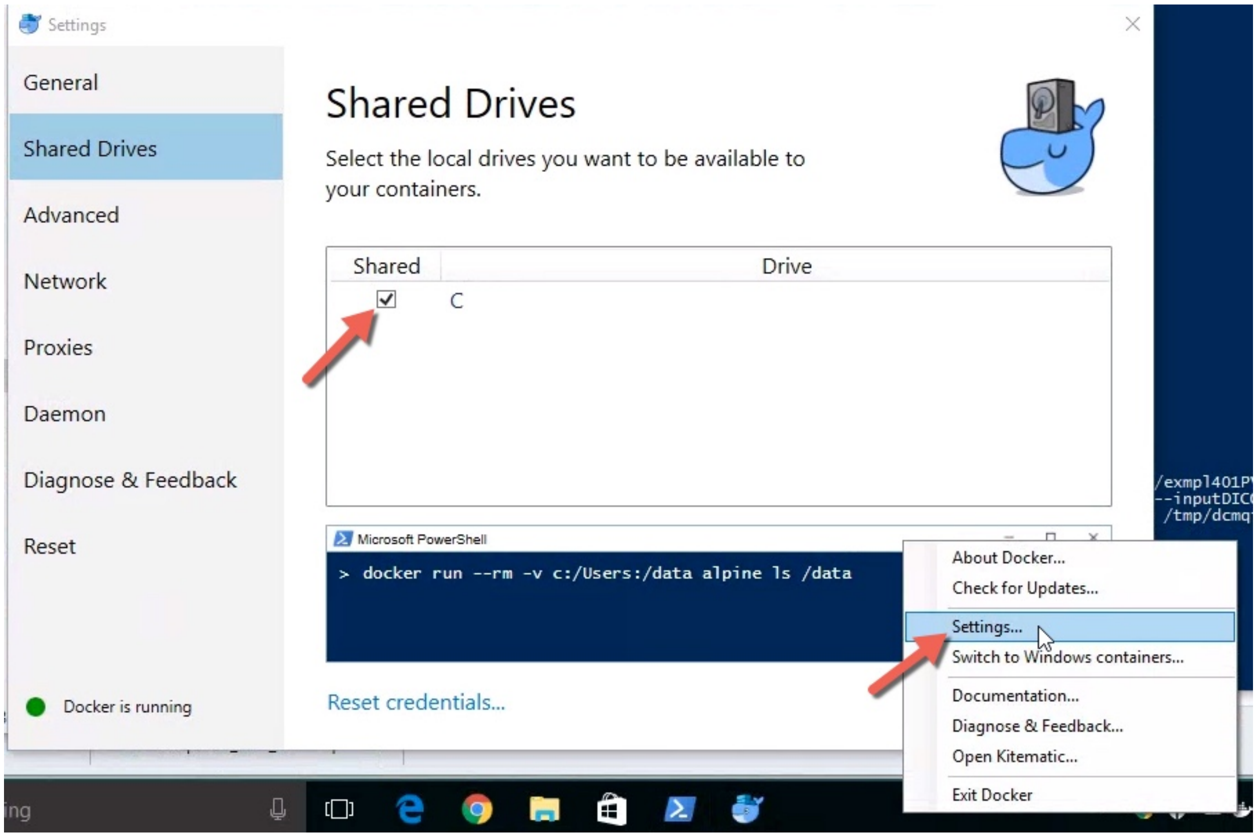
**IMPORTANT**: You will also need to share the drive you will be using to communicate data to and from Docker image in Docker Settings as shown in the screenshot below.

Most likely you will experience the display of an error message similar to the one shown below:

```
C:\Program Files\Docker\Docker\Resources\bin\docker.exe: Error response from daemon:
C: drive is not shared. Please share it in Docker for Windows Settings.
See 'C:\Program Files\Docker\Docker\Resources\bin\docker.exe run --help'.
```

If you have this error, make sure that the drive, where the `HOST_DIR` is located, is shared:

1. right click onto the Docker task bar icon and choose "Settings"
2. choose "Shared Drives" from the left menu (a list of drives that are available to share will be displayed)
3. select the drive for your `HOST_DIR` to be shared
4. confirm with Apply and continue



```
> docker run --rm -v c:/Users:/data alpine ls /data
```

## 2.2 Usage

### 2.2.1 Instruction Video

### 2.2.2 Example

- PyRadiomics example code and data is available in the Github repository
- The sample data is provided in `pyradiomics/data`
- Use jupyter to run the helloRadiomics example, located in `pyradiomics/notebooks`
- Jupyter can also be used to run the example notebook as shown in the instruction video
  - The example notebook can be found in `pyradiomics/notebooks`
  - The parameter file used in the instruction video is available in `pyradiomics/examples/exampleSettings`
- If jupyter is not installed, run the python script alternatives contained in the folder (`pyradiomics/examples`):
  - `python helloRadiomics.py` (segment-based extraction)
  - `python helloVoxel.py` (voxel-based extraction)

### 2.2.3 Voxel-based extraction

As of version 2.0, pyradiomics also implements a voxel-based extraction. It is both available from the command line and in the interactive use. See below for details.

Important to know here is that this extraction takes longer (features have to be calculated for each voxel), and that the output is a SimpleITK image of the parameter map instead of a float value *for each feature*.

### 2.2.4 Command Line Use

PyRadiomics can be used directly from the commandline via the entry point `pyradiomics`. Depending on the input provided, PyRadiomics is run in either single-extraction or batch-extraction mode. All options available on the commandline can be listed by running:

```
pyradiomics -h
```

#### Single image/mask

To extract features from a single image and segmentation run:

```
pyradiomics <path/to/image> <path/to/segmentation>
```

#### Batch Mode

To extract features from a batch run:

```
pyradiomics <path/to/input>
```

The input file for batch processing is a CSV file where the first row is contains headers and each subsequent row represents one combination of an image and a segmentation and contains at least 2 elements: 1) path/to/image, 2) path/to/mask. The headers specify the column names and **must** be “Image” and “Mask” for image and mask location, respectively (capital sensitive). Additional columns may also be specified, all columns are copied to the output in the same order (with calculated features appended after last column). To specify custom values for `label` in each combination, a column “Label” can optionally be added, which specifies the desired extraction label for each combination. Values specified in this column take precedence over label values specified in the parameter file or on the commandline. If a row contains no value, the default (or globally customized) value is used instead. Similarly, an optional value for the `label_channel` setting can be provided in a column “Label_channel”.

**Note:** All headers should be unique and different from headers provided by PyRadiomics (`<filter>_<class>_<feature>`). In case of conflict, values are overwritten by the PyRadiomics values.

**Note:** In batch processing, it is possible to speed up the process by applying multiprocessing. This is done on the case-level (i.e. each thread processes a single case). You can enable this by adding the `--jobs` parameter, specifying how many parallel threads you want to use.

### Customization

Extraction can be customized by specifying a *parameter file* in the `--param` argument and/or by specifying override settings (only *type 3 customization*) in the `--setting` argument. Multiple overrides can be used by specifying `--setting` multiple times.

### Output

By default, results are printed out to the console window. To store the results in a CSV-structured text file, add the `-o <PATH>` and `-f csv` arguments, where `<PATH>` specifies the filepath where the results should be stored. e.g.:

```
pyradiomics <path/to/image> <path/to/segmentation> -o results.csv -f csv
pyradiomics <path/to/input> -o results.csv -f csv
```

### Voxel-based Radiomics

To extract feature maps ("voxel-based" extraction), simply add the argument `--mode voxel`. The calculated feature maps are then stored as images (NRRD format) in the current working directory. The name convention used is "Case-<idx>_<FeatureName>.nrrd". An alternative output directory can be provided in the `--out-dir` command line switch. The results that are printed to the console window or the out file will still contain the diagnostic information, and the value of the extracted features is set to the location where the feature maps are stored.

## 2.2.5 Interactive Use

- (LINUX) To run from source code, add pyradiomics to the environment variable PYTHONPATH (Not necessary when PyRadiomics is installed):
  - `setenv PYTHONPATH /path/to/pyradiomics/radiomics`
- Start the python interactive session:
  - `python`
- Import the necessary classes:

```
import os

import SimpleITK as sitk
import six

from radiomics import featureextractor, getTestCase
```

- Set up a pyradiomics directory variable:

```
dataDir = '/path/to/pyradiomics'
```

- You will find sample data files brain1_image.nrrd and brain1_label.nrrd in that directory. Note that NRRD format used here does not mean that your image and label must always be in this format. Any format readable by ITK is suitable (e.g., NIfTI, MHA, MHD, HDR, etc). See more details in **'this section of FAQ https://pyradiomics.readthedocs.io/en/latest/faq.html#what-file-types-are-supported-by-pyradiomics-for-input-image-and-mask'_**.
- Store the path of your image and mask in two variables:

```
imageName, maskName = getTestCase('brain1', dataDir)
```

- Also store the path to the file containing the extraction settings:

```
params = os.path.join(dataDir, "examples", "exampleSettings", "Params.yaml")
```

- Instantiate the feature extractor class with the parameter file:

```
extractor = featureextractor.RadiomicsFeatureExtractor(params)
```

- Calculate the features (segment-based):

```
result = extractor.execute(imageName, maskName)
for key, val in six.iteritems(result):
  print("\t%s: %s" %(key, val))
```

- Calculate the features (voxel-based):

```
result = extractor.execute(imageName, maskName, voxelBased=True)
for key, val in six.iteritems(result):
  if isinstance(val, sitk.Image):  # Feature map
    sitk.WriteImage(val, key + '.nrrd', True)
    print("Stored feature %s in %s" % (key, key + ".nrrd"))
  else:  # Diagnostic information
    print("\t%s: %s" %(key, val))
```

- See the *feature extractor class* for more information on using this core class.

### 2.2.6 PyRadiomics in 3D Slicer

A convenient front-end interface is provided as the 'Radiomics' extension for 3D Slicer. It is available here.

### 2.2.7 Setting Up Logging

PyRadiomics features extensive logging to help track down any issues with the extraction of features. By default PyRadiomics logging reports messages of level WARNING and up (reporting any warnings or errors that occur), and prints this to the output (stderr). By default, PyRadiomics does not create a log file.

To change the amount of information that is printed to the output, use `setVerbosity()` in interactive use and the optional `--verbosity` argument in commandline use.

When using PyRadiomics in interactive mode, enable storing the PyRadiomics logging in a file by adding an appropriate handler to the pyradiomics logger:

```
import radiomics

log_file = 'path/to/log_file.txt'
handler = logging.FileHandler(filename=log_file, mode='w')   # overwrites log_files␣
↪from previous runs. Change mode to 'a' to append.
formatter = logging.Formatter("%(levelname)s:%(name)s: %(message)s")   # format string␣
↪for log messages
handler.setFormatter(formatter)
radiomics.logger.addHandler(handler)

# Control the amount of logging stored by setting the level of the logger. N.B. if␣
↪the level is higher than the
```

(continues on next page)

(continued from previous page)

```
# Verbositiy level, the logger level will also determine the amount of information␣
↪printed to the output
radiomics.logger.setLevel(logging.DEBUG)
```

To store a log file when running pyradiomics from the commandline, specify a file location in the optional `--log-file` argument. The amount of logging that is stored is controlled by the `--logging-level` argument (default level WARNING and up).

## 2.3 Customizing the Extraction

### 2.3.1 Types of Customization

There are 4 ways in which the feature extraction can be customized in PyRadiomics:

1. Specifying which image types (original/derived) to use to extract features from
2. Specifying which feature(class) to extract
3. Specifying settings, which control the pre processing and customize the behaviour of enabled filters and feature classes.
4. Specifying the voxel-based specific settings, which are only needed when using PyRadiomics to generate feature maps

**Warning:** At initialization of the feature extractor or an individual feature class, settings can be provided as keyword arguments in `**kwargs`. These consist *only* of type 3 parameters (setting). Parameters of type 1 (image type) and 2 (feature class) can only provided at initialization when using the parameter file. When the parameter file is not used, or when these parameters have to be changed after initialization, use the respective function calls.

#### Image Types

These are the image types (either the original image or derived images using filters) that can be used to extract features from. The image types that are available are determined dynamically (all are functions in `imageoperations.py` that fit the *signature* of an image type).

The enabled types are stored in the `_enabledImageTypes` dictionary in the feature extractor class instance and can be changed using the functions `enableAllImageTypes()`, `disableAllImageTypes()`, `enableImageTypeByName()` and `enableImageTypes()`. Moreover, custom settings can be provided for each enabled image type, which will then only be applied during feature extraction for that image type. Please note that this will only work for settings that are applied at or after any filter is applied (i.e. not at the feature extractor level).

**Note:** This type of customization can be included in the *Parameter File* using key `imageType`.

**Note:** By default, only the “Original” image type is enabled.

Currently available image types are:

- Original: No filter applied

- Wavelet: Wavelet filtering, yields 8 decompositions per level (all possible combinations of applying either a High or a Low pass filter in each of the three dimensions. See also `getWaveletImage()`
- LoG: Laplacian of Gaussian filter, edge enhancement filter. Emphasizes areas of gray level change, where sigma defines how coarse the emphasised texture should be. A low sigma emphasis on fine textures (change over a short distance), where a high sigma value emphasises coarse textures (gray level change over a large distance). See also `getLoGImage()`
- Square: Takes the square of the image intensities and linearly scales them back to the original range. Negative values in the original image will be made negative again after application of filter.
- SquareRoot: Takes the square root of the absolute image intensities and scales them back to original range. Negative values in the original image will be made negative again after application of filter.
- Logarithm: Takes the logarithm of the absolute intensity + 1. Values are scaled to original range and negative original values are made negative again after application of filter.
- Exponential: Takes the the exponential, where filtered intensity is e^(absolute intensity). Values are scaled to original range and negative original values are made negative again after application of filter.
- Gradient: Returns the magnitude of the local gradient. See also `getGradientImage()`
- LocalBinaryPattern2D: Computes the Local Binary Pattern in a by-slice operation (2D). See also `getLBP2DImage()`
- LocalBinaryPattern3D: Computes the Local Binary Pattern in 3D using spherical harmonics. See also `getLBP3DImage()`

### Enabled Features

These are the features that are extracted from each (original and/or derived) image type. The available features are determined dynamically, and are ordered in feature classes. For more information on the signature used to identify features and feature classes, see the *Developers* section.

The enable features are stored in the `_enabledFeatures` dictionary in the feature extractor class instance and can be changed using the functions `enableAllFeatures()`, `disableAllFeatures()`, `enableFeatureClassByName()` and `enableFeaturesByName()`. Each key-value pair in the dictionary represents one enabled feature class with the feature class name as the key and a list of enabled feature names as value. If the value is `None` or an empty list, all features in that class are enabled. Otherwise only the features specified.

**Note:** This type of customization can be included in the *Parameter File* using key `featureClass`.

**Note:** By default, all feature classes and all features are enabled.

Currently available feature classes are:

- firstorder
- shape
- glcm
- glrlm
- glszm
- gldm
- ngtdm

An individual feature can be enabled by submitting the feature name as defined in the unique part of the function signature (e.g. the First Order feature defined by `get10PercentileFeatureValue()` is enabled by specifying `{firstorder: ['10Percentile']}`). Function signatures for all features are available in the *Radiomic Features* section.

### Settings

Besides customizing what to extract (image types, features), PyRadiomics exposes various settings customizing how the features are extracted. These settings operate at different levels. E.g. resampling is done just after the images are loaded (in the feature extractor), so settings controlling the resampling operate only on the feature extractor level. Settings are stored in the `setttings` dictionary in the feature extractor class instance, where the key is the case sensitive setting name. Custom settings are provided as keyword arguments at initialization of the feature extractor (with the setting name as keyword and value as the argument value, e.g. `binWidth=25`), or by interacting directly with the `settings` dictionary.

**Note:** This type of customization can be included in the *Parameter File* using key `setting`.

**Note:** When using the feature classes directly, feature class level settings can be customized by providing them as keyword arguments at initialization of the feature class.

Below are the settings that control the behaviour of the extraction, ordered per level and category. Each setting is listed as it's unique, case sensitive name, followed by it's default value in brackets. After the default value is the documentation on the type of the value and what the setting controls.

### Feature Extractor Level

*Image Normalization*

- `normalize` [False]: Boolean, set to True to enable normalizing of the image before any resampling. See also *`normalizeImage()`*.
- `normalizeScale` [1]: Float, > 0, determines the scale after normalizing the image. If normalizing is disabled, this has no effect.
- `removeOutliers` [None]: Float, > 0, defines the outliers to remove from the image. An outlier is defined as values that differ more than $n\sigma_x$ from the mean, where $n > 0$ and equal to the value of this setting. If this parameter is omitted (providing it without a value (i.e. None) in the parameter file will throw an error), no outliers are removed. If normalizing is disabled, this has no effect. See also *`normalizeImage()`*.

*Resampling the image/mask*

- `resampledPixelSpacing` [None]: List of 3 floats (>= 0), sets the size of the voxel in (x, y, z) plane when resampling. A value of 0 is replaced with the spacing for that dimension as it is in the original (non-resampled) image or mask. For example, to perform only in-plane resampling, the x and y values alone should be edited (e.g.: [2,2,0]). In-plane resolution is always relative to image acquisition plane (i.e. axial, coronal or sagittal).
- `interpolator` [sitkBSpline]: SimpleITK constant or string name thereof, sets interpolator to use for resampling. The choice of `interpolator` is only applied to resampling images, while `sitkNearestNeighbor` is always used for resampling masks in order to preserve label values. Enumerated value, possible values:
  - sitkNearestNeighbor (= 1)
  - sitkLinear (= 2)

- sitkBSpline (= 3)
- sitkGaussian (= 4)
- sitkLabelGaussian (= 5)
- sitkHammingWindowedSinc (= 6)
- sitkCosineWindowedSinc (= 7)
- sitkWelchWindowedSinc (= 8)
- sitkLanczosWindowedSinc (= 9)
- sitkBlackmanWindowedSinc (= 10)

- `padDistance` [5]: Integer, $\geq 0$, set the number of voxels pad cropped tumor volume with during resampling. Padding occurs in new feature space and is done on all faces, i.e. size increases in x, y and z direction by 2*padDistance. Padding is needed for some filters (e.g. LoG). Value of padded voxels are set to original gray level intensity, padding does not exceed original image boundaries. **N.B. After application of filters image is cropped again without padding.**

**Note:** Resampling is disabled when either *resampledPixelSpacing* or *interpolator* is set to *None*

*Pre-Cropping*

- `preCrop` [False]: Boolean, if true and resampling is disabled, crops the image onto the bounding box with additional padding as specified in `padDistance`. Similar to padding after resampling, padding does not exceed original image bounds after pre-cropping. Setting `preCrop` to true speeds up extraction and makes it less memory intensive, especially in the case of large images with only small ROIs.

**Note:** Because image and mask are also cropped onto the bounding box before they are passed to the feature classes, pre-crop is only beneficial when filters are enabled.

*Resegmentation*

- `resegmentRange` [None]: List of 1 or 2 floats, specifies the lower and and optionally upper threshold, respectively. Segmented voxels outside this range are removed from the mask prior to feature calculation. When the value is None (default), no resegmentation is performed. Resegemented size is checked (using parameter `minimumROISize`, default 1) and upon fail, an error is logged and extraction is skipped for this case.
- `resegmentMode` ['absolute']: string, specifying the method to use for defining the resegmentation thresholds:
  - 'absolute': The resegmentRange values are treated as absolute values, i.e. used directly to perform resegmentation.
  - 'relative': The resegmentRange values are treated as relative to the maximum in the ROI, i.e. the actual threshold used is defined as threshold $=$ value $* X_{max}$.
  - 'sigma': The resegmentRange values indicate a distance from the mean of the ROI in standard deviations. E.g. to exclude outliers farther from the mean than 3 sigma, specify mode 'sigma' and range [-3, 3]. Threshold is defined as threshold $= \mu +$ value $* \sigma$.
- `resegmentShape` [False]: Boolean, if set to True, the resegmented mask is also used for shape calculation. If set to False (default), only first order and texture classes are calculated using the resegmented mask (known in IBSI as the intensity mask). Shape is then calculated using the mask after any optional resampling and corrections (known in IBSI as the morphologic mask).

*Mask validation*

- `minimumROIDimensions` [2]: Integer, range 1-3, specifies the minimum dimensions (1D, 2D or 3D, respectively). Single-voxel segmentations are always excluded.
- `minimumROISize` [None]: Integer, > 0, specifies the minimum number of voxels required. Test is skipped if this parameter is omitted (specifying it as None in the parameter file will throw an error).
- `geometryTolerance` [None]: Float, determines the tolarance used by SimpleITK to compare origin, direction and spacing between image and mask. Affects the fist step in *`checkMask()`*. If set to `None`, PyRadiomics will use SimpleITK default (1e-16).
- `correctMask` [False]: Boolean, if set to true, PyRadiomics will attempt to resample the mask to the image geometry when the first step in *`checkMask()`* fails. This uses a nearest neighbor interpolator. Mask check will still fail if the ROI defined in the mask includes areas outside of the image physical space.

*Miscellaneous*

- `additionalInfo` [True]: boolean, set to False to disable inclusion of additional information on the extraction in the output. See also `addProvenance()`.

## Filter Level

*Laplacian of Gaussian settings*

- `sigma`: List of floats or integers, must be greater than 0. Sigma values to use for the filter (determines coarseness).

**Warning:** Setting for sigma must be provided if LoG filter is enabled. If omitted, no LoG image features are calculated and the function will return an empty dictionary.

*Wavelet settings*

- `start_level` [0]: integer, 0 based level of wavelet which should be used as first set of decompositions from which a signature is calculated
- `level` [1]: integer, number of levels of wavelet decompositions from which a signature is calculated.
- `wavelet` ["coif1"]: string, type of wavelet decomposition. Enumerated value, validated against possible values present in the `pyWavelet.wavelist()`. Current possible values (pywavelet version 0.4.0) (where an aditional number is needed, range of values is indicated in []):
  - haar
  - dmey
  - sym[2-20]
  - db[1-20]
  - coif[1-5]
  - bior[1.1, 1.3, 1.5, 2.2, 2.4, 2.6, 2.8, 3.1, 3.3, 3.5, 3.7, 3.9, 4.4, 5.5, 6.8]
  - rbio[1.1, 1.3, 1.5, 2.2, 2.4, 2.6, 2.8, 3.1, 3.3, 3.5, 3.7, 3.9, 4.4, 5.5, 6.8]

*Gradient settings*

- `gradientUseSpacing` [True]: Boolean, if true, image spacing is taken into account when computing the gradient magnitude in the image.

*Local Binary Pattern 2D*

- `lbp2DRadius` [1]: Float, > 0, specifies the radius in which the neighbours should be sampled

- `lbp2DSamples` [9]: Integer, $\geq 1$, specifies the number of samples to use
- `lbp2DMethod` ['uniform']: String, specifies the method for computing the LBP to use.

**Warning:** Requires package `skimage` to function.

*Local Binary Pattern 3D*

- `lbp3DLevels` [2]: integer, $\geq 1$, specifies the the number of levels in spherical harmonics to use.
- `lbp3DIcosphereRadius` [1]: Float, > 0, specifies the radius in which the neighbours should be sampled
- `lbp3DIcosphereSubdivision` [1]: Integer, $\geq 0$, specifies the number of subdivisions to apply in the icosphere

**Warning:** Requires package `scipy` and `trimesh` to function.

## Feature Class Level

- `Label` [1]: Integer, label value of Region of Interest (ROI) in labelmap.

*Image discretization*

- `binWidth` [25]: Float, > 0, size of the bins when making a histogram and for discretization of the image gray level.
- `binCount` [None]: integer, > 0, specifies the number of bins to create. The width of the bin is then determined by the range in the ROI. No definitive evidence is available on which method of discretization is superior, we advise a fixed bin width. See more *here*.

*Forced 2D extraction*

- `force2D` [False]: Boolean, set to true to force a by slice texture calculation. Dimension that identifies the 'slice' can be defined in `force2Ddimension`. If input ROI is already a 2D ROI, features are automatically extracted in 2D.
- `force2Ddimension` [0]: int, range 0-2. Specifies the 'slice' dimension for a by-slice feature extraction. A value of 0 represents the native acquisition plane for the images (usually axial for CT and axial, coronal or sagittal for MRI). Similarly, 1 identifies the out-of plane y dimension (e.g. coronal plane for an axial image) and 2 the out-of-plane x dimension (e.g. sagittal plane for an acial image). if `force2D` is set to False, this parameter has no effect.

*Texture matrix weighting*

- `weightingNorm` [None]: string, indicates which norm should be used when applying distance weighting. Enumerated setting, possible values:
  - 'manhattan': first order norm
  - 'euclidean': second order norm
  - 'infinity': infinity norm.
  - 'no_weighting': GLCMs are weighted by factor 1 and summed
  - None: Applies no weighting, mean of values calculated on separate matrices is returned.

  In case of other values, an warning is logged and option 'no_weighting' is used.

**Note:** This only affects the GLCM and GLRLM feature classes. Moreover, weighting is applied differently in those classes. For more information on how weighting is applied, see the documentation on *GLCM* and *GLRLM*.

*Distance to neighbour*

- `distances` [[1]]: List of integers. This specifies the distances between the center voxel and the neighbor, for which angles should be generated.

**Note:** This only affects the GLCM and NGTDM feature classes. The GLSZM and GLRLM feature classes use a fixed distance of 1 (infinity norm) to define neighbours.

### Feature Class Specific Settings

*First Order*

- `voxelArrayShift` [0]: Integer, This amount is added to the gray level intensity in features Energy, Total Energy and RMS, this is to prevent negative values. *If using CT data, or data normalized with mean 0, consider setting this parameter to a fixed value (e.g. 2000) that ensures non-negative numbers in the image. Bear in mind however, that the larger the value, the larger the volume confounding effect will be.*

*GLCM*

- `symmetricalGLCM` [True]: boolean, indicates whether co-occurrences should be assessed in two directions per angle, which results in a symmetrical matrix, with equal distributions for $i$ and $j$. A symmetrical matrix corresponds to the GLCM as defined by Haralick et al.

*GLDM*

- `gldm_a` [0]: float, $\alpha$ cutoff value for dependence. A neighbouring voxel with gray level $j$ is considered dependent on center voxel with gray level $i$ if $|i - j| \le \alpha$

### Voxel-based specific settings

When using PyRadiomics to generate feature maps, additional customization options exist. These control the neighborhood around each voxel that is used for calculation (kernel) and what the background value should be, i.e. the value of voxels for which there is no calculated value.

- `kernelRadius` [1]: integer, specifies the size of the kernel to use as the radius from the center voxel. Therefore the actual size is `2 * kernelRadius + 1`. E.g. a value of 1 yields a 3x3x3 kernel, a value of 2 5x5x5, etc. In case of 2D extraction, the generated kernel will also be a 2D shape (square instead of cube).
- `maskedKernel` [True]: boolean, specifies whether to mask the kernel with the overall mask. If True, only voxels in the kernel that are also segmented in the mask are used for calculation. Otherwise, all voxels inside the kernel are used. Moreover, gray value discretization is performed over the ROI if the setting is set to True, and over the entire image if False.
- `initValue` [0]: float, value to use for voxels outside the ROI, or voxels where calculation failed. If set to `nan`, 3D slicer will treat them as transparent voxels
- `voxelBatch` [-1]: integer > 0, this value controls the maximum number of voxels that are calculated in one batch. Larger batches mean less loops in Python and therefore a quicker extraction, but do require more memory. This setting allows the user to compromise between extraction speed and memory usage. When providing this setting, the value is constrained to be > 0, only by not providing it is the default value of -1 used (which means: all voxels in 1 batch).

### 2.3.2 Parameter File

All 4 categories of customization can be provided in a single yaml or JSON structured text file, which can be provided in an optional argument (`--param`) when running pyradiomics from the command line. In interactive mode, it can be provided during initialization of the *feature extractor*, or using `loadParams()` after initialization. This removes the need to hard code a customized extraction in a python script through use of functions described above. Additionally, this also makes it more easy to share settings for customized extractions. We encourage users to share their parameter files in the PyRadiomics repository. See *Submitting a parameter file* for more information on how to submit your parameter file.

**Note:** For an extensive list of possible settings, see *Image Types*, *Feature Classes* and *Settings*, which can be provided in the parameter file using key `imageType`, `featureClass` and `setting`, respectively.

**Note:** Examples of the parameter file are provided in the `pyradiomics/examples/exampleSettings` folder.

The paramsFile is written according to the YAML-convention (www.yaml.org) and is checked by the code for consistency. Only one yaml document per file is allowed. Parameters must be grouped by customization category as mentioned above. This is reflected in the structure of the document as follows:

```
<Customization Category>:
  <Setting Name>: <value>
  ...
<Customization Categort>:
  ...
```

Blank lines may be inserted to increase readability, these are ignored by the parser. Additional comments are also possible, these are preceded by an '#' and can be inserted on a blank line, or on a line containing parameters:

```
# This is a line containing only comments
setting: # This is a comment placed after the declaration of the 'setting' category.
```

Any keyword, such as a customization category or setting name may only be mentioned once. Multiple instances do not raise an error, but only the last one encountered is used.

The three setting types are named as follows:

1. **imageType:** image type to calculate features on. <value> is custom kwarg settings (dictionary). if <value> is an empty dictionary ('{}'), no custom settings are added for this input image.
2. **featureClass:** Feature class to enable, <value> is list of strings representing enabled features. If no <value> is specified or <value> is an empty list ('[]'), all features for this class are enabled.
3. **setting:** Setting to use for pre processing and class specific settings. if no <value> is specified, the value for this setting is set to None.
4. **voxelSetting:** Settings used to control the voxel-based specific settings. E.g. the size of the kernel used and the background value in the parameter maps.

Example:

```
# This is a non-active comment on a separate line
imageType:
    Original: {}
    LoG: {'sigma' : [1.0, 3.0]}  # This is a non active comment on a line with active␣
↪code preceding it.
```

(continues on next page)

(continued from previous page)

```
    Wavelet:
        binWidth: 10

featureClass:
    glcm:
    glrlm: []
    firstorder: ['Mean',
                 'StandardDeviation']
    shape:
        - Volume
        - SurfaceArea

setting:
    binWidth: 25
    resampledPixelSpacing:
```

In this example, 3 image types are enabled ("Original", "LoG" (Laplacian of Gaussian) and "Wavelet"), with custom settings specified for "LoG" ("sigma") and "Wavelet" ("binWidth"). Note that the manner of specifying the custom settings for "LoG" and "Wavelet" is equivalent.

Next, 4 feature classes are defined. "glcm" and "glrlm" are both enabled with all possible features in the respective class, whereas only "Mean" and "StandardDeviation" are enabled for "firstorder", and only "Volume" and "SurfaceArea" for shape. Note that the manner of specifying individual features for "firstorder" and "shape" is equivalent.

Finally, 2 settings are specified: "binWidth", whose value has been set to 25 (but will be set to 10 during extraction of "Wavelet" derived features), and "resampledPixelSpacing", where no value is provided, which is equivalent to a python "None" value.

**Note:**

- settings not specified in parameters are set to their default value.
- enabledFeatures are replaced by those in parameters (i.e. only specified features/classes are enabled. If the 'featureClass' customization type is omitted, all feature classes and features are enabled.
- ImageTypes are replaced by those in parameters (i.e. only specified types are used to extract features from. If the 'inputImage' customization type is omitted, only "Original" image type is used for feature extraction, with no additional custom settings.

## 2.4 Pipeline Modules

This section contains the documentation on the various modules used to define the PyRadiomics pipeline and pre-process the input data. Feature class modules, which contain the feature definitions are documented in the *Radiomic Features* section.

Additionally, this section contains the documentation for the *radiomics.generalinfo module*, which provides the additional information about the extraction in the output. This additional information is added to enhance reproducibility of the results.

Finally, this section contains documentation for the *global functions*, which are used throughout the toolbox (such as logging and the C extensions) and the *radiomics.base module*, which defines the common interface for the feature classes.

### 2.4.1 Feature Extractor

**class** `radiomics.featureextractor.`**`RadiomicsFeatureExtractor`**(**args*, ***kwargs*)

Bases: `object`

Wrapper class for calculation of a radiomics signature. At and after initialisation various settings can be used to customize the resultant signature. This includes which classes and features to use, as well as what should be done in terms of preprocessing the image and what images (original and/or filtered) should be used as input.

Then a call to `execute()` generates the radiomics signature specified by these settings for the passed image and labelmap combination. This function can be called repeatedly in a batch process to calculate the radiomics signature for all image and labelmap combinations.

At initialization, a parameters file (string pointing to yaml or json structured file) or dictionary can be provided containing all necessary settings (top level containing keys "setting", "imageType" and/or "featureClass). This is done by passing it as the first positional argument. If no positional argument is supplied, or the argument is not either a dictionary or a string pointing to a valid file, defaults will be applied. Moreover, at initialisation, custom settings (*NOT enabled image types and/or feature classes*) can be provided as keyword arguments, with the setting name as key and its value as the argument value (e.g. `binWidth=25`). Settings specified here will override those in the parameter file/dict/default settings. For more information on possible settings and customization, see *Customizing the Extraction*.

By default, all features in all feature classes are enabled. By default, only *Original* input image is enabled (No filter applied).

**`addProvenance`**(*provenance_on=True*)

Enable or disable reporting of additional information on the extraction. This information includes toolbox version, enabled input images and applied settings. Furthermore, additional information on the image and region of interest (ROI) is also provided, including original image spacing, total number of voxels in the ROI and total number of fully connected volumes in the ROI.

To disable this, call `addProvenance(False)`.

**`loadParams`**(*paramsFile*)

Parse specified parameters file and use it to update settings, enabled feature(Classes) and image types. For more information on the structure of the parameter file, see *Customizing the extraction*.

If supplied file does not match the requirements (i.e. unrecognized names or invalid values for a setting), a pykwalify error is raised.

**`loadJSONParams`**(*JSON_configuration*)

Pars JSON structured configuration string and use it to update settings, enabled feature(Classes) and image types. For more information on the structure of the parameter file, see *Customizing the extraction*.

If supplied string does not match the requirements (i.e. unrecognized names or invalid values for a setting), a pykwalify error is raised.

**`execute`**(*imageFilepath*, *maskFilepath*, *label=None*, *label_channel=None*, *voxelBased=False*)

Compute radiomics signature for provide image and mask combination. It comprises of the following steps:

1. Image and mask are loaded and normalized/resampled if necessary.
2. Validity of ROI is checked using `checkMask()`, which also computes and returns the bounding box.
3. If enabled, provenance information is calculated and stored as part of the result. (Not available in voxel-based extraction)
4. Shape features are calculated on a cropped (no padding) version of the original image. (Not available in voxel-based extraction)

5. If enabled, resegment the mask based upon the range specified in `resegmentRange` (default None: resegmentation disabled).
6. Other enabled feature classes are calculated using all specified image types in `_enabledImageTypes`. Images are cropped to tumor mask (no padding) after application of any filter and before being passed to the feature class.
7. The calculated features is returned as `collections.OrderedDict`.

**Parameters**

- **`imageFilepath`** – SimpleITK Image, or string pointing to image file location
- **`maskFilepath`** – SimpleITK Image, or string pointing to labelmap file location
- **`label`** – Integer, value of the label for which to extract features. If not specified, last specified label is used. Default label is 1.
- **`label_channel`** – Integer, index of the channel to use when maskFilepath yields a SimpleITK.Image with a vector pixel type. Default index is 0.
- **`voxelBased`** – Boolean, default False. If set to true, a voxel-based extraction is performed, segment-based otherwise.

**Returns** dictionary containing calculated signature ("<imageType>_<featureClass>_<featureName>":value). In case of segment-based extraction, value type for features is float, if voxel-based, type is SimpleITK.Image. Type of diagnostic features differs, but can always be represented as a string.

**`static loadImage`**(*ImageFilePath*, *MaskFilePath*, *generalInfo=None*, ***kwargs*)

Load and pre-process the image and labelmap. If ImageFilePath is a string, it is loaded as SimpleITK Image and assigned to `image`, if it already is a SimpleITK Image, it is just assigned to `image`. All other cases are ignored (nothing calculated). Equal approach is used for assignment of `mask` using MaskFilePath. If necessary, a segmentation object (i.e. mask volume with vector-image type) is then converted to a labelmap (=scalar image type). Data type is forced to UInt32. See also `getMask()`.

If normalizing is enabled image is first normalized before any resampling is applied.

If resampling is enabled, both image and mask are resampled and cropped to the tumor mask (with additional padding as specified in padDistance) after assignment of image and mask.

**Parameters**

- **`ImageFilePath`** – SimpleITK.Image object or string pointing to SimpleITK readable file representing the image to use.
- **`MaskFilePath`** – SimpleITK.Image object or string pointing to SimpleITK readable file representing the mask to use.
- **`generalInfo`** – GeneralInfo Object. If provided, it is used to store diagnostic information of the pre-processing.
- **`kwargs`** – Dictionary containing the settings to use for this particular image type.

**Returns** 2 SimpleITK.Image objects representing the loaded image and mask, respectively.

**`computeShape`**(*image*, *mask*, *boundingBox*, ***kwargs*)

Calculate the shape (2D and/or 3D) features for the passed image and mask.

**Parameters**

- **`image`** – SimpleITK.Image object representing the image used
- **`mask`** – SimpleITK.Image object representing the mask used

- **boundingBox** – The boundingBox calculated by `checkMask()`, i.e. a tuple with lower (even indices) and upper (odd indices) bound of the bounding box for each dimension.
- **kwargs** – Dictionary containing the settings to use.

**Returns** collections.OrderedDict containing the calculated shape features. If no features are calculated, an empty OrderedDict will be returned.

**computeFeatures**(*image*, *mask*, *imageTypeName*, ***kwargs*)

Compute signature using image, mask and **kwargs settings.

This function computes the signature for just the passed image (original or derived), it does not pre-process or apply a filter to the passed image. Features / Classes to use for calculation of signature are defined in `self.enabledFeatures`. See also *`enableFeaturesByName()`*.

**Parameters**

- **image** – The cropped (and optionally filtered) SimpleITK.Image object representing the image used
- **mask** – The cropped SimpleITK.Image object representing the mask used
- **imageTypeName** – String specifying the filter applied to the image, or "original" if no filter was applied.
- **kwargs** – Dictionary containing the settings to use for this particular image type.

**Returns** collections.OrderedDict containing the calculated features for all enabled classes. If no features are calculated, an empty OrderedDict will be returned.

**Note:** shape descriptors are independent of gray level and therefore calculated separately (handled in *execute*). In this function, no shape features are calculated.

**enableAllImageTypes**()

Enable all possible image types without any custom settings.

**disableAllImageTypes**()

Disable all image types.

**enableImageTypeByName**(*imageType*, *enabled=True*, *customArgs=None*)

Enable or disable specified image type. If enabling image type, optional custom settings can be specified in customArgs.

Current possible image types are:

- Original: No filter applied
- Wavelet: Wavelet filtering, yields 8 decompositions per level (all possible combinations of applying either a High or a Low pass filter in each of the three dimensions. See also *`getWaveletImage()`*
- LoG: Laplacian of Gaussian filter, edge enhancement filter. Emphasizes areas of gray level change, where sigma defines how coarse the emphasised texture should be. A low sigma emphasis on fine textures (change over a short distance), where a high sigma value emphasises coarse textures (gray level change over a large distance). See also *`getLoGImage()`*
- Square: Takes the square of the image intensities and linearly scales them back to the original range. Negative values in the original image will be made negative again after application of filter.
- SquareRoot: Takes the square root of the absolute image intensities and scales them back to original range. Negative values in the original image will be made negative again after application of filter.

- Logarithm: Takes the logarithm of the absolute intensity + 1. Values are scaled to original range and negative original values are made negative again after application of filter.
- Exponential: Takes the the exponential, where filtered intensity is e^(absolute intensity). Values are scaled to original range and negative original values are made negative again after application of filter.
- Gradient: Returns the gradient magnitude.
- LBP2D: Calculates and returns a local binary pattern applied in 2D.
- LBP3D: Calculates and returns local binary pattern maps applied in 3D using spherical harmonics. Last returned image is the corresponding kurtosis map.

For the mathmetical formulas of square, squareroot, logarithm and exponential, see their respective functions in *imageoperations* (`getSquareImage()`, `getSquareRootImage()`, `getLogarithmImage()`, `getExponentialImage()`, `getGradientImage()`, `getLBP2DImage()` and `getLBP3DImage()`, respectively).

**`enableImageTypes`**(***enabledImagetypes*)

Enable input images, with optionally custom settings, which are applied to the respective input image. Settings specified here override those in kwargs. The following settings are not customizable:

- interpolator
- resampledPixelSpacing
- padDistance

Updates current settings: If necessary, enables input image. Always overrides custom settings specified for input images passed in inputImages. To disable input images, use `enableInputImageByName()` or `disableAllInputImages()` instead.

**Parameters** **`enabledImagetypes`** – dictionary, key is imagetype (original, wavelet or log) and value is custom settings (dictionary)

**`enableAllFeatures`**()

Enable all classes and all features.

**Note:** Individual features that have been marked "deprecated" are not enabled by this function. They can still be enabled manually by a call to `enableFeatureByName()`, `enableFeaturesByName()` or in the parameter file (by specifying the feature by name, not when enabling all features). However, in most cases this will still result only in a deprecation warning.

**`disableAllFeatures`**()

Disable all classes.

**`enableFeatureClassByName`**(*featureClass*, *enabled=True*)

Enable or disable all features in given class.

**Note:** Individual features that have been marked "deprecated" are not enabled by this function. They can still be enabled manually by a call to `enableFeatureByName()`, `enableFeaturesByName()` or in the parameter file (by specifying the feature by name, not when enabling all features). However, in most cases this will still result only in a deprecation warning.

**`enableFeaturesByName`**(***enabledFeatures*)

Specify which features to enable. Key is feature class name, value is a list of enabled feature names.

To enable all features for a class, provide the class name with an empty list or None as value. Settings for feature classes specified in enabledFeatures.keys are updated, settings for feature classes not yet present in enabledFeatures.keys are added. To disable the entire class, use *disableAllFeatures()* or *enableFeatureClassByName()* instead.

## 2.4.2 Image Processing and Filters

radiomics.imageoperations.**getMask**(*mask*, ***kwargs*)

Function to get the correct mask. Includes enforcing a correct pixel data type (UInt32).

Also supports extracting the mask for a segmentation (stored as SimpleITK Vector image) if necessary. In this case, the mask at index `label_channel` is extracted. The resulting 3D volume is then treated as it were a scalar input volume (i.e. with the region of interest defined by voxels with value matching `label`).

Finally, checks if the mask volume contains an ROI identified by `label`. Raises a value error if the label is not present (including a list of valid labels found).

**Parameters**

- **mask** – SimpleITK Image object representing the mask. Can be a vector image to allow for overlapping masks.
- **kwargs** – keyword arguments. If argument `label_channel` is present, this is used to select the channel. Otherwise label_channel `0` is assumed.

**Returns** SimpleITK.Image with pixel type UInt32 representing the mask volume

radiomics.imageoperations.**getBinEdges**(*parameterValues*, ***kwargs*)

Calculate and return the histogram using parameterValues (1D array of all segmented voxels in the image).

**Fixed bin width:**

Returns the bin edges, a list of the edges of the calculated bins, length is N(bins) + 1. Bins are defined such, that the bin edges are equally spaced from zero, and that the leftmost edge $\leq \min(X_{gl})$. These bin edges represent the half-open ranges of each bin [

radiomics.imageoperations.**binImage**(*parameterMatrix*, *parameterMatrixCoordinates=None*, ***kwargs*)

Discretizes the parameterMatrix (matrix representation of the gray levels in the ROI) using the binEdges calculated using *getBinEdges()*. Only voxels defined by parameterMatrixCoordinates (defining the segmentation) are used for calculation of histogram and subsequently discretized. Voxels outside segmentation are left unchanged.

radiomics.imageoperations.**checkMask**(*imageNode*, *maskNode*, ***kwargs*)

Checks whether the Region of Interest (ROI) defined in the mask size and dimensions match constraints, specified in settings. The following checks are performed.

1. Check whether the mask corresponds to the image (i.e. has a similar size, spacing, direction and origin). **N.B. This check is performed by SimpleITK, if it fails, an error is logged, with additional error information from SimpleITK logged with level DEBUG (i.e. logging-level has to be set to debug to store this information in the log file).** The tolerance can be increased using the `geometryTolerance` parameter. Alternatively, if the `correctMask` parameter is `True`, PyRadiomics will check if the mask contains a valid ROI (inside image physical area) and if so, resample the mask to image geometry. See *Settings* for more info.
2. Check if the label is present in the mask
3. Count the number of dimensions in which the size of the ROI > 1 (i.e. does the ROI represent a single voxel (0), a line (1), a surface (2) or a volume (3)) and compare this to the minimum number of dimension required (specified in `minimumROIDimensions`).
4. Optional. Check if there are at least N voxels in the ROI. N is defined in `minimumROISize`, this test is skipped if `minimumROISize = None`.

This function returns a tuple of two items. The first item is the bounding box of the mask. The second item is the mask that has been corrected by resampling to the input image geometry (if that resampling was successful).

If a check fails, a ValueError is raised. No features will be extracted for this mask. If the mask passes all tests, this function returns the bounding box, which is used in the `cropToTumorMask()` function.

The bounding box is calculated during (1.) and used for the subsequent checks. The bounding box is calculated by SimpleITK.LabelStatisticsImageFilter() and returned as a tuple of indices: (L_x, U_x, L_y, U_y, L_z, U_z), where 'L' and 'U' are lower and upper bound, respectively, and 'x', 'y' and 'z' the three image dimensions.

By reusing the bounding box calculated here, calls to SimpleITK.LabelStatisticsImageFilter() are reduced, improving performance.

Uses the following settings:

- minimumROIDimensions [1]: Integer, range 1-3, specifies the minimum dimensions (1D, 2D or 3D, respectively). Single-voxel segmentations are always excluded.
- minimumROISize [None]: Integer, > 0, specifies the minimum number of voxels required. Test is skipped if this parameter is set to None.

**Note:** If the first check fails there are generally 2 possible causes:

1. The image and mask are matched, but there is a slight difference in origin, direction or spacing. The exact cause, difference and used tolerance are stored with level DEBUG in a log (if enabled). For more information on setting up logging, see "*setting up logging*" and the helloRadiomics examples (located in the `pyradiomics/examples` folder). This problem can be fixed by changing the global tolerance (`geometryTolerance` parameter) or enabling mask correction (`correctMask` parameter).
2. The image and mask do not match, but the ROI contained within the mask does represent a physical volume contained within the image. If this is the case, resampling is needed to ensure matching geometry between image and mask before features can be extracted. This can be achieved by enabling mask correction using the `correctMask` parameter.

`radiomics.imageoperations.`**`cropToTumorMask`**(*imageNode*, *maskNode*, *boundingBox*, ***kwargs*)

Create a sitkImage of the segmented region of the image based on the input label.

Create a sitkImage of the labelled region of the image, cropped to have a cuboid shape equal to the ijk boundaries of the label.

**Parameters**

- **`boundingBox`** – The bounding box used to crop the image. This is the bounding box as returned by `checkMask()`.
- **`label`** – [1], value of the label, onto which the image and mask must be cropped.

**Returns** Cropped image and mask (SimpleITK image instances).

`radiomics.imageoperations.`**`resampleImage`**(*imageNode*, *maskNode*, ***kwargs*)

Resamples image and mask to the specified pixel spacing (The default interpolator is Bspline).

Resampling can be enabled using the settings 'interpolator' and 'resampledPixelSpacing' in the parameter file or as part of the settings passed to the feature extractor. See also *feature extractor*.

'imageNode' and 'maskNode' are SimpleITK Objects, and 'resampledPixelSpacing' is the output pixel spacing (sequence of 3 elements).

If only in-plane resampling is required, set the output pixel spacing for the out-of-plane dimension (usually the last dimension) to 0. Spacings with a value of 0 are replaced by the spacing as it is in the original mask.

Only part of the image and labelmap are resampled. The resampling grid is aligned to the input origin, but only voxels covering the area of the image ROI (defined by the bounding box) and the padDistance are resampled. This results in a resampled and partially cropped image and mask. Additional padding is required as some filters also sample voxels outside of segmentation boundaries. For feature calculation, image and mask are cropped to the bounding box without any additional padding, as the feature classes do not need the gray level values outside the segmentation.

The resampling grid is calculated using only the input mask. Even when image and mask have different directions, both the cropped image and mask will have the same direction (equal to direction of the mask). Spacing and size are determined by settings and bounding box of the ROI.

**Note:** Before resampling the bounds of the non-padded ROI are compared to the bounds. If the ROI bounding box includes areas outside of the physical space of the image, an error is logged and (None, None) is returned. No features will be extracted. This enables the input image and mask to have different geometry, so long as the ROI defines an area within the image.

**Note:** The additional padding is adjusted, so that only the physical space within the mask is resampled. This is done to prevent resampling outside of the image. Please note that this assumes the image and mask to image the same physical space. If this is not the case, it is possible that voxels outside the image are included in the resampling grid, these will be assigned a value of 0. It is therefore recommended, but not enforced, to use an input mask which has the same or a smaller physical space than the image.

`radiomics.imageoperations.`**`normalizeImage`**`(`*image*, ***kwargs*`)`

Normalizes the image by centering it at the mean with standard deviation. Normalization is based on all gray values in the image, not just those inside the segmentation.

$f(x) = \frac{s(x-\mu_x)}{\sigma_x}$

Where:

- $x$ and $f(x)$ are the original and normalized intensity, respectively.
- $\mu_x$ and $\sigma_x$ are the mean and standard deviation of the image instensity values.
- $s$ is an optional scaling defined by `scale`. By default, it is set to 1.

Optionally, outliers can be removed, in which case values for which $x > \mu_x + n\sigma_x$ or $x < \mu_x - n\sigma_x$ are set to $\mu_x + n\sigma_x$ and $\mu_x - n\sigma_x$, respectively. Here, $n > 0$ and defined by `outliers`. This, in turn, is controlled by the `removeOutliers` parameter. Removal of outliers is done after the values of the image are normalized, but before `scale` is applied.

`radiomics.imageoperations.`**`resegmentMask`**`(`*imageNode*, *maskNode*, ***kwargs*`)`

Resegment the Mask based on the range specified by the threshold(s) in `resegmentRange`. Either 1 or 2 thresholds can be defined. In case of 1 threshold, all values equal to or higher than that threshold are included. If there are 2 thresholds, all voxels with a value inside the closed-range defined by these thresholds is included (i.e. a voxels is included if $T_{lower} \leq X_gl \leq T_{upper}$). The resegmented mask is therefore always equal or smaller in size than the original mask. In the case where either resegmentRange or resegmentMode contains illigal values, a ValueError is raised.

There are 3 modes for defining the threshold:

1. absolute (default): The values in resegmentRange define as absolute values (i.e. corresponding to the gray values in the image
2. relative: The values in resegmentRange define the threshold as relative to the maximum value found in the ROI. (e.g. 0.5 indicates a threshold at 50% of maximum gray value)

3. sigma: The threshold is defined as the number of sigma from the mean. (e.g. resegmentRange [-3, 3] will include all voxels that have a value that differs 3 or less standard deviations from the mean).

`radiomics.imageoperations.`**`getOriginalImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

This function does not apply any filter, but returns the original image. This function is needed to dynamically expose the original image as a valid image type.

**Returns** Yields original image, 'original' and `kwargs`

`radiomics.imageoperations.`**`getLoGImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Applies a Laplacian of Gaussian filter to the input image and yields a derived image for each sigma value specified.

A Laplacian of Gaussian image is obtained by convolving the image with the second derivative (Laplacian) of a Gaussian kernel.

The Gaussian kernel is used to smooth the image and is defined as

$$G(x, y, z, \sigma) = \frac{1}{(\sigma \sqrt{2 \pi})^3}e^{-\frac{x^2 + y^2 + z^2}{2\sigma^2}}$$

The Gaussian kernel is convolved by the laplacian kernel $\nabla^2G(x, y, z)$, which is sensitive to areas with rapidly changing intensities, enhancing edges. The width of the filter in the Gaussian kernel is determined by $\sigma$ and can be used to emphasize more fine (low $\sigma$ values) or coarse (high $\sigma$ values) textures.

**Warning:** The LoG filter implemented in PyRadiomics is a 3D LoG filter, and therefore requires 3D input. Features using a single slice (2D) segmentation can still be extracted, but the input image *must* be a 3D image, with a minimum size in all dimensions $\geq \sigma$. If input image is too small, a warning is logged and $\sigma$ value is skipped. Moreover, the image size *must* be at least 4 voxels in each dimensions, if this constraint is not met, no LoG derived images can be generated.

Following settings are possible:

- sigma: List of floats or integers, must be greater than 0. Filter width (mm) to use for the Gaussian kernel (determines coarseness).

**Warning:** Setting for sigma must be provided. If omitted, no LoG image features are calculated and the function will return an empty dictionary.

Returned filter name reflects LoG settings: log-sigma-<sigmaValue>-3D.

References:

- SimpleITK Doxygen documentation
- ITK Doxygen documentation
- https://en.wikipedia.org/wiki/Blob_detection#The_Laplacian_of_Gaussian

**Returns** Yields log filtered image for each specified sigma, corresponding image type name and `kwargs` (customized settings).

`radiomics.imageoperations.`**`getWaveletImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Applies wavelet filter to the input image and yields the decompositions and the approximation.

Following settings are possible:

- start_level [0]: integer, 0 based level of wavelet which should be used as first set of decompositions from which a signature is calculated
- level [1]: integer, number of levels of wavelet decompositions from which a signature is calculated.
- wavelet ["coif1"]: string, type of wavelet decomposition. Enumerated value, validated against possible values present in the `pyWavelet.wavelist()`. Current possible values (pywavelet version 0.4.0) (where an aditional number is needed, range of values is indicated in []):
  - haar
  - dmey
  - sym[2-20]
  - db[1-20]
  - coif[1-5]
  - bior[1.1, 1.3, 1.5, 2.2, 2.4, 2.6, 2.8, 3.1, 3.3, 3.5, 3.7, 3.9, 4.4, 5.5, 6.8]
  - rbio[1.1, 1.3, 1.5, 2.2, 2.4, 2.6, 2.8, 3.1, 3.3, 3.5, 3.7, 3.9, 4.4, 5.5, 6.8]

Returned filter name reflects wavelet type: wavelet[level]-<decompositionName>

N.B. only levels greater than the first level are entered into the name.

**Returns** Yields each wavelet decomposition and final approximation, corresponding imaget type name and `kwargs` (customized settings).

`radiomics.imageoperations.`**`getSquareImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Computes the square of the image intensities.

Resulting values are rescaled on the range of the initial original image and negative intensities are made negative in resultant filtered image.

$$f(x) = (cx)^2,\text{ where } c = \frac{1}{\sqrt{\max(|x|)}}$$

Where $x$ and $f(x)$ are the original and filtered intensity, respectively.

**Returns** Yields square filtered image, 'square' and `kwargs` (customized settings).

`radiomics.imageoperations.`**`getSquareRootImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Computes the square root of the absolute value of image intensities.

Resulting values are rescaled on the range of the initial original image and negative intensities are made negative in resultant filtered image.

$$f(x) = \left\{ {\begin{array}{lcl} \sqrt{cx} & \mbox{for} & x \geq 0 \\ -\sqrt{-cx} & \mbox{for} & x < 0 \end{array}} \right.,\text{ where } c = \max(|x|)$$

Where $x$ and $f(x)$ are the original and filtered intensity, respectively.

**Returns** Yields square root filtered image, 'squareroot' and `kwargs` (customized settings).

`radiomics.imageoperations.`**`getLogarithmImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Computes the logarithm of the absolute value of the original image + 1.

Resulting values are rescaled on the range of the initial original image and negative intensities are made negative in resultant filtered image.

$$f(x) = \left\{ {\begin{array}{lcl} c\log{(x + 1)} & \mbox{for} & x \geq 0 \\ -c\log{(-x + 1)} & \mbox{for} & x < 0 \end{array}} \right.,\text{ where } c = \frac{\max(|x|)}{\log(\max(|x|) + 1)}$$

Where $x$ and $f(x)$ are the original and filtered intensity, respectively.

**Returns** Yields logarithm filtered image, 'logarithm' and `kwargs` (customized settings)

`radiomics.imageoperations.`**`getExponentialImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Computes the exponential of the original image.

Resulting values are rescaled on the range of the initial original image.

$$f(x) = e^{cx}, \text{ where } c = \frac{\log(\max(|x|))}{\max(|x|)}$$

Where $x$ and $f(x)$ are the original and filtered intensity, respectively.

**Returns** Yields exponential filtered image, 'exponential' and `kwargs` (customized settings)

`radiomics.imageoperations.`**`getGradientImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Compute and return the Gradient Magnitude in the image. By default, takes into account the image spacing, this can be switched off by specifying `gradientUseSpacing = False`.

References:

- SimpleITK documentation
- https://en.wikipedia.org/wiki/Image_gradient

`radiomics.imageoperations.`**`getLBP2DImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Compute and return the Local Binary Pattern (LBP) in 2D. If `force2D` is set to false (= feature extraction in 3D) a warning is logged, as this filter processes the image in a by-slice operation. The plane in which the LBP is applied can be controlled by the `force2Ddimension` parameter (see also `generateAngles()`).

Following settings are possible (in addition to `force2Ddimension`):

- `lbp2DRadius` [1]: Float, specifies the radius in which the neighbours should be sampled
- `lbp2DSamples` [9]: Integer, specifies the number of samples to use
- `lbp2DMethod` ['uniform']: String, specifies the method for computing the LBP to use.

For more information see scikit documentation

**Returns** Yields LBP filtered image, 'lbp-2D' and `kwargs` (customized settings)

**Note:** LBP can often return only a very small number of different gray levels. A customized bin width is often needed.

**Warning:** Requires package `scikit-image` to function. If not available, this filter logs a warning and does not yield an image.

References:

- T. Ojala, M. Pietikainen, and D. Harwood (1994), "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions", Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR 1994), vol. 1, pp. 582 - 585.
- T. Ojala, M. Pietikainen, and D. Harwood (1996), "A Comparative Study of Texture Measures with Classification Based on Feature Distributions", Pattern Recognition, vol. 29, pp. 51-59.

`radiomics.imageoperations.`**`getLBP3DImage`**`(`*inputImage*, *inputMask*, ***kwargs*`)`

Compute and return the Local Binary Pattern (LBP) in 3D using spherical harmonics. If `force2D` is set to true (= feature extraction in 2D) a warning is logged.

LBP is only calculated for voxels segmented in the mask

Following settings are possible:

- `lbp3DLevels` [2]: integer, specifies the the number of levels in spherical harmonics to use.
- `lbp3DIcosphereRadius` [1]: Float, specifies the radius in which the neighbours should be sampled
- `lbp3DIcosphereSubdivision` [1]: Integer, specifies the number of subdivisions to apply in the icosphere

**Returns** Yields LBP filtered image for each level, 'lbp-3D-m<level>' and `kwargs` (customized settings). Additionally yields the kurtosis image, 'lbp-3D-k' and `kwargs`.

**Note:** LBP can often return only a very small number of different gray levels. A customized bin width is often needed.

**Warning:** Requires package `scipy` and `trimesh` to function. If not available, this filter logs a warning and does not yield an image.

References:

- Banerjee, J, Moelker, A, Niessen, W.J, & van Walsum, T.W. (2013), "3D LBP-based rotationally invariant region description." In: Park JI., Kim J. (eds) Computer Vision - ACCV 2012 Workshops. ACCV 2012. Lecture Notes in Computer Science, vol 7728. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-37410-4_3

### 2.4.3 General Info Module

**class** `radiomics.generalinfo.`**`GeneralInfo`**

Bases: `object`

**`getGeneralInfo`**`()`

Return a dictionary containing all general info items. Format is <info_item>:<value>, where the type of the value is preserved. For CSV format, this will result in conversion to string and quotes where necessary, for JSON, the values will be interpreted and stored as JSON strings.

**`addStaticElements`**`()`

Adds the following elements to the general info:

- Version: current version of PyRadiomics
- NumpyVersion: version of numpy used
- SimpleITKVersion: version SimpleITK used
- PyWaveletVersion: version of PyWavelet used
- PythonVersion: version of the python interpreter running PyRadiomics

**`addImageElements`**(*image*, *prefix='original'*)

Calculates provenance info for the image

Adds the following:

- Hash: sha1 hash of the mask, which can be used to check if the same mask was used during reproducibility tests. (Only added when prefix is "original")

- Dimensionality: Number of dimensions (e.g. 2D, 3D) in the image. (Only added when prefix is “original”)
- Spacing: Pixel spacing (x, y, z) in mm.
- Size: Dimensions (x, y, z) of the image in number of voxels.
- Mean: Mean intensity value over all voxels in the image.
- Minimum: Minimum intensity value among all voxels in the image.
- Maximum: Maximum intensity value among all voxels in the image.

A prefix is added to indicate what type of image is described:

- original: Image as loaded, without pre-processing.
- interpolated: Image after it has been resampled to a new spacing (includes cropping).

**`addMaskElements`**(*image*, *mask*, *label*, *prefix='original'*)
Calculates provenance info for the mask

Adds the following:

- MaskHash: sha1 hash of the mask, which can be used to check if the same mask was used during reproducibility tests. (Only added when prefix is “original”)
- BoundingBox: bounding box of the ROI defined by the specified label: Elements 0, 1 and 2 are the x, y and z coordinates of the lower bound, respectively. Elements 3, 4 and 5 are the size of the bounding box in x, y and z direction, respectively.
- VoxelNum: Number of voxels included in the ROI defined by the specified label.
- VolumeNum: Number of fully connected (26-connectivity) volumes in the ROI defined by the specified label.
- CenterOfMassIndex: x, y and z coordinates of the center of mass of the ROI in terms of the image coordinate space (continuous index).
- CenterOfMass: the real-world x, y and z coordinates of the center of mass of the ROI
- ROIMean: Mean intensity value over all voxels in the ROI defined by the specified label.
- ROIMinimum: Minimum intensity value among all voxels in the ROI defined by the specified label.
- ROIMaximum: Maximum intensity value among all voxels in the ROI defined by the specified label.

A prefix is added to indicate what type of mask is described:

- original: Mask as loaded, without pre-processing.
- corrected: Mask after it has been corrected by `imageoperations.checkMask()`.
- interpolated: Mask after it has been resampled to a new spacing (includes cropping).
- resegmented: Mask after resegmentation has been applied.

**`addGeneralSettings`**(*settings*)
Add a string representation of the general settings. Format is {<settings_name>:<value>, . . . }.

**`addEnabledImageTypes`**(*enabledImageTypes*)
Add a string representation of the enabled image types and any custom settings for each image type. Format is {<imageType_name>:{<setting_name>:<value>, . . . }, . . . }.

### 2.4.4 Feature Class Base

**class** radiomics.base.**RadiomicsFeaturesBase**(*inputImage*, *inputMask*, ***kwargs*)

Bases: `object`

This is the abstract class, which defines the common interface for the feature classes. All feature classes inherit (directly of indirectly) from this class.

At initialization, image and labelmap are passed as SimpleITK image objects (`inputImage` and `inputMask`, respectively.) The motivation for using SimpleITK images as input is to keep the possibility of reusing the optimized feature calculators implemented in SimpleITK in the future. If either the image or the mask is None, initialization fails and a warning is logged (does not raise an error).

Logging is set up using a child logger from the parent 'radiomics' logger. This retains the toolbox structure in the generated log. The child logger is named after the module containing the feature class (e.g. 'radiomics.glcm').

Any pre calculations needed before the feature functions are called can be added by overriding the `_initSegmentBasedCalculation` function, which prepares the input for feature extraction. If image discretization is needed, this can be implemented by adding a call to `_applyBinning` to this initialization function, which also instantiates coefficients holding the maximum ('Ng') and unique ('GrayLevels') that can be found inside the ROI after binning. This function also instantiates the *matrix* variable, which holds the discretized image (the *imageArray* variable will hold only original gray levels).

The following variables are instantiated at initialization:

- kwargs: dictionary holding all customized settings passed to this feature class.
- label: label value of Region of Interest (ROI) in labelmap. If key is not present, a default value of 1 is used.
- featureNames: list containing the names of features defined in the feature class. See `getFeatureNames()`
- inputImage: SimpleITK image object of the input image (dimensions x, y, z)

The following variables are instantiated by the `_initSegmentBasedCalculation` function:

- inputMask: SimpleITK image object of the input labelmap (dimensions x, y, z)
- imageArray: numpy array of the gray values in the input image (dimensions z, y, x)
- maskArray: numpy boolean array with elements set to `True` where labelmap = label, `False` otherwise, (dimensions z, y, x).
- labelledVoxelCoordinates: tuple of 3 numpy arrays containing the z, x and y coordinates of the voxels included in the ROI, respectively. Length of each array is equal to total number of voxels inside ROI.
- matrix: copy of the imageArray variable, with gray values inside ROI discretized using the specified binWidth. This variable is only instantiated if a call to `_applyBinning` is added to an override of `_initSegmentBasedCalculation` in the feature class.

**Note:** Although some variables listed here have similar names to customization settings, they do *not* represent all the possible settings on the feature class level. These variables are listed here to help developers develop new feature classes, which make use of these variables. For more information on customization, see *Customizing the Extraction*, which includes a comprehensive list of all possible settings, including default values and explanation of usage.

**enableFeatureByName**(*featureName*, *enable=True*)

Enables or disables feature specified by `featureName`. If feature is not present in this class, a lookup error is raised. `enable` specifies whether to enable or disable the feature.

**enableAllFeatures**()

Enables all features found in this class for calculation.

**Note:** Features that have been marked "deprecated" are not enabled by this function. They can still be enabled manually by a call to `enableFeatureByName()`, `enableFeaturesByName()` or in the parameter file (by specifying the feature by name, not when enabling all features). However, in most cases this will still result only in a deprecation warning.

**disableAllFeatures**()

Disables all features. Additionally resets any calculated features.

**classmethod getFeatureNames**()

Dynamically enumerates features defined in the feature class. Features are identified by the `get<Feature>FeatureValue` signature, where <Feature> is the name of the feature (unique on the class level).

Found features are returned as a dictionary of the feature names, where the value `True` if the feature is deprecated, `False` otherwise (`{<Feature1>:<deprecated>, <Feature2>:<deprecated>, ...}`).

This function is called at initialization, found features are stored in the `featureNames` variable.

**execute**()

Calculates all features enabled in `enabledFeatures`. A feature is enabled if it's key is present in this dictionary and it's value is True.

Calculated values are stored in the `featureValues` dictionary, with feature name as key and the calculated feature value as value. If an exception is thrown during calculation, the error is logged, and the value is set to NaN.

### 2.4.5 Global Toolbox Functions

`radiomics.`**deprecated**(*func*)

Decorator function to mark functions as deprecated. This is used to ensure deprecated feature functions are not added to the enabled features list when enabling 'all' features.

`radiomics.`**getFeatureClasses**()

Iterates over all modules of the radiomics package using pkgutil and subsequently imports those modules.

Return a dictionary of all modules containing featureClasses, with modulename as key, abstract class object of the featureClass as value. Assumes only one featureClass per module

This is achieved by inspect.getmembers. Modules are added if it contains a member that is a class, with name starting with 'Radiomics' and is inherited from *`radiomics.base.RadiomicsFeaturesBase`*.

This iteration only runs once (at initialization of toolbox), subsequent calls return the dictionary created by the first call.

`radiomics.`**getImageTypes**()

Returns a list of possible image types (i.e. the possible filters and the "Original", unfiltered image type). This function finds the image types dynamically by matching the signature ("get<imageType>Image") against functions defined in *imageoperations*. Returns a list containing available image type names (<imageType> part of the corresponding function name).

This iteration only occurs once, at initialization of the toolbox. Found results are stored and returned on subsequent calls.

radiomics.**getParameterValidationFiles**()

Returns file locations for the parameter schema and custom validation functions, which are needed when validating a parameter file using `PyKwalify.core`. This functions returns a tuple with the file location of the schema as first and python script with custom validation functions as second element.

radiomics.**getProgressReporter**(**args*, ***kwargs*)

This function returns an instance of the progressReporter, if it is set and the logging level is defined at level INFO or DEBUG. In all other cases a dummy progress reporter is returned.

To enable progress reporting, the progressReporter variable should be set to a class object (NOT an instance), which fits the following signature:

1. Accepts an iterable as the first positional argument and a keyword argument ('desc') specifying a label to display
2. Can be used in a 'with' statement (i.e. exposes a __enter__ and __exit__ function)
3. Is iterable (i.e. at least specifies an __iter__ function, which iterates over the iterable passed at initialization).

It is also possible to create your own progress reporter. To achieve this, additionally specify a function *__next__*, and have the *__iter__* function return *self*. The *__next__* function takes no arguments and returns a call to the *__next__* function of the iterable (i.e. *return self.iterable.__next__()*). Any prints/progress reporting calls can then be inserted in this function prior to the return statement.

radiomics.**getTestCase**(*testCase*, *dataDirectory=None*)

This function provides an image and mask for testing PyRadiomics. One of seven test cases can be selected:

- brain1
- brain2
- breast1
- lung1
- lung2
- test_wavelet_64x64x64
- test_wavelet_37x37x37

Checks if the test case (consisting of an image and mask file with signature <testCase>_image.nrrd and <testCase>_label.nrrd, respectively) is available in the `dataDirectory`. If not available, the testCase is downloaded from the GitHub repository and stored in the `dataDirectory`. Also creates the `dataDirectory` if necessary. If no `dataDirectory` has been specified, PyRadiomics will use a temporary directory: <TEMPDIR>/pyradiomics/data.

If the test case has been found or downloaded successfully, this function returns a tuple of two strings: `(path/to/image.nrrd, path/to/mask.nrrd)`. In case of an error `(None, None)` is returned.

**Note:** To get the testcase with the corresponding single-slice label, append "_2D" to the testCase.

radiomics.**setVerbosity**(*level*)

Change the amount of information PyRadiomics should print out during extraction. The lower the level, the more information is printed to the output (stderr).

Using the `level` (Python defined logging levels) argument, the following levels are possible:

- 60: Quiet mode, no messages are printed to the stderr
- 50: Only log messages of level "CRITICAL" are printed

- 40: Log messages of level "ERROR" and up are printed
- 30: Log messages of level "WARNING" and up are printed
- 20: Log messages of level "INFO" and up are printed
- 10: Log messages of level "DEBUG" and up are printed (i.e. all log messages)

By default, the radiomics logger is set to level "INFO" and the stderr handler to level "WARNING". Therefore a log storing the extraction log messages from level "INFO" and up can be easily set up by adding an appropriate handler to the radiomics logger, while the output to stderr will still only contain warnings and errors.

**Note:** This function assumes the handler added to the radiomics logger at initialization of the toolbox is not removed from the logger handlers and therefore remains the first handler.

**Note:** This does not affect the level of the logger itself (e.g. if verbosity level = 3, log messages with DEBUG level can still be stored in a log file if an appropriate handler is added to the logger and the logging level of the logger has been set to the correct level. *Exception: In case the verbosity is set to DEBUG, the level of the logger is also lowered to DEBUG. If the verbosity level is then raised again, the logger level will remain DEBUG.*

## 2.5 Radiomic Features

This section contains the definitions of the various features that can be extracted using PyRadiomics. They are subdivided into the following classes:

- *First Order Statistics* (19 features)
- *Shape-based (3D)* (16 features)
- *Shape-based (2D)* (10 features)
- *Gray Level Cooccurence Matrix* (24 features)
- *Gray Level Run Length Matrix* (16 features)
- *Gray Level Size Zone Matrix* (16 features)
- *Neighbouring Gray Tone Difference Matrix* (5 features)
- *Gray Level Dependence Matrix* (14 features)

All feature classes, with the exception of shape can be calculated on either the original image and/or a derived image, obtained by applying one of several filters. The shape descriptors are independent of gray value, and are extracted from the label mask. If enabled, they are calculated separately of enabled input image types, and listed in the result as if calculated on the original image.

Most features defined below are in compliance with feature definitions as described by the Imaging Biomarker Standardization Initiative (IBSI), which are available in a separate document by Zwanenburg et al. (2016)[1]. Where features differ, a note has been added specifying the difference.

[1] Zwanenburg, A., Leger, S., Vallières, M., and Löck, S. (2016). Image biomarker standardisation initiative - feature definitions. In eprint arXiv:1612.07003 [cs.CV]

### 2.5.1 First Order Features

**class** radiomics.firstorder.**RadiomicsFirstOrder**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

First-order statistics describe the distribution of voxel intensities within the image region defined by the mask through commonly used and basic metrics.

Let:

- **X** be a set of $N_p$ voxels included in the ROI
- $\mathbf{P}(i)$ be the first order histogram with $N_g$ discrete intensity levels, where $N_g$ is the number of non-zero bins, equally spaced from 0 with a width defined in the `binWidth` parameter.
- $p(i)$ be the normalized first order histogram and equal to $\frac{\mathbf{P}(i)}{N_p}$

Following additional settings are possible:

- voxelArrayShift [0]: Integer, This amount is added to the gray level intensity in features Energy, Total Energy and RMS, this is to prevent negative values. *If using CT data, or data normalized with mean 0, consider setting this parameter to a fixed value (e.g. 2000) that ensures non-negative numbers in the image. Bear in mind however, that the larger the value, the larger the volume confounding effect will be.*

**Note:** In the IBSI feature definitions, no correction for negative gray values is implemented. To achieve similar behaviour in PyRadiomics, set `voxelArrayShift` to 0.

**getEnergyFeatureValue**()

**1. Energy**

$$\textit{energy} = \sum_{i=1}^{N_p} (\mathbf{X}(i) + c)^2$$

Here, $c$ is optional value, defined by `voxelArrayShift`, which shifts the intensities to prevent negative values in **X**. This ensures that voxels with the lowest gray values contribute the least to Energy, instead of voxels with gray level intensity closest to 0.

Energy is a measure of the magnitude of voxel values in an image. A larger values implies a greater sum of the squares of these values.

**Note:** This feature is volume-confounded, a larger value of $c$ increases the effect of volume-confounding.

**getTotalEnergyFeatureValue**()

**2. Total Energy**

$$\textit{total energy} = V_{voxel} \sum_{i=1}^{N_p} (\mathbf{X}(i) + c)^2$$

Here, $c$ is optional value, defined by `voxelArrayShift`, which shifts the intensities to prevent negative values in **X**. This ensures that voxels with the lowest gray values contribute the least to Energy, instead of voxels with gray level intensity closest to 0.

Total Energy is the value of Energy feature scaled by the volume of the voxel in cubic mm.

**Note:** This feature is volume-confounded, a larger value of $c$ increases the effect of volume-confounding.

**Note:** Not present in IBSI feature definitions

`getEntropyFeatureValue()`

**3. Entropy**

$$\textit{entropy} = -\sum_{i=1}^{N_g} p(i) \log_2 \big(p(i) + \epsilon\big)$$

Here, $\epsilon$ is an arbitrarily small positive number ($\approx 2.2 \times 10^{-16}$).

Entropy specifies the uncertainty/randomness in the image values. It measures the average amount of information required to encode the image values.

**Note:** Defined by IBSI as Intensity Histogram Entropy.

`getMinimumFeatureValue()`

**4. Minimum**

$$\textit{minimum} = \min(\mathbf{X})$$

`get10PercentileFeatureValue()`

**5. 10th percentile**

The 10th percentile of **X**

`get90PercentileFeatureValue()`

**6. 90th percentile**

The 90th percentile of **X**

`getMaximumFeatureValue()`

**7. Maximum**

$$\textit{maximum} = \max(\mathbf{X})$$

The maximum gray level intensity within the ROI.

`getMeanFeatureValue()`

**8. Mean**

$$\textit{mean} = \frac{1}{N_p}\sum_{i=1}^{N_p} \mathbf{X}(i)$$

The average gray level intensity within the ROI.

`getMedianFeatureValue()`

**9. Median**

The median gray level intensity within the ROI.

`getInterquartileRangeFeatureValue()`

**10. Interquartile Range**

$$\textit{interquartile range} = \mathbf{P}_{75} - \mathbf{P}_{25}$$

Here $\mathbf{P}_{25}$ and $\mathbf{P}_{75}$ are the 25th and 75th percentile of the image array, respectively.

`getRangeFeatureValue()`

**11. Range**

$$\textit{range} = \max(\mathbf{X}) - \min(\mathbf{X})$$

The range of gray values in the ROI.

`getMeanAbsoluteDeviationFeatureValue()`

**12. Mean Absolute Deviation (MAD)**

$$\textit{MAD} = \frac{1}{N_p}\sum^{N_p}_{i=1}{|\mathbf{X}(i)-\bar{X}|}$$

Mean Absolute Deviation is the mean distance of all intensity values from the Mean Value of the image array.

`getRobustMeanAbsoluteDeviationFeatureValue()`

**13. Robust Mean Absolute Deviation (rMAD)**

$$\textit{rMAD} = \frac{1}{N_{10-90}}\sum^{N_{10-90}}_{i=1}{|\mathbf{X}_{10-90}(i)-\bar{X}_{10-90}|}$$

Robust Mean Absolute Deviation is the mean distance of all intensity values from the Mean Value calculated on the subset of image array with gray levels in between, or equal to the $10^{\text{th}}$ and $90^{\text{th}}$ percentile.

`getRootMeanSquaredFeatureValue()`

**14. Root Mean Squared (RMS)**

$$\textit{RMS} = \sqrt{\frac{1}{N_p}\sum^{N_p}_{i=1}{(\mathbf{X}(i) + c)^2}}$$

Here, $c$ is optional value, defined by `voxelArrayShift`, which shifts the intensities to prevent negative values in $\mathbf{X}$. This ensures that voxels with the lowest gray values contribute the least to RMS, instead of voxels with gray level intensity closest to 0.

RMS is the square-root of the mean of all the squared intensity values. It is another measure of the magnitude of the image values. This feature is volume-confounded, a larger value of $c$ increases the effect of volume-confounding.

`getStandardDeviationFeatureValue()`

**15. Standard Deviation**

$$\textit{standard deviation} = \sqrt{\frac{1}{N_p}\sum^{N_p}_{i=1}{(\mathbf{X}(i)-\bar{X})^2}}$$

Standard Deviation measures the amount of variation or dispersion from the Mean Value. By definition, $\textit{standard deviation} = \sqrt{\textit{variance}}$

**Note:** As this feature is correlated with variance, it is marked so it is not enabled by default. To include this feature in the extraction, specify it by name in the enabled features (i.e. this feature will not be enabled if no individual features are specified (enabling 'all' features), but will be enabled when individual features are specified, including this feature). Not present in IBSI feature definitions (correlated with variance)

**getSkewnessFeatureValue**()

**16. Skewness**

$$\textit{skewness} = \frac{\mu_3}{\sigma^3} = \frac{\frac{1}{N_p}\sum_{i=1}^{N_p}{(\textbf{X}(i)-\bar{X})^3}}{\left(\sqrt{\frac{1}{N_p}\sum_{i=1}^{N_p}{(\textbf{X}(i)-\bar{X})^2}}\right)^3}$$

Where $\mu_3$ is the 3rd central moment.

Skewness measures the asymmetry of the distribution of values about the Mean value. Depending on where the tail is elongated and the mass of the distribution is concentrated, this value can be positive or negative.

Related links:

https://en.wikipedia.org/wiki/Skewness

**Note:** In case of a flat region, the standard deviation and 4rd central moment will be both 0. In this case, a value of 0 is returned.

**getKurtosisFeatureValue**()

**17. Kurtosis**

$$\textit{kurtosis} = \frac{\mu_4}{\sigma^4} = \frac{\frac{1}{N_p}\sum_{i=1}^{N_p}{(\textbf{X}(i)-\bar{X})^4}}{\left(\frac{1}{N_p}\sum_{i=1}^{N_p}{(\textbf{X}(i)-\bar{X})^2}\right)^2}$$

Where $\mu_4$ is the 4th central moment.

Kurtosis is a measure of the 'peakedness' of the distribution of values in the image ROI. A higher kurtosis implies that the mass of the distribution is concentrated towards the tail(s) rather than towards the mean. A lower kurtosis implies the reverse: that the mass of the distribution is concentrated towards a spike near the Mean value.

Related links:

https://en.wikipedia.org/wiki/Kurtosis

**Note:** In case of a flat region, the standard deviation and 4rd central moment will be both 0. In this case, a value of 0 is returned.

**Note:** The IBSI feature definition implements excess kurtosis, where kurtosis is corrected by -3, yielding 0 for normal distributions. The PyRadiomics kurtosis is not corrected, yielding a value 3 higher than the IBSI kurtosis.

**getVarianceFeatureValue**()

**18. Variance**

$$\textit{variance} = \frac{1}{N_p}\displaystyle\sum^{N_p}_{i=1}{(\textbf{X}(i)-\bar{X})^2}$$

Variance is the the mean of the squared distances of each intensity value from the Mean value. This is a measure of the spread of the distribution about the mean. By definition, $\textit{variance} = \sigma^2$

**getUniformityFeatureValue**()

**19. Uniformity**

$$\textit{uniformity} = \sum_{i=1}^{N_g} p(i)^2$$

Uniformity is a measure of the sum of the squares of each intensity value. This is a measure of the homogeneity of the image array, where a greater uniformity implies a greater homogeneity or a smaller range of discrete intensity values.

**Note:** Defined by IBSI as Intensity Histogram Uniformity.

### 2.5.2 Shape Features (3D)

**class** radiomics.shape.**RadiomicsShape**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

In this group of features we included descriptors of the three-dimensional size and shape of the ROI. These features are independent from the gray level intensity distribution in the ROI and are therefore only calculated on the non-derived image and mask.

Unless otherwise specified, features are derived from the approximated shape defined by the triangle mesh. To build this mesh, vertices (points) are first defined as points halfway on an edge between a voxel included in the ROI and one outside the ROI. By connecting these vertices a mesh of connected triangles is obtained, with each triangle defined by 3 adjacent vertices, which shares each side with exactly one other triangle.

This mesh is generated using a marching cubes algorithm. In this algorithm, a 2x2 cube is moved through the mask space. For each position, the corners of the cube are then marked ‘segmented’ (1) or ‘not segmented’ (0). Treating the corners as specific bits in a binary number, a unique cube-index is obtained (0-255). This index is then used to determine which triangles are present in the cube, which are defined in a lookup table.

These triangles are defined in such a way, that the normal (obtained from the cross product of vectors describing 2 out of 3 edges) are always oriented in the same direction. For PyRadiomics, the calculated normals are always pointing outward. This is necessary to obtain the correct signed volume used in calculation of `MeshVolume`.

Let:

- $N_v$ represent the number of voxels included in the ROI
- $N_f$ represent the number of faces (triangles) defining the Mesh.
- $V$ the volume of the mesh in mm$^3$, calculated by *getMeshVolumeFeatureValue()*
- $A$ the surface area of the mesh in mm$^2$, calculated by `getMeshSurfaceAreaFeatureValue()`

References:

- Lorensen WE, Cline HE. Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Comput Graph Internet. 1987;21:163-9.

**getMeshVolumeFeatureValue**()

**1. Mesh Volume**

$$V_i = \frac{Oa_i \cdot (Ob_i \times Oc_i)}{6} \ (1)$$

$$V = \sum_{i=1}^{N_f} V_i \ (2)$$

The volume of the ROI $V$ is calculated from the triangle mesh of the ROI. For each face $i$ in the mesh, defined by points $a_i, b_i$ and $c_i$, the (signed) volume $V_f$ of the tetrahedron defined by that face and the origin of the image ($O$) is calculated. (1) The sign of the volume is determined by the sign of the normal, which must be consistently defined as either facing outward or inward of the ROI.

Then taking the sum of all $V_i$, the total volume of the ROI is obtained (2)

**Note:** For more extensive documentation on how the volume is obtained using the surface mesh, see the IBSI document, where this feature is defined as `Volume`.

**getVoxelVolumeFeatureValue**()

**2. Voxel Volume**

$$V_{voxel} = \sum_{k=1}^{N_v} V_k$$

The volume of the ROI $V_{voxel}$ is approximated by multiplying the number of voxels in the ROI by the volume of a single voxel $V_k$. This is a less precise approximation of the volume and is not used in subsequent features. This feature does not make use of the mesh and is not used in calculation of other shape features.

**Note:** Defined in IBSI as `Approximate Volume`.

**getSurfaceAreaFeatureValue**()

**3. Surface Area**

$$A_i = \frac{1}{2}|\text{a}_i\text{b}_i \times \text{a}_i\text{c}_i| \text{ (1)}$$
$$A = \sum_{i=1}^{N_f} A_i \text{ (2)}$$

where:

$\text{a}_i\text{b}_i$ and $\text{a}_i\text{c}_i$ are edges of the $i^{\text{th}}$ triangle in the mesh, formed by vertices $\text{a}_i$, $\text{b}_i$ and $\text{c}_i$.

To calculate the surface area, first the surface area $A_i$ of each triangle in the mesh is calculated (1). The total surface area is then obtained by taking the sum of all calculated sub-areas (2).

**Note:** Defined in IBSI as `Surface Area`.

**getSurfaceVolumeRatioFeatureValue**()

**4. Surface Area to Volume ratio**

$$\textit{surface to volume ratio} = \frac{A}{V}$$

Here, a lower value indicates a more compact (sphere-like) shape. This feature is not dimensionless, and is therefore (partly) dependent on the volume of the ROI.

**getSphericityFeatureValue**()

**5. Sphericity**

$$\textit{sphericity} = \frac{\sqrt[3]{36\pi V^2}}{A}$$

Sphericity is a measure of the roundness of the shape of the tumor region relative to a sphere. It is a dimensionless measure, independent of scale and orientation. The value range is $0 < \textit{sphericity} \leq 1$, where a value of 1 indicates a perfect sphere (a sphere has the smallest possible surface area for a given volume, compared to other solids).

**Note:** This feature is correlated to Compactness 1, Compactness 2 and Spherical Disproportion. In the default parameter file provided in the `pyradiomics/examples/exampleSettings` folder, Compactness 1 and Compactness 2 are therefore disabled.

**getCompactness1FeatureValue**()

**6. Compactness 1**

$$\textit{compactness 1} = \frac{V}{\sqrt{\pi A^3}}$$

Similar to Sphericity, Compactness 1 is a measure of how compact the shape of the tumor is relative to a sphere (most compact). It is therefore correlated to Sphericity and redundant. It is provided here for completeness. The value range is $0 < compactness\ 1 \leq \frac{1}{6\pi}$, where a value of $\frac{1}{6\pi}$ indicates a perfect sphere.

By definition, $compactness\ 1 = \frac{1}{6\pi}\sqrt{compactness\ 2} = \frac{1}{6\pi}\sqrt{sphericity^3}$.

**Note:** This feature is correlated to Compactness 2, Sphericity and Spherical Disproportion. Therefore, this feature is marked, so it is not enabled by default (i.e. this feature will not be enabled if no individual features are specified (enabling 'all' features), but will be enabled when individual features are specified, including this feature). To include this feature in the extraction, specify it by name in the enabled features.

**getCompactness2FeatureValue**()

**7. Compactness 2**

$$\textit{compactness 2} = 36\pi\frac{V^2}{A^3}$$

Similar to Sphericity and Compactness 1, Compactness 2 is a measure of how compact the shape of the tumor is relative to a sphere (most compact). It is a dimensionless measure, independent of scale and orientation. The value range is $0 < compactness\ 2 \leq 1$, where a value of 1 indicates a perfect sphere.

By definition, $compactness\ 2 = (sphericity)^3$

**Note:** This feature is correlated to Compactness 1, Sphericity and Spherical Disproportion. Therefore, this feature is marked, so it is not enabled by default (i.e. this feature will not be enabled if no individual features are specified (enabling 'all' features), but will be enabled when individual features are specified, including this feature). To include this feature in the extraction, specify it by name in the enabled features.

**getSphericalDisproportionFeatureValue**()

**8. Spherical Disproportion**

$$\textit{spherical disproportion} = \frac{A}{4\pi R^2} = \frac{A}{\sqrt[3]{36\pi V^2}}$$

Where $R$ is the radius of a sphere with the same volume as the tumor, and equal to $\sqrt[3]{\frac{3V}{4\pi}}$.

Spherical Disproportion is the ratio of the surface area of the tumor region to the surface area of a sphere with the same volume as the tumor region, and by definition, the inverse of Sphericity. Therefore, the value range is $spherical\ disproportion \geq 1$, with a value of 1 indicating a perfect sphere.

**Note:** This feature is correlated to Compactness 2, Compactness2 and Sphericity. Therefore, this feature is marked, so it is not enabled by default (i.e. this feature will not be enabled if no individual features are specified (enabling 'all' features), but will be enabled when individual features are specified, including this feature). To include this feature in the extraction, specify it by name in the enabled features.

`getMaximum3DDiameterFeatureValue()`

**9. Maximum 3D diameter**

Maximum 3D diameter is defined as the largest pairwise Euclidean distance between tumor surface mesh vertices.

Also known as Feret Diameter.

`getMaximum2DDiameterSliceFeatureValue()`

**10. Maximum 2D diameter (Slice)**

Maximum 2D diameter (Slice) is defined as the largest pairwise Euclidean distance between tumor surface mesh vertices in the row-column (generally the axial) plane.

`getMaximum2DDiameterColumnFeatureValue()`

**11. Maximum 2D diameter (Column)**

Maximum 2D diameter (Column) is defined as the largest pairwise Euclidean distance between tumor surface mesh vertices in the row-slice (usually the coronal) plane.

`getMaximum2DDiameterRowFeatureValue()`

**12. Maximum 2D diameter (Row)**

Maximum 2D diameter (Row) is defined as the largest pairwise Euclidean distance between tumor surface mesh vertices in the column-slice (usually the sagittal) plane.

`getMajorAxisLengthFeatureValue()`

**13. Major Axis Length**

$$\textit{major axis} = 4\sqrt{\lambda_{major}}$$

This feature yield the largest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component $\lambda_{major}$.

The principal component analysis is performed using the physical coordinates of the voxel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

`getMinorAxisLengthFeatureValue()`

**14. Minor Axis Length**

$$\textit{minor axis} = 4\sqrt{\lambda_{minor}}$$

This feature yield the second-largest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component $\lambda_{minor}$.

The principal component analysis is performed using the physical coordinates of the voxel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

`getLeastAxisLengthFeatureValue()`

**15. Least Axis Length**

$$\textit{least axis} = 4\sqrt{\lambda_{least}}$$

This feature yield the smallest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component $\lambda_{least}$. In case of a 2D segmentation, this value will be 0.

The principal component analysis is performed using the physical coordinates of the voxel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

**getElongationFeatureValue**()

**16. Elongation**

Elongation shows the relationship between the two largest principal components in the ROI shape. For computational reasons, this feature is defined as the inverse of true elongation.

$$\textit{elongation} = \sqrt{\frac{\lambda_{minor}}{\lambda_{major}}}$$

Here, $\lambda_{\text{major}}$ and $\lambda_{\text{minor}}$ are the lengths of the largest and second largest principal component axes. The values range between 1 (where the cross section through the first and second largest principal moments is circle-like (non-elongated)) and 0 (where the object is a maximally elongated: i.e. a 1 dimensional line).

The principal component analysis is performed using the physical coordinates of the voxel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

**getFlatnessFeatureValue**()

**17. Flatness**

Flatness shows the relationship between the largest and smallest principal components in the ROI shape. For computational reasons, this feature is defined as the inverse of true flatness.

$$\textit{flatness} = \sqrt{\frac{\lambda_{least}}{\lambda_{major}}}$$

Here, $\lambda_{\text{major}}$ and $\lambda_{\text{least}}$ are the lengths of the largest and smallest principal component axes. The values range between 1 (non-flat, sphere-like) and 0 (a flat object, or single-slice segmentation).

The principal component analysis is performed using the physical coordinates of the voxel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

### 2.5.3 Shape Features (2D)

**class** radiomics.shape2D.**RadiomicsShape2D**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

In this group of features we included descriptors of the two-dimensional size and shape of the ROI. These features are independent from the gray level intensity distribution in the ROI and are therefore only calculated on the non-derived image and mask.

Unless otherwise specified, features are derived from the approximated shape defined by the circumference mesh. To build this mesh, vertices (points) are first defined as points halfway on an edge between a pixel included in the ROI and one outside the ROI. By connecting these vertices a mesh of connected lines is obtained, with each line defined by 2 adjacent vertices, which shares each a point with exactly one other line.

This mesh is generated using an adapted version marching cubes algorithm. In this algorithm, a 2x2 square is moved through the mask space (2d). For each position, the corners of the square are then marked 'segmented' (1) or 'not segmented' (0). Treating the corners as specific bits in a binary number, a unique square-index is obtained (0-15). This index is then used to determine which lines are present in the square, which are defined in a lookup table.

These lines are defined in such a way, that the normal of the triangle defined by these points and the origin is always oriented in the a consistent direction. This results in signed values for the surface area of each triangle, so that when summed, the superfluous (postive) area included by triangles partly inside and outside the ROI is perfectly cancelled out by the (negative) area of triangles entirely outside the ROI.

Let:

- $N_p$ represent the number of pixels included in the ROI
- $N_f$ represent the number of lines defining the circumference (perimeter) Mesh.
- $A$ the surface area of the mesh in mm$^2$, calculated by *getMeshSurfaceFeatureValue()*
- $P$ the perimeter of the mesh in mm, calculated by *getPerimeterFeatureValue()*

**Note:** This class can **only** be calculated for truly 2D masks. To ensure correct processing, it is required that `force2D` is set to `True` and `force2Ddimension` to the dimension that is out-of plane (e.g. 0 (z-axis) for an axial slice). Furthermore, this dimension is required to have size 1. If not set correctly, a ValueError is raised.

References:

- Lorensen WE, Cline HE. Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Comput Graph Internet. 1987;21:163-9.

**getMeshSurfaceFeatureValue**()

**1. Mesh Surface**

$$A_i = \frac{1}{2}\text{Oa}_i \times \text{Ob}_i \text{ (1)}$$
$$A = \sum_{i=1}^{N_f} A_i \text{ (2)}$$

where:

$\text{O}_i\text{a}_i$ and $\text{O}_i\text{b}_i$ are edges of the $i^{\text{th}}$ triangle in the mesh, formed by vertices $\text{a}_i$, $\text{b}_i$ of the perimiter and the origin O.

To calculate the surface area, first the signed surface area $A_i$ of each triangle in the mesh is calculated (1). The total surface area is then obtained by taking the sum of all calculated sub-areas (2), where the sign will ensure correct surface area, as the negative area of triangles outside the ROI will cancel out the surplus area included by triangles partly inside and partly outside the ROI.

**getPixelSurfaceFeatureValue**()

**2. Pixel Surface**

$$A_{pixel} = \sum_{k=1}^{N_v} A_k$$

The surface area of the ROI $A_{pixel}$ is approximated by multiplying the number of pixels in the ROI by the surface area of a single pixel $A_k$. This is a less precise approximation of the surface area. This feature does not make use of the mesh and is not used in calculation of other 2D shape features.

**getPerimeterFeatureValue**()

**3. Perimeter**

$$P_i = \sqrt{(\text{a}_i - \text{b}_i)^2} \text{ (1)}$$
$$P = \sum_{i=1}^{N_f} P_i \text{ (2)}$$

where:

$\text{a}_i$ and $\text{b}_i$ are vertices of the $i^{\text{th}}$ line in the perimeter mesh.

To calculate the perimeter, first the perimeter $A_i$ of each line in the mesh circumference is calculated (1). The total perimeter is then obtained by taking the sum of all calculated sub-areas (2).

**getPerimeterSurfaceRatioFeatureValue**()

**4. Perimeter to Surface ratio**

$$\textit{perimeter to surface ratio} = \frac{P}{A}$$

Here, a lower value indicates a more compact (circle-like) shape. This feature is not dimensionless, and is therefore (partly) dependent on the surface area of the ROI.

**getSphericityFeatureValue**()

**5. Sphericity**

$$\textit{sphericity} = \frac{2\pi R}{P} = \frac{2\sqrt{\pi A}}{P}$$

Where $R$ is the radius of a circle with the same surface as the ROI, and equal to $\sqrt{\frac{A}{\pi}}$.

Sphericity is the ratio of the perimeter of the tumor region to the perimeter of a circle with the same surface area as the tumor region and therefore a measure of the roundness of the shape of the tumor region relative to a circle. It is a dimensionless measure, independent of scale and orientation. The value range is $0 < sphericity \leq 1$, where a value of 1 indicates a perfect circle (a circle has the smallest possible perimeter for a given surface area, compared to other shapes).

**Note:** This feature is correlated to Spherical Disproportion. Therefore, only this feature is enabled by default.

**getSphericalDisproportionFeatureValue**()

**6. Spherical Disproportion**

$$\textit{spherical disproportion} = \frac{P}{2\sqrt{\pi A}}$$

Spherical Disproportion is the ratio of the perimeter of the tumor region to the perimeter of a circle with the same surface area as the tumor region, and by definition, the inverse of Sphericity. Therefore, the value range is $spherical\ disproportion \geq 1$, with a value of 1 indicating a perfect sphere.

**Note:** This feature is correlated to Sphericity. Therefore, this feature is marked, so it is not enabled by default (i.e. this feature will not be enabled if no individual features are specified (enabling 'all' features), but will be enabled when individual features are specified, including this feature). To include this feature in the extraction, specify it by name in the enabled features.

**getMaximumDiameterFeatureValue**()

**7. Maximum 2D diameter**

Maximum diameter is defined as the largest pairwise Euclidean distance between tumor surface mesh vertices.

**getMajorAxisLengthFeatureValue**()

**8. Major Axis Length**

$$\textit{major axis} = 4\sqrt{\lambda_{major}}$$

This feature yield the largest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component $\lambda_{major}$.

The principal component analysis is performed using the physical coordinates of the pixel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

`getMinorAxisLengthFeatureValue()`

**9. Minor Axis Length**

$$\textit{minor axis} = 4\sqrt{\lambda_{minor}}$$

This feature yield the second-largest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component $\lambda_{minor}$.

The principal component analysis is performed using the physical coordinates of the pixel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

`getElongationFeatureValue()`

**10. Elongation**

Elongation shows the relationship between the two largest principal components in the ROI shape. For computational reasons, this feature is defined as the inverse of true elongation.

$$\textit{elongation} = \sqrt{\frac{\lambda_{minor}}{\lambda_{major}}}$$

Here, $\lambda_{\text{major}}$ and $\lambda_{\text{minor}}$ are the lengths of the largest and second largest principal component axes. The values range between 1 (where the cross section through the first and second largest principal moments is circle-like (non-elongated)) and 0 (where the object is a maximally elongated: i.e. a 1 dimensional line).

The principal component analysis is performed using the physical coordinates of the pixel centers defining the ROI. It therefore takes spacing into account, but does not make use of the shape mesh.

### 2.5.4 Gray Level Co-occurrence Matrix (GLCM) Features

**class** radiomics.glcm.**RadiomicsGLCM**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

A Gray Level Co-occurrence Matrix (GLCM) of size $N_g \times N_g$ describes the second-order joint probability function of an image region constrained by the mask and is defined as $\textbf{P}(i,j|\delta,\theta)$. The $(i,j)^{\text{th}}$ element of this matrix represents the number of times the combination of levels $i$ and $j$ occur in two pixels in the image, that are separated by a distance of $\delta$ pixels along angle $\theta$. The distance $\delta$ from the center voxel is defined as the distance according to the infinity norm. For $\delta = 1$, this results in 2 neighbors for each of 13 angles in 3D (26-connectivity) and for $\delta = 2$ a 98-connectivity (49 unique angles).

Note that pyradiomics by default computes symmetrical GLCM!

As a two dimensional example, let the following matrix **I** represent a 5x5 image, having 5 discrete grey levels:

$$\textbf{I} = \begin{bmatrix} 1 & 2 & 5 & 2 & 3 \\ 3 & 2 & 1 & 3 & 1 \\ 1 & 3 & 5 & 5 & 2 \\ 1 & 1 & 1 & 1 & 2 \\ 1 & 2 & 4 & 3 & 5 \end{bmatrix}$$

For distance $\delta = 1$ (considering pixels with a distance of 1 pixel from each other) and angle $\theta = 0^\circ$ (horizontal plane, i.e. voxels to the left and right of the center voxel), the following symmetrical GLCM is obtained:

$$\textbf{P} = \begin{bmatrix} 6 & 4 & 3 & 0 & 0 \\ 4 & 0 & 2 & 1 & 3 \\ 3 & 2 & 0 & 1 & 2 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 3 & 2 & 0 & 2 \end{bmatrix}$$

Let:

- $\epsilon$ be an arbitrarily small positive number ($\approx 2.2\times 10^{-16}$)
- $\textbf{P}(i,j)$ be the co-occurence matrix for an arbitrary $\delta$ and $\theta$
- $p(i,j)$ be the normalized co-occurence matrix and equal to $\frac{\textbf{P}(i,j)}{\sum{\textbf{P}(i,j)}}$
- $N_g$ be the number of discrete intensity levels in the image
- $p_x(i) = \sum^{N_g}_{j=1}{p(i,j)}$ be the marginal row probabilities
- $p_y(j) = \sum^{N_g}_{i=1}{p(i,j)}$ be the marginal column probabilities
- $\mu_x$ be the mean gray level intensity of $p_x$ and defined as $\mu_x = \displaystyle\sum^{N_g}_{i=1}{p_x(i)i}$
- $\mu_y$ be the mean gray level intensity of $p_y$ and defined as $\mu_y = \displaystyle\sum^{N_g}_{j=1}{p_y(j)j}$
- $\sigma_x$ be the standard deviation of $p_x$
- $\sigma_y$ be the standard deviation of $p_y$
- $p_{x+y}(k) = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)},\text{ where }i+j=k,\text{ and }k=2,3,\dots,2N_g$
- $p_{x-y}(k) = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)},\text{ where }|i-j|=k,\text{ and }k=0,1,\dots,N_g-1$
- $HX = -\sum^{N_g}_{i=1}{p_x(i)\log_2\big(p_x(i)+\epsilon\big)}$ be the entropy of $p_x$
- $HY = -\sum^{N_g}_{j=1}{p_y(j)\log_2\big(p_y(j)+\epsilon\big)}$ be the entropy of $p_y$
- $HXY = -\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)\log_2\big(p(i,j)+\epsilon\big)}$ be the entropy of $p(i,j)$
- $HXY1 = -\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)\log_2\big(p_x(i)p_y(j)+\epsilon\big)}$
- $HXY2 = -\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p_x(i)p_y(j)\log_2\big(p_x(i)p_y(j)+\epsilon\big)}$

By default, the value of a feature is calculated on the GLCM for each angle separately, after which the mean of these values is returned. If distance weighting is enabled, GLCM matrices are weighted by weighting factor W and then summed and normalised. Features are then calculated on the resultant matrix. Weighting factor W is calculated for the distance between neighbouring voxels by:

$W = e^{-\|d\|^2}$, where d is the distance for the associated angle according to the norm specified in setting 'weightingNorm'.

The following class specific settings are possible:

- distances [[1]]: List of integers. This specifies the distances between the center voxel and the neighbor, for which angles should be generated.
- symmetricalGLCM [True]: boolean, indicates whether co-occurrences should be assessed in two directions per angle, which results in a symmetrical matrix, with equal distributions for $i$ and $j$. A symmetrical matrix corresponds to the GLCM as defined by Haralick et al.
- weightingNorm [None]: string, indicates which norm should be used when applying distance weighting. Enumerated setting, possible values:
  - 'manhattan': first order norm
  - 'euclidean': second order norm
  - 'infinity': infinity norm.
  - 'no_weighting': GLCMs are weighted by factor 1 and summed

– None: Applies no weighting, mean of values calculated on separate matrices is returned.

In case of other values, an warning is logged and option 'no_weighting' is used.

References

- Haralick, R., Shanmugan, K., Dinstein, I; Textural features for image classification; IEEE Transactions on Systems, Man and Cybernetics; 1973(3), p610-621
- https://en.wikipedia.org/wiki/Co-occurrence_matrix
- http://www.fp.ucalgary.ca/mhallbey/the_glcm.htm

**`getAutocorrelationFeatureValue()`**

**1. Autocorrelation**

$$\textit{autocorrelation} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)ij}$$

Autocorrelation is a measure of the magnitude of the fineness and coarseness of texture.

**`getJointAverageFeatureValue()`**

**2. Joint Average**

$$\textit{joint average} = \mu_x = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)i}$$

Returns the mean gray level intensity of the $i$ distribution.

**Warning:** As this formula represents the average of the distribution of $i$, it is independent from the distribution of $j$. Therefore, only use this formula if the GLCM is symmetrical, where $p_x(i) = p_y(j)$, where $i = j$.

**`getClusterProminenceFeatureValue()`**

**3. Cluster Prominence**

$$\textit{cluster prominence} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{\big(i+j-\mu_x-\mu_y\big)^4p(i,j)}$$

Cluster Prominence is a measure of the skewness and asymmetry of the GLCM. A higher values implies more asymmetry about the mean while a lower value indicates a peak near the mean value and less variation about the mean.

**`getClusterShadeFeatureValue()`**

**4. Cluster Shade**

$$\textit{cluster shade} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{\big(i+j-\mu_x-\mu_y\big)^3p(i,j)}$$

Cluster Shade is a measure of the skewness and uniformity of the GLCM. A higher cluster shade implies greater asymmetry about the mean.

**`getClusterTendencyFeatureValue()`**

**5. Cluster Tendency**

$$\textit{cluster tendency} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{\big(i+j-\mu_x-\mu_y\big)^2p(i,j)}$$

Cluster Tendency is a measure of groupings of voxels with similar gray-level values.

`getContrastFeatureValue()`

**6. Contrast**

$$\textit{contrast} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{(i-j)^2p(i,j)}$$

Contrast is a measure of the local intensity variation, favoring values away from the diagonal $(i = j)$. A larger value correlates with a greater disparity in intensity values among neighboring voxels.

`getCorrelationFeatureValue()`

**7. Correlation**

$$\textit{correlation} = \frac{\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)ij-\mu_x\mu_y}}{\sigma_x(i)\sigma_y(j)}$$

Correlation is a value between 0 (uncorrelated) and 1 (perfectly correlated) showing the linear dependency of gray level values to their respective voxels in the GLCM.

**Note:** When there is only 1 discrete gray value in the ROI (flat region), $\sigma_x$ and $\sigma_y$ will be 0. In this case, an arbitrary value of 1 is returned instead. This is assessed on a per-angle basis.

`getDifferenceAverageFeatureValue()`

**8. Difference Average**

$$\textit{difference average} = \sum^{N_g-1}_{k=0}{kp_{x-y}(k)}$$

Difference Average measures the relationship between occurrences of pairs with similar intensity values and occurrences of pairs with differing intensity values.

`getDifferenceEntropyFeatureValue()`

**9. Difference Entropy**

$$\textit{difference entropy} = \sum^{N_g-1}_{k=0}{p_{x-y}(k)\log_2\big(p_{x-y}(k)+\epsilon\big)}$$

Difference Entropy is a measure of the randomness/variability in neighborhood intensity value differences.

`getDifferenceVarianceFeatureValue()`

**10. Difference Variance**

$$\textit{difference variance} = \sum^{N_g-1}_{k=0}{(k-DA)^2p_{x-y}(k)}$$

Difference Variance is a measure of heterogeneity that places higher weights on differing intensity level pairs that deviate more from the mean.

`getDissimilarityFeatureValue()`

**DEPRECATED. Dissimilarity**

$$\textit{dissimilarity} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{|i-j|p(i,j)}$$

**Warning:** This feature has been deprecated, as it is mathematically equal to Difference Average `getDifferenceAverageFeatureValue()`. See *here* for the proof. **Enabling this feature will result in the logging of a DeprecationWarning (does not interrupt extraction of other features), no value is calculated for this features**

**getJointEnergyFeatureValue**()

**11. Joint Energy**

$$\textit{joint energy} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_g}_{j=1}{\big(p(i,j)\big)^2}$$

Energy is a measure of homogeneous patterns in the image. A greater Energy implies that there are more instances of intensity value pairs in the image that neighbor each other at higher frequencies.

**Note:** Defined by IBSI as Angular Second Moment.

**getJointEntropyFeatureValue**()

**12. Joint Entropy**

$$\textit{joint entropy} = -\displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_g}_{j=1}{p(i,j)\log_2\big(p(i,j)+\epsilon\big)}$$

Joint entropy is a measure of the randomness/variability in neighborhood intensity values.

**Note:** Defined by IBSI as Joint entropy

**getHomogeneity1FeatureValue**()

**DEPRECATED. Homogeneity 1**

$$\textit{homogeneity 1} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_g}_{j=1}{\frac{p(i,j)}{1+|i-j|}}$$

**Warning:** This feature has been deprecated, as it is mathematically equal to Inverse Difference `getIdFeatureValue()`. **Enabling this feature will result in the logging of a DeprecationWarning (does not interrupt extraction of other features), no value is calculated for this features**

**getHomogeneity2FeatureValue**()

**DEPRECATED. Homogeneity 2**

$$\textit{homogeneity 2} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_g}_{j=1}{\frac{p(i,j)}{1+|i-j|^2}}$$

**Warning:** This feature has been deprecated, as it is mathematically equal to Inverse Difference Moment `getIdmFeatureValue()`. **Enabling this feature will result in the logging of a DeprecationWarning (does not interrupt extraction of other features), no value is calculated for this features**

**getImc1FeatureValue**()

**13. Informational Measure of Correlation (IMC) 1**

$$\textit{IMC 1} = \frac{HXY - HXY1}{\max\{HX, HY\}}$$

IMC1 assesses the correlation between the probability distributions of $i$ and $j$ (quantifying the complexity of the texture), using mutual information I(x, y):

$$\begin{aligned}I(i,j) &= \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}p(i,j)\log_2\big(\frac{p(i,j)}{p_x(i)p_y(j)}\big)\\ &= \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}p(i,j)\big(\log_2(p(i,j)) - \log_2(p_x(i)p_y(j))\big)\\ &= \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}p(i,j)\log_2\big(p(i,j)\big) - \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}p(i,j)\log_2\big(p_x(i)p_y(j)\big)\\ &= -HXY + HXY1\end{aligned}$$

However, in this formula, the numerator is defined as HXY - HXY1 (i.e. $-I(x,y)$), and is therefore $\leq 0$. This reflects how this feature is defined in the original Haralick paper.

In the case where the distributions are independent, there is no mutual information and the result will therefore be 0. In the case of uniform distribution with complete dependence, mutual information will be equal to $\log_2(N_g)$.

Finally, $HXY - HXY1$ is divided by the maximum of the 2 marginal entropies, where in the latter case of complete dependence (not necessarily uniform; low complexity) it will result in $IMC1 = -1$, as $HX = HY = I(i,j)$.

**Note:** In the case where both HX and HY are 0 (as is the case in a flat region), an arbitrary value of 0 is returned to prevent a division by 0. This is done on a per-angle basis (i.e. prior to any averaging).

**getImc2FeatureValue**()

**14. Informational Measure of Correlation (IMC) 2**

$$\textit{IMC 2} = \sqrt{1-e^{-2(HXY2 - HXY)}}$$

IMC2 also assesses the correlation between the probability distributions of $i$ and $j$ (quantifying the complexity of the texture). Of interest is to note that $HXY1 = HXY2$ and that $HXY2 - HXY \geq 0$ represents the mutual information of the 2 distributions. Therefore, the range of IMC2 = [0, 1), with 0 representing the case of 2 independent distributions (no mutual information) and the maximum value representing the case of 2 fully dependent and uniform distributions (maximal mutual information, equal to $\log_2(N_g)$). In this latter case, the maximum value is then equal to $\sqrt{1-e^{-2\log_2(N_g)}}$, approaching 1.

**Note:** Due to machine precision errors, it is possble that HXY > HXY2, which would result in returning complex numbers. In these cases, a value of 0 is returned for IMC2. This is done on a per-angle basis (i.e. prior to any averaging).

**getIdmFeatureValue**()

**15. Inverse Difference Moment (IDM)**

$$\textit{IDM} = \sum^{N_g-1}_{k=0}{\frac{p_{x-y}(k)}{1+k^2}}$$

IDM (a.k.a Homogeneity 2) is a measure of the local homogeneity of an image. IDM weights are the inverse of the Contrast weights (decreasing exponentially from the diagonal i=j in the GLCM).

**getMCCFeatureValue()**

**16. Maximal Correlation Coefficient (MCC)**

$$\textit{MCC} = \sqrt{\text{second largest eigenvalue of Q}}$$

$$Q(i,j) = \sum^{N_g}_{k=0}{\frac{p(i,k)p(j,k)}{p_x(i)p_y(k)}}$$

The Maximal Correlation Coefficient is a measure of complexity of the texture and $0 \leq MCC \leq 1$.

In case of a flat region, each GLCM matrix has shape (1, 1), resulting in just 1 eigenvalue. In this case, an arbitrary value of 1 is returned.

**getIdmnFeatureValue()**

**17. Inverse Difference Moment Normalized (IDMN)**

$$\textit{IDMN} = \sum^{N_g-1}_{k=0}{ \frac{p_{x-y}(k)}{1+\left(\frac{k^2}{N_g^2}\right)} }$$

IDMN (inverse difference moment normalized) is a measure of the local homogeneity of an image. IDMN weights are the inverse of the Contrast weights (decreasing exponentially from the diagonal $i=j$ in the GLCM). Unlike Homogeneity2, IDMN normalizes the square of the difference between neighboring intensity values by dividing over the square of the total number of discrete intensity values.

**getIdFeatureValue()**

**18. Inverse Difference (ID)**

$$\textit{ID} = \sum^{N_g-1}_{k=0}{\frac{p_{x-y}(k)}{1+k}}$$

ID (a.k.a. Homogeneity 1) is another measure of the local homogeneity of an image. With more uniform gray levels, the denominator will remain low, resulting in a higher overall value.

**getIdnFeatureValue()**

**19. Inverse Difference Normalized (IDN)**

$$\textit{IDN} = \sum^{N_g-1}_{k=0}{ \frac{p_{x-y}(k)}{1+\left(\frac{k}{N_g}\right)} }$$

IDN (inverse difference normalized) is another measure of the local homogeneity of an image. Unlike Homogeneity1, IDN normalizes the difference between the neighboring intensity values by dividing over the total number of discrete intensity values.

**getInverseVarianceFeatureValue()**

**20. Inverse Variance**

$$\textit{inverse variance} = \sum^{N_g-1}_{k=1}{\frac{p_{x-y}(k)}{k^2}}$$

Note that $k=0$ is skipped, as this would result in a division by 0.

**getMaximumProbabilityFeatureValue()**

**21. Maximum Probability**

$$\textit{maximum probability} = \max\big(p(i,j)\big)$$

Maximum Probability is occurrences of the most predominant pair of neighboring intensity values.

**Note:** Defined by IBSI as Joint maximum

**getSumAverageFeatureValue**()

**22. Sum Average**

$$\textit{sum average} = \sum_{k=2}^{2N_g} p_{x+y}(k)k$$

Sum Average measures the relationship between occurrences of pairs with lower intensity values and occurrences of pairs with higher intensity values.

**Warning:** When GLCM is symmetrical, $\mu_x = \mu_y$, and therefore Sum Average $= \mu_x + \mu_y = 2\mu_x = 2 * JointAverage$. See formulas (4.), (5.) and (6.) defined *here* for the proof that Sum Average $= \mu_x + \mu_y$. In the default parameter files provided in the `examples/exampleSettings`, this feature has been disabled.

**getSumVarianceFeatureValue**()

**DEPRECATED. Sum Variance**

$$\textit{sum variance} = \sum_{k=2}^{2N_g} (k - SA)^2 p_{x+y}(k)$$

**Warning:** This feature has been deprecated, as it is mathematically equal to Cluster Tendency *getClusterTendencyFeatureValue()*. See *here* for the proof. **Enabling this feature will result in the logging of a DeprecationWarning (does not interrupt extraction of other features), no value is calculated for this features**

**getSumEntropyFeatureValue**()

**23. Sum Entropy**

$$\textit{sum entropy} = \sum_{k=2}^{2N_g} p_{x+y}(k) \log_2 \big(p_{x+y}(k) + \epsilon\big)$$

Sum Entropy is a sum of neighborhood intensity value differences.

**getSumSquaresFeatureValue**()

**24. Sum of Squares**

$$\textit{sum squares} = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (i - \mu_x)^2 p(i,j)$$

Sum of Squares or Variance is a measure in the distribution of neigboring intensity level pairs about the mean intensity level in the GLCM.

**Warning:** This formula represents the variance of the distribution of $i$ and is independent from the distribution of $j$. Therefore, only use this formula if the GLCM is symmetrical, where $p_x(i) = p_y(j)$, where $i = j$

**Note:** Defined by IBSI as Joint Variance

### 2.5.5 Gray Level Size Zone Matrix (GLSZM) Features

**class** radiomics.glszm.**RadiomicsGLSZM**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

A Gray Level Size Zone (GLSZM) quantifies gray level zones in an image. A gray level zone is defined as a the number of connected voxels that share the same gray level intensity. A voxel is considered connected if the distance is 1 according to the infinity norm (26-connected region in a 3D, 8-connected region in 2D). In a gray level size zone matrix $P(i,j)$ the $(i,j)^{\text{th}}$ element equals the number of zones with gray level $i$ and size $j$ appear in image. Contrary to GLCM and GLRLM, the GLSZM is rotation independent, with only one matrix calculated for all directions in the ROI.

As a two dimensional example, consider the following 5x5 image, with 5 discrete gray levels:

$$\mathbf{I} = \begin{bmatrix} 5 & 2 & 5 & 4 & 4 \\ 3 & 3 & 3 & 1 & 3 \\ 2 & 1 & 1 & 1 & 3 \\ 4 & 2 & 2 & 2 & 3 \\ 3 & 5 & 3 & 3 & 2 \end{bmatrix}$$

The GLSZM then becomes:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Let:

- $N_g$ be the number of discrete intensity values in the image
- $N_s$ be the number of discrete zone sizes in the image
- $N_p$ be the number of voxels in the image
- $N_z$ be the number of zones in the ROI, which is equal to $\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)$ and $1 \leq N_z \leq N_p$
- $\mathbf{P}(i,j)$ be the size zone matrix
- $p(i,j)$ be the normalized size zone matrix, defined as $p(i,j) = \frac{\mathbf{P}(i,j)}{N_z}$

**Note:** The mathematical formulas that define the GLSZM features correspond to the definitions of features extracted from the GLRLM.

References

- Guillaume Thibault; Bernard Fertil; Claire Navarro; Sandrine Pereira; Pierre Cau; Nicolas Levy; Jean Sequeira; Jean-Luc Mari (2009). “Texture Indexes and Gray Level Size Zone Matrix. Application to Cell Nuclei Classification”. Pattern Recognition and Information Processing (PRIP): 140-145.
- https://en.wikipedia.org/wiki/Gray_level_size_zone_matrix

`getSmallAreaEmphasisFeatureValue()`

**1. Small Area Emphasis (SAE)**

$$\textit{SAE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\frac{\textbf{P}(i,j)}{j^2}}}{N_z}$$

SAE is a measure of the distribution of small size zones, with a greater value indicative of more smaller size zones and more fine textures.

`getLargeAreaEmphasisFeatureValue()`

**2. Large Area Emphasis (LAE)**

$$\textit{LAE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\textbf{P}(i,j)j^2}}{N_z}$$

LAE is a measure of the distribution of large area size zones, with a greater value indicative of more larger size zones and more coarse textures.

`getGrayLevelNonUniformityFeatureValue()`

**3. Gray Level Non-Uniformity (GLN)**

$$\textit{GLN} = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_s}_{j=1}{\textbf{P}(i,j)}\right)^2}{N_z}$$

GLN measures the variability of gray-level intensity values in the image, with a lower value indicating more homogeneity in intensity values.

`getGrayLevelNonUniformityNormalizedFeatureValue()`

**4. Gray Level Non-Uniformity Normalized (GLNN)**

$$\textit{GLNN} = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_s}_{j=1}{\textbf{P}(i,j)}\right)^2}{N_z^2}$$

GLNN measures the variability of gray-level intensity values in the image, with a lower value indicating a greater similarity in intensity values. This is the normalized version of the GLN formula.

`getSizeZoneNonUniformityFeatureValue()`

**5. Size-Zone Non-Uniformity (SZN)**

$$\textit{SZN} = \frac{\sum^{N_s}_{j=1}\left(\sum^{N_g}_{i=1}{\textbf{P}(i,j)}\right)^2}{N_z}$$

SZN measures the variability of size zone volumes in the image, with a lower value indicating more homogeneity in size zone volumes.

`getSizeZoneNonUniformityNormalizedFeatureValue()`

**6. Size-Zone Non-Uniformity Normalized (SZNN)**

$$\textit{SZNN} = \frac{\sum^{N_s}_{j=1}\left(\sum^{N_g}_{i=1}{\textbf{P}(i,j)}\right)^2}{N_z^2}$$

SZNN measures the variability of size zone volumes throughout the image, with a lower value indicating more homogeneity among zone size volumes in the image. This is the normalized version of the SZN formula.

`getZonePercentageFeatureValue()`

**7. Zone Percentage (ZP)**

$$\textit{ZP} = \frac{N_z}{N_p}$$

ZP measures the coarseness of the texture by taking the ratio of number of zones and number of voxels in the ROI.

Values are in range $\frac{1}{N_p} \leq ZP \leq 1$, with higher values indicating a larger portion of the ROI consists of small zones (indicates a more fine texture).

`getGrayLevelVarianceFeatureValue()`

**8. Gray Level Variance (GLV)**

$$\textit{GLV} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_s}_{j=1}{p(i,j)(i - \mu)^2}$$

Here, $\mu = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_s}_{j=1}{p(i,j)i}$

GLV measures the variance in gray level intensities for the zones.

`getZoneVarianceFeatureValue()`

**9. Zone Variance (ZV)**

$$\textit{ZV} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_s}_{j=1}{p(i,j)(j - \mu)^2}$$

Here, $\mu = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_s}_{j=1}{p(i,j)j}$

ZV measures the variance in zone size volumes for the zones.

`getZoneEntropyFeatureValue()`

**10. Zone Entropy (ZE)**

$$\textit{ZE} = -\displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_s}_{j=1}{p(i,j)\log_{2}(p(i,j)+\epsilon)}$$

Here, $\epsilon$ is an arbitrarily small positive number ($\approx 2.2\times10^{-16}$).

ZE measures the uncertainty/randomness in the distribution of zone sizes and gray levels. A higher value indicates more heterogeneneity in the texture patterns.

`getLowGrayLevelZoneEmphasisFeatureValue()`

**11. Low Gray Level Zone Emphasis (LGLZE)**

$$\textit{LGLZE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\frac{\textbf{P}(i,j)}{i^2}}}{N_z}$$

LGLZE measures the distribution of lower gray-level size zones, with a higher value indicating a greater proportion of lower gray-level values and size zones in the image.

`getHighGrayLevelZoneEmphasisFeatureValue()`

**12. High Gray Level Zone Emphasis (HGLZE)**

$$\textit{HGLZE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\textbf{P}(i,j)i^2}}{N_z}$$

HGLZE measures the distribution of the higher gray-level values, with a higher value indicating a greater proportion of higher gray-level values and size zones in the image.

**getSmallAreaLowGrayLevelEmphasisFeatureValue**()

**13. Small Area Low Gray Level Emphasis (SALGLE)**

$$\textit{SALGLE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\frac{\textbf{P}(i,j)}{i^2j^2}}}{N_z}$$

SALGLE measures the proportion in the image of the joint distribution of smaller size zones with lower gray-level values.

**getSmallAreaHighGrayLevelEmphasisFeatureValue**()

**14. Small Area High Gray Level Emphasis (SAHGLE)**

$$\textit{SAHGLE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\frac{\textbf{P}(i,j)i^2}{j^2}}}{N_z}$$

SAHGLE measures the proportion in the image of the joint distribution of smaller size zones with higher gray-level values.

**getLargeAreaLowGrayLevelEmphasisFeatureValue**()

**15. Large Area Low Gray Level Emphasis (LALGLE)**

$$\textit{LALGLE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\frac{\textbf{P}(i,j)j^2}{i^2}}}{N_z}$$

LALGLE measures the proportion in the image of the joint distribution of larger size zones with lower gray-level values.

**getLargeAreaHighGrayLevelEmphasisFeatureValue**()

**16. Large Area High Gray Level Emphasis (LAHGLE)**

$$\textit{LAHGLE} = \frac{\sum^{N_g}_{i=1}\sum^{N_s}_{j=1}{\textbf{P}(i,j)i^2j^2}}{N_z}$$

LAHGLE measures the proportion in the image of the joint distribution of larger size zones with higher gray-level values.

## 2.5.6 Gray Level Run Length Matrix (GLRLM) Features

**class** radiomics.glrlm.**RadiomicsGLRLM**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

A Gray Level Run Length Matrix (GLRLM) quantifies gray level runs, which are defined as the length in number of pixels, of consecutive pixels that have the same gray level value. In a gray level run length matrix $\textbf{P}(i,j|\theta)$, the $(i,j)^{\text{th}}$ element describes the number of runs with gray level $i$ and length $j$ occur in the image (ROI) along angle $\theta$.

As a two dimensional example, consider the following 5x5 image, with 5 discrete gray levels:

$$\textbf{I} = \begin{bmatrix} 5 & 2 & 5 & 4 & 4\\ 3 & 3 & 3 & 1 & 3\\ 2 & 1 & 1 & 1 & 3\\ 4 & 2 & 2 & 2 & 3\\ 3 & 5 & 3 & 3 & 2 \end{bmatrix}$$

The GLRLM for $\theta = 0$, where 0 degrees is the horizontal direction, then becomes:

$$\textbf{P} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \\ 4 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Let:

- $N_g$ be the number of discrete intensity values in the image
- $N_r$ be the number of discrete run lengths in the image
- $N_p$ be the number of voxels in the image
- $N_r(\theta)$ be the number of runs in the image along angle $\theta$, which is equal to $\sum^{N_g}_{i=1}\sum^{N_r}_{j=1}{\textbf{P}(i,j|\theta)}$ and $1 \leq N_r(\theta) \leq N_p$
- $\textbf{P}(i,j|\theta)$ be the run length matrix for an arbitrary direction $\theta$
- $p(i,j|\theta)$ be the normalized run length matrix, defined as $p(i,j|\theta) = \frac{\textbf{P}(i,j|\theta)}{N_r(\theta)}$

By default, the value of a feature is calculated on the GLRLM for each angle separately, after which the mean of these values is returned. If distance weighting is enabled, GLRLMs are weighted by the distance between neighbouring voxels and then summed and normalised. Features are then calculated on the resultant matrix. The distance between neighbouring voxels is calculated for each angle using the norm specified in 'weightingNorm'.

The following class specific settings are possible:

- weightingNorm [None]: string, indicates which norm should be used when applying distance weighting. Enumerated setting, possible values:
  - 'manhattan': first order norm
  - 'euclidean': second order norm
  - 'infinity': infinity norm.
  - 'no_weighting': GLCMs are weighted by factor 1 and summed
  - None: Applies no weighting, mean of values calculated on separate matrices is returned.

  In case of other values, an warning is logged and option 'no_weighting' is used.

References

- Galloway MM. 1975. Texture analysis using gray level run lengths. Computer Graphics and Image Processing, 4(2):172-179.
- Chu A., Sehgal C.M., Greenleaf J. F. 1990. Use of gray value distribution of run length for texture analysis. Pattern Recognition Letters, 11(6):415-419
- Xu D., Kurani A., Furst J., Raicu D. 2004. Run-Length Encoding For Volumetric Texture. International Conference on Visualization, Imaging and Image Processing (VIIP), p. 452-458
- Tang X. 1998. Texture information in run-length matrices. IEEE Transactions on Image Processing 7(11):1602-1609.
- Tustison N., Gee J. Run-Length Matrices For Texture Analysis. Insight Journal 2008 January - June.

**`getShortRunEmphasisFeatureValue`**`()`

**1. Short Run Emphasis (SRE)**

$$\textit{SRE} = \frac{\sum^{N_g}_{i=1}\sum^{N_r}_{j=1}{\frac{\textbf{P}(i,j|\theta)}{j^2}}}{N_r(\theta)}$$

SRE is a measure of the distribution of short run lengths, with a greater value indicative of shorter run lengths and more fine textural textures.

`getLongRunEmphasisFeatureValue()`

**2. Long Run Emphasis (LRE)**

$$\textit{LRE} = \frac{\sum^{N_g}_{i=1}\sum^{N_r}_{j=1}{\textbf{P}(i,j|\theta)j^2}}{N_r(\theta)}$$

LRE is a measure of the distribution of long run lengths, with a greater value indicative of longer run lengths and more coarse structural textures.

`getGrayLevelNonUniformityFeatureValue()`

**3. Gray Level Non-Uniformity (GLN)**

$$\textit{GLN} = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_r}_{j=1}{\textbf{P}(i,j|\theta)}\right)^2}{N_r(\theta)}$$

GLN measures the similarity of gray-level intensity values in the image, where a lower GLN value correlates with a greater similarity in intensity values.

`getGrayLevelNonUniformityNormalizedFeatureValue()`

**4. Gray Level Non-Uniformity Normalized (GLNN)**

$$\textit{GLNN} = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_r}_{j=1}{\textbf{P}(i,j|\theta)}\right)^2}{N_r(\theta)^2}$$

GLNN measures the similarity of gray-level intensity values in the image, where a lower GLNN value correlates with a greater similarity in intensity values. This is the normalized version of the GLN formula.

`getRunLengthNonUniformityFeatureValue()`

**5. Run Length Non-Uniformity (RLN)**

$$\textit{RLN} = \frac{\sum^{N_r}_{j=1}\left(\sum^{N_g}_{i=1}{\textbf{P}(i,j|\theta)}\right)^2}{N_r(\theta)}$$

RLN measures the similarity of run lengths throughout the image, with a lower value indicating more homogeneity among run lengths in the image.

`getRunLengthNonUniformityNormalizedFeatureValue()`

**6. Run Length Non-Uniformity Normalized (RLNN)**

$$\textit{RLNN} = \frac{\sum^{N_r}_{j=1}\left(\sum^{N_g}_{i=1}{\textbf{P}(i,j|\theta)}\right)^2}{N_r(\theta)^2}$$

RLNN measures the similarity of run lengths throughout the image, with a lower value indicating more homogeneity among run lengths in the image. This is the normalized version of the RLN formula.

`getRunPercentageFeatureValue()`

**7. Run Percentage (RP)**

$$\textit{RP} = {\frac{N_r(\theta)}{N_p}}$$

RP measures the coarseness of the texture by taking the ratio of number of runs and number of voxels in the ROI.

Values are in range $\frac{1}{N_p} \leq RP \leq 1$, with higher values indicating a larger portion of the ROI consists of short runs (indicates a more fine texture).

**Note:** Note that when weighting is applied and matrices are merged before calculation, $N_p$ is multiplied by $n$ number of matrices merged to ensure correct normalization (as each voxel is considered $n$ times)

**getGrayLevelVarianceFeatureValue()**

**8. Gray Level Variance (GLV)**

$$\textit{GLV} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_r}_{j=1}{p(i,j|\theta)(i - \mu)^2}$$

Here, $\mu = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_r}_{j=1}{p(i,j|\theta)i}$

GLV measures the variance in gray level intensity for the runs.

**getRunVarianceFeatureValue()**

**9. Run Variance (RV)**

$$\textit{RV} = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_r}_{j=1}{p(i,j|\theta)(j - \mu)^2}$$

Here, $\mu = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_r}_{j=1}{p(i,j|\theta)j}$

RV is a measure of the variance in runs for the run lengths.

**getRunEntropyFeatureValue()**

**10. Run Entropy (RE)**

$$\textit{RE} = -\displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_r}_{j=1}{p(i,j|\theta)\log_{2}(p(i,j|\theta)+\epsilon)}$$

Here, $\epsilon$ is an arbitrarily small positive number ($\approx 2.2\times10^{-16}$).

RE measures the uncertainty/randomness in the distribution of run lengths and gray levels. A higher value indicates more heterogeneity in the texture patterns.

**getLowGrayLevelRunEmphasisFeatureValue()**

**11. Low Gray Level Run Emphasis (LGLRE)**

$$\textit{LGLRE} = \frac{\sum^{N_g}_{i=1}\sum^{N_r}_{j=1}{\frac{\textbf{P}(i,j|\theta)}{i^2}}}{N_r(\theta)}$$

LGLRE measures the distribution of low gray-level values, with a higher value indicating a greater concentration of low gray-level values in the image.

**getHighGrayLevelRunEmphasisFeatureValue()**

**12. High Gray Level Run Emphasis (HGLRE)**

$$\textit{HGLRE} = \frac{\sum^{N_g}_{i=1}\sum^{N_r}_{j=1}{\textbf{P}(i,j|\theta)i^2}}{N_r(\theta)}$$

HGLRE measures the distribution of the higher gray-level values, with a higher value indicating a greater concentration of high gray-level values in the image.

**getShortRunLowGrayLevelEmphasisFeatureValue**()

**13. Short Run Low Gray Level Emphasis (SRLGLE)**

$$\textit{SRLGLE} = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\frac{\textbf{P}(i,j|\theta)}{i^2j^2}}{N_r(\theta)}$$

SRLGLE measures the joint distribution of shorter run lengths with lower gray-level values.

**getShortRunHighGrayLevelEmphasisFeatureValue**()

**14. Short Run High Gray Level Emphasis (SRHGLE)**

$$\textit{SRHGLE} = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\frac{\textbf{P}(i,j|\theta)i^2}{j^2}}{N_r(\theta)}$$

SRHGLE measures the joint distribution of shorter run lengths with higher gray-level values.

**getLongRunLowGrayLevelEmphasisFeatureValue**()

**15. Long Run Low Gray Level Emphasis (LRLGLE)**

$$\textit{LRLGLRE} = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\frac{\textbf{P}(i,j|\theta)j^2}{i^2}}{N_r(\theta)}$$

LRLGLRE measures the joint distribution of long run lengths with lower gray-level values.

**getLongRunHighGrayLevelEmphasisFeatureValue**()

**16. Long Run High Gray Level Emphasis (LRHGLE)**

$$\textit{LRHGLRE} = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\textbf{P}(i,j|\theta)i^2j^2}{N_r(\theta)}$$

LRHGLRE measures the joint distribution of long run lengths with higher gray-level values.

### 2.5.7 Neighbouring Gray Tone Difference Matrix (NGTDM) Features

**class** radiomics.ngtdm.**RadiomicsNGTDM**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

A Neighbouring Gray Tone Difference Matrix quantifies the difference between a gray value and the average gray value of its neighbours within distance $\delta$. The sum of absolute differences for gray level $i$ is stored in the matrix. Let $\textbf{X}_{gl}$ be a set of segmented voxels and $x_{gl}(j_x,j_y,j_z) \in \textbf{X}_{gl}$ be the gray level of a voxel at postion $(j_x,j_y,j_z)$, then the average gray level of the neigbourhood is:

$$\begin{aligned}\bar{A}_i &= \bar{A}(j_x,j_y,j_z)\\ &= \frac{1}{W}\sum_{k_x=-\delta}^{\delta}\sum_{k_y=-\delta}^{\delta}\sum_{k_z=-\delta}^{\delta}x_{gl}(j_x+k_x,j_y+k_y,j_z+k_z),\\ &\text{where } (k_x,k_y,k_z)\neq(0,0,0) \text{ and } x_{gl}(j_x+k_x,j_y+k_y,j_z+k_z) \in \textbf{X}_{gl}\end{aligned}$$

Here, $W$ is the number of voxels in the neighbourhood that are also in $\textbf{X}_{gl}$.

As a two dimensional example, let the following matrix **I** represent a 4x4 image, having 5 discrete grey levels, but no voxels with gray level 4:

$$\textbf{I} = \begin{bmatrix}1 & 2 & 5 & 2\\ 3 & 5 & 1 & 3\\ 1 & 3 & 5 & 5\\ 3 & 1 & 1 & 1\end{bmatrix}$$

The following NGTDM can be obtained:

| $i$ | $n_i$ | $p_i$ | $s_i$ |
|---|---|---|---|
| 1 | 6 | 0.375 | 13.35 |
| 2 | 2 | 0.125 | 2.00 |
| 3 | 4 | 0.25 | 2.63 |
| 4 | 0 | 0.00 | 0.00 |
| 5 | 4 | 0.25 | 10.075 |

6 pixels have gray level 1, therefore:

$s_1 = |1-10/3| + |1-30/8| + |1-15/5| + |1-13/5| + |1-15/5| + |1-11/3| = 13.35$

For gray level 2, there are 2 pixels, therefore:

$s_2 = |2-15/5| + |2-9/3| = 2$

Similar for gray values 3 and 5:

$s_3 = |3-12/5| + |3-18/5| + |3-20/8| + |3-5/3| = 3.03$
$s_5 = |5-14/5| + |5-18/5| + |5-20/8| + |5-11/5| = 10.075$

Let:

$n_i$ be the number of voxels in $X_{gl}$ with gray level $i$

$N_{v,p}$ be the total number of voxels in $X_{gl}$ and equal to $\sum n_i$ (i.e. the number of voxels with a valid region; at least 1 neighbor). $N_{v,p} \leq N_p$, where $N_p$ is the total number of voxels in the ROI.

$p_i$ be the gray level probability and equal to $n_i/N_v$

$s_i = \left\{ \begin{array}{rcl} \sum^{n_i} |i-\bar{A}_i| & \text{for} & n_i \neq 0 \\ 0 & \text{for} & n_i = 0 \end{array} \right.$ be the sum of absolute differences for gray level $i$

$N_g$ be the number of discrete gray levels

$N_{g,p}$ be the number of gray levels where $p_i \neq 0$

The following class specific settings are possible:

- distances [[1]]: List of integers. This specifies the distances between the center voxel and the neighbor, for which angles should be generated.

References

- Amadasun M, King R; Textural features corresponding to textural properties; Systems, Man and Cybernetics, IEEE Transactions on 19:1264-1274 (1989). doi: 10.1109/21.44046

**getCoarsenessFeatureValue**()

Calculate and return the coarseness.

$Coarseness = \frac{1}{\sum^{N_g}_{i=1}{p_{i}s_{i}}}$

Coarseness is a measure of average difference between the center voxel and its neighbourhood and is an indication of the spatial rate of change. A higher value indicates a lower spatial change rate and a locally more uniform texture.

N.B. $\sum^{N_g}_{i=1}{p_{i}s_{i}}$ potentially evaluates to 0 (in case of a completely homogeneous image). If this is the case, an arbitrary value of $10^6$ is returned.

**getContrastFeatureValue**()

Calculate and return the contrast.

$Contrast = \left(\frac{1}{N_{g,p}(N_{g,p}-1)}\displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_g}_{j=1}{p_{i}p_{j}(i-j)^2}\right)\left(\frac{1}{N_{v,p}}\displaystyle\sum^{N_g}_{i=1}{s_i}\right)$, where $p_i \neq 0, p_j \neq 0$

Contrast is a measure of the spatial intensity change, but is also dependent on the overall gray level dynamic range. Contrast is high when both the dynamic range and the spatial change rate are high, i.e. an image with a large range of gray levels, with large changes between voxels and their neighbourhood.

N.B. In case of a completely homogeneous image, $N_{g,p} = 1$, which would result in a division by 0. In this case, an arbitray value of 0 is returned.

**getBusynessFeatureValue**()

Calculate and return the busyness.

$Busyness = \frac{\sum^{N_g}_{i=1}{p_{i}s_{i}}}{\sum^{N_g}_{i = 1}\sum^{N_g}_{j = 1}{|ip_i - jp_j|}}$, where $p_i \neq 0, p_j \neq 0$

A measure of the change from a pixel to its neighbour. A high value for busyness indicates a 'busy' image, with rapid changes of intensity between pixels and its neighbourhood.

N.B. if $N_{g,p} = 1$, then $busyness = \frac{0}{0}$. If this is the case, 0 is returned, as it concerns a fully homogeneous region.

**getComplexityFeatureValue**()

Calculate and return the complexity.

$$Complexity = \frac{1}{N_{v,p}}\displaystyle\sum^{N_g}_{i = 1}\displaystyle\sum^{N_g}_{j = 1}{|i - j| \frac{p_{i}s_{i} + p_{j}s_{j}}{p_i + p_j}}, \text{ where } p_i \neq 0, p_j \neq 0$$

An image is considered complex when there are many primitive components in the image, i.e. the image is non-uniform and there are many rapid changes in gray level intensity.

**getStrengthFeatureValue**()

Calculate and return the strength.

$Strength = \frac{\sum^{N_g}_{i = 1}\sum^{N_g}_{j = 1}{(p_i + p_j)(i-j)^2}}{\sum^{N_g}_{i = 1}{s_i}}$, where $p_i \neq 0, p_j \neq 0$

Strength is a measure of the primitives in an image. Its value is high when the primitives are easily defined and visible, i.e. an image with slow change in intensity but more large coarse differences in gray level intensities.

N.B. $\sum^{N_g}_{i=1}{s_i}$ potentially evaluates to 0 (in case of a completely homogeneous image). If this is the case, 0 is returned.

## 2.5.8 Gray Level Dependence Matrix (GLDM) Features

**class** radiomics.gldm.**RadiomicsGLDM**(*inputImage*, *inputMask*, ***kwargs*)

Bases: *radiomics.base.RadiomicsFeaturesBase*

A Gray Level Dependence Matrix (GLDM) quantifies gray level dependencies in an image. A gray level dependency is defined as a the number of connected voxels within distance $\delta$ that are dependent on the center voxel. A neighbouring voxel with gray level $j$ is considered dependent on center voxel with gray level $i$ if $|i-j|\le\alpha$. In a gray level dependence matrix $\textbf{P}(i,j)$ the $(i,j)^{\text{th}}$ element describes the number of times a voxel with gray level $i$ with $j$ dependent voxels in its neighbourhood appears in image.

As a two dimensional example, consider the following 5x5 image, with 5 discrete gray levels:

$$\textbf{I} = \begin{bmatrix} 5 & 2 & 5 & 4 & 4\\ 3 & 3 & 3 & 1 & 3\\ 2 & 1 & 1 & 1 & 3\\ 4 & 2 & 2 & 2 & 3\\ 3 & 5 & 3 & 3 & 2 \end{bmatrix}$$

For $\alpha = 0$ and $\delta = 1$, the GLDM then becomes:

$$\textbf{P} = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 0 \\ 1 & 4 & 4 & 0 \\ 1 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

Let:

- $N_g$ be the number of discrete intensity values in the image
- $N_d$ be the number of discrete dependency sizes in the image
- $N_z$ be the number of dependency zones in the image, which is equal to $\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\textbf{P}(i,j)}$
- $\textbf{P}(i,j)$ be the dependence matrix
- $p(i,j)$ be the normalized dependence matrix, defined as $p(i,j) = \frac{\textbf{P}(i,j)}{N_z}$

**Note:** Because incomplete zones are allowed, every voxel in the ROI has a dependency zone. Therefore, $N_z = N_p$, where $N_p$ is the number of voxels in the image. Due to the fact that $Nz = N_p$, the Dependence Percentage and Gray Level Non-Uniformity Normalized (GLNN) have been removed. The first because it would always compute to 1, the latter because it is mathematically equal to first order - Uniformity (see `getUniformityFeatureValue()`). For mathematical proofs, see *here*.

The following class specific settings are possible:

- distances [[1]]: List of integers. This specifies the distances between the center voxel and the neighbor, for which angles should be generated.
- gldm_a [0]: float, $\alpha$ cutoff value for dependence. A neighbouring voxel with gray level $j$ is considered dependent on center voxel with gray level $i$ if $|i-j|\le\alpha$

References:

- Sun C, Wee WG. Neighboring Gray Level Dependence Matrix for Texture Classification. Comput Vision, Graph Image Process. 1983;23:341-352

**`getSmallDependenceEmphasisFeatureValue`**`()`

**1. Small Dependence Emphasis (SDE)**

$$SDE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\frac{\textbf{P}(i,j)}{i^2}}}{N_z}$$

A measure of the distribution of small dependencies, with a greater value indicative of smaller dependence and less homogeneous textures.

**`getLargeDependenceEmphasisFeatureValue`**`()`

**2. Large Dependence Emphasis (LDE)**

$$LDE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\textbf{P}(i,j)j^2}}{N_z}$$

A measure of the distribution of large dependencies, with a greater value indicative of larger dependence and more homogeneous textures.

**`getGrayLevelNonUniformityFeatureValue`**`()`

**3. Gray Level Non-Uniformity (GLN)**

$$GLN = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_d}_{j=1}{\textbf{P}(i,j)}\right)^2}{N_z}$$

Measures the similarity of gray-level intensity values in the image, where a lower GLN value correlates with a greater similarity in intensity values.

**getGrayLevelNonUniformityNormalizedFeatureValue**()

**DEPRECATED. Gray Level Non-Uniformity Normalized (GLNN)**

$$GLNN = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_d}_{j=1}{\textbf{P}(i,j)}\right)^2}{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\textbf{P}(i,j)}^2}$$

**Warning:** This feature has been deprecated, as it is mathematically equal to First Order - Uniformity `getUniformityFeatureValue()`. See *here* for the proof. **Enabling this feature will result in the logging of a DeprecationWarning (does not interrupt extraction of other features), no value is calculated for this feature**

**getDependenceNonUniformityFeatureValue**()

**4. Dependence Non-Uniformity (DN)**

$$DN = \frac{\sum^{N_d}_{j=1}\left(\sum^{N_g}_{i=1}{\textbf{P}(i,j)}\right)^2}{N_z}$$

Measures the similarity of dependence throughout the image, with a lower value indicating more homogeneity among dependencies in the image.

**getDependenceNonUniformityNormalizedFeatureValue**()

**5. Dependence Non-Uniformity Normalized (DNN)**

$$DNN = \frac{\sum^{N_d}_{j=1}\left(\sum^{N_g}_{i=1}{\textbf{P}(i,j)}\right)^2}{N_z^2}$$

Measures the similarity of dependence throughout the image, with a lower value indicating more homogeneity among dependencies in the image. This is the normalized version of the DLN formula.

**getGrayLevelVarianceFeatureValue**()

**6. Gray Level Variance (GLV)**

$$GLV = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_d}_{j=1}{p(i,j)(i - \mu)^2} \text{, where} \mu = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_d}_{j=1}{ip(i,j)}$$

Measures the variance in grey level in the image.

**getDependenceVarianceFeatureValue**()

**7. Dependence Variance (DV)**

$$DV = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_d}_{j=1}{p(i,j)(j - \mu)^2} \text{, where} \mu = \displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_d}_{j=1}{jp(i,j)}$$

Measures the variance in dependence size in the image.

**getDependenceEntropyFeatureValue**()

**8. Dependence Entropy (DE)**

$$Dependence Entropy = -\displaystyle\sum^{N_g}_{i=1}\displaystyle\sum^{N_d}_{j=1}{p(i,j)\log_{2}(p(i,j)+\epsilon)}$$

`getDependencePercentageFeatureValue()`

**DEPRECATED. Dependence Percentage**

$$\textit{dependence percentage} = \frac{N_z}{N_p}$$

**Warning:** This feature has been deprecated, as it would always compute 1. See *here* for more details. **Enabling this feature will result in the logging of a DeprecationWarning (does not interrupt extraction of other features), no value is calculated for this features**

`getLowGrayLevelEmphasisFeatureValue()`

**9. Low Gray Level Emphasis (LGLE)**

$$LGLE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\frac{\textbf{P}(i,j)}{i^2}}}{N_z}$$

Measures the distribution of low gray-level values, with a higher value indicating a greater concentration of low gray-level values in the image.

`getHighGrayLevelEmphasisFeatureValue()`

**10. High Gray Level Emphasis (HGLE)**

$$HGLE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\textbf{P}(i,j)i^2}}{N_z}$$

Measures the distribution of the higher gray-level values, with a higher value indicating a greater concentration of high gray-level values in the image.

`getSmallDependenceLowGrayLevelEmphasisFeatureValue()`

**11. Small Dependence Low Gray Level Emphasis (SDLGLE)**

$$SDLGLE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\frac{\textbf{P}(i,j)}{i^2j^2}}}{N_z}$$

Measures the joint distribution of small dependence with lower gray-level values.

`getSmallDependenceHighGrayLevelEmphasisFeatureValue()`

**12. Small Dependence High Gray Level Emphasis (SDHGLE)**

Measures the joint distribution of small dependence with higher gray-level values.

`getLargeDependenceLowGrayLevelEmphasisFeatureValue()`

**13. Large Dependence Low Gray Level Emphasis (LDLGLE)**

$$LDLGLE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\frac{\textbf{P}(i,j)j^2}{i^2}}}{N_z}$$

Measures the joint distribution of large dependence with lower gray-level values.

`getLargeDependenceHighGrayLevelEmphasisFeatureValue()`

**14. Large Dependence High Gray Level Emphasis (LDHGLE)**

$$LDHGLE = \frac{\sum^{N_g}_{i=1}\sum^{N_d}_{j=1}{\textbf{P}(i,j)i^2j^2}}{N_z}$$

Measures the joint distribution of large dependence with higher gray-level values.

## 2.6 Excluded Radiomic Features

Some commonly know features are not supported (anymore) in PyRadiomics. These features are listed here, so as to provide a complete overview, as well as argumentation for why these features are excluded from PyRadiomics

### 2.6.1 Excluded GLCM Features

For included features and class definition, see *Gray Level Co-occurrence Matrix (GLCM) Features*.

#### Sum Variance

$$\textit{sum variance} = \sum^{2N_g}_{k=2}{(k-SA)^2p_{x+y}(k)}$$

Sum Variance is a measure of heterogeneity that places higher weights on neighboring intensity level pairs that deviate more from the mean.

This feature has been removed, as it is mathematically identical to Cluster Tendency (see `getClusterTendencyFeatureValue()`).

The mathematical proof is as follows:

(1) As defined in GLCM, $p_{x+y}(k) = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)},\text{ where }i+j=k, k \in \{2, 3, \dots, 2N_g\}$

(2) Starting with cluster tendency as defined in GLCM:

$$\textit{cluster tendency} = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{\big(i+j-\mu_x-\mu_y\big)^2p(i,j)}$$
$$= \sum^{2N_g}_{k=2}{\Big[\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{\big(i+j-\mu_x-\mu_y\big)^2p(i,j)}\text{, where }i+j=k\Big]}$$
$$= \sum^{2N_g}_{k=2}{\Big[\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{\big(k-(\mu_x+\mu_y)\big)^2p(i,j)}\text{, where }i+j=k\Big]}$$

**Note:** Because inside the sum $\sum^{2N_g}_{k=2}$, $k$ is a constant, and so are $\mu_x$ and $\mu_y$, $\big(k-(\mu_x+\mu_y)\big)^2$ is constant and can be taken outside the inner sum $\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}$.

$$= \sum^{2N_g}_{k=2}{\Big[\big(k-(\mu_x+\mu_y)\big)^2\sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{p(i,j)}\text{, where }i+j=k\Big]}$$

(3) Using (1.) and (2.)

$$\textit{cluster tendency} = \sum^{2N_g}_{k=2}{\Big[\big(k-(\mu_x+\mu_y)\big)^2p_{x+y}(k)\Big]}$$

(4) As defined in GLCM, $p_x(i) = \sum^{N_g}_{j=1}{P(i,j)}$ and $\mu_x = \sum^{N_g}_{i=1}{p_x(i)i}$, therefore $\mu_x = \sum^{N_g}_{i=1}\sum^{N_g}_{j=1}{P(i,j)i}$

(5) Similarly as in (4.), $\mu_y = \sum^{N_g}_{j=1}\sum^{N_g}_{i=1}{P(i,j)j}$

(6) Using (4.) and (5.), $\mu_x$ and $\mu_y$ can then be combined as follows:

$$\begin{aligned}\mu_x + \mu_y &= \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} P(i,j)i + \sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)j \\ &= \sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)i + P(i,j)j \\ &= \sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)(i+j) \\ &= \sum_{k=2}^{2N_g}\left[\sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)(i+j), \text{ where } k = i+j\right] \\ &= \sum_{k=2}^{2N_g} p_{x+y}(k)k = \textit{sum average (SA)}\end{aligned}$$

(7) Combining (3.) and (6.) yields the following formula:

$$\text{Cluster Tendency} = \sum_{k=2}^{2N_g}\left[\left(k - SA\right)^2 p_{x+y}(k)\right] = \textit{ sum variance}$$

Q.E.D

## Dissimilarity

$$\textit{dissimilarity} = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)|i-j|$$

Dissimilarity is a measure of local intensity variation defined as the mean absolute difference between the neighbouring pairs. A larger value correlates with a greater disparity in intensity values among neighboring voxels.

This feature has been removed, as it is mathematically identical to Difference Average (see `getDifferenceAverageFeatureValue()`).

The mathematical proof is as follows:

(1) As defined in GLCM, $p_{x-y}(k) = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j), \text{ where } |i-j| = k$

(2) Starting with Dissimilarity as defined in GLCM:

$$\begin{aligned}\textit{dissimilarity} &= \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)|i-j| \\ &= \sum_{k=0}^{N_g-1}\left[\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)|i-j|, \text{ where } |i-j| = k\right] \\ &= \sum_{k=0}^{N_g-1}\left[\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)k, \text{ where } |i-j| = k\right]\end{aligned}$$

(3) Using (1.) and (2.)

$$\textit{dissimilarity} = \sum_{k=0}^{N_g-1} p_{x-y}(k)k = \textit{difference average}$$

Q.E.D.

### 2.6.2 Excluded GLDM Features

For included features and class definition, see *Gray Level Dependence Matrix (GLDM) Features*.

#### Dependence percentage

$$\textit{dependence percentage} = \frac{N_z}{N_p}$$

Dependence percentage is the ratio between voxels with a dependence zone and the total number of voxels in the image. Because PyRadiomics allows for incomplete dependence zones, all voxels have a dependence zone and $N_z = N_p$. Therefore, this feature would always compute to 1.

#### Gray Level Non-Uniformity Normalized

$$\textit{GLNN} = \frac{\sum^{N_g}_{i=1}\left(\sum^{N_d}_{j=1}{\textbf{P}(i,j)}\right)^2}{N_z^2}$$

Measures the similarity of gray-level intensity values in the image, where a lower GLNN value correlates with a greater similarity in intensity values. This is the normalized version of the GLN formula.

This formula has been removed, because due to the definition of GLDM matrix (allowing incomplete zones), this feature is equal to first order Uniformity (see `getUniformityFeatureValue()`).

The mathematical proof is as follows:

(1) Starting with Gray Level Non-Uniformity Normalized as defined in GLDM,

$$\begin{aligned} \textit{GLNN} &= \frac{\sum^{N_g}_{i=1}\left(\sum^{N_d}_{j=1}{\textbf{P}(i,j)}\right)^2}{N_z^2} \\ &= \sum^{N_g}_{i=1}\frac{\left(\sum^{N_d}_{j=1}{\textbf{P}(i,j)}\right)^2}{N_z^2} \\ &= \sum^{N_g}_{i=1}\left(\frac{\sum^{N_d}_{j=1}{\textbf{P}(i,j)}}{N_z}\right)^2 \\ &= \sum^{N_g}_{i=1}\left(\sum^{N_d}_{j=1}{\frac{\textbf{P}(i,j)}{N_z}}\right)^2 \end{aligned}$$

(2) As defined in GLDM, $p(i,j) = \frac{\textbf{P}(i,j)}{N_z}$

(3) Using (1.) and (2.)

$$\textit{GLNN} = \sum^{N_g}_{i=1}\left(\sum^{N_d}_{j=1}{p(i,j)}\right)^2$$

(4) Because in the PyRadiomics definition incomplete dependence zones are allowed, every voxel in the ROI has a dependence zone. Therefore, $N_z = N_p$ and $\sum^{N_d}_{j=1}{\textbf{P}(i,j)}$ equals the number of voxels with gray level $i$ and is equal to $\textbf{P}(i)$, the first order histogram with $N_g$ discrete gray levels, as defined in first order.

(5) As defined in first order, $p(i) = \frac{\textbf{P}(i)}{N_p}$

(6) Using (2.), (4.) and (5.)

$$\sum_{j=1}^{N_d}\textbf{P}(i,j) = \textbf{P}(i)$$

$$\frac{\sum_{j=1}^{N_d}\textbf{P}(i,j)}{N_z} = \frac{\textbf{P}(i)}{N_p}$$

$$\sum_{j=1}^{N_d}\frac{\textbf{P}(i,j)}{N_z} = \frac{\textbf{P}(i)}{N_p}$$

$$\sum_{j=1}^{N_d}p(i,j) = p(i)$$

(7) Combining (3.) and (6.) yields:

$$\textit{GLNN} = \sum_{i=1}^{N_g}p(i)^2 = Uniformity$$

Q.E.D.

## 2.7 Contributing to pyradiomics

There are many ways to contribute to pyradiomics, with varying levels of effort. Do try to look through the documentation first if something is unclear, and let us know how we can do better.

- Ask a question on the pyradiomics email list
- Submit a parameter file you used for your extraction
- Submit a feature request or bug, or add to the discussion on the pyradiomics issue tracker
- Submit a Pull Request to improve pyradiomics or its documentation

We encourage a range of Pull Requests, from patches that include passing tests and documentation, all the way down to half-baked ideas that launch discussions.

### 2.7.1 The PR Process, Circle CI, and Related Gotchas

#### How to submit a PR ?

If you are new to pyradiomics development and you don't have push access to the pyradiomics repository, here are the steps:

1. Fork and clone the repository.
2. Create a branch.
3. Push the branch to your GitHub fork.
4. Create a Pull Request.

This corresponds to the *Fork & Pull Model* mentioned in the GitHub flow guides.

If you have push access to pyradiomics repository, you could simply push your branch into the main repository and create a Pull Request. This corresponds to the *Shared Repository Model* and will facilitate other developers to checkout your topic without having to configure a remote. It will also simplify the workflow when you are *co-developing* a branch.

When submitting a PR, make sure to add a `cc: @Radiomics/developers` comment to notify pyradiomics developers of your awesome contributions. Based on the comments posted by the reviewers, you may have to revisit your patches.

#### How to integrate a PR ?

Getting your contributions integrated is relatively straightforward, here is the checklist:

- Your changes include an update of the documentation if necessary
  - Documentation on modules, classes and functions is contained in the respective docstrings
  - More global documentation is contained in the `docs` folder.
  - New modules need to be added to the auto-generated documentation. See here for more information on adding new modules to the documentation.
- Your changes are added in the changelog in the *Next Release* section.
- All tests pass
- Consensus is reached. This usually means that at least one reviewer reviewed and approved your changes or added a `LGTM` comment, which is an acronym for *Looks Good to Me*.

Next, there are two scenarios:

- You do NOT have push access: A pyradiomics core developer will integrate your PR.
- You have push access: Simply click on the "Merge pull request" button.

Then, click on the "Delete branch" button that appears afterward.

#### Automatic testing of pull requests

Every pull request is tested automatically using CircleCI, TravisCI and AppVeyor each time you push a commit to it. The Github UI will restrict users from merging pull requests until the builds have returned with a successful result indicating that all tests have passed and there were no problems detected by the linter. These tests include the following

- flake8 to check adherence to the code style. See `.flake8` and `.editorconfig` for styles, exceptions to the PEP8 style, etc.
- If a feature class has a function `_calculateCMatrix()`, identifying it as a C enhanced class, output from the C extension is compared to the output from full python calculation. A absolute difference of 1e-3 is allowed to account for machine precision errors.
- All implemented features and feature classes have docstrings at the class level and feature definition level.
- A baseline is available for all features extracted from the 5 included test cases and calculated features match this baseline to within 3% (allowing for machine precision errors)

### 2.7.2 Submitting a parameter file

Different inputs into PyRadiomics require different settings. We encourage users to share their parameter file to help others extract features using the best settings for their use case.

#### How to submit your parameter file?

Parameter files are stored in the repository under `examples/exampleSettings`. If you wish to submit your parameters to the community, you can add your file here via a pull request (see above for details on making PRs). To help you along, here is a small checklist:

- The filename should at least contain the modality (e.g. "MR") for which it is intended, and optionally the body part (e.g. "prostate").
- Ensure the file has the correct extension (either ".yml" or ".yaml")
- Using comments in the parameter file, briefly explain your use case.

After you've opened a PR to submit your parameter file, it will be checked for validity by the automatic testing. You don't have to specify your file anywhere, as parameter files are detected automatically in the `exampleSettings` folder. If you want, you can also check your file manually using `bin/testParams.py`.

## 2.8 Developers

This section contains information on how to add or customize the feature classes and filters available in PyRadiomics. PyRadiomics enumerates the available feature classes and input image types at initialization of the toolbox. These are available from the global `radiomics` namespace by use of the functions `getFeatureClasses()` and `getImageTypes()`, respectively. Individual features in a feature class are enumerated at initialization of the class. See also the *contributing guidelines*.

### 2.8.1 Signature of a feature class

Each feature class is defined in a separate module, the module name is used as the feature class name (e.g. if module tex.py matches the feature class signature, it is available in the PyRadiomics toolbox as the 'tex' feature class). In the module a class should be defined that fits the following signature:

```
[required imports]
from radiomics import base

class Radiomics[Name](base.RadiomicsFeaturesBase):
    """
    Feature class docstring
    """

    def __init__(self, inputImage, inputMask, **kwargs):
        super(Radiomics[Name], self).__init__(inputImage, inputMask, **kwargs)
        # Feature class specific init

    def get[Feature]FeatureValue(self):
        """
        Feature docstring
        """
        # value = feature calculation using member variables of RadiomicsFeatureBase␣
↪and this class.
        return [value]
```

- At the top should be the import statements for packages required by the feature class. Unused import statements should be removed (flake8 will fail if unused import statements are encountered, or import statements are not structured as defined by appnexus).

- The class name should be 'Radiomics' followed by the name of the class (usually similar to the module name. However, this name is not used elsewhere and may be named arbitrarily).
- The class should inherit (directly or indirectly) from `base.RadiomicsFeaturesBase`, which is an abstract class defining the common interface for the feature classes
- Additional initialization steps should be called in the `__init__` function. For default variables initialized, see *Feature Class Base*.
- Documentation is required! Both at the class level (Feature class docstring) and at the level of the individual features (Feature docstring).
- If the feature class uses C extensions for matrix calculation enhancement, which should be tested using `test_matrices`, matrix calculation should be implemented as follows:
  - The function calculating the matrix using the C extension should be defined in a function called `_calculateMatrix`.
  - The functions to calculate the matrix accept no additional input arguments other than the `self` argument, and return the fully processed matrix as a numpy array.
  - The fully processed matrix should be assigned to a variable in the feature class named `P_[Name]`, where `[Name]` is identical to the feature class name (module name) (e.g. in feature class `glcm`, matrix is stored in variable `P_glcm`
- A feature class specific logger is created by the base class, which will be a child logger (i.e. the 'radiomics.tex' logger in case of the feature class 'tex'). It is exposed in the feature class as `self.logger`. Any log messages generated by the feature class should make use of this logger to ensure that the hierarchy of classes is correctly reflected in generated logs (i.e. `self.logger.debug('message')` to generate a debug log message).

### 2.8.2 Adding the baseline

During testing, calculated features are compared to a fixed baseline. If you implement a new class, it must be added to the baseline, otherwise testing will fail. Fortunately, adding a new class to the baseline is fairly easy: just run the `add_baseline.py` script located in the `tests` folder. In case you change a feature and need to rebuild the baseline for that class, run the script with the class name as an argument (e.g. `python add_baseline.py glcm`).

### 2.8.3 Signature of individual features

Each individual feature is defined as a function in the feature class with the `get[Name]FeatureValue(self)` signature, where `[Name]` is the feature name (unique on the feature class level). It accepts no input arguments, and should return a scalar value. The `self` argument represents the instantiated feature class that defines the function, and identifies the feature function as non-static.

### 2.8.4 Signature of an image type

All image types are defined in the *imageoperations module*, and identified by the signature `get[Name]Image(inputImage, inputMask, **kwargs)`. Here, `[Name]` represents the unique name for the image type, which is also used to identify the image type during extraction. The input of a image type function is fixed and consists of the `inputImage` and `inputMask`, SimpleITK Image objects of the original image and mask, respectively and `**kwargs`, which are the customized settings that should be used for the extraction of features from the derived image.

One or more derived images are returned using the 'yield' statement: `yield derivedImage, imageTypeName, kwargs`. Here, `derivedImage` is one SimpleITK image object representing the filtered image, `imageTypeName` is a unique string identifying features calculated using this filter in the output and `kwargs` are the customized settings for the extraction (`**kwargs` passed as input, without the double asterisk). Multiple derived images can be returned by multiple yield statements, or yield statements inside a loop. Please note that only one derived image should be returned on each call to yield and that `imageTypeName` is a unique name for *each* returned derived image. Derived images must have the same dimensions and occupy the same physical space to ensure compatibility with the mask.

### 2.8.5 Progress Reporting

When operating in full-python mode, the calculation of the texture matrices can take some time. Therefor PyRadiomics provides the possibility to report the progress for calculation of GLCM and GLSZM. This is only enabled in full-python mode when the verbosity (`setVerbosity()`) is set to INFO or DEBUG. By default, none is provided and no progress of matrix calculation will be reported.

To enable progress reporting, the `radiomics.progressReporter` variable should be set to a class object (NOT an instance), which fits the following signature:

1. Accepts an iterable as the first positional argument and a keyword argument ('desc') specifying a label to display
2. Can be used in a 'with' statement (i.e. exposes a `__enter__` and `__exit__` function)
3. Is iterable (i.e. at least specifies an `__iter__` function, which iterates over the iterable passed at initialization)

It is also possible to create your own progress reporter. To achieve this, additionally specify a function `__next__`, and have the `__iter__` function return `self`. The `__next__` function takes no arguments and returns a call to the `__next__` function of the iterable (i.e. `return self.iterable.__next__()`). Any prints/progress reporting calls can then be inserted in this function prior to the return statement.

In `radiomics\__init__.py` a dummy progress reporter (`_DummyProgressReporter`) is defined, which is used when calculating in full-python mode, but progress reporting is not enabled (verbosity > INFO) or the `progressReporter` variable is not set.

To design a custom progress reporter, the following code can be adapted and used as progressReporter:

```
class MyProgressReporter(object):
    def __init__(self, iterable, desc=''):
        self.desc = desc  # A description is which describes the progress that is␣
↪reported
        self.iterable = iterable  # Iterable is required

    # This function identifies the class as iterable and should return an object␣
↪which exposes
    # the __next__ function that should be used to iterate over the object
    def __iter__(self):
        return self  # return self to 'intercept' the calls to __next__ to insert␣
↪reporting code.

    def __next__(self):
        nextElement = self.iterable.__next__()
        # Insert custom progress reporting code here. This is called for every␣
↪iteration in the loop
        # (once for each unique gray level in the ROI for GLCM and GLSZM)

        # By inserting after the call `self.iterable.__next__()` the function will␣
↪exit before the
        # custom code is run when the stopIteration error is raised.
        return nextElement

    # This function is called when the 'with' statement is entered
```

(continues on next page)

(continued from previous page)

```
    def __enter__(self):
        print (self.desc)  # Print out the description upon start of the loop
        return self  # The __enter__ function should return itself

    # This function is called when the 'with' statement is exited
    def __exit__(self, exc_type, exc_value, tb):
        pass  # If nothing needs to be closed or handled, so just specify 'pass'
```

### 2.8.6 Using feature classes directly

- This represents an example where feature classes are used directly, circumventing checks and preprocessing done by the radiomics feature extractor class, and is not intended as standard use.
- (LINUX) To run from source code, add pyradiomics to the environment variable PYTHONPATH (Not necessary when PyRadiomics is installed):
  - `setenv PYTHONPATH /path/to/pyradiomics/radiomics`
- Start the python interactive session:
  - `python`
- Import the necessary classes:

```
from radiomics import firstorder, glcm, imageoperations, shape, glrlm, glszm,␣
↪getTestCase
import SimpleITK as sitk
import six
import sys, os
```

- Set up a data directory variable:

```
dataDir = '/path/to/pyradiomics/data'
```

- You will find sample data files brain1_image.nrrd and brain1_label.nrrd in that directory.
- Use SimpleITK to read a the brain image and mask:

```
imageName, maskName = getTestCase('brain1', dataDir)
image = sitk.ReadImage(imageName)
mask = sitk.ReadImage(maskName)
```

- Calculate the first order features:

```
firstOrderFeatures = firstorder.RadiomicsFirstOrder(image,mask)
firstOrderFeatures.enableAllFeatures()  # On the feature class level, all␣
↪features are disabled by default.
firstOrderFeatures.calculateFeatures()
for (key,val) in six.iteritems(firstOrderFeatures.featureValues):
  print("\t%s: %s" % (key, val))
```

- See the *Radiomic Features* section for more features that you can calculate.

### 2.8.7 Addtional points for attention

### Code style

To keep the PyRadiomics code consistent and as readable as possible, some style rules are enforced. These are part of the continuous testing and implemented using flake8. See also the `.flake8` configuration file in the root of the repository. To aid in keeping a consistent code style, a `.editorconfig` file is provided in the root of the folder.

Module names should be lowercase, without underscores or spaces. Class names, function names and variables should be declared using camelcase, with uppercase first letter for class names and lowercase first letter otherwise. Private helper functions (which should not be included in the documentation) should be declared using a '_' prefix. This is consistent with the python style for marking them as 'private', and will automatically exclude them from the generated documentation.

### Documentation

The documentation of PyRadiomics is auto-generated from static files contained in the `docs` folder and the docstrings of the Python code files. When a new feature class is added, this has to be added to the static file (`features.rst`) describing the feature classes as well. If done so, sphinx will take care of the rest. A featureclass can be added as follows:

```
<Class Name> Features
---------------------

.. automodule:: radiomics.<module name>
    :members:
    :undoc-members:
    :show-inheritance:
    :member-order: bysource
```

Documentation providing information of the feature class as a whole (e.g. how the feature matrix is calculated) should be provided in the docstring of the class. Definition of individual features, including the mathematical formulas should be provided in the docstrings of the feature functions. A docstring of the module is not required.

The presence of a docstring at the class level and at the level of each individual feature is required and checked during testing. Missing docstrings will cause the test to fail.

If you make edits to the documentation, or if you want to test how documentation for the new classes you added is rendered, you can generate Sphinx documentation locally:

1. `pip install sphinx`
2. `pip install sphinx_rtd_theme`
3. Run this command in the `docs` folder: `make html`
4. HTML version of the Sphinx documentation root will be in `_build/html/index.html`

### Testing

To ensure consistency in the extraction provided by PyRadiomics, continuous testing is used to test the PyRadiomics source code after each commit. These tests are defined in the test folder and used to run tests for the following environments:

- Python 2.7 64 bits (Windows, Linux and Mac)
- Python 3.4 64 bits (Windows and Linux)
- Python 3.5 64 bits (Windows and Linux)

**Note:** Python 3 testing for mac is currently disabled for Mac due to some issues with the SimpleITK package for python 3.

There are 3 testing scripts run for PyRadiomics. The first test is `test_cmatrices`, which asserts if the matrices calculated by the C extensions match those calculated by Python. A threshold of 1e-3 is used to allow for machine precision errors. The second test is `test_docstrings`, which asserts if there is missing documentation as described above. The final and most important test is `test_features`, which compares the features calculated by PyRadiomics against a known baseline using 5 test cases. These test cases and the baseline are stored in the `data` folder of the repository. This ensures that changes to the code do not silently change the calculated values of the features.

To add a new feature class to the baseline, run the `addClassToBaseline.py` script, contained in the `bin` folder. This script detects if there are feature classes in PyRadiomics, for which there is no baseline available. If any are found, a new baseline if calculated for these classes in the full-python mode and added to the baseline files. These new baseline files then needed to be included in the repository and committed.

## 2.9 pyradiomics labs

pyradiomics labs is a collection of exploratory/experimental features that are part of the repository, but are not part of the core functionality. We welcome user feedback about those features. Those scripts and features may change in the future.

### 2.9.1 pyradiomics-dcm

#### About

This is an experimental script to support the use of pyradiomics with DICOM data.

The script will accept as input a directory with a single DICOM image study for the input image, and the file name pointing to a DICOM Segmentation Image (DICOM SEG) object.

The script will transparently convert the DICOM image into a representation suitable by pyradiomics using either plastimatch or dcm2niix.

#### Why?

- medical image data usually comes in DICOM, and pyradiomics users often ask for help working with DICOM data
- there are public collections of data on TCIA where segmentations are stored as DICOM SEG
- the use of DICOM representation for radiomics features
  - introduces standardized formalism for the attributes that should be stored to accompany the features
  - allows to link results of calculations with the various ontologies describing the anatomy of the regions analyzed, and the features itself (e.g., the SR document produced by the script will utilize IBSI nomenclature to describe those features implemented in pyradiomics that have correspondence in IBSI)
  - allows to reference (by unique identifiers) the DICOM image series and DICOM segmentation used for feature calculation
  - enables harmonized representation of data for images, segmentations and features (i.e., same data management system can be used for all data types)

   – does not prevent the use of the results in software tools that are not DICOM-aware - dcmqi can be used to convert DICOM segmentations and DICOM SR with the measurements into non-DICOM representation (ITK-readable image formats for segmentations, and JSON for measurements); a separate tool is available to generate tab-delimited representation for DICOM attributes and measurements stored in those SRs: https://github.com/QIICR/dcm2tables

### Prerequisites

- plastimatch or dcm2niix for image volume reconstruction
- dcmqi[1][2] (build from fedb41 or later) for reading DICOM SEG and converting to a representation suitable by pyradiomics, and for storing the resulting features as a DICOM Structured Report, instantiating SR TID 1500
- prior to using this script, you might want to sort your DICOM data such that individual series are stored in separate directories. You might find this tool useful for this purpose: https://github.com/pieper/dicomsort
- if you segmentations are not stored as DICOM SEG, you can use dcmqi for generating standard representation of those segmentations: https://github.com/QIICR/dcmqi

### Usage

Example usage from command line:

```
$ python pyradiomics-dcm.py -h
usage: pyradiomics-dcm.py --input-image <dir> --input-seg <name> --output-sr <name>

Warning: This is a "pyradiomics labs" script, which means it is an experimental␣
↪feature in development!
The intent of this helper script is to enable pyradiomics feature extraction directly␣
↪from/to DICOM data.
The segmentation defining the region of interest must be defined as a DICOM␣
↪Segmentation image.
Support for DICOM Radiotherapy Structure Sets for defining region of interest may be␣
↪added in the future.

optional arguments:
  -h, --help            show this help message and exit
  --input-image-dir Input DICOM image directory
                    Directory with the input DICOM series. It is expected
                    that a single series is corresponding to a single
                    scalar volume.
  --input-seg-file Input DICOM SEG file
                    Input segmentation defined as aDICOM Segmentation
                    object.
  --output-dir Directory to store the output file
                    Directory for saving the resulting DICOM file.
  --parameters pyradiomics extraction parameters
  --temp-dir Temporary directory
```

(continues on next page)

[1] Herz C, Fillion-Robin J-C, Onken M, Riesmeier J, Lasso A, Pinter C, Fichtinger G, Pieper S, Clunie D, Kikinis R, Fedorov A. dcmqi: An Open Source Library for Standardized Communication of Quantitative Image Analysis Results Using DICOM. Cancer Research. 2017;77(21):e87–e90 http://cancerres.aacrjournals.org/content/77/21/e87

[2] Fedorov A, Clunie D, Ulrich E, Bauer C, Wahle A, Brown B, Onken M, Riesmeier J, Pieper S, Kikinis R, Buatti J, Beichel RR. (2016) DICOM for quantitative imaging biomarker development: a standards based approach to sharing clinical data and structured PET/CT analysis results in head and neck cancer research. PeerJ 4:e2057 https://doi.org/10.7717/peerj.2057

(continued from previous page)

```
  --features-dict Dictionary mapping pyradiomics feature names to the IBSI defined␣
↪features.
  --volume-reconstructor Choose the tool to be used for reconstructing image volume␣
↪from the DICOM image series. Allowed options are plastimatch or dcm2niix (should be␣
↪installed on the system). plastimatch will be used by default.
```

### Sample invocation

```
$ python pyradiomics-dcm.py --input-image-dir CT --input-seg SEG/1.dcm \
   --output-dir OutputSR --temp-dir TempDir --parameters Pyradiomics_Params.yaml
dcmqi repository URL: https://github.com/QIICR/dcmqi.git revision: 3638930 tag:␣
↪latest-4-g3638930
Row direction: 1 0 0
Col direction: 0 1 0
Z direction: 0 0 1
Total frames: 177
Total frames with unique IPP: 177
Total overlapping frames: 0
Origin: [-227.475, -194.775, -1223]
dcmqi repository URL: https://github.com/QIICR/dcmqi.git revision: 3638930 tag:␣
↪latest-4-g3638930
Total measurement groups: 1
Adding to compositeContext: 1.dcm
Composite Context initialized
SR saved!

$ dsrdump OutputSR/1.2.276.0.7230010.3.1.4.0.60427.1539113881.935517.dcm
Enhanced SR Document

Patient             : interobs05 (#interobs05)
ENH: include pyradiomics identification and version
Study               : interobs05_20170910_CT
Series              : GTV segmentation - Reader AB - pyradiomics features (#1)
Manufacturer        : QIICR (https://github.com/QIICR/dcmqi.git, #0)
Completion Flag     : PARTIAL
Verification Flag   : UNVERIFIED
Content Date/Time   : 2018-10-09 15:38:01

<CONTAINER:(,,"Imaging Measurement Report")=SEPARATE>
  <has concept mod CODE:(,,"Language of Content Item and Descendants")=(eng,RFC5646,
↪"English")>
  <has obs context CODE:(,,"Observer Type")=(121007,DCM,"Device")>
  <has obs context UIDREF:(,,"Device Observer UID")="1.3.6.1.4.1.43046.3.1.4.0.60427.
↪1539113880.935515">
  <has obs context TEXT:(,,"Device Observer Name")="pyradiomics">
  <has obs context TEXT:(,,"Device Observer Model Name")="2.1.0.post10.dev0+g51bc87f">
  <has concept mod CODE:(,,"Procedure reported")=(P0-0099A,SRT,"Imaging procedure")>
  <contains CONTAINER:(,,"Image Library")=SEPARATE>
    <contains CONTAINER:(,,"Image Library Group")=SEPARATE>
      <has acq context CODE:(,,"Modality")=(CT,DCM,"Computed Tomography")>
      <has acq context DATE:(,,"Study Date")="20170910">
      <has acq context UIDREF:(,,"Frame of Reference UID")="1.3.6.1.4.1.40744.29.
↪28518703451127075549995420991770873582">

...
```

(continues on next page)

(continued from previous page)

```
  <contains CONTAINER:(,,"Imaging Measurements")=SEPARATE>
    <contains CONTAINER:(,,"Measurement Group")=SEPARATE>
      <has obs context TEXT:(,,"Tracking Identifier")="Gross Target Volume">
      <has obs context UIDREF:(,,"Tracking Unique Identifier")="1.3.6.1.4.1.43046.3.1.
↪4.0.60427.1539113881.935516"
>
      <contains CODE:(,,"Finding")=(C112913,NCIt,"Gross Target Volume")>
      <contains IMAGE:(,,"Referenced Segment")=(SG image,,1)>
      <contains UIDREF:(,,"Source series for segmentation")="1.3.6.1.4.1.40744.29.
↪18397950185694012790332812250603
612437">
      <has concept mod CODE:(,,"Finding Site")=(T-28000,SRT,"Lung")>
      <contains NUM:(,,"shape_MeshVolume")="7.255467E+04" (1,UCUM,"no units")>
      <contains NUM:(,,"Maximum 3D diameter")="7.491328E+01" (1,UCUM,"no units")>
      <contains NUM:(,,"shape_Maximum2DDiameterSlice")="6.767570E+01" (1,UCUM,"no␣
↪units")>
      <contains NUM:(,,"Elongation")="7.993260E-01" (1,UCUM,"no units")>
      <contains NUM:(,,"shape_MinorAxisLength")="4.699969E+01" (1,UCUM,"no units")>
      <contains NUM:(,,"Flatness")="6.517569E-01" (1,UCUM,"no units")>
      <contains NUM:(,,"shape_Maximum2DDiameterColumn")="6.746851E+01" (1,UCUM,"no␣
↪units")>
      <contains NUM:(,,"Surface to volume ratio")="1.572168E-01" (1,UCUM,"no units")>
      <contains NUM:(,,"shape_Maximum2DDiameterRow")="6.072891E+01" (1,UCUM,"no units
↪")>
      <contains NUM:(,,"shape_VoxelVolume")="7.285600E+04" (1,UCUM,"no units")>
      <contains NUM:(,,"Sphericity")="7.375024E-01" (1,UCUM,"no units")>
      <contains NUM:(,,"Surface area")="1.140681E+04" (1,UCUM,"no units")>
      <contains NUM:(,,"shape_MajorAxisLength")="5.879915E+01" (1,UCUM,"no units")>
      <contains NUM:(,,"shape_LeastAxisLength")="3.832275E+01" (1,UCUM,"no units")>
      <contains NUM:(,,"Small zone emphasis")="7.384502E-01" (1,UCUM,"no units")>
      <contains NUM:(,,"glszm_SmallAreaLowGrayLevelEmphasis")="3.381883E-03" (1,UCUM,
↪"no units")>
      <contains NUM:(,,"Normalised grey level non-uniformity")="3.136554E-02" (1,UCUM,
↪"no units")>
      <contains NUM:(,,"glszm_SmallAreaHighGrayLevelEmphasis")="5.478214E+02" (1,UCUM,
↪"no units")>
      <contains NUM:(,,"Large zone emphasis")="3.873234E+03" (1,UCUM,"no units")>

...
```

### Questions?

Please post your feedback and questions on the pyradiomics email list.

### References

## 2.10 Frequently Asked Questions

### 2.10.1 Feature Extraction: Input, Customization and Reproducibility

### Does PyRadiomics adhere to IBSI definitions of features?

For the most part, yes.

PyRadiomics development is also involved in the standardisation effort by the IBSI team. Still, there are some differences between PyRadiomics and feature extraction as defined in the IBSI documents. These arise in places where several equally valid alternatives exist. In some of these cases, PyRadiomics opted for the one alternative, while the IBSI standard recommends another. For the sake of consistency in PyRadiomics development, we have opted not to change the PyRadiomics implementation, but rather document the difference.

Most notably are the differences in gray value discretization (just for the fixed bin size type) and resampling. These differences cannot be corrected by customization settings alone and require replacement by custom functions:

- **Binning:** When performing gray value discretization using a fixed bin width (AKA IBSI: FBS, Fixed Bins Size), and with Resegmentation set (most notable Case A and C), IBSI computes the bin edges as equally spaced from the minimum of the resegmentation range if absolute-resegmentation, and min(intensities) when sigma-resegmentation.

  In PyRadiomics, gray value discretization using a fixed bin width always utilizes bin edges which are equally spaced from 0 and where the lowest gray level is ensured to be discretized to the first bin. Regardless of any resegmentation, etc.

- **Resampling:**
  - Alignment of the grid: In IBSI, resampling grid is aligned by aligning the center of the image, whereas in Pyradiomics, we align the corner of the origin voxel. This can result in slightly different interpolated results, and even slightly different sizes of the resampled image and ROI, which in turn causes differences in extracted feature values.
  - **Gray value rounding:** In IBSI, they reason that if the original intensity values are from some lower precision datatype, resampled values (which are floating point numbers, usually 64-bit) should be resampled to a similar resolution. In the case of the IBSI phantoms, resampling to the nearest integer. PyRadiomics does not implement this, as differences are likely to be small and therefore serve more to add complexity than increase the meaning of the extracted values. Especially considering gray values are discretized anyway prior to calculation of most (all except firstorder) features. If some kind of normalization is performed, then meaning of the gray values changes as well. Differences arise here, because small rounding differences can cause a voxel to be assigned to a different bin, which can be a marked change in feature value results, especially in small ROIs.
  - Mask resampling: In IBSI, different interpolators can also be selected for resampling of the mask, with an additional thresholding the retrieve the binary mask. This only works if the mask is limited to zero and non-zero (i.e. 1) values. PyRadiomics also supports masks with different value labels, allowing extraction from different ROIs in the same mask file by indicating different label values. To prevent any incorrect re-assignment, PyRadiomics forces the mask resampling to be nearest neighbor.

Next, there are also some differences which can be solved by custom settings, in this case as only applies to Configuration E, where both absolute AND sigma resegmentation are performed. In PyRadiomics, both types are implemented, but only 1 can be selected at a time. To simulate applying both types, I calculated the absolute range after resegmentation and used that as the absolute resegment range: [ -718, 400 ]

Finally, there is a difference between PyRadiomics and IBSI in the calculation of firstorder: Kurtosis. IBSI calculates Excess Kurtosis, which is equal to Kurtosis - 3. PyRadiomics calculates Kurtosis, which is always +3 compared to IBSI. The difference of 3 stems from the fact that a gaussian distribution has a kurtosis of 3.

So in summary, the cause of the difference between PyRadiomics results and IBSI benchmark, per case:

Configuration C: Due to differences in gray value discretization and resampling Configuration D: Due to differences in resampling Configuration E: Due to differences in resampling and resegmentation

### What about gray value discretization? Fixed bin width? Fixed bin count?

Currently, although many studies favour a fixed bin count over a fixed bin width, there is no hard evidence favouring either a fixed bin width or a fixed bin count in all cases. Therefore PyRadiomics implements both the option for setting a fixed bin count (`binCount`) and a fixed bin width (`binWidth`, default).

The reason the a fixed bin width has been chosen as the default parameter is based in part on studies in PET that show a better reproducibility of features when implementing a fixed bin width[1]. Furthermore, our reasoning is best illustrated by the following example: Given an input with 2 images with 2 ROIs, with the range of gray values in the first being {0-100} and in the second {0-10}. If you use a fixed bin count, the "meaning" of 1 (discretized) gray value difference is different (in the first it means 10 gray values different, in the second just 1). This means you are looking at texture based on very different contrasts.

This example does assume that the original gray values mean the same thing in both images, and in case of images with definite/absolute gray values (e.g. HU in CT, SUV in PET imaging), this holds true. However, in case of arbitrary/relative gray values (e.g. signal intensity in MR), this is not necessarily the case. In this latter case, we still recommend a fixed bin width, but with additional pre-processing (e.g. normalization) to ensure better comparability of gray values. Use of a fixed bin count would be possible here, but then the calculated features may still be very influenced by the range of gray values seen in the image, as well as noise caused by the fact that the original gray values are less comparable. Moreover, regardless of type of gray value discretization, steps must be taken to ensure good comparability, as the first order features largely use the original gray values (without discretization).

Finally, there is the issue of what value to use for the width of the bin. Again, there are currently no specific guidelines from literature as to what constitutes an optimal bin width. We try to choose a bin width in such a way, that the resulting amount of bins is somewhere between 30 and 130 bins, which shows good reproducibility and performance in literature for a fixed bin count[2]. This allows for differing ranges of intensity in ROIs, while still keeping the texture features informative (and comparable inter lesion!).

### What modalities does PyRadiomics support? 2D? 3D? Color?

PyRadiomics is not developed for one specific modality. Multiple modalities can be processed by PyRadiomics, although the optimal settings may differ between modalities. There are some constraints on the input however:

1. Gray scale volume: PyRadiomics currently does not provide extraction from color images or images with complex values. In those cases, each pixel/voxel has multiple values and PyRadiomics does not know how you'd want to combine those. It is possible to select a color channel and use that as input:

```python
import SimpleITK as sitk

from radiomics import featureextractor

# Instantiate extractor with parameter file
extractor = featureextractor.RadiomicsFeatureExtractor(r'path/to/params.yml')

# Set path to mask
ma_path = 'path/to/maskfile.nrrd'
label = 1  # Change this if the ROI in your mask is identified by a different␣
↪value

# Load the image and extract a color channel
color_channel = 0
im = sitk.ReadImage(r'path/to/image.jpg')
```

(continues on next page)

[1] Leijenaar RTH, Nalbantov G, Carvalho S, van Elmpt WJC, Troost EGC, Boellaard R, et al. ; The effect of SUV discretization in quantitative FDG-PET Radiomics: the need for standardized methodology in tumor texture analysis; Sci Rep. 2015;5(August):11075

[2] Tixier F, Cheze-Le Rest C, Hatt M, Albarghach NM, Pradier O, Metges J-P, et al. *Intratumor Heterogeneity Characterized by Textural Features on Baseline 18F-FDG PET Images Predicts Response to Concomitant Radiochemotherapy in Esophageal Cancer.* J Nucl Med. 2011;52:369–78.

(continued from previous page)

```
selector = sitk.VectorIndexSelectionCastImageFilter()
selector.SetIndex(color_channel)
im = selector.Execute(im)

# Run the extractor
results = extractor.execute(im, ma_path, label=label)
```

2. File format: Currently, PyRadiomics requires the image and mask input to be either a string pointing to a single file containing the image/mask, or a SimpleITK.Image object (only possible in interactive mode). When e.g. using DICOMs, the separate files need to be first combined into 1 volume prior to extracting features by either converting to e.g. NRRD or NIFTII, or reading the DICOM in a python script and calling PyRadiomics from that script. See also *What file types are supported by PyRadiomics for input image and mask?*.

3. Dimensionality: PyRadiomics supports both 2D and 3D input images, but be aware that feature class `shape` only extracts 3D shape descriptors and `shape2D` only 2D shape descriptors. If you have a 3D volume, but a single-slice segmentation and want the results to include 2D shape descriptors, enable `shape2D` and set `force2D=True`. This allows you to extract 2D shape features from a 3D volume with single-slice segmentation (but fails when segmentation represents a volume segmentation spanning multiple slices).

### What file types are supported by PyRadiomics for input image and mask?

PyRadiomics uses SimpleITK for image loading and handling. Therefore, all image types supported by SimpleITK can be used as input for PyRadiomics. Please note that only one file location can be provided for image/mask.

If your input images are DICOM, things become more complicated. If you want to process a single 2D image slice stored in DICOM format, you can use it as any other format. However, if you are processing a volumetric dataset, you should first confirm the DICOM files you have correspond to a single image series. If you are not sure, you can sort the data such that you have a single directory per series using, for example, dicomsort. You can then convert the DICOM series into an ITK-readable volumetric format using plastimatch convert or dcm2niix.

If your label is defined in DICOM format, this can mean different things. First, check what is the modality of the dataset with the label. You can check that by using dcmdump, and then checking the line that says "Modality". You can find the binary packages of this tool here (you can also use dicom-dump package if you want to look at DICOM files more conveniently from the Atom editor).

- If the modality is an image (e.g., CT or MR), use *plastimatch* or *dcm2niix* to convert the image into a 3D volume.
- If the modality is RT, use *plastimatch* to convert contours of the structure sets into 3D volumes.
- If the modality is SEG, use dcmqi to convert voxel segmentations into 3D volumes.

We also provide a "labs" (experimental) script pyradiomics-dcm that can do those conversions automatically and also saves the resulting features as DICOM SR.

### I want to customize my extraction. How do I do that?

See also *Customizing the Extraction*. PyRadiomics can be customized in various ways, but it's most easy to do this by providing a *parameter file*. In this yaml structured text file you can define your custom settings and which features and input image types to enable. We strongly recommend to use this method for customization, as it contains all the customization in 1 human-readable file, which can be shared for other users and, more importantly, is checked for validity prior to the extraction.

#### Does PyRadiomics support voxel-wise feature extraction?

Yes, as of version 2.0, voxelwise calculation has been implemented. However, as this entails the calculations of features for each voxel, performing a voxelwise extraction is much slower and as the output consists of a feature map for each feature, output size is also much larger. See more on enabling a voxel-based extraction in the *usage section*.

#### How should the input file for `pyradiomics` in batch-mode be structured?

Currently, the batch input file for `pyradiomics` is a csv file specifying the combinations of images and masks for which to extract features. It must contain a header line, where at least header "Image" and "Mask" should be specified (capital sensitive). These identify the columns that contain the file location of the image and the mask, respectively. Each subsequent line represents one combination of an image and a mask. Additional columns are also allowed, these are copied to the output in the same order as the input, with the additional columns of the calculated features appended at the end. *N.B. All header names should be unique and not match any of the produced header names by pyradiomics.*

## 2.10.2 Common Errors

#### Error loading parameter file

When I try to load my own parameter file, I get error:"CoreError: Unable to load any data from source yaml file"

This error is thrown by PyKwalify when it is unable to read the parameter file. The most common cause is when the file is indented using tabs, which throws a "token ('t') not recognized error". Instead, ensure that all indentations are made using 2 or 4 spaces.

#### Geometry mismatch between image and mask

My mask was generated using a another image than my input image, can I still extract features?

For various reasons, both image and mask must have the same geometry (i.e. same spacing, size, direction and origin) when passed the feature classes. To this end PyRadiomics includes checks in the pipeline to ensure this is the case. For more information on the mask checks, see `checkMask()`. If the geometry error is due to a small difference in origin, spacing or direction, you can increase the tolerance by setting `geometryTolerance`. If the error is large, or the dimensions do not match, you could also resample the mask to image reference space. An example of this is provided in `bin\resampleMask.py` and can be enabled in PyRadiomics by setting `correctMask` to `True`, which will only perform this correction in case of a geometery mismatch where the mask contains a valid ROI (i.e. the mask contains the label value which does not include areas outside image physical bounds).

#### ValueError: ('Label (. . . ) not present in mask. Choose from [. . . ]'

This error indicates that the ROI identified by the value of `label` does not exist. I.e. there are no voxels in the mask volume that have the value specified in `label`. To help you fix the error, this error also lists the possible alternative label values that have been found. If no values are listed at the end of this error, it means that there are no segmented voxels in your mask.

**Note:** It is possible that in the original mask there were voxels segmented, but were lost during resampling (when upsampling). In that case, the ROI is too small for the requested `resampledPixelSpacing` and should be treated as 'nothing is segmented'.

### I'm unable to calculate texture matrices and getting a RunTimeError instead

This error means that something went wrong during the calculation of the matrices in the C extensions. There are several potential causes for this error:

- "Error parsing array arguments."

This error is thrown when either the Image or the Mask provided to the function could not be interpreted as a numpy array.

- "Expected image and mask to have equal number of dimensions."

Thrown when the Image and Mask Provided did not have the same number of dimensions. N-Dimensional extraction is supported, but the image and mask are still required to match in both the size and number of dimensions.

- "Dimensions of image and mask do not match."

This means that the size of the mask array does not match the size of the image array. Because numpy arrays do not contain information on the transformation to the physical world, input arrays of differing sizes cannot be matched. You can solve this error by resampling the SimplITK-Image object of the Mask to the geometry of the Image before converting them to their respective numpy arrays for feature calculation. See also *Geometry mismatch between image and mask*.

- "Error parsing distances array."

This error is shown if the C extension was not able to interpret the distances argument that was provided. In the settings, the `distances` parameter should be either a tuple or a list of values.

- "Expecting distances array to be 1-dimensional."

Again an error in the provided distances. The list provided should be 1 dimensional (i.e. no nested lists).

- "Error calculating angles."

This error means there was an issue in generating the angles based on the distances provided. Currently, this only occurs when distances < 1 are provided.

- "Number of elements in <Matrix> would overflow index variable! (...)"

This error is shown when the size of the (flattened) output array would be larger than the maximum integer value (~2 mln). This is generally caused by a too large number of bins after discretization, resulting in a too large range of gray values in the discretized image used for texture calculation. We generally advise to chose a bin width so, that the number of bins after discretization does not exceed 150-200. Running the code with DEBUG logging enabled shows the number of bins that are generated and may help to give an indication as to how large your matrices are.

- "Failed to initialize output array for <Matrix>"

This means that the computer was unable to allocate memory for the output. This is most likely due to a too large output size or too little free memory being available. Similar as above, run with DEBUG logging to see how many bins are generated (giving an indication on how large the output matrices are).

- "Calculation of <Matrix> Failed."

This error means there was a problem in the calculation of the matrix itself. It is generally thrown if the code tries to set an element in the output array that is out-of-range. This can happen if there are voxels inside the ROI that have gray values that are larger than the `Ng` parameter that is provided when calling the C function from Python.

### I'm able to extract features, but many are NaN, 0 or 1. What happened?

It is possible that the segmentation was too small to extract a valid texture. Check the value of `VoxelNum`, which is part of the additional information in the output. This is the number of voxels in the ROI after pre processing and therefore the number of voxels that are used for feature calculation.

Another problem can be that you have to many or too few gray values after discretization. You can check this by comparing the range of gray values in the ROI (a First Order feature) with the value for your `binWidth` parameter. More bins capture smaller differences in gray values, but too many bins (compared to number of voxels) will yield low probabilities in the texture matrices, resulting in non-informative features. There is no definitive answer for the ideal number of discretized gray values, and this may differ between modalities. One study[2] assessed the number of bins in PET and found that in the range of 16 - 128 bins, texture features did not differ significantly.

#### I'm missing features from my output. How can I see what went wrong?

If calculation of features or application of filters fails, a warning is logged. If you want to know exactly what happens inside the toolbox, PyRadiomics provides extensive debug logging. You can enable this to be printed to the out, or stored in a separate log file. The output is regulated by `radiomics.setVerbosity()` and the PyRadiomics logger can be accessed via `radiomics.logger`. See also *here* and the examples included in the repository on how to set up logging.

#### Radiomics module not found in jupyter notebook

I installed PyRadiomics, but when I run the jupyter notebook, I get `ImportError: No module named radiomics`

This can have two possible causes:

1) When installing PyRadiomics from the repository, your python path variable will be updated to enable python to find the package. However, this value is only updated in commandline windows when they are restarted. If your jupyter notebook was running during installation, you first need to restart it.
2) Multiple versions of python can be installed on your machine simultaneously. Ensure PyRadiomics is installed on the same version you are using in your Jupyter notebook.

### 2.10.3 Building PyRadiomics from source

#### During setup, python is unable to compile the C extensions.

This can occur when no compiler is available for python. If you're installing on Windows, you can find free compilers for python here.

#### Error loading C extensions.

When I try to run PyRadiomics, I get `Error loading C extensions`

When PyRadiomics is installed, the C extensions are compiled and copied to the installation folder, by default the `site-packages` folder. However, when the notebook is run form the repository, it is possible that PyRadiomics uses the source code directly (i.e. runs in development mode). You can check this by checking the `radiomics.__path__` field, which will be something like `['radiomics']` when it is running in development mode and `['path/to/python/Lib/site-packages']` when running from the installed folder. If running in development mode, the C extensions are not available by default. To make them available in development mode, run `python setup.py build_ext --inplace` from the commandline, which is similar to the `install` command, but just compiles the C extensions end copies them to the source folder (making them available when running from the source tree).

### 2.10.4 Miscellaneous

#### Which python versions is PyRadiomics compatible with?

PyRadiomics is compatible with python 3. Python 2 support was dropped in PyRadiomics version 3.0, though compatibility code was retained. However, the automated testing only uses python versions 3.5, 3.6 and 3.7 (64 bits architecture). Python < 2.6 is not supported. Other python versions may be compatible with PyRadiomics, but this is not actively tested and therefore not guaranteed to work. Pre-built wheels are only available for the tested versions of python (3.5, 3.6 and 3.7)

#### A new version of PyRadiomics is available! Where can I find out what changed?

When a new version is released, a changelog is included in the release statement. Between releases, changes are not explicitly documented, but all significant changes are implemented using pull requests. Check the merged pull request for the latest changes.

#### I have some ideas for PyRadiomics. How can I contribute?

We welcome suggestions and contributions to PyRadiomics. Check our guidelines to see how you can contribute to PyRadiomics. Signatures and code styles used in PyRadiomics are documented in the *Developers* section.

#### I found a bug! Where do I report it?

We strive to keep PyRadiomics as bug free as possible by thoroughly testing new additions before including them in the stable version. However, nothing is perfect, and some bugs may therefore exist. Report yours by opening an issue on the GitHub. If you want to help in fixing it, we'd welcome you to open up a pull request with your suggested fix.

#### My question is not listed here...

If you have a question that is not listed here, check the pyradiomics forum on 3D Slicer discourse or the issues on GitHub. Feel free to post a new question or issue and we'll try to get back to you ASAP.

## 2.11 Release Notes

### 2.11.1 Next Release

### 2.11.2 PyRadiomics 3.0.1

#### Bug Fixes

- Fix bug causing `IndexError` when no gray levels are 'empty'. (#592)
- Fail initialization of feature extractor when the passed parameter file path does not point to existing file. (#587)
- Fix out-of-range check in GLSZM C calculation. (#635)
- Fix bug in Travis CI testing (MacOS platform). (#643, #646)
- Fix cmake URL and remove python2 support from DockerFiles. (#645)

#### Examples

- Add example settings for forced-2D extraction in MR. (#613, #644)

#### Documentation

- Fix typos in documentation. (9d26a6b8, 896682d7, e100f1d0, #639)
- Further clarify resampling. (#599)

#### Internal API

- Fail gracefully when grayvalues < 1 are encountered in the discretized image. (#602)
- Add optional progress reporting for voxel-based extraction. (#636)

#### Labs

- Expose mask correction and geometry tolerance settings in pyradiomics-dcm CLI (#724)

### 2.11.3 PyRadiomics 3.0

**Warning:** As of this release, Python 2.7 testing is removed. Compatibility code such as it is will be left in place, but future changes will not be checked for backwards compatibility. Moreover, no pre-built binaries for python 2.7 will be distributed on PyPi or Conda. Finally, some deprecated code is removed (`commandlinebatch.py` and `calculateFeatures()`).

#### Bug Fixes

- Fix broken Conda deployment (51c5849)
- Fix error in IBSI mapping (labs/pyradiomics-dcm) (54d6689)
- Fix resampling error when spacing is correct, but sizes are different (ac7458e)
- Fix label channel selection (54a3782)
- Use local scope of settings, preventing race conditions in parallel extraction (43578f7)
- Fix resampling for 2D input (#545)

#### Internal API

- Update C API to use large datatype for index pointers (#500, #501)
- Update docker CLI to use python 3.6.9 and fix bugs to allow integration with pyradiomics-dcm lab (#527)
- Add option to force path to UNIX style paths, even on windows (3c0708a)
- Removed deprecated code (fedaa5e)

#### Testing

- Remove testing and deployment for python 2.7 (a5a7e61)

#### Documentation

- Refactor documentation (#536)
- Fix various typos/wording
- Clarify use of force2D, and add example settings file (#558)

### 2.11.4 PyRadiomics 2.2.0

**Warning:** In this release, the main interface class, `RadiomicsFeaturesExtractor`, was renamed to `RadiomicsFeatureExtractor` (no 's' between 'Feature' and 'Extractor'). This was done to avoid confusion between the module and class name. (#481)

#### New Features

- Add 2D shape features (#442)
- Expose voxel-based feature extraction on the PyRadiomics command line interface. (#457)

#### Labs

- Add notebook investigating reproducibility between PyRadiomics and USF tool (ITK-based; #458)

#### Bug Fixes

- Flatten array when applying gray value discretization of the entire image (voxel-based, full kernel; f87abcf)
- Fix incorrect removal of 'empty gray levels' in GLDM and GLRLM (voxel-based; 4b18ce2)
- Fix incorrect instantiation of firstorder voxel-based extraction. (81e713a)
- Force cast coefficients to float. Prevents overflow and type errors in feature calculation. (e9d60c7)

#### Tests

- Removed support and continuous integration for Python 3.4 (not maintained since March 2019). Added support and CI for Python 3.7. (#486)

#### Internal API

- Update C-extensions:
  - Rewrite C code to work with N-Dimensional input. (#463)
  - Add batch-calculation of kernels and vectorized feature calculation to improve voxel-based extraction duration. (#466)

- Add support for segmentation objects (multi-layer labelmaps; #445)
- Refactor the commandline interface (#481)
  - Extractor instantiated once (resulting in only 1 validation of the parameter file, outside of paralellization loop)
  - Simplify construction of the python generator of the cases that are to be extracted
  - Remove now unnecessary functions

#### Documentation

- Update documentation (#446, 690891d)
- Fix some rendering errors (723d868, e3eb427)

### 2.11.5 PyRadiomics 2.1.2

#### Labs

- Include algorithm details in dcm output. (f03145b)

### 2.11.6 PyRadiomics 2.1.1

#### New Features

- Implement validation of commandline input. (#433)
- Implement thread-safe logging for python >= 3.2 (#441, d8db675)

#### Labs

- Add script for using PyRadiomics with DICOM input and output. (#437)

#### Bug Fixes

- Fix memory error in calculation of GLCM-MCC. (167888b)
- Fix error in serialization for JSON output. (9d992fe)

#### Tests

- Expand testing to include more parts of PyRadiomics. (#410)

#### Internal API

- Force cast the mask to an integer datatype on load. (#431)

#### Dependencies

- Fix PyWavelets version to > 0.4.0, <= 1.0.0, due to compilation issue in SlicerRadiomics. (c828b99, SlicerRadiomics#50)

### 2.11.7 PyRadiomics 2.1.0

#### Feature Calculation Changes

- Switch Shape - Volume calculation to a mesh-based instead of a voxel-based one. This also affects all features derived from Volume. Original Volume calculation is retained as `VoxelVolume`. Also switch calculation of maximum diameter to mesh based. Only PCA-derived are not affected. (#427)

#### New Features

- Add GLCM - Maximal Correlation Coefficient. (#411)

#### New Parameters

- Update resegmentation function, add support for single (lower) threshold and new modes `relative` and `sigma`, customizable in parameter `resegmentMode`. (#420)
- Add `resegmentShape`. Default `False`, if set to `True`, the resegmented mask (intensity mask) will also be used for shape calculation. Otherwise, the non-resegmented mask (morphological mask) is used for shape. (#428)

#### Bug fixes

- Fix bug in dimension checking in `checkMask`. (623b836)
- Fix some errors in the testUtils and baseline generation script. (c285c15)
- Prevent division by 0 in NGTDM - Coarseness. Return 0 instead. (a59861e)
- Remove duplicate key in settings file example. (828a7ac)
- Prevent duplicate log entries in parallel batch extraction. (8cedd8f)
- Build PyWavelets from source for AppVeyor (Windows) python 3.4 testing. Requires pre-installation of numpy and cython. (6223d35)

#### Tests

- Integrate automatic distribution to conda upon release. (#422)

#### Documentation

- Update README and Setup.py with additional classifiers, urls. Update section in README on Docker usage. (0fe737e)

### Internal API

- Use `ValueError` exceptions when feature extraction pipeline fails (exceptions of individual features) (#420)
- Update generation and names of general info features (provenance information) (#420, #426)
- Rewrite signatures of pre-processing functions to accept all customization arguments in 1 `**kwargs` dict. Necessary parameters are obtained using `kwargs.get` inside the function. Full settings are passed to the function. (#425)

## 2.11.8 PyRadiomics 2.0.1

### New Features

- Add Center of Mass to general info output. (#416)

### Bug fixes

- Fix invocation of numpy.histogram when using a fixed bin count. (2a9fd79)
- Fix assignment of x and y pixelspacing in shape (no changes in results). (#404)
- Fix generation of approximation name (LLL or LL) in wavelet. (#405)
- Add missing requirements for new filters in Docker CLI file. (#409)
- Fix memory leak in C extensions. (#419)
- Fix Label column parsing in batch processing. (217a840)

### Documentation

- Fix math rendering in GLCM. (c6a1f21)
- Add reference to GLDM feature class. (9f9361a)
- Correct typo in IMC1 and 2 formulas. (4ba909a)
- Update warning message in ROI check. (1f16b9e)
- Update usage section in documentation on command line usage. (fe0e2c3)

### Internal API

- Simplify calculation of various GLCM features (no changes in results). (#407)

## 2.11.9 PyRadiomics 2.0.0

### Feature Calculation Changes

- Change calculation of filter coefficients to reflect absolute maximum (take into account negative values). (#319)
- Mark duplicate features as 'deprecated' and document mathematical proof of the equality. (#321)
- Fix error in calculation of NGTDM's Complexity and Contrast features (#351)

### New Features

- Add `preCrop`, which crops the image onto the bounding box with an additional padding specified in `padDistance`. This is similar to cropping as performed during resampling and serves to decrease memory consumption and computation time. N.B. To ensure calculated values are not changed, a sufficient padding is required when using filters which include values outside of ROI (e.g. Wavelet, LoG). (#317)
- Add `skip-nans` as a commandline argument. If specified, features that compute NaN are removed from the output. In batch mode, NaN is replaced by an empty string. (#318)
- Add support to configure the feature extractor using a JSON structured string. (#334)
- Add Gradient Magnitude Filter. (#356)
- Add Local Binary Pattern Filter (2D/3D). (#357)
- Add support for Gray Value discretization using a fixed bin count. (#386)

### Bug fixes

- Ensure PyKwalify has a log handler, which is needed when parameter file validation fails. (#309)
- Fix bug in error handling in *`checkMask()`* (compatibility issue between python 2 and 3).
- Fix bug in GLCM (incorrect use of `self.maskArray`) (#322)
- Fix bug in error handling during geometry checks of image and mask. (0257217)
- Fix broken continuous testing integration due to unavailability of pip script. (#333)
- Fix incorrect path separator in example scripts. (c7c5d2e)
- Fix bug in the calculation of Wavelet. (#346)
- Fix machine-precision errors in Eigenvalue calculation (Shape) (#355)
- Update validation rule for image filters (remove hardcoded filters by package-detected filters). (#364)
- Add missing requirements for LBP filters in the dockerfile. (#389)
- Fix deprecation error in feature extractor. (da1fc16)
- Fix axis definition in wavelet. (4027a52)
- Fix erroneous double return of wavelet approximation. (c8ceee2)

### Tests

- Improve testing badge layout. (#312)
- Remove unused testing configuration files. (#313)
- Add testing for wavelet output. (#387)
- Integrate publication to PyPi into the Continuous Integration, revise the CI workflow to test python 2.7, 3.4, 3.5 and 3.6 for all 3 platforms (Windows, Mac and Linux). **N.B. This makes PyRadiomics installable via pip** (#394)

#### Documentation

- Update documentation of `base.py` (#306)
- Update notebooks to reflect most recent version of PyRadiomics. (ac66e6c)
- Add documentation detailing rationale of enforcing a fixed bin width. (#320)
- Update reference to official publication. (b395904)
- Update installation instructions for docker. (#329)
- Add version of NumPy, SimpleITK and PyWavelet to the additional information in the output. (#342)
- Add documentation for the calculation of Laplacian of Gaussian. (#345)
- Add refrences for the newly implemented filters (4464d1c)
- Fix an error in the firstorder-Uniformity documentation. (da7321d)

#### Examples

- Add example for batchprocessing using a multithreaded approach. (#305)

#### Internal API

- Update batch script for the commandline interface. Ensures all required input is available and relative filepaths are relative to the input file, not the current working directory. (#307)
- Remove support for 32-bits python, as memory errors can arise when extracting from many or large images in 32-bits python. (#310)
- Simplify Calculation of Wavelet Filter. Does not change output. (#323)
- Refactor commandline interface to work with only 1 entry point (`pyradiomics`). Also add parallel-processing option for batch-processing (argument `-j`, which specifies number of CPU cores to use). (#347)
- Reconfigur testing to allow the removal of testcases from the repository itself (still available as binary data attached to release 1.0.0) and store the baseline in a different format (allowing for easier change-tracking) (#353)
- Add a check for number of bins generated (preventing construction of too large matrices in C) (#391, #393)

### 2.11.10 PyRadiomics 1.3.0

#### Feature Calculation Changes

- Remove feature *Sum Variance*, as this is mathematically equal to *Cluster Tendency*. (#300)
- Fix feature formula error in NGTDM (incorrect use of square in *Complexity* and *Contrast*). (#351)

#### New Features

- Add a row by row customization of the extraction label in the batch processing command line script, as well as both batchprocessing examples. (#262)

- Allow value 0 for a resampled pixel spacing (per dimension). Values of 0 are replaced by the spacing for that dimension as it is in the original (non-resampled) mask. This allows resampling over a subset of dimension (e.g. only in-plane resampling when out-of-plane spacing is set to 0). (#299)
- Add optional resegmentation of mask based on customizable threshold. (#302)
- Add Neighbouring Gray Tone Difference Matrix (NGTDM) (#296)
- Add Add Gray Level Dependence Matrix (GLDM) (#295)
- Add a docker file that exposes the PyRadiomics commandline tools. (#297, #301)
- Add voxel-based calculation, allowing for extraction of feature maps (values per voxel instead of per ROI). (#337)

## Bug fixes

- In GLCM, the matrix is made symmetrical by adding the transposed matrix. However, `numpy.transpose` returns a view and not a copy of the array, causing erroneous results when adding it to the original array. use `numpy.ndarray.copy` to prevent this bug. **N.B. This affects the feature values calculated by GLCM when symmetrical matrix is enabled (as is the default setting).** (#261)
- Use a python implementation to compute eigenvalues for `shape.py` instead of SimpleITK. The implementation in SimpleITK assumes segmented voxels to be consecutive on the x-axis lines. Furthermore, it also assumes that all voxels on a given line of x have the same values for y and z (which is not necessarily the case). (#264)
- Removal of outliers was not applied to returned object in `normalizeImage`. (#277)
- Fix python 3 incompatibility when using `urllib` (#285)
- Fix broken URL link in feature visualization notebooks.
- Update docker manually install python2 support (since recently not supported by default in jupyter/datascience-notebook). (#287)
- For GLRLM and GLSZM, force2D keyword is passed manually, but was incorrectly named and therefore ignored. Fix name to enable forced 2D extraction for GLRLM and GLSZM. (26b9ef3)
- Fix bug in the calculation of eigen values due to machine precision errors. (#355)

## Tests

- Update the C Matrices test, so that the C and python calculated matrices will have the same dimensions when compared (In the previous implementation, the `_calculateCoefficients` function was applied to the C calculated matrix, but not in the python calculated matrix, for some texture matrices, this function can change the dimension of the matrix). This update ensures that `_calculateCoefficients` is applied to neither matrix. (#265)
- Add a test to check validity of parameter files included in `examples/exampleSettings`. (#294)

## Documentation

version 1.3.0 docs

- Update reference. (#271)
- Move section "Customizing the Extraction" to the top level, to make it more visible. (#271)
- Change License to 3-clause BSD (#272

- Document the extend of compliance between PyRadiomics and the IBSI feature definitions (#289)
- Fix typos in documentation.
- Expand documentation on customizing the extraction (#291)
- Include contributing guidelines in sphinx-generated documentation and add a section on sharing parameter files. (#294)
- Insert missing line to enable all features in documentation on using the feature classes directly. (5ce9f48)
- Fix typo in NGTDM documentation. (ea9a6ce)
- Fix some typos in documentation of firstorder - std and gldm - GLN (#369)
- Add additional comments to the code of the Wavelet filter (`_swt3`). (#375)
- Add references to the new filter functions. (4464d1c)

#### Examples

- Add example settings for CT, MR (3 scenarios). (#273)

#### Internal API

- Remove unnecessary rows and columns from texture matrices prior to feature calculation. This does not affect the value of the calculated features, as the i and j vectors are updated accordingly, but it does reduce both computation time and memory requirements. This is especially the case when calculating GLSZM on large segmentations, where there may be many 'empty' zone sizes (i.e. no zones of that size are present in the ROI). This reduces the size of the matrix, which therefore reduces the memory needed and the number of calculations performed in the vectorized operations. (#265)
- Remove circular import statement in `__init__.py` (circular with `radiomics.base`) (#270)
- Revise initialization of the feature class. (#274)
- Rename parts of the customization variables and functions to better reflect their definition (#291)
- Update C extensions: Make python wrapping more similar for different feature classes, simplify calculation of surface area, remove deprecated Numpy C-API references and implement angle-generation in C. (#360)
- Remove Python equivalents of C extensions: Some, but not all C extensions had python equivalents, which calculated equal values but, by using a python-only implementation, are much slower than the C extension. Only advantage is that it would also work when compiling the code fails. Also update the tests to check consistency of the calculated matrices against a baseline file (binary numpy array file) instead of python calculated matrices. (#373)

#### License

- Switch to 3-clause BSD license. (#272)

### 2.11.11 PyRadiomics 1.2.0

### Feature Calculation Changes

- Remove feature *SumVariance*, rename *SumVariance2* to *SumVariance*. *SumVariance* reflected the formula as is defined in the paper by Haralick et al[1]. However, the variance is calculated by subtracting the entropy as opposed to subtracting the average, most likely due to a typo('f8' instead of 'f6'). *SumVariance2* reflected the formula where the average is subtracted and is retained as the only *SumVariance*. (#233)
- Redefine features *Elongation* and *Flatness* as the inverse of the original definition. This prevents a returned value of NaN when the shape is completely flat. (#234)
- In certain edge cases, the calculated maximum diameters may be too small when calculating using the python implementation. This is corrected by the C extension and a warning is now logged when calculating these features in python. **N.B. As of this change, maximum diameter is not available for calculation in full-python mode** (#257)
- For certain formulas, a NaN value is returned in some edge cases. Catch this and return a predefined value instead. Document this behaviour in the docstrings of the features affected. (#248)

### New Features

- Add Region of Interest checks. (#223, #227)
- Add variable column support for batch input file (#228)
- Add Docker support (#236)

### Bug fixes

- Instantiate output with input in `commandlinebatch`
- Correct `Np` when weighting is applied in GLRLM (#229)
- Update CSV generators to reflect variable number of columns for input CSV in batch processing (#246)
- Return corrected mask when it had to be resampled due to geometry mismatch errors (#260)

### Requirements

- Remove `tqdm` requirement (#232)
- Reorganize requirements, with requirements only needed during development moved to `requirements-dev.txt` (#231)

### Documentation

version 1.2.0 docs

- Update feature docstrings, making them more easily adaptable for article supplements (#233)
- Add FAQ concerning the cmatrices lib path (#233)
- Add developer install step to documentation (#245)
- Remove use of `sudo` (#233)
- Fix subclass name in feature class signature (section "Developers")

[1] Haralick R, Shanmugan K, Dinstein I: Textural features for image classification. IEEE Trans Syst Man Cybern 1973:610–621.

- Add subsection on customizing the extraction to the "Usage" section (#252)
- Remove SimpleITK installation workaround, this is no longer needed (#249)
- Add a changelog to keep track of changes and integrate this into the auto generated documentation (#255)

#### Examples

- Add `pandas` example, showing how to process PyRadiomics output/input using the `pandas` library (#228)

#### Internal API

- Add function to get or download test case (#235)
- Rewrite C Extension algorithm for GSLZM. Instead of searching over the image for the next voxel when growing a region, store all unprocessed voxels in a stack. This yields a significant increase in performance, especially in large ROIs. Requires slightly more memory (1 array, type integer, size equal to number of voxels in the ROI) (#257)
- Implement C extension for calculation of maximum diameters. (#257)

#### Cleanups

- Restructure repository (#254)
  - Move jupyter notebooks to separate root folder (`root/notebooks`)
  - Move example script to separate root folder (`root/examples`), with example settings in separate subfolder (`root/examples/exampleSettings`)
  - `bin` folder now only contains support scripts for the core code (i.e. generators for input files for batch processing and scripts to generate new baselines or to resample a mask to the image geometry)

### 2.11.12 PyRadiomics 1.1.1

#### Feature Calculation Changes

- Correct error in formula for *Compactness1*. **N.B. Baseline updated!** (#218)
- Remove feature *Roundness*, as this feature is identical to feature *Sphericity*, but uses different implementation for surface area calculation (all implemented in SimpleITK) (#218)
- Change handling of cases where `max(X) mod binwidth = 0` during image discretization. These used to be assigned to topmost bin, but this produces unexpected behaviour (i.e. in range 1, 2, 3, 4, 5 with binwidth 1, value 5 would be discretized to 4 in stead of 5). Value now assigned is topmost bin + 1 (in concordance with default behavior of `numpy.digitize`) (#219)
- Change default value for `voxelArrayShift` (from 2000 to 0), this is to prevent unknowingly using a too large shift when not necessary. Document effect of this parameter in the first order formulas affected. (#219)

#### New features

- Add forced 2D extraction (as alternative to resampling for handling anisotropy in voxels spacing)
- Enable specification of distances between neighbors for GLCM matrix calculation

(#215)

### New Parameters

- `force2D`, Boolean default `False`. Set to `True` to force a by slice texture calculation. Dimension that identifies the 'slice' can be defined in `force2Ddimension`. If input ROI is already a 2D ROI, features are automatically extracted in 2D.
- `force2Ddimension`, int, range 0-2, default 0. Specifies the 'slice' dimension for a by-slice feature extraction. Value 0 identifies the 'z' dimension (axial plane feature extraction), and features will be extracted from the xy plane. Similarly, 1 identifies the y dimension (coronal plane) and 2 the x dimension (saggital plane).
- `distances`, List of integers, default `[1]`. This specifies the distances between the center voxel and the neighbor, for which angles should be generated.

(#215)

### Bug fixes

- Add some missing python 3 compatibility lines to the supporting script `addClassToBaseline` and command line script `pyradiomicsbatch` (#210, #214)
- Fix bug when loading image as file path and mask as SimpleITK object. (#211)
- Change location of parameter schema files. These files are otherwise not included in the wheel distribution. (#221)

### Requirements

- Add sphinx_rtd_theme to requirements (needed to build documentation). (#222)

### Documentation

version 1.1.1 docs

- Split package documentation into "Pipeline Modules" (all non-feature-class modules) and "Feature Definitions" (feature class modules)
- Add developers section with documentation on how to implement new filters, feature and feature classes.
- Add FAQ section with some trouble shooting tips
- Rename some GLSZM features, this is to make them more consistent with GLRLM features, which are similar, but calculated on a different matrix
- Add documentation for Elongation and Flatness
- Document mathematical correlation between various Shape features.

(#216)

### Internal API

- Update logging with more extensive debug logging and more informative info log messages. (#220)

- Replace parameter verbose with output printing implemented in logging. Control verbosity level to output (stderr) by calling *`setVerbosity()`*, where level determines the verbosity level (as defined in python logging). This prints out the requested levels of the log messaging, where process reports with parameter verbose are now classified as INFO-level messages (i.e. specify INFO or DEBUG to enable these). **N.B. parameter verbose is not longer supported and will throw an error if passed in the parameter file** (#220)
- Add feature class and input image type checks in `featureextractor` when changing these settings. (#213)
- Remove usage of `eval` (replaced by implementations of `getattr`), this is a more secure approach. (#216)
- Define default settings in featureextractor in a separate function. This is to ensure consistency in applied default settings, as well as make them easily available outside of featureextractor (#216)
- Update reference for citing PyRadiomics (#224)

#### Cleanups

- Remove unused variable (`self.provenance_on` in `featureextractor`, this value is now replaced by a customizable setting)

### 2.11.13 PyRadiomics 1.1.0

#### New features

- Image normalization. This feature enables the normalization of image intensity values prior to feeding them to the extraction pipeline (i.e. before any other preprocessing steps are performed). Normalization is based on the all gray values contained within the image, not just those defined by the ROI in the mask.
- C Extensions for texture matrix and surface area calculation. These extensions enhance performance of texture matrix calculation associated GLCM, GLRLM and GLSZM features and of surface area calculation. Below shows the decrease in computation time for the 5 test cases included in PyRadiomics. (#158, #200, #202)
  - GLCM 6913 ms -> 3 ms
  - GLRLM 1850 ms -> 10 ms
  - GLSZM 12064 ms -> 58 ms
  - Surface Area 3241 ms -> 1 ms

#### New Parameters

- `additionalInfo` Boolean, default `True`. Enables additional information in the output if set to `True`. (#190)
- `enableCExtensions` Boolean, defailt `True`. Enables enhanced performance for texture matrix calculation using C extensions if set to `True`. (#202)
- `normalize` Boolean, default ‘‘ False‘‘. If set to true, normalizes image before feeding it into the extraction pipeline. (#209)
- `normalizeScale` Float, > 0, default 1. Enables scaling of normalized intensities by specified value. (#209)
- `removeOutliers` Float, > 0, default `None`. If set, outliers (defined by the value specified) are removed by setting them to the outlier value. Outlier value is defined on the non-scaled values. (#209)

### Bug fixes

- Unlink venv only when needed in Circle CI testing (#199)
- Fix datatype error when calling `SimpleITK.ResampleImageFilter.SetSize()` (only causes error in python 3, #205)

### Requirements

- Add requirement for `six>=1.10.0`, needed to make PyRadiomics compatible with both python 2 and 3.

### Documentation

version 1.1.0 docs

- Documentation on installation and usage is upgraded, with the addition of an embedded instruction video (in section "Usage", cued at the section on usage examples). (#187)
- Updated contact information to point to the google groups.
- Updated the classifiers in the setup script to reflect the more advanced status of Pyradiomics. (#193)

### Tests

- Add support for multiple python versions and platforms, now including python 2.7, 3.4, 3.5 (32/64bits) for Linux, Windows and Mac. (#183, #191, #199)
- Testing output is upgraded to ensure unique feature names (#195, #197)
- Add `test_cmatrices` to assert conformity between output from Python and C based texture matrix calculation.

### Internal API

- *`getFeatureClasses()`* and `getInputImageTypes()` are moved from *Feature Extractor <radiomics-featureextractor-label>* to the global radiomics namespace. This enumerates the possible feature classes and filters at initialization of the toolbox, and ensures feature classes are imported at initialization. (#190, #198)
- Python 3 Compatibility. Add support for compatibility with python 2.7 and python >= 3.4. This is achieved using package `six`.
- Standardize function names for calculating matrices in python and with C extensions to `_calculateMatrix` and `_calculateCMatrix`, respectively.
- Make C code consistent with C89 convention. All variables (pointers for python objects) are initialized at top of each block.
- Optimize GLSZM calculation (C extension)
  - Define temporary array for holding the calculated zones. During calculation, the matrix must be able to store all possible zones, ranging from zone size 1 to total number of voxels (Ns), for each gray level (Ng). In this case, the GLSZM would be initialized with size Ng * Ns, which is very memory intensive. Instead, use a temporary array of size (Ns * 2) + 1, which stores all calculated zones in pairs of 2 elements: the first element holds the gray level, the second the size of the calculated zone. The first element after the last zone is set to -1 to serve as a stop sign for the second function, which translates the temporary array into the final GLSZM, which can be directly initialized at optimum size.

  - Use `calloc` and `free` for the temporary array holding the calculated zones.
  - Use `char` datatype for mask. (signed char in GLSZM).
  - Uses `while` loops. This allows to reduce the memory usage. Additionally, we observed that with recursive functions it was 'unexpectedly' failing.
  - Optimized search that finds a new index to process in the region growing.

### 2.11.14 PyRadiomics 1.0.1

#### New features

- Added 2 commandline scripts ( pyradiomics and pyradiomicsbatch), which enable feature extraction directly from the commandline. For help on usage, run script with "-h" argument. (#188, #194, #196, #205)

#### Bug fixes

- Fix hardcoded label in shape (#175)
- Fix incorrect axis when deleting empty angles in GLCM (#176)
- Numpy slicing error in application of wavelet filters. This error caused the derived image to be erroneously rotated and flipped, with misaligned mask as a result.(#182)

#### Requirements

- Revert numpy minimum requirement to `1.9.2`. All operations in PyRadiomics are supported by this version, and it is the version used by Slicer. By reverting the minimum required version, installing PyRadiomics in the slicer extension does not cause an update of the numpy package distributed by slicer. (#180)

#### Documentation

version 1.0.1 docs

- Update on the documentation, reflecting recent changes in the code.
- Add developers and affiliations to ReadMe and documentation (#177)
- Added additional references and updated installation and usage section.

#### Internal API

- Different implementation of the various filters. No changes to calculation, but has a changed signature.

  **N.B. This results in inputImages to be differently defined (different capitalization, e.g. "orginal" should now be "Original"). See documentation for definition of inputImages (featureextractor section).**

### 2.11.15 PyRadiomics 1.0

#### New features

- Initial Release of PyRadiomics

### Work in progress

- Full python calculation (C matrices branch not stable and reserved for later release)

### Documentation

- Documentation published at readthedocs

CHAPTER 3

# Feature Classes

Currently supports the following feature classes:

- *First Order Statistics*
- *Shape-based (3D)*
- *Shape-based (2D)*
- *Gray Level Cooccurence Matrix* (GLCM)
- *Gray Level Run Length Matrix* (GLRLM)
- *Gray Level Size Zone Matrix* (GLSZM)
- *Neigbouring Gray Tone Difference Matrix* (NGTDM)
- *Gray Level Dependence Matrix* (GLDM)

On average, Pyradiomics extracts $\approx 1500$ features per image, which consist of the 16 shape descriptors and features extracted from original and derived images (LoG with 5 sigma levels, 1 level of Wavelet decomposistions yielding 8 derived images and images derived using Square, Square Root, Logarithm and Exponential filters).

Detailed description on feature classes and individual features is provided in section *Radiomic Features*.

# CHAPTER 4

# Filter Classes

Aside from the feature classes, there are also some built-in optional filters:

- `Laplacian of Gaussian` (LoG, based on SimpleITK functionality)
- `Wavelet` (using the PyWavelets package)
- `Square`
- `Square Root`
- `Logarithm`
- `Exponential`
- `Gradient`
- `Local Binary Pattern (2D)`
- `Local Binary Pattern (3D)`

For more information, see also *Image Processing and Filters*.

CHAPTER 5

# Supporting reproducible extraction

Aside from calculating features, the pyradiomics package includes additional information in the output. This information contains information on used image and mask, as well as applied settings and filters, thereby enabling fully reproducible feature extraction. For more information, see *General Info Module*.

CHAPTER 6

# 3rd-party packages used in pyradiomics

- SimpleITK (Image loading and preprocessing)
- numpy (Feature calculation)
- PyWavelets (Wavelet filter)
- pykwalify (Enabling yaml parameters file checking)
- six (Python 3 Compatibility)

See also the requirements file.

# CHAPTER 7

# Installation

PyRadiomics is OS independent and compatible with and Python >=3.5. Pre-built binaries are available on PyPi and Conda. To install PyRadiomics, ensure you have python installed and run:

- `python -m pip install pyradiomics`

For more detailed installation instructions and building from source, see *Installation* section.

# CHAPTER 8

# Pyradiomics Indices and Tables

- modindex
- genindex
- search

# CHAPTER 9

# License

This package is covered by the open source 3-clause BSD License.

# CHAPTER 10

# Developers

- Joost van Griethuysen[1,3,4]
- Andriy Fedorov[2]
- Nicole Aucoin[2]
- Jean-Christophe Fillion-Robin[5]
- Ahmed Hosny[1]
- Steve Pieper[6]
- Hugo Aerts (PI)[1,2]

[1]Department of Radiation Oncology, Dana-Farber Cancer Institute, Brigham and Women's Hospital, Harvard Medical School, Boston, MA, [2]Department of Radiology, Brigham and Women's Hospital, Harvard Medical School, Boston, MA [3]Department of Radiology, Netherlands Cancer Institute, Amsterdam, The Netherlands, [4]GROW-School for Oncology and Developmental Biology, Maastricht University Medical Center, Maastricht, The Netherlands, [5]Kitware, [6]Isomics

CHAPTER 11

# Contact

We are happy to help you with any questions. Please contact us on the Radiomics community section of the 3D Slicer Discourse.

We'd welcome your contributions to PyRadiomics. Please read the *contributing guidelines* on how to contribute to PyRadiomics. Information on adding / customizing feature classes and filters can be found in the *Developers* section.

# Python Module Index

# Index