pyradiomics Documentation

Release 1.3.0.post8+gaca0d08

pyradiomics community

Oct 09, 2017

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.

If you publish any work which uses this package, please cite the following publication: *Joost JM van Griethuysen, Andriy Fedorov, Chintan Parmar, Ahmed Hosny, Nicole Aucoin, Vivek Narayan, Regina GH Beets-Tan, Jean-Christophe Fillion-Robin, Steve Pieper, Hugo JWL Aerts, "Computational Radiomics System to Decode the Radiographic Phenotype"; Accepted Cancer Research 2017*

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

Table of Contents

Installation

Get the code

- Ensure you have the version control system `git` installed on your machine.
- Ensure that you have `python` installed on your machine, at least version 2.7 or 3.4.
- Clone the repository:
  - `git clone git://github.com/Radiomics/pyradiomics`

Installation on your system

- 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 develop`
  - 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`

Usage

Instruction Video

Example

- PyRadiomics example code and data is available in the Github repository
- The sample sample data is provided in `pyradiomics/data`
- Use jupyter to run the helloRadiomics example, located in `pyradiomics/examples/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/examples/Notebooks`
  - The parameter file used in the instruction video is available in `pyradiomics/examples/exampleSettings`
- If jupyter is not installed, run the python script alternative (`pyradiomics/examples/helloRadiomics.py`):
  - `python helloRadiomics.py`

Command Line Use

- PyRadiomics has 2 commandline scripts, `pyradiomics` is for single image feature extraction and `pyradiomicsbatch` is for feature extraction from a batch of images and segmentations.
- Both scripts can be run directly from a command line window, anywhere in your system.
- To extract features from a single image and segmentation run:

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

- To extract features from a batch run:

```
pyradiomicsbatch <path/to/input> <path/to/output>
```

- 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 label values for 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.

Note: All headers should be unique and different from headers provided by PyRadiomics (`<filter>_<class>_<feature>`).

- For more information on the possible command line arguments, run:

```
pyradiomics -h
pyradiomicsbatch -h
```

Interactive Use

- (LINUX) Add pyradiomics to the environment variable PYTHONPATH:
  - `setenv PYTHONPATH /path/to/pyradiomics/radiomics`
- Start the python interactive session:
  - `python`
- Import the necessary classes:

```
from radiomics import featureextractor, getTestCase
import six
import sys, os
```

- 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.
- 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.RadiomicsFeaturesExtractor(params)
```

- Calculate the features:

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

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

PyRadiomics in 3D Slicer

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

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 example.

- (LINUX) Add pyradiomics to the environment variable PYTHONPATH:
  - `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.

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 INFO and up (giving some information on progress during extraction and 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
# 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 `--log-level` argument.

Customizing the Extraction

Types of Customization

There are 3 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.

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).

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.

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.

By default, all feature classes and all features are enabled.

Currently available feature classes are:

- firstorder
- shape
- glcm
- glrlm
- glszm

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: 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

- 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) mask, thereby enabling only in-plane resampling, for example.
- interpolator [sitkBSpline]: SimpleITK constant or string name thereof, sets interpolator to use for resampling. 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, ≥ 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*

Resegmentation

- resegmentRange [None]: List of 2 floats, specifies the lower and 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 the mask is reset to the original mask.

Note: This only affects first order and texture classes. No resegmentation is performed prior to calculating shape features.

Mask validation

- 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 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

- enableCExtensions [True]: Boolean, set to False to force calculation to full-python mode. See also *enableCExtensions()*.
- 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]

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.

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. See also *`generateAngles()`*
- force2Ddimension [0]: int, range 0-2. 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). if `force2Dextraction` is set to False, this parameter has no effect. See also *`generateAngles()`*

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. See also *`generateAngles()`*

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, α cutoff value for dependence. A neighbouring voxel with gray level j is considered dependent on center voxel with gray level i if $|i - j| \leq \alpha$

Parameter File

All 3 categories of customization can be provided in a single yaml-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: 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.

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.
    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.

Pipeline Modules

This section contains the documentation on the various modules used to define the PyRadiomics pipeline and preprocess 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.

Feature Extractor

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

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 can be provided containing all necessary settings. This is done by passing the location of the file as the single argument in the initialization call, without specifying it as a keyword argument. If such a file location is provided, any additional kwargs are ignored. Alternatively, 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`). 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.

`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.

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

disableAllFeatures()
Disable all classes.

enableFeatureClassByName(*featureClass*, *enabled=True*)
Enable or disable all features in given class.

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.

execute(*imageFilepath*, *maskFilepath*, *label=None*)
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.

4. Shape features are calculated on a cropped (no padding) version of the original image.

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.

Returns dictionary containing calculated signature ("<imageType>_<featureClass>_<featureName>":value).

loadImage (*ImageFilePath*, *MaskFilePath*)

Preprocess 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 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.

getProvenance (*imageFilepath*, *maskFilepath*, *mask*)

Generates provenance information for reproducibility. Takes the original image & mask filepath, as well as the resampled mask which is passed to the feature classes. Returns a dictionary with keynames coded as "general_info_<item>". For more information on generated items, see *generalinfo*

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

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

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

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

---

getFeatureClassNames ()

Returns a list of all possible feature classes.

getFeatureNames (*featureClassName*)

Returns a list of all possible features in provided featureClass

Image Processing and Filters

`radiomics.imageoperations.`**getBinEdges** (*binwidth*, *parameterValues*)

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

Parameter `binWidth` determines the fixed width of each bin. This ensures comparable voxels after binning, a fixed bin count would be dependent on the intensity range in the segmentation.

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 $\leq \min(X_{gl})$.

Example: for a ROI with values ranging from 54 to 166, and a bin width of 25, the bin edges will be [50, 75, 100, 125, 150, 175].

This value can be directly passed to `numpy.histogram` to generate a histogram or `numpy.digitize` to discretize the ROI gray values. See also *binImage()*.

References

•Leijenaar RTH, Nalbantov G, Carvalho S, 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.

`radiomics.imageoperations.`**`binImage`**(*binwidth*, *parameterMatrix*, *parameterMatrixCoordinates*)

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.

$X_{b,i} = \lfloor \frac{X_{gl,i}}{W} \rfloor - \lfloor \frac{\min(X_{gl})}{W} \rfloor + 1$

Here, $X_{gl,i}$ and $X_{b,i}$ are gray level intensities before and after discretization, respectively. W is the bin width value (specfied in `binWidth` parameter). The first part of the formula ensures that the bins are equally spaced from 0, whereas the second part ensures that the minimum gray level intensity inside the ROI after binning is always 1.

If the range of gray level intensities is equally dividable by the binWidth, i.e. $(\max(X_{gl}) - \min(X_{gl})) \mod W = 0$, the maximum intensity will be encoded as numBins + 1, therefore the maximum number of gray level intensities in the ROI after binning is number of bins + 1.

Warning: This is different from the assignment of voxels to the bins by `numpy.histogram` , which has half-open bins, with the exception of the rightmost bin, which means this maximum values are assigned to the topmost bin. `numpy.digitize` uses half-open bins, including the rightmost bin.

Note: This method is slightly different from the fixed bin size discretization method described by IBSI. The two most notable differences are 1) that PyRadiomics uses a floor division (and adds 1), as opposed to a ceiling division and 2) that in PyRadiomics, bins are always equally spaced from 0, as opposed to equally spaced from the minimum gray level intensity.

`radiomics.imageoperations.`**`generateAngles`**(*size*, ***kwargs*)

Generate all possible angles for specified distances in `distances` in 3D. E.g. for d = 1, 13 angles are generated and for d = 2, 49 angles are generated (representing the 26 connected region for distance 1, and the 98 connected region for distance 2). Angles are generated with the following steps:

1. All angles for distance = 1 to the maximum distance specified in `distances` are generated.
2. Only angles are retained, for which the maximum step size in any dimension (i.e. the infinity norm distance from the center voxel) is present in `distances`.
3. "Impossible" angles (where 'neighbouring' voxels will always be outside delineation) are deleted.

4. If `force2Dextraction` is enabled, all angles defining a step in the `force2Ddimension` are removed (e.g. if this dimension is 0, all angles that have a non-zero step size at index 0 (z dimension) are removed, resulting in angles that only move in the x and/or y dimension).

Parameters

- **`size`** – dimensions (z, x, y) of the bounding box of the tumor mask.
- **`kwargs`** – The following additional parameters can be specified here (default values in brackets):
  - distances [[1]]: List of integers. This specifies the distances between the center voxel and the neighbor, for which angles should be generated.
  - 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. 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). if `force2Dextraction` is set to False, this parameter has no effect.

Returns numpy array with shape (N, 3), where N is the number of unique angles

`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 (if not None) 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, an error is logged and a (None,None) tuple is returned. 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*)

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*, *resampledPixelSpacing*, *interpolator=3*, *label=1*, *padDistance=5*)

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*, *scale=1*, *outliers=None*)

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 segementation.

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

Where:

- x and $f(x)$ are the original and normalized intensity, respectively.
- μ_x and σ_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*, *resegmentRange*, *label=1*)

Resegment the Mask based on the range specified in `resegmentRange`. All voxels with a gray level outside the range specified are removed from the mask. The resegmented mask is therefore always equal or smaller in size than the original mask. The resegemented mask is then checked for size (as specified by parameter `minimumROISize`, defaults to minimum size 1). When this check fails, an error is logged and maskArray is reset to the original mask.

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

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

Following settings are possible:

- 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 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.

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

`radiomics.imageoperations.`**`getWaveletImage`**(*inputImage*, ***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*, ***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*, ***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.

$$f(x) = \begin{cases} \sqrt{cx} & \text{for} \quad x \geq 0 \\ -\sqrt{-cx} & \text{for} \quad x < 0 \end{cases}, \text{ where } c = \max(x)$$

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*, ***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.

$$f(x) = \begin{cases} c\log(x+1) & \text{for} \quad x \geq 0 \\ -c\log(-x+1) & \text{for} \quad x < 0 \end{cases}, \text{where } c = \begin{cases} \frac{\max(x)}{\log(\max(x)+1)} & if \quad \max(x) \geq 0 \\ \frac{\max(x)}{-\log(-\max(x)-1)} & if \quad \max(x) < 0 \end{cases}$$

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*, ***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)

General Info Module

class radiomics.generalinfo.**GeneralInfo**(*imagePath*, *maskPath*, *resampledMask*, *settings*, *enabledImageTypes*)

execute()

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.

getBoundingBoxValue()

Calculate and return the boundingbox extracted using 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.

Values are based on the resampledMask.

getGeneralSettingsValue()

Return a string representation of the general settings. Format is {<settings_name>:<value>, ...}.

getImageHashValue()

Returns the sha1 hash of the image. This enables checking whether two images are the same, regardless of the file location.

If the reading of the image fails, an empty string is returned.

getImageSpacingValue()

Returns the original spacing (before any resampling) of the image.

If the reading of the image fails, an empty string is returned.

getEnabledImageTypesValue()

Return a string representation of the enabled image types and any custom settings for each image type. Format is {<imageType_name>:{<setting_name>:<value>, ...}, ...}.

getMaskHashValue()
Returns the sha1 hash of the mask. This enables checking whether two masks are the same, regardless of the file location.

If the reading of the mask fails, an empty string is returned. Uses the original mask, specified in maskPath.

getVersionValue()
Return the current version of this package.

getVolumeNumValue()
Calculate and return the number of zones within the mask for the specified label. A zone is defined as a group of connected neighbours that are segmented with the specified label, and a voxel is considered a neighbour using 26-connectedness for 3D and 8-connectedness for 2D.

Values are based on the resampledMask.

getVoxelNumValue()
Calculate and return the number of voxels that have been segmented using the specified label.

Values are based on the resampledMask.

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.

•binWidth: bin width, as specified in `**kwargs`. If key is not present, a default value of 25 is used.

•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.
- boundingBoxSize: tuple of 3 integers containing the z, x and y sizes of the ROI bounding box, respectively.
- 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.

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 list of the feature names (`[<Feature1>, <Feature2>, ...]`).

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

calculateFeatures()
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.

Global Toolbox Functions

radiomics.**cMatsEnabled**()
Returns a boolean indicating whether or not the C extensions are enabled. This function is called by the feature classes to switch between C-enhanced calculation and full python mode.

radiomics.**enableCExtensions**(*enabled=True*)
By default, calculation of GLCM, GLRLM and GLSZM is done in C, using extension `_cmatrices.py`

If an error occurs during loading of this extension, a warning is logged and the extension is disabled, matrices are then calculated in python. The C extension can be disabled by calling this function as `enableCExtensions(False)`, which forces the calculation of the matrices to full-python mode.

Re-enabling use of C implementation is also done by this function, but if the extension is not loaded correctly, a warning is logged and matrix calculation is forced to full-python mode.

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*, *repoDirectory=None*)

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

- brain1
- brain2
- breast1
- lung1
- lung2

If the repository is available locally (including all five test cases, the path to the root folder of the repository can be specified in `repoDirectory`, preventing unnecessary downloads. If the repository is not found, or

the repository does not contain the requested test case, PyRadiomics checks if it is run in development mode (directly from the source code in the repository), and if so, if it can find the test case relative to it's own location.

If the requested test case could not be found in the repository, PyRadiomics downloads the test case from the GitHub repository and stores it in temporary files. If the test case was already downloaded, this is returned instead.

Returns a tuple of two strings: `(path/to/image.nrrd, path/to/mask.nrrd)`

`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.*

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 Features* (19 features)
- *Shape Features* (16 features)
- *Gray Level Co-occurrence Matrix (GLCM) Features* (26 features)
- *Gray Level Size Zone Matrix (GLSZM) Features* (16 features)
- *Gray Level Run Length Matrix (GLRLM) Features* (16 features)
- *Neigbouring Gray Tone Difference Matrix (NGTDM) Features* (5 features)
- *Gray Level Dependence Matrix (GLDM) Features* (15 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.

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 voxels included in the ROI
- $\mathbf{P}(i)$ be the first order histogram with N_l discrete intensity levels, where N_l 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 $\frac{\mathbf{P}(i)}{\sum \mathbf{P}(i)}$

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

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

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.

---

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

getTotalEnergyFeatureValue()

2. Total Energy

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

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 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

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

Here, ϵ 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 10^{th} percentile of $\mathbf{X}$

get90PercentileFeatureValue()

6. 90th percentile

The 90^{th} percentile of $\mathbf{X}$

getMaximumFeatureValue()

7. Maximum

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

The maximum gray level intensity within the ROI.

getMeanFeatureValue()

8. Mean

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

The average gray level intensity within the ROI.

getMedianFeatureValue()

9. Median

The median gray level intensity within the ROI.

getInterquartileRangeFeatureValue()

10. Interquartile Range

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

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

getRangeFeatureValue()

11. Range

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

The range of gray values in the ROI.

getMeanAbsoluteDeviationFeatureValue()

12. Mean Absolute Deviation (MAD)

$$MAD = \frac{1}{N}\sum_{i=1}^{N}|\mathbf{X}(i) - \bar{X}|$$

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)

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

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 10th and 90th percentile.

getRootMeanSquaredFeatureValue()

14. Root Mean Squared (RMS)

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{X}(i) + c)^2}$$

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

$$standard\ deviation = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{X}(i) - \bar{X})^2}$$

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

Note: Not present in IBSI feature definitions (correlated with variance)

---

getSkewnessFeatureValue(*axis=0*)

16. Skewness

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

Where μ_3 is the 3[rd] 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 4[rd] central moment will be both 0. In this case, a value of 0 is returned.

---

getKurtosisFeatureValue(*axis=0*)

17. Kurtosis

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

Where μ_4 is the 4[th] 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 4[rd] 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

$$variance = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{X}(i) - \bar{X})^2$$

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, $variance = \sigma^2$

getUniformityFeatureValue()

19. Uniformity

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

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

Note: Defined by IBSI as Intensity Histogram Uniformity.

Shape Features

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.

Let:

- V the volume of the ROI in mm^3
- A the surface area of the ROI in mm^2

getVolumeFeatureValue()

1. Volume

$$V = \sum_{i=1}^{N} V_i$$

The volume of the ROI V is approximated by multiplying the number of voxels in the ROI by the volume of a single voxel V_i.

Note: In the IBSI feature definitions, a more precise approximation of the volume is used. That method uses tetrahedrons consisting of the origin and faces in the ROI. Although the method implemented here overestimates the volume, especially in small volumes, the difference will be negligible in large ROIs.

getSurfaceAreaFeatureValue()

2. Surface Area

$$A = \sum_{i=1}^{N} \frac{1}{2} |\mathrm{a}_i\mathrm{b}_i \times \mathrm{a}_i\mathrm{c}_i|$$

Where:

N is the number of triangles forming the surface mesh of the volume (ROI)

$\mathrm{a}_i\mathrm{b}_i$ and $\mathrm{a}_i\mathrm{c}_i$ are the edges of the i^{th} triangle formed by points a_i, b_i and c_i

Surface Area is an approximation of the surface of the ROI in mm2, calculated using a marching cubes algorithm.

References:

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

getSurfaceVolumeRatioFeatureValue()

3. 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()

4. 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 $0 < \mathit{sphericity} \leq 1$, 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()

5. 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 $0 < \mathit{compactness}\ 1 \leq \frac{1}{6\pi}$, where a value of $\frac{1}{6\pi}$ indicates a perfect sphere.

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

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

---

getCompactness2FeatureValue()

6. Compactness 2

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

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. In the default parameter file provided in the `pyradiomics/examples/exampleSettings` folder, Compactness 1 and Compactness 2 are therefore disabled.

---

getSphericalDisproportionFeatureValue()

7. Spherical Disproportion

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

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 1, Compactness 2 and Sphericity. In the default parameter file provided in the `pyradiomics/examples/exampleSettings` folder, Compactness 1 and Compactness 2 are therefore disabled.

---

getMaximum3DDiameterFeatureValue()

8. Maximum 3D diameter

Maximum 3D diameter is defined as the largest pairwise Euclidean distance between surface voxels in the ROI.

Also known as Feret Diameter.

Warning: This feature is only available when C Extensions are enabled

getMaximum2DDiameterSliceFeatureValue()

9. Maximum 2D diameter (Slice)

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

Warning: This feature is only available when C Extensions are enabled

getMaximum2DDiameterColumnFeatureValue()

10. Maximum 2D diameter (Column)

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

> **Warning:** This feature is only available when C Extensions are enabled

getMaximum2DDiameterRowFeatureValue()

11. Maximum 2D diameter (Row)

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

> **Warning:** This feature is only available when C Extensions are enabled

getMajorAxisFeatureValue()

12. Major Axis

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

getMinorAxisFeatureValue()

13. Minor Axis

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

getLeastAxisFeatureValue()

14. Least Axis

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

getElongationFeatureValue()

15. Elongation

Elongation is calculated using its implementation in SimpleITK, and is defined as:

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

Here, λ_{major} and λ_{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 single point or 1 dimensional line).

getFlatnessFeatureValue()

16. Flatness

Flatness is calculated using its implementation in SimpleITK, and is defined as:

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

Here, λ_{major} and λ_{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).

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 $\mathbf{P}(i, j|\delta, \alpha)$. 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 δ pixels along angle α. The distance δ 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 $\mathbf{I}$ represent a 5x5 image, having 5 discrete grey levels:

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

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

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

Let:

- ϵ be an arbitrarily small positive number ($\approx 2.2 \times 10^{-16}$)
- $\mathbf{P}(i, j)$ be the co-occurence matrix for an arbitrary δ and α
- $p(i, j)$ be the normalized co-occurence matrix and equal to $\frac{\mathbf{P}(i,j)}{\sum \mathbf{P}(i,j)}$
- N_g be the number of discrete intensity levels in the image
- $p_x(i) = \sum_{j=1}^{N_g} P(i, j)$ be the marginal row probabilities
- $p_y(j) = \sum_{i=1}^{N_g} P(i, j)$ be the marginal column probabilities
- μ_x be the mean gray level intensity of p_x and defined as $\mu_x = \sum_{i=1}^{N_g} p_x(i)i$
- μ_y be the mean gray level intensity of p_y and defined as $\mu_y = \sum_{j=1}^{N_g} p_y(j)j$
- σ_x be the standard deviation of p_x
- σ_y be the standard deviation of p_y
- $p_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i, j),$ where $i + j = k,$ and $k = 2, 3, \ldots, 2N_g$
- $p_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i, j),$ where $|i - j| = k,$ and $k = 0, 1, \ldots, N_g - 1$
- $HX = -\sum_{i=1}^{N_g} p_x(i) \log_2 \big(p_x(i) + \epsilon\big)$ be the entropy of p_x

- $HY = -\sum_{j=1}^{N_g} p_y(j) \log_2 \left(p_y(j) + \epsilon\right)$ be the entropy of p_y
- $HXY = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log_2 \left(p(i,j) + \epsilon\right)$ be the entropy of $p(i,j)$
- $HXY1 = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log_2 \left(p_x(i)p_y(j) + \epsilon\right)$
- $HXY2 = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_x(i)p_y(j) \log_2 \left(p_x(i)p_y(j) + \epsilon\right)$

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. See also *generateAngles()*
- 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

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

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

getJointAverageFeatureValue()

2. Joint Average

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

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

$$\textit{cluster prominence} = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}\left(i + j - \mu_x - \mu_y\right)^4 p(i,j)$$

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

$$\textit{cluster shade} = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}\left(i + j - \mu_x - \mu_y\right)^3 p(i,j)$$

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

$$\textit{cluster tendency} = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}\left(i + j - \mu_x - \mu_y\right)^2 p(i,j)$$

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

Note: Cluster Tendency is mathematically identical to Sum Variance, the latter has therefore been removed from PyRadiomics. See *here* for the proof.

---

getContrastFeatureValue()

6. Contrast

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

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

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

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 discreet gray value in the ROI (flat region), σ_x and σ_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

$$\textit{difference average} = \sum_{k=0}^{N_g-1} k p_{x-y}(k)$$

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

$$\textit{difference entropy} = \sum_{k=0}^{N_g-1} p_{x-y}(k) \log_2 \left(p_{x-y}(k) + \epsilon\right)$$

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

getDifferenceVarianceFeatureValue()

10. Difference Variance

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

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

getDissimilarityFeatureValue()

11. Dissimilarity

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

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.

getJointEnergyFeatureValue()

12. Joint Energy

$$\textit{joint energy} = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \left(p(i,j)\right)^2$$

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()

13. Joint Entropy

$$\textit{joint entropy} = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log_2 \left(p(i,j) + \epsilon\right)$$

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

Note: Defined by IBSI as Joint entropy

getHomogeneity1FeatureValue()

14. Homogeneity 1

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

Homogeneity 1 is a measure of the similarity in intensity values for neighboring voxels. It is a measure of local homogeneity that increases with less contrast in the window.

Note: Not present in IBSI feature definitions

getHomogeneity2FeatureValue()

15. Homogeneity 2

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

Homogeneity 2 is a measure of the similarity in intensity values for neighboring voxels.

Note: Not present in IBSI feature definitions

getImc1FeatureValue()

16. Informal Measure of Correlation (IMC) 1

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

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

getImc2FeatureValue()

17. Informal Measure of Correlation (IMC) 2

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

Note: In the case where HXY = HXY2, an arbitrary value of 0 is returned to prevent returning complex numbers. This is done on a per-angle basis.

getIdmFeatureValue()

18. Inverse Difference Moment (IDM)

$$\textit{IDM} = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}\frac{p(i,j)}{1+|i-j|^2}$$

IDM (inverse difference moment) 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).

getIdmnFeatureValue()

19. Inverse Difference Moment Normalized (IDMN)

$$IDMN = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} \frac{p(i,j)}{1 + \left(\frac{|i-j|^2}{N_g^2}\right)}$$

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()

20. Inverse Difference (ID)

$$ID = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} \frac{p(i,j)}{1 + |i-j|}$$

ID (inverse difference) 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()

21. Inverse Difference Normalized (IDN)

$$IDN = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} \frac{p(i,j)}{1 + \left(\frac{|i-j|}{N_g}\right)}$$

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()

22. Inverse Variance

$$\textit{inverse variance} = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} \frac{p(i,j)}{|i-j|^2}, i \neq j$$

getMaximumProbabilityFeatureValue()

23. 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()

24. 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.

getSumEntropyFeatureValue()

25. Sum Entropy

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

Sum Entropy is a sum of neighborhood intensity value differences.

getSumSquaresFeatureValue()

26. 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

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:

- $\mathbf{P}(i, j)$ be the size zone matrix
- $p(i, j)$ be the normalized size zone matrix, defined as $p(i, j) = \frac{\mathbf{P}(i,j)}{\sum \mathbf{P}(i,j)}$
- N_g be the number of discreet intensity values in the image
- N_s be the number of discreet zone sizes in the image
- N_p be the number of voxels in the image

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)

$$SAE = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_s} \frac{\mathbf{P}(i,j)}{j^2}}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_s} \mathbf{P}(i, j)}$$

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)

$$LAE = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_s} \mathbf{P}(i, j) j^2}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_s} \mathbf{P}(i, j)}$$

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)

$$GLN = \frac{\sum_{i=1}^{N_g} \left(\sum_{j=1}^{N_s} \mathbf{P}(i, j) \right)^2}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_s} \mathbf{P}(i, j)}$$

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)

$$GLNN = \frac{\sum_{i=1}^{N_g}\left(\sum_{j=1}^{N_s}\mathbf{P}(i,j)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)^2}$$

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)

$$SZN = \frac{\sum_{j=1}^{N_s}\left(\sum_{i=1}^{N_g}\mathbf{P}(i,j)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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)

$$SZNN = \frac{\sum_{j=1}^{N_s}\left(\sum_{i=1}^{N_g}\mathbf{P}(i,j)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)^2}$$

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)

$$ZP = \sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\frac{\mathbf{P}(i,j)}{N_p}$$

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)

$$GLV = \sum_{i=1}^{N_g}\sum_{j=1}^{N_s}p(i,j)(i-\mu)^2$$

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

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

getZoneVarianceFeatureValue()

9. Zone Variance (ZV)

$$ZV = \sum_{i=1}^{N_g}\sum_{j=1}^{N_s}p(i,j)(j-\mu)^2$$

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

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

getZoneEntropyFeatureValue()

10. Zone Entropy (ZE)

$$ZE = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_s} p(i,j)\log_2(p(i,j)+\epsilon)$$

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

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)

$$LGLZE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\frac{\mathbf{P}(i,j)}{i^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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)

$$HGLZE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)i^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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)

$$SALGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\frac{\mathbf{P}(i,j)}{i^2 j^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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)

$$SAHGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\frac{\mathbf{P}(i,j)i^2}{j^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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)

$$LALGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\frac{\mathbf{P}(i,j)j^2}{i^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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)

$$LAHGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)i^2j^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_s}\mathbf{P}(i,j)}$$

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

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 $\mathbf{P}(i,j|\theta)$, 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 θ.

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

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

Let:

- $\mathbf{P}(i,j|\theta)$ be the run length matrix for an arbitrary direction θ
- $p(i,j|\theta)$ be the normalized run length matrix, defined as $p(i,j|\theta) = \frac{\mathbf{P}(i,j|\theta)}{\sum\mathbf{P}(i,j|\theta)}$
- N_g be the number of discreet intensity values in the image
- N_r be the number of discreet run lengths in the image
- N_p be the number of voxels in the image

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)

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

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)

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

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)

$$GLN = \frac{\sum_{i=1}^{N_g}\left(\sum_{j=1}^{N_r}\mathbf{P}(i,j|\theta)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\mathbf{P}(i,j|\theta)}$$

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)

$$GLNN = \frac{\sum_{i=1}^{N_g}\left(\sum_{j=1}^{N_r}\mathbf{P}(i,j|\theta)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\mathbf{P}(i,j|\theta)^2}$$

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)

$$RLN = \frac{\sum_{j=1}^{N_r}\left(\sum_{i=1}^{N_g}\mathbf{P}(i,j|\theta)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\mathbf{P}(i,j|\theta)}$$

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)

$$RLNN = \frac{\sum_{j=1}^{N_r}\left(\sum_{i=1}^{N_g}\mathbf{P}(i,j|\theta)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\mathbf{P}(i,j|\theta)}$$

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)

$$RP = \sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\frac{\mathbf{P}(i,j|\theta)}{N_p}$$

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)

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

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

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

getRunVarianceFeatureValue()

9. Run Variance (RV)

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

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

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

getRunEntropyFeatureValue()

10. Run Entropy (RE)

$$RE = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} p(i,j|\theta) \log_2(p(i,j|\theta) + \epsilon)$$

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

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)

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

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)

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

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)

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

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

getShortRunHighGrayLevelEmphasisFeatureValue()

14. Short Run High Gray Level Emphasis (SRHGLE)

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

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

getLongRunLowGrayLevelEmphasisFeatureValue()

15. Long Run Low Gray Level Emphasis (LRLGLE)

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

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

getLongRunHighGrayLevelEmphasisFeatureValue()

16. Long Run High Gray Level Emphasis (LRHGLE)

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

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

Neigbouring 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 δ. The sum of absolute differences for gray level i is stored in the matrix. Let $\mathbf{X}_{gl}$ be a set of segmented voxels and $x_{gl}(j_x, j_y, j_z) \in \mathbf{X}_{gl}$ be the gray level of a voxel at postion (j_x, j_y, j_z), then the average gray level of the neigbourhood is:

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

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

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

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

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:

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

Similar for gray values 3 and 5:

$s_3 = |3 - 14/5| + |3 - 18/5| + |3 - 20/8| + |3 - 5/3| = 2.63$
$s_5 = |5 - 14/5| + |5 - 18/5| + |5 - 20/8| + |5 - 11/5| = 10.075$

Let:

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

N_v be the total number of voxels in X_{gl} and equal to $\sum n_i$

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

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

N_g be the number of discreet 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. See also *generateAngles()*

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.

$Coarseness = \frac{1}{\sum_{i=1}^{N_g} p_i s_i}$

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. $\sum_{i=1}^{N_g} p_i s_i$ 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.

$Contrast = \left(\frac{1}{N_{g,p}(N_{g,p}-1)} \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_i p_j (i-j)^2 \right) \left(\frac{1}{N_p^2} \sum_{i=1}^{N_g} s_i \right)$, 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.

getBusynessFeatureValue()

Calculate and return the busyness.

$Busyness = \frac{\sum_{i=1}^{N_g} p_i s_i}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} |ip_i - jp_j|}$, 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.

$Complexity = \frac{1}{N_p^2} \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} |i-j| \frac{p_i s_i + p_j s_j}{p_i + p_j}$, 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.

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

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

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 δ 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 $|i-j| \leq \alpha$. In a gray level dependence matrix $\mathbf{P}(i,j)$ 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:

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

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

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

Let:

$\mathbf{P}(i,j)$ be the dependence matrix

$p(i,j)$ be the normalized dependence matrix, defined as $p(i,j) = \frac{\mathbf{P}(i,j)}{\sum \mathbf{P}(i,j)}$

N_g be the number of discreet intensity values in the image

N_d be the number of discreet dependency sizes in the image

N_p be the number of voxels in the image

The following class specific settings are possible:

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

getSmallDependenceEmphasisFeatureValue()

1. Small Dependence Emphasis (SDE)

$SDE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\frac{\mathbf{P}(i,j)}{i^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$

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)

$$LDE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)j^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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)

$$GLN = \frac{\sum_{i=1}^{N_g}\left(\sum_{j=1}^{N_d}\mathbf{P}(i,j)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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)

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

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.

getDependenceNonUniformityFeatureValue()

5. Dependence Non-Uniformity (DN)

$$DN = \frac{\sum_{j=1}^{N_d}\left(\sum_{i=1}^{N_g}\mathbf{P}(i,j)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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

getDependenceNonUniformityNormalizedFeatureValue()

6. Dependence Non-Uniformity Normalized (DNN)

$$DNN = \frac{\sum_{j=1}^{N_d}\left(\sum_{i=1}^{N_g}\mathbf{P}(i,j)\right)^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)^2}$$

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()

7. Gray Level Variance (GLV)

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

$$\mu = \sum_{i=1}^{N_g}\sum_{j=1}^{N_d} ip(i,j)$$

Measures the variance in grey level in the image.

getDependenceVarianceFeatureValue()

8. Dependence Variance (DV)

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

$$\mu = \sum_{i=1}^{N_g}\sum_{j=1}^{N_d} jp(i,j)$$

Measures the variance in dependence size in the image.

getDependenceEntropyFeatureValue()

9. Dependence Entropy (DE)

$$DependenceEntropy = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_d} p(i,j)\log_2(p(i,j)+\epsilon)$$

getLowGrayLevelEmphasisFeatureValue()

10. Low Gray Level Emphasis (LGLE)

$$LGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\frac{\mathbf{P}(i,j)}{i^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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()

11. High Gray Level Emphasis (HGLE)

$$HGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)i^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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()

12. Small Dependence Low Gray Level Emphasis (SDLGLE)

$$SDLGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\frac{\mathbf{P}(i,j)}{i^2j^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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

getSmallDependenceHighGrayLevelEmphasisFeatureValue()

13. Small Dependence High Gray Level Emphasis (SDHGLE)

$$SDHGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\frac{\mathbf{P}(i,j)i^2}{j^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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

getLargeDependenceLowGrayLevelEmphasisFeatureValue()

14. Large Dependence Low Gray Level Emphasis (LDLGLE)

$$LDLGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\frac{\mathbf{P}(i,j)j^2}{i^2}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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

getLargeDependenceHighGrayLevelEmphasisFeatureValue()

15. Large Dependence High Gray Level Emphasis (LDHGLE)

$$LDHGLE = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)i^2j^2}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_d}\mathbf{P}(i,j)}$$

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

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

Excluded GLCM Features

For included features and class definition, see *Gray Level Co-occurrence Matrix (GLCM) Features*.

1. Sum Variance

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

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, $p_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j),$ where $i + j = k, k \in \{2, 3, \ldots, 2N_g\}$
2. Starting with cluster tendency as defined in GLCM:

$$\begin{aligned}
\textit{cluster tendency} &= \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \big(i + j - \mu_x - \mu_y\big)^2 p(i,j) \\
&= \sum_{k=2}^{2N_g} \Big[\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \big(i + j - \mu_x - \mu_y\big)^2 p(i,j), \text{ where } i + j = k \Big] \\
&= \sum_{k=2}^{2N_g} \Big[\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \big(k - (\mu_x + \mu_y)\big)^2 p(i,j), \text{ where } i + j = k \Big]
\end{aligned}$$

Note: Because inside the sum $\sum_{k=2}^{2N_g}$, k is a constant, and so are μ_x and μ_y, $\big(k - (\mu_x + \mu_y)\big)^2$ is constant and can be taken outside the inner sum $\sum_{i=1}^{N_g} \sum_{j=1}^{N_g}$.

$$= \sum_{k=2}^{2N_g} \Big[\big(k - (\mu_x + \mu_y)\big)^2 \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j), \text{ where } i + j = k \Big]$$

3. Using (1.) and (2.)

$$\textit{cluster tendency} = \sum_{k=2}^{2N_g} \Big[\big(k - (\mu_x + \mu_y)\big)^2 p_{x+y}(k) \Big]$$

4. As defined in GLCM, $p_x(i) = \sum_{j=1}^{N_g} P(i,j)$ and $\mu_x = \sum_{i=1}^{N_g} p_x(i) i$, therefore $\mu_x = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} P(i,j) i$
5. Similarly as in (4.), $\mu_y = \sum_{j=1}^{N_g} \sum_{i=1}^{N_g} P(i,j) j$
6. Using (4.) and (5.), μ_x and μ_y can then be combined as follows:

$$\mu_x + \mu_y = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} P(i,j)i + \sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)j$$
$$= \sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)i + P(i,j)j$$
$$= \sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)(i+j)$$
$$= \sum_{k=2}^{2N_g}\left[\sum_{j=1}^{N_g}\sum_{i=1}^{N_g} P(i,j)(i+j), \text{ where } k = i+j\right]$$
$$= \sum_{k=2}^{2N_g} p_{x+y}(k)k = \textit{sum average (SA)}$$

7. Combining (3.) and (6.) yields the following formula:

$$\text{Cluster Tendency} = \sum_{k=2}^{2N_g}\left[(k - SA)^2 p_{x+y}(k)\right] = \textit{ sum variance}$$

Q.E.D

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.

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)

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`.

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*.

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_cmatrices`, matrix calculation should be implemented as follows:
  - The function calculating the matrix in python should be defined in a function called `_calculateMatrix`.
  - The function calculating the matrix using the C extension should be defined in a function called `_calculateCMatrix`.
  - 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).

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.

Signature of an image type

All image types are defined in the *imageoperations module*, and identified by the signature `get[Name]Image(inputImage, **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`, a SimpleITK Image object of the original image 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.

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
    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'
```

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.

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 32 and 64 bits (Windows, Linux and Mac)
- Python 3.4 32 and 64 bits (Windows and Linux)
- Python 3.5 32 and 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.

Frequently Asked Questions

Installation

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.

I installed PyRadiomics successfully from the repository, but when I run the notebook, I get `Error loading C extensions, switching to python calculation`

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 develop` from the commandline, which is similar to the `install` command, but installs pyradiomics to the source folder instead (i.e. does nothing to the python files, but compiles the C extensions and copies them to the source folder).

Which python versions is PyRadiomics compatible with?

PyRadiomics is compatible with both python 2 and python 3. The automated testing uses python versions 2.7, 3.4 and 3.5 (both 32 and 64 bits). 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.

Input / Customization

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.

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.

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 you want to provide the image in DICOM format, load the DICOM images using SimpleITK functionality and pass the resultant image object instead.

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 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 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).

What modalities does PyRadiomics support?

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
2. 3D or slice: Although PyRadiomics supports single slice (2D) feature extraction, the input is still required to have 3 dimensions (where in case of 2D, a dimension may be of size 1).

Can I use DICOM-RT struct for the input mask?

PyRadiomics does not support DICOM-RT struct as input directly. We recommend to convert these using for example SlicerRT. We are working on providing support for DICOM-RT in the Slicer extension, but this is not thoroughly tested yet.

Usage

How should the input file for `pyradiomicsbatch` be structured?

Currently, the input file for `pyradiomicsbatch` 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.*

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.

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.

I'm able to extract features, but many are NaN, 0 or 1. What happend?

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[1] assessed the number of bins in PET and found that in the range of 16 - 128 bins, texture features did not differ significantly.

Does PyRadiomics support voxel-wise feature extraction?

No, currently PyRadiomics only supports lesion-based feature extraction. However, voxel-based feature extraction may be a good addition in the future. If you have thoughts or ideas on how to implement this, we'd welcome your input on the pyradiomics email list.

Miscellaneous

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.

[1] 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.

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 or contact us at the pyradiomics email list. 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 email list or the issues on GitHub. Feel free to post a new question or issue and we'll try to get back to you ASAP.

Release Notes

Next Release

PyRadiomics 1.3.0

Feature Calculation Changes

- Remove feature *Sum Variance*, as this is mathematically equal to *Cluster Tendency*. (#300)

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)

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)

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)

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)

License

- Switch to 3-clause BSD license. (#272)

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)

[1] Haralick R, Shanmugan K, Dinstein I: Textural features for image classification. IEEE Trans Syst Man Cybern 1973:610–621.

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")
- 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)

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)

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.

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).

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 2

Feature Classes

Currently supports the following feature classes:

- *First Order Statistics*
- *Shape-based*
- *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 ≈ 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 3

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*

For more information, see also *Image Processing and Filters*.

CHAPTER 4

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 5

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 6

Installation

PyRadiomics is OS independent and compatible with both Python 2.7 and Python >=3.4.

- Clone the repository
  - `git clone git://github.com/Radiomics/pyradiomics`
- Install on your system (Linux, Mac OSX), with prerequisites:
  - `cd pyradiomics`
  - `python -m pip install -r requirements.txt`
  - `python setup.py install`
- For more detailed installation instructions and installation on Windows see *Installation Details*

CHAPTER 7

Pyradiomics Indices and Tables

- modindex
- genindex
- search

CHAPTER 8

License

This package is covered by the open source 3-clause BSD License.

CHAPTER 9

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 10

Contact

We are happy to help you with any questions. Please contact us on the pyradiomics email list.

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

L

N

R

S