pyradiomics Documentation

Release 1.1.1

pyradiomics community

Mar 23, 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"; Submitted 2017*

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

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

    - If the installation of SimpleITK fails (newer versions not available on the default servers), you can get it manually from sourceforge
      - For linux:

```
wget 'https://sourceforge.net/projects/simpleitk/files/SimpleITK/0.10.
↪0/Python/SimpleITK-0.10.0-1-cp27-cp27m-manylinux1_x86_64.whl'
pip install --user 'SimpleITK-0.10.0-1-cp27-cp27m-manylinux1_x86_64.whl
↪'
```

· For Mac:

```
wget 'https://sourceforge.net/projects/simpleitk/files/SimpleITK/0.10.
↪0/Python/SimpleITK-0.10.0-cp27-cp27m-macosx_10_6_intel.whl'
pip install --user 'SimpleITK-0.10.0-cp27-cp27m-macosx_10_6_intel.whl'
```

- For Windows:
  - `cd pyradiomics`
  - `python -m pip install -r requirements.txt`
  - `python setup.py install`
  - If the installation of SimpleITK fails (newer versions not available on the default servers), you can install it manually:

```
pip install --trusted-host www.simpleitk.org -f https://sourceforge.net/
↪projects/simpleitk/files/SimpleITK/0.10.0/Python/ SimpleITK==0.10.0
```

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/bin/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/bin/Notebooks`
  - The parameter file used in the instruction video is available in `pyradiomics/bin`
- If jupyter is not installed, run the python script alternative (`pyradiomics/bin/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 each row represents one combination of an image and a segmentation and contains 5 elements: 1) patient ID, 2) sequence name (image identifier), 3) reader (segmentation identifier), 4) path/to/image, 5) path/to/mask.
- For more information on passing parameter files, setting up logging and controlling output format, 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
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 = os.path.join(dataDir, "data", 'brain1_image.nrrd')
maskName = os.path.join(dataDir, "data",  'brain1_label.nrrd')
```

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

```
params = os.path.join(dataDir, "bin", "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
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 = str(dataDir + os.path.sep + 'brain1_image.nrrd')
maskName = str(dataDir + os.path.sep + 'brain1_label.nrrd')
image = sitk.ReadImage(imageName)
mask = sitk.ReadImage(maskName)
```

- Calculate the first order features:

```
firstOrderFeatures = firstorder.RadiomicsFirstOrder(image,mask)
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.

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.

It initialisation, 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, the following general settings can be specified in the parameter file or `kwargs`, with default values in brackets:

- 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()*.
- binWidth [25]: Float, size of the bins when making a histogram and for discretization of the image gray level.
- normalize [False]: Boolean, set to True to enable normalizing of the image before any resampling. See also `normalizeImage()`.
- normalizeScale [1]: Float, determines the scale after normalizing the image. If normalizing is disabled, this has no effect.
- removeOutliers [None]: Float, 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()`.
- resampledPixelSpacing [None]: List of 3 floats, sets the size of the voxel in (x, y, z) plane when resampling.
- interpolator [sitkBSpline]: Simple ITK 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, 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.**
- 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()*
- 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()*

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

In addition to these general settings, filter or feature class specific settings can be defined here also. For more information on possible settings, see the respective filters and feature classes.

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 in kwargs, enabled feature(Classes) and input images:

- settings not specified in parameters are set to their default value.
- enabledFeatures are replaced by those in parameters. If no featureClass parameters were specified, all featureClasses and features are enabled.
- inputImages are replaced by those in parameters. If no inputImage parameters were specified, only original image is used for feature extraction, with no additional custom settings

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. Settings must be grouped by setting type as mentioned above are reflected in the structure of the document as follows:

```
<Setting Type>:
  <Setting Name>: <value>
  ...
<Setting Type>:
  ...
```

Blank lines may be inserted to increase readability, the 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 settings:

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

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

The three setting types are named as follows:

- setting: Setting to use for preprocessing and class specific settings (`kwargs` arguments). if no <value> is specified, the value for this setting is set to None.
- 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.
- inputImage: input image 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.

If supplied params file does not match the requirements, a pykwalify error is raised.

enableAllInputImages()
Enable all possible input images without any custom settings.

disableAllInputImages()
Disable all input images.

enableInputImageByName(*inputImage*, *enabled=True*, *customArgs=None*)
Enable or disable specified input image. If enabling input image, optional custom settings can be specified in customArgs.

Current possible input images 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).

enableInputImages(****inputImages*)
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 **inputImages** – 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. First, image and mask are loaded and normalized/resampled if necessary. Second, if enabled, provenance information is calculated and stored as part of the result. Next, shape features are calculated on a cropped (no padding) version of the original image. Then other featureclasses are calculated using all specified input images in `inputImages`. Images are cropped to tumor mask (no padding) after application of any filter and before being passed to the feature class. Finally, the dictionary containing all calculated features is returned.

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 ("<filter>_<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*, *inputImageName*, ***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 (handeled 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.

N.B. 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.

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

Generate all possible angles from distance 1 until the maximum distance in `distances` in 3D. E.g. for d = 1, 13 angles are generated (representing the 26-connected region). For d = 2, 13 + 49 = 62 angles are generated (representing the 26 connected region for distance 1, and the 98 connected region for distance 2)

First, only generated 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`. Next, impossible angles (where 'neighbouring' voxels will always be outside delineation) are deleted. Finally, 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.`**`cropToTumorMask`**(*imageNode*, *maskNode*, *label=1*, *boundingBox=None*)

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.

Returns both the cropped version of the image and the cropped version of the labelmap, as well as the computed bounding box. The bounding box is 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.

This can be used in subsequent calls to this function for the same images. This improves computation time, as it will reduce the number of calls to SimpleITK.LabelStatisticsImageFilter().

Parameters

- **`label`** – [1], value of the label, onto which the image and mask must be cropped.
- **`boundingBox`** – [None], during a subsequent call, the boundingBox of a previous call can be passed here, removing the need to recompute it. During a first call to this function for a image/mask with a certain label, this value must be None or omitted.

Returns Cropped image and mask (SimpleITK image instances) and the bounding box generated by SimpleITK LabelStatisticsImageFilter.

radiomics.imageoperations.**resampleImage**(*imageNode*, *maskNode*, *resampledPixelSpacing*, *interpolator=3*, *label=1*, *padDistance=5*)

Resamples image or label to the specified pixel spacing (The default interpolator is Bspline)

'imageNode' is a SimpleITK Object, and 'resampledPixelSpacing' is the output pixel spacing (list of 3 elements).

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 defined by the bounding box and the padDistance are resampled. This results in a resampled and partially cropped return image and labelmap. 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.

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.**applyThreshold**(*inputImage*, *lowerThreshold*, *upperThreshold*, *insideValue=None*, *outsideValue=0*)

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

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

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

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

Apply Laplacian of Gaussian filter to input image and compute signature for each filtered image.

Following settings are possible:

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

N.B. 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 filter name and `kwargs`

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

Apply wavelet filter to image and compute signature for each filtered image.

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 filter name and `kwargs`

`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`

`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`

`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`

`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`

General Info Module

class `radiomics.generalinfo.`**`GeneralInfo`**(*imagePath*, *maskPath*, *resampledMask*, *kwargs*, *inputImages*)

`execute`()

Calculate and return a dictionary containing all general info items. Format is <info_item>:<value>, where any ',' in <value> are replaced by ';' to prevent column alignment errors in csv formatted output.

`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 settings contained in kwargs. 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 of the image.

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

`getInputImagesValue`()

Return a string representation of the enabled filters and any custom settings for the filter. Format is {<filter_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 following variables are instantiated at initialization:

- 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.
- verbose: boolean indication whether or not to provide progress reporting to the output.
- featureNames: list containing the names of features defined in the feature class. See *getFeatureNames()*
- inputImage: Simple ITK image object of the input image
- inputMask: Simple ITK image object of the input labelmap
- imageArray: numpy array of the gray values in the input image
- maskArray: numpy array with elements set to 1 where labelmap = label, 0 otherwise
- matrix: numpy array of the gray values in the input image (with gray values inside ROI discretized when necessary in the texture feature classes).
- matrixCoordinates: 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.
- targetVoxelArray: flattened numpy array of gray values inside ROI.

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.**debug**(*debug_on=True*)
Control level of logger and stderr output of the toolbox. By default, this output reflects module hierarchy, as child loggers are created by module. This is achieved by the following line in base.py: `self.logger = logging.getLogger(self.__module__)`. To use same instance in each module, set `self.logger=logging.getLogger('radiomics')`.

At command line, turn on debugging output to stderr for all pyradiomics functions with:

```
import radiomics
radiomics.debug()
```

This set the level of both the logger and the handler for output to stderr to level = DEBUG. If level of logger is already at level "DEBUG" or "NOTSET", level of logger is not changed.

Turn off debugging with (only changes the level of the handler for output to stderr):

```
radiomics.debug(False)
```

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.

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.**getInputImageTypes**()
Returns a list of possible input image types. This function finds the image types dynamically by matching the signature ("get<inputImage>Image") against functions defined in *imageoperations*. Returns a list containing available input image names (<inputImage> 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.**setVerbosity**(*level*)
Assumes the handler added to the radiomics logger at initialization of the toolbox is not removed from the logger handlers.

Using the `level` (Python defined logging levels) argument, determine how much PyRadiomics should print out to the stderr, 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)

N.B. 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

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* (13 features)
- *Gray Level Co-occurrence Matrix (GLCM) Features* (28 features)
- *Gray Level Size Zone Matrix (GLSZM) Features* (16 features)
- *Gray Level Run Length Matrix (GLRLM) Features* (16 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.

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:

$\mathbf{X}$ denote the three dimensional image matrix with N voxels

$\mathbf{P}(i)$ the first order histogram with N_l discrete intensity levels, where N_l is defined by the number of levels, calculated based on 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.__

getEnergyFeatureValue()

Calculate the Energy of the image array.

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

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

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

getTotalEnergyFeatureValue()

Calculate the Total Energy of the image array.

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

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

getEntropyFeatureValue()

Calculate the Entropy of the image array.

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

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.

getMinimumFeatureValue()

Calculate the Minimum Value in the image array.

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

get10PercentileFeatureValue()

Calculate the 10$^{\text{th}}$ percentile in the image array.

get90PercentileFeatureValue()

Calculate the 90$^{\text{th}}$ percentile in the image array.

getMaximumFeatureValue()
Calculate the Maximum Value in the image array.

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

getMeanFeatureValue()
Calculate the Mean Value for the image array.

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

getMedianFeatureValue()
Calculate the Median Value for the image array.

getInterquartileRangeFeatureValue()
Calculate the interquartile range of the image array.

$interquartile\ range = \mathbf{P}_{75} - \mathbf{P}_{25}$, where $\mathbf{P}_{25}$ and $\mathbf{P}_{75}$ are the 25$^{\text{th}}$ and 75$^{\text{th}}$ percentile of the image array, respectively.

getRangeFeatureValue()
Calculate the Range of Values in the image array.

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

getMeanAbsoluteDeviationFeatureValue()
Calculate the Mean Absolute Deviation for the image array.

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

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

getRobustMeanAbsoluteDeviationFeatureValue()
Calculate the Robust Mean Absolute Deviation for the image array.

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

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 10$^{\text{th}}$ and 90$^{\text{th}}$ percentile.

getRootMeanSquaredFeatureValue()
Calculate the Root Mean Squared of the image array.

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

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()
Calculate the Standard Deviation of the image array.

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

Standard Deviation measures the amount of variation or dispersion from the Mean Value.

getSkewnessFeatureValue(*axis=0*)
Calculate the Skewness of the image array.

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

Where μ_3 is the 3rd central moment.

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

Related links:

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

getKurtosisFeatureValue(*axis=0*)
Calculate the Kurtosis of the image array.

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

Where μ_4 is the 4th central moment.

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

Related links:

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

getVarianceFeatureValue()
Calculate the Variance in the image array.

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

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.

getUniformityFeatureValue()
Calculate the Uniformity of the image array.

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

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.

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 tumor region. Let in the following definitions denote V the volume (mm^3) and A the surface area of the volume (mm^2) of interest.

getVolumeFeatureValue()
Calculate the volume of the tumor region in cubic millimeters.

getSurfaceAreaFeatureValue()
Calculate the surface area of the tumor region in square millimeters using a marching cubes algorithm.

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

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

getSurfaceVolumeRatioFeatureValue()
Calculate the surface area to volume ratio of the tumor region

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

Here, a lower value indicates a more compact (sphere-like) shape.

getSphericityFeatureValue()
Calculate the Sphericity of the tumor region.

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

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 $0 < sphericity \leq 1$, 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\bin` folder, Compactness 1 and Compactness 2 are therefore disabled.

---

getCompactness1FeatureValue()
Calculate the compactness (1) of the tumor region.

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

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

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

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

---

getCompactness2FeatureValue()
Calculate the Compactness (2) of the tumor region.

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

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\bin` folder, Compactness 1 and Compactness 2 are therefore disabled.

---

getSphericalDisproportionFeatureValue()

Calculate the Spherical Disproportion of the tumor region.

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

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, Sphericity and Spherical Disproportion. In the default parameter file provided in the `pyradiomics\bin` folder, Compactness 1 and Compactness 2 are therefore disabled.

---

getMaximum3DDiameterFeatureValue()

Calculate the largest pairwise euclidean distance between tumor surface voxels. Also known as Feret Diameter.

getMaximum2DDiameterSliceFeatureValue()

Calculate the largest pairwise euclidean distance between tumor surface voxels in the row-column plane.

getMaximum2DDiameterColumnFeatureValue()

Calculate the largest pairwise euclidean distance between tumor surface voxels in the row-slice plane.

getMaximum2DDiameterRowFeatureValue()

Calculate the largest pairwise euclidean distance between tumor surface voxels in the column-slice plane.

getElongationFeatureValue()

Calculate the elongation of the shape, which is defined as:

$$Elongation = \frac{\lambda_{\text{longest}}}{\lambda_{\text{intermediate}}}$$

Here, λ_{longest} and $\lambda_{\text{intermediate}}$ are the lengths of the largest and second largest principal component axes.

References:

•Andrey P, Kieu K, Kress C, Lehmann G, Tirichine L, Liu Z, et al. Statistical analysis of 3D images detects regular spatial distributions of centromeres and chromocenters in animal and plant nuclei. PLoS Comput Biol. 2010;6:27.

getFlatnessFeatureValue()

Calculate the flatness of the shape, which is defined as:

$$Flatness = \frac{\lambda_{\text{intermediate}}}{\lambda_{\text{shortest}}}$$

Here, $\lambda_{\text{intermediate}}$ and $\lambda_{\text{shortest}}$ are the lengths of the second largest and smallest principal component axes.

References:

•Andrey P, Kieu K, Kress C, Lehmann G, Tirichine L, Liu Z, et al. Statistical analysis of 3D images detects regular spatial distributions of centromeres and chromocenters in animal and plant nuclei. PLoS Comput Biol. 2010;6:27.

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 in direction α, and N_g is the number of discrete gray level intensities. The distance δ from the center voxel is defined as the distance according to the infinity norm. For $\delta = 1$, this assumes 26-connectivity in 3D and for $\delta = 2$ a 98-connectivity.

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) in directions $\alpha = 0°$ and opposite $\alpha = 180°$ (i.e., to the left and right from the pixel with the given value), 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 $\mu_x = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i, j)i$

μ_y be the mean gray level intensity of p_y and defined as $\mu_y = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i, j)j$

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 $k = 0, 1, \ldots, N_g - 1$

$HX = -\sum_{i=1}^{N_g} p_x(i) \log_2 \left(p_x(i) + \epsilon\right)$ 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:

- 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()

Calculate and return the mean 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.

getAverageIntensityFeatureValue()

Return the mean gray level intensity of the i distribution.

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

> **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()

Using coefficients μ_x and μ_y, calculate and return the mean Cluster Prominence.

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

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()

Using coefficients μ_x and μ_y, calculate and return the mean Cluster Shade.

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

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

getClusterTendencyFeatureValue()

Using coefficients μ_x and μ_y, calculate and return the mean Cluster Tendency.

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

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

getContrastFeatureValue()

Using the squared difference between gray values of neighbouring paris, calculate and return the mean Contrast.

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

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

getCorrelationFeatureValue()

Using coefficients μ_x, μ_y, σ_x and σ_y, calculate and return the mean Correlation.

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

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.

getDifferenceAverageFeatureValue()

Using coefficient p_{x-y}, calculate and return the mean Difference Average.

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

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

getDifferenceEntropyFeatureValue()

Using coefficient p_{x-y}, calculate and return the mean Difference Entropy.

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

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

getDifferenceVarianceFeatureValue()

Using coefficients p_{x-y} and DifferenceAverage (DA) calculate and return the mean Difference Variance.

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

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

getDissimilarityFeatureValue()

Calculate and return the mean Dissimilarity.

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

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.

getEnergyFeatureValue()

Calculate and return the mean Energy.

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

Energy (or Angular Second Moment)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.

getEntropyFeatureValue()

Calculate and return the mean Entropy.

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

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

getHomogeneity1FeatureValue()

Calculate and return the mean Homogeneity 1.

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

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.

getHomogeneity2FeatureValue()
Calculate and return the mean Homogeneity 2.

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

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

getImc1FeatureValue()
Using coefficients HX, HY, HXY and $HXY1$, calculate and return the mean Informal Measure of Correlation 1.

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

getImc2FeatureValue()
Using coefficients HXY and $HXY2$, calculate and return the mean Informal Measure of Correlation 2.

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

getIdmFeatureValue()
Calculate and return the mean Inverse Difference Moment.

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

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()
Calculate and return the mean Inverse Difference Moment Normalized.

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

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()
Calculate and return the mean Inverse Difference.

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

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()
Calculate and return the mean Inverse Difference Normalized.

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

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()

Calculate and return the mean Inverse Variance.

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

getMaximumProbabilityFeatureValue()

Calculate and return the mean Maximum Probability.

$$maximum\ probability = \max\left(p(i,j)\right)$$

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

getSumAverageFeatureValue()

Coefficient p_{x+y}, calculate and return the mean Sum Average.

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

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

getSumEntropyFeatureValue()

Using coefficient p_{x+y}, calculate and return the mean Sum Entropy.

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

Sum Entropy is a sum of neighborhood intensity value differences.

getSumVarianceFeatureValue()

Using coefficients p_{x+y} and SumEntropy (SE) calculate and return the mean Sum Variance.

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

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

getSumVariance2FeatureValue()

Using coefficients p_{x+y} and SumAvarage (SA) calculate and return the mean Sum Variance 2.

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

Sum Variance 2 is a measure of heterogeneity that places higher weights on neighboring intensity level pairs that deviate more from the mean.

This formula differs from SumVariance in that instead of subtracting the SumEntropy from the intensity, it subtracts the SumAvarage, which is the mean of intensities and not its entropy

getSumSquaresFeatureValue()

Using coefficients i and μ_x, calculate and return the mean Sum of Squares (also known as Variance) of the i distribution.

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

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 $i = j$

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. This yields a 26-connected region in a 3D image, and an 8-connected region in a 2D image. In a gray level size zone matrix $P(i, j)$ the $(i, j)^{\text{th}}$ element describes the number of times a gray level zone with gray level i and size j appears in image.

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

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

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

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()

Calculate and return the Small Area Emphasis (SAE) value.

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

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

getLargeAreaEmphasisFeatureValue()

Calculate and return the Large Area Emphasis (LAE) value.

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

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()

Calculate and return the Gray Level Non-Uniformity (GLN) value.

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

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

getGrayLevelNonUniformityNormalizedFeatureValue()

Calculate and return the Gray Level Non-Uniformity Normalized (GLNN) value.

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

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()

Calculate and return the Size-Zone Non-Uniformity (SZN) value.

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

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

getSizeZoneNonUniformityNormalizedFeatureValue()

Calculate and return the Size-Zone Non-Uniformity Normalized (SZNN) value.

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

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()

Calculate and return the Zone Percentage (ZP) value.

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

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()

Calculate and return the Gray Level Variance (GLV) value.

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

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

Measures the variance in gray level intensities for the zones.

getZoneVarianceFeatureValue()
Calculate and return the Zone Variance (ZV) value.

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

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

Measures the variance in zone size volumes for the zones.

getZoneEntropyFeatureValue()
Calculate and return the Zone Entropy (ZE) value.

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

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

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

getLowGrayLevelZoneEmphasisFeatureValue()
Calculate and return the Low Gray Level Zone Emphasis (LGLZE) value.

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

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()
Calculate and return the High Gray Level Zone Emphasis (HGLZE) value.

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

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()
Calculate and return the Small Area Low Gray Level Emphasis (SALGLE) value.

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

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

getSmallAreaHighGrayLevelEmphasisFeatureValue()
Calculate and return the Small Area High Gray Level Emphasis (SAHGLE) value.

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

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

getLargeAreaLowGrayLevelEmphasisFeatureValue()
Calculate and return the Large Area Low Gray Level Emphasis (LALGLE) value.

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

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

getLargeAreaHighGrayLevelEmphasisFeatureValue()
Calculate and return the Large Area High Gray Level Emphasis (LAHGLE) value.

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

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 in an image. A gray level run is 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 times a gray level i appears consecutively j times in the direction specified by 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()

Calculate and return the mean Short Run Emphasis (SRE) value for all GLRLMs.

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

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

getLongRunEmphasisFeatureValue()

Calculate and return the mean Long Run Emphasis (LRE) value for all GLRLMs.

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

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

getGrayLevelNonUniformityFeatureValue()

Calculate and return the mean Gray Level Non-Uniformity (GLN) value for all GLRLMs.

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

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()

Calculate and return the Gray Level Non-Uniformity Normalized (GLNN) value.

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

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()

Calculate and return the mean Run Length Non-Uniformity (RLN) value for all GLRLMs.

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

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

getRunLengthNonUniformityNormalizedFeatureValue()
Calculate and return the mean Run Length Non-Uniformity Normalized (RLNN) value for all GLRLMs.

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

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()
Calculate and return the mean Run Percentage (RP) value for all GLRLMs.

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

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).

getGrayLevelVarianceFeatureValue()
Calculate and return the Gray Level Variance (GLV) value.

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

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

Measures the variance in gray level intensity for the runs.

getRunVarianceFeatureValue()
Calculate and return the Run Variance (RV) value.

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

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

Measures the variance in runs for the run lengths.

getRunEntropyFeatureValue()
Calculate and return the Run Entropy (RE) value.

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

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

getLowGrayLevelRunEmphasisFeatureValue()
Calculate and return the mean Low Gray Level Run Emphasis (LGLRE) value for all GLRLMs.

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

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()
Calculate and return the mean High Gray Level Run Emphasis (HGLRE) value for all GLRLMs.

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

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()
Calculate and return the mean Short Run Low Gray Level Emphasis (SRLGLE) value for all GLRLMs.

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

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

getShortRunHighGrayLevelEmphasisFeatureValue()
Calculate and return the mean Short Run High Gray Level Emphasis (SRHGLE) value for all GLRLMs.

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

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

getLongRunLowGrayLevelEmphasisFeatureValue()
Calculate and return the mean Long Run Low Gray Level Emphasis (LRLGLE) value for all GLRLMs.

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

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

getLongRunHighGrayLevelEmphasisFeatureValue()
Calculate and return the mean Long Run High Gray Level Emphasis (LRHGLE) value for all GLRLMs.

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

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

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 *`getInputImageTypes()`*, 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(RadiomicsGLCM, self).__init__(inputImage, inputMask, **kwargs)

    def get[Feature]FeatureValue(self):
        """
        Feature docstring
        """

        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`

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 a filter

All filters 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 filter during extraction. The input of a filter is fixed and consists of the `inputImage`, a SimpleITK Image object of the original image and `**kwargs`, which are the customized setting 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, inputImageName, kwargs`. Here, `derivedImage` is one SimpleITK image object representing the filtered imag e, `inputImageName` 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 `inputImageName` 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.

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.

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 the *Usage* section. 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. Possible general settings, as well as possible image types are documented *here*, with additional feature class specific settings documented at the start of the *feature class*.

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.

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 does not contain a header line, and each line represents one such combination. Each line has 5 elements: Patient, Image, Mask, Image location, Mask location. Only the last two elements are 'active' (used during extraction), the are the locations of the image and mask files on the computer. The first three elements are copied to the output and are there to enable easy clustering of the results or correlation to outcome. These three elements are chosen, because patients can have multiple scans (or multiple sequences, e.g. in MRI), and each image can be segmented by different readers. As they are not actively used by PyRadiomics, any value is valid, including empty values, as long as they are provided (e.g. ",,,<path/to/image>,<path/to/mask>" is valid, "<path/to/image>,<path/to/mask>" is not).

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.

When I try to extract features, I get an error stating that image and mask do not occupy the same space.

During extraction, SimpleITK checks whether the mask matches the image dimensions and physical space and raises an error when this is not the case. This is to prevent attempts to extract features from an image using a mask that does not match the image. If it is the correct mask, you can change the tolerance for differences in direction and origin between the image and mask by setting `SimpleITK.ProcessObject.SetGlobalDefaultDirectionTolerance` and `SimpleITK.ProcessObject.SetGlobalDefaultDirectionTolerance`, respectively. By default, these are set to 1e-6 mm. Alternatively, you can resample your mask with SimpleITK.ResampleImageFilter(), using the image as a reference image and Nearest Neighbour as interpolator. See the SimpleITK documentation for more information.

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 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 (for the generation of colormaps)?

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.

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.

[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.

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)

On average, Pyradiomics extracts $\approx$ 1300 features per image, which consist of the 13 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)
- tqdm (Progressbar)
- six (Python 3 Compatibility)
- sphinx (Generating documentation)
- sphinx_rtd_theme (Template for documentation)
- nose-parameterized (Testing)

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`
  - `sudo python -m pip install -r requirements.txt`
  - `sudo 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 3D Slicer 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