Computing Bone Morphometric Feature Maps from 3-Dimensional Images

Release 2.0.0

Jean-Baptiste Vimort[1], Matthew McCormick[1] and Beatriz Paniagua[1]

November 2, 2017

[1]Kitware Inc., Carrboro, NC

Abstract

This document describes a new remote module implemented for the Insight Toolkit (ITK, www.itk.org), itkBoneMorphometry. This module contains bone analysis filters that compute features from N-dimensional images that represent the internal architecture of bone. The computation of the bone morphometry features in this module is based on well known methods. The two filters contained in this module are itkBoneMorphometryFeaturesFilter. which computes a set of features that describe the whole input image in the form of a feature vector, and itkBoneMorphometryFeaturesImageFilter, which computes an N-D feature map that locally describes the input image (i.e. for every voxel). itkBoneMorphometryFeaturesImageFilter can be configured based in the locality of the desired morphometry features by specifying the neighborhood size. This paper is accompanied by the source code, the input data, the choice of parameters and the output data that we have used for validating the algorithms described. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3588]
Distributed under Creative Commons Attribution License

Contents

Distributed under Creative Commons Attribution License

1 Introduction

Morphometry (or morphometrics) refers to the quantitative analysis of form and it is done by analyzing different aspects of an object such as the size or the shape of the studied object. The first bone morphometry analyses were performed in the 60s, thanks to a method called histomorphometry. Histomorphometry consists of slicing piece of *ex-vivo* bone and performing a succession of 2D morphometry analysis on the tissue slices obtained. Therefore, this technique was limited by the destructive nature of the procedure, which did not allow clinical application. Additionally, due to the 2D nature of the images, certain types of features such as bone volume density (*BV/TV*) and bone surface density (*BS/BV*)[2] could be computed, but computation of other types of 3D features, such as trabecular thickness (*Tb.Th*), trabecular separation (*Tb.Sp*), and trabecular number (*Tb.N*), were not possible[3].

In the past decade, improvements in 3D medical imaging technologies in terms of contrast, resolution and reconstruction have enabled the study bone structure *in-vivo*. Combined with the ever increasing computational power available, the development of tools to compute quantitative biomarkers of bone morphometry is now possible. Several free packages already offer a way to compute 3D bone morphometry features such as Microview or BoneJ. However, none of those tools contain bone morphometry filters that are able to compute bone morphometry N-dimensional feature maps.

We have created a new remote module containing two bone morphometry filters for ITK: the first one is able to compute a set of feature characterizing the whole input image (`itk::BoneMorphometryFeaturesFilter`) and a second one is optimized for the computation of feature maps characterizing the input image locally for every one of its voxels (`itk::BoneMorphometryFeaturesImageFilter`).

Significant computational power is required to create the feature maps described, so the `itk::BoneMorphometryFeaturesImageFilter` algorithms have been optimized thanks to ITK's multithreading, `itk::NeighborhoodIterator`, and `itk::ImageBoundaryFacesCalculator`. These improvements, only possible thanks to the internal ITK infrastructure, allow fast and efficient feature maps computation.

All the features available in *itkBoneMorphometry* are presented in Section 2. Section 3 describes the filters specifications (templates, inputs, parameters) of each filter and how to customize the use of these filters. Section 4 contain examples of code using itkBoneMorphometry filters in Python and C++. Finally, Sections 5 and 6 present several scenarios, results and conclusions obtained with *itkBoneMorphometry*.

Distributed under Creative Commons Attribution License

2 Bone Morphometry Features Available

The computation of the bone morphometry features is based on the following parameters:

- The total number of voxels in the studied volume N_{Total}: It represents the total number of voxels contained in the mask. If no N_{Total} is specified, the total number voxels in the whole image will be considered by default. In the particular case of the figure 1 as no mask is specified and the image is a square of 25 by 25 voxels $N_{Total} = 625$
- The number of voxels that are part of the bone N_{Bone}: It represents the number of pixels with an intensity higher than the specified threshold. Figure 1 shows the pixels that are part of the bone highlighted in brown and $N_{Bone} = 292$
- The number of voxels that are part of the bone/non-bone boundary $N_{Boundary}$: This number represents the separation for that can be separated for each direction $N_{BoundaryX}$, $N_{BoundaryY}$ and $N_{BoundaryZ}$: This parameter represents the number of boundary bone/non-bone voxels, it is important to notice that a single voxel can be part of the $N_{Boundary}$ several times. In the figure 1 example $N_{Boundary} = 218$

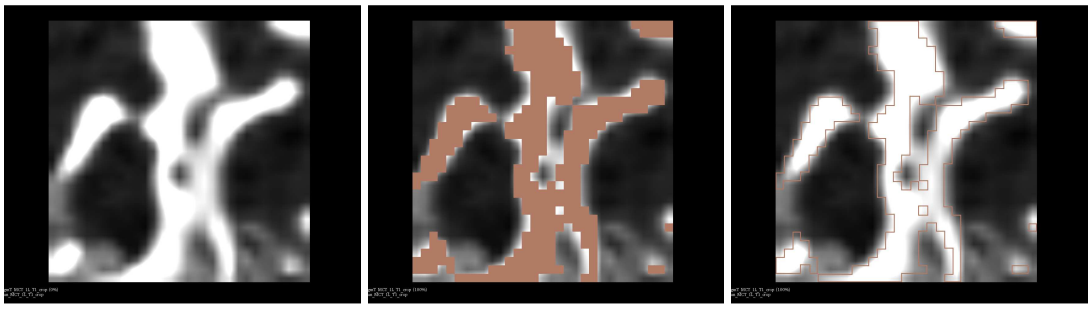

Figure 1: N_{Total}(left), N_{Bone} (center) and $N_{Boundary}$ (right)

The following features can be computed thanks to the itkBoneMorphometry filters:

Bone volume density or BvTv (which stands for stands for Bone Volume Bv over Total Volume Tv ratio) indicates the fraction of a given volume of interest (VOI, i.e. the Total Volume Tv) that is occupied by mineralized bone (Bone Volume Bv).

$$BvTv = \frac{N_{Bone}}{N_{Total}} \tag{1}$$

Trabecular number (TbN) is taken as the inverse of the mean distance between the mid-axes of the structure to be examined.

$$TbN = (TbN_x + TbN_y + TbN_z)/3 \tag{2}$$

with

$$TbN_{x/y/z} = \frac{N_{Boundaryx/y/z}}{N_{Total} * ImSpacing_{x/y/z}} \tag{3}$$

Bone surface density or BsBv (which stands for stands for Bone Surface Bs over Bone Volume Bv) gives an indication on how many bone lining cells cover a given volume of bone (Bv).

$$BsBv = 2\frac{TbN}{BsBv} \tag{4}$$

Distributed under Creative Commons Attribution License

Trabecular thickness (TbTh) is determined by filling maximal spheres into the structure using a distance transform. Then the average thickness of all maximal spheres is calculated to give an estimate of mean TbTh.

$$TbTh = \frac{BsBv}{TbN} \tag{5}$$

Trabecular separation (TbSp) is calculated in the same way than TbTh, but this time the voxels representing non-bone parts are filled with maximal spheres. TbSp can thus be expressed as the average thickness of the marrow cavities.

$$TbSp = \frac{1 - BsBv}{TbN} \tag{6}$$

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3588]
Distributed under Creative Commons Attribution License

3 Filter Usage

3.1 itk::BoneMorphometryFeaturesFilter

For a given N-dimensional input image, `itk::BoneMorphometryFeaturesFilter` will provide a set of 5 bone morphometry features summarizing the whole image. This filter behaves as a filter with an input image and output value. Thus it can be inserted in a pipeline with other filters and the metrics will only be recomputed if a downstream filter changes.

Template Parameters (if used in C++):

- The input image type: it must be a ND image of any type.
- The mask image type: it also must be a ND image of any type (will be unsigned char by default)

Inputs and parameters:

- An input image
- A mask defining the region over which features will be calculated. (Optional)
- A Threshold that will be used to determine if each voxel is part of the bone or not. Every voxel with an intensity higher than the threshold will be considered as part of the bone.

3.2 itk::BoneMorphometryFeaturesImageFilter

For each voxel of the input image, the itkBoneMorphometryFeaturesImageFilter will compute a 5-D vector containing a local bone morphometry feature for that voxel. The output of the filter is a N-D image where each pixel will contain a vector of 5 scalars. Each feature map can be extracted from the output image afterward thanks to `itk::NthElementImageAdaptor`. By default the morphometry features are computed for each spatial direction and averaged afterward.

Template Parameters:

- The input image type: must be a N-D image of any type.
- The output image type: must be a N-D image where the pixel type must be a vector of floating points or an ImageVector.
- The mask image type: must be a N-D image of any type (will be unsigned char by default)

Inputs and parameters:

- An input image
- A mask defining the region over which features will be calculated. (Optional)
- A threshold that will be used to determine if each voxel is part of the bone or not. Every voxel with an intensity higher than the threshold will be considered as part of the bone.
- The size of the neighborhood radius. (Optional, defaults to 2.)

Distributed under Creative Commons Attribution License

3.3 Recommendations

To obtain significant results, it is important to carefully choose the parameters depending on the input data and the significant information that need to be revealed by the output. The radius of the neighborhood should be chosen depending on the scale of the trabecular spaces and resolution of the input data and the size of the anomaly/object that needs to be detected in the input image. The threshold will need to be specifically adapted to every input data; it is possible to use a segmentation of the bone as an input by setting the threshold to 1.

The usage of a Region Of Interest (ROI) mask is strongly advised, it will reduce the computation time by avoiding computing features for the noisy/background parts of the image.

In addition to the settings, particular attention should be payed to the input data. Please consider cropping the input so it contains only areas that will be interesting for the analysis. This will both help improve the computation time, and avoid memory problems due to large output data (consider that the output data is 8 or 10 times bigger than the input data).

The memory problems arise from output data that is too large, separate the output feature map image into several scalar feature images with the ITK class `itk::VectorIndexSelectionCastImageFilter`.

3.4 Python Packages

Python wheels allow easily installation of *itkBoneMorphometry* filters and all their dependencies to use with any other Python tools. They have been generated for the three main operating systems (Mac, Linux and Windows) and three versions of Python (2.7, 3.5 and 3.6). To install the Python package, use the following command from your shell:

```
python -m pip install itk-bonemorphometry
```


Distributed under Creative Commons Attribution License

4 Practical examples

4.1 C++

itk::BoneMorphometryFeaturesFilter

```
#include "itkBoneMorphometryFeaturesFilter.h"
#include "itkMath.h"
#include "itkImage.h"
#include "itkVector.h"
#include "itkImageFileReader.h"
#include "itkTestingMacros.h"

int BoneMorphometryFeaturesFilterInstantiationTest( int argc, char *argv[] )
{
    if( argc < 3 )
      {
      std::cerr << "Missing parameters." << std::endl;
      std::cerr << "Usage: " << argv[0]
        << " inputImageFile"
        << " maskImageFile"
        << " threshold"
        << std::endl;
      return EXIT_FAILURE;
      }

  const unsigned int ImageDimension = 3;

  // Declare types
  typedef float                                          InputPixelType;
  typedef itk::Image< InputPixelType, ImageDimension >   InputImageType;
  typedef itk::ImageFileReader< InputImageType >         ReaderType;

  // Create and set up a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );

  // Create and set up a maskReader
  ReaderType::Pointer maskReader = ReaderType::New();
  maskReader->SetFileName( argv[2] );

  // Create the filter
  typedef itk::BoneMorphometryFeaturesFilter<InputImageType> FilterType;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( reader->GetOutput() );
  filter->SetMaskImage( maskReader->GetOutput() );
  filter->SetThreshold( std::atoi( ragv[3]) );
  filter->Update();

  filter->GetBVTV();
  filter->GetTbN();
  filter->GetTbTh();
  filter->GetTbSp();
  filter->GetBSBV();

  return EXIT_SUCCESS;
}
```


Distributed under Creative Commons Attribution License

itk::BoneMorphometryFeaturesImageFilter

```
#include "itkBoneMorphometryFeaturesImageFilter.h"

#include "itkMath.h"
#include "itkImage.h"
#include "itkVector.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkTestingMacros.h"

int BoneMorphometryFeaturesImageFilterInstantiationTest( int argc, char *argv[] )
{
    if( argc < 5 )
      {
      std::cerr << "Missing parameters." << std::endl;
      std::cerr << "Usage: " << argv[0]
        << " inputImageFile"
        << " maskImageFile"
        << " outputImageFile"
        << " threshold"
        << " neighborhoodRadius"
        << std::endl;
      return EXIT_FAILURE;
      }

  const unsigned int ImageDimension = 3;
  const unsigned int VectorComponentDimension = 5;

  // Declare types
  typedef float                                          InputPixelType;
  typedef itk::Image< InputPixelType, ImageDimension >  InputImageType;
  typedef itk::ImageFileReader< InputImageType >        ReaderType;
  typedef itk::Neighborhood<typename InputImageType::PixelType,
      InputImageType::ImageDimension>                   NeighborhoodType;
  typedef float                                          OutputPixelComponentType;
  typedef itk::Vector< OutputPixelComponentType, VectorComponentDimension >
                                                        OutputPixelType;
  typedef itk::Image< OutputPixelType, ImageDimension > OutputImageType;

  // Create and set up a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );

  // Create and set up a maskReader
  ReaderType::Pointer maskReader = ReaderType::New();
  maskReader->SetFileName( argv[2] );

  // Create the filter
  typedef itk::BoneMorphometryFeaturesImageFilter<InputImageType, OutputImageType, InputImageType> FilterType;
  FilterType::Pointer filter = FilterType::New();

  filter->SetInput( reader->GetOutput() );
  filter->SetMaskImage( maskReader->GetOutput() );
  filter->SetThreshold( std::atoi( ragv[4]) );
  NeighborhoodType neighborhood;
  neighborhood.SetRadius( std::atoi(argv[5]) );
  filter->SetNeighborhoodRadius(neighborhood.GetRadius());
```


Distributed under Creative Commons Attribution License

```
  filter->Update();

  // Create and set up a writer
  typedef itk::ImageFileWriter< OutputImageType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName( argv[3] );
  writer->SetInput( filter->GetOutput() );
  writer->Update();

  return EXIT_SUCCESS;
}
```

4.2 Python

itk.BoneMorphometryFeaturesFilter

```
import itk, sys

if len(sys.argv) != 4:
    print("Usage: " + sys.argv[0] + " <inputImagePath> "
                                    " <maskImagePath> "
                                    " <threshold> ")
    sys.exit(1)


Dimension = 3

#Input scan reader
InputPixelType = itk.ctype('signed short')
InputImageType = itk.Image[InputPixelType, Dimension]
imReader = itk.ImageFileReader[InputImageType].New()
imReader.SetFileName(sys.argv[1])

#Input mask reader
MaskPixelType = itk.ctype('unsigned char')
MaskImageType = itk.Image[MaskPixelType, Dimension]
maskReader = itk.ImageFileReader[MaskImageType].New()
maskReader.SetFileName(sys.argv[2])

im = imReader.GetOutput()
mask = maskReader.GetOutput()

filtr = itk.BoneMorphometryFeaturesFilter.New(im)
filtr.SetMaskImage(mask)
filtr.SetThreshold(int(sys.argv[3]))

filtr.Update()

print filtr.GetBVTV()
print filtr.GetTbN()
print filtr.GetTbTh()
print filtr.GetTbSp()
print filtr.GetBSBV()
```


Distributed under Creative Commons Attribution License

itk.BoneMorphometryFeaturesImageFilter

```
import itk, sys

if len(sys.argv) != 6:
    print("Usage: " + sys.argv[0] + " <inputImagePath> "
                                   " <maskImagePath> "
                                   " <outputImagePath> "
                                   " <threshold> "
                                   " <neigborhoodRadius> ")
    sys.exit(1)


Dimension = 3

#Input scan reader
InputPixelType = itk.ctype('signed short')
InputImageType = itk.Image[InputPixelType, Dimension]
imReader = itk.ImageFileReader[InputImageType].New()
imReader.SetFileName(sys.argv[1])

#Input mask reader
MaskPixelType = itk.ctype('unsigned char')
MaskImageType = itk.Image[MaskPixelType, Dimension]
maskReader = itk.ImageFileReader[MaskImageType].New()
maskReader.SetFileName(sys.argv[2]t

im = imReader.GetOutput()
mask = maskReader.GetOutput()

filtr = itk.BoneMorphometryFeaturesImageFilter.New(im)
filtr.SetMaskImage(mask)
filtr.SetThreshold(int(sys.argv[4]))
filtr.SetNeighborhoodRadius([int(sys.argv[5]),int(sys.argv[5]),int(sys.argv[5])])

result = filtr.GetOutput()

itk.imwrite(result, sys.argv[3])
```


Distributed under Creative Commons Attribution License

5 Results

We presented concrete use case scenarios of the itkBoneMorphometry's filters in this section. We used itkBoneMorphometry to characterize subchondral bone structure in temporomandibular joint (TMJ) Osteoarthritis (OA). To date, there is no single sign, symptom, or test that can clearly diagnose early stages of TMJ OA. However, it has been observed that changes in the subchondral bone occur in very early stages of this disease involving structural changes in the subchondral bone (i.e. bone marrow).

The different tools presented in this document can aid highlighting those structural variations to help clinicians to detect TMJ OA earlier in disease progression.

In the test case (figure 2), the lower part of the condyle is healthy (normal bone trabeculae density) whereas the upper part is characteristic of a TMJ OA case (low bone trabeculae density).

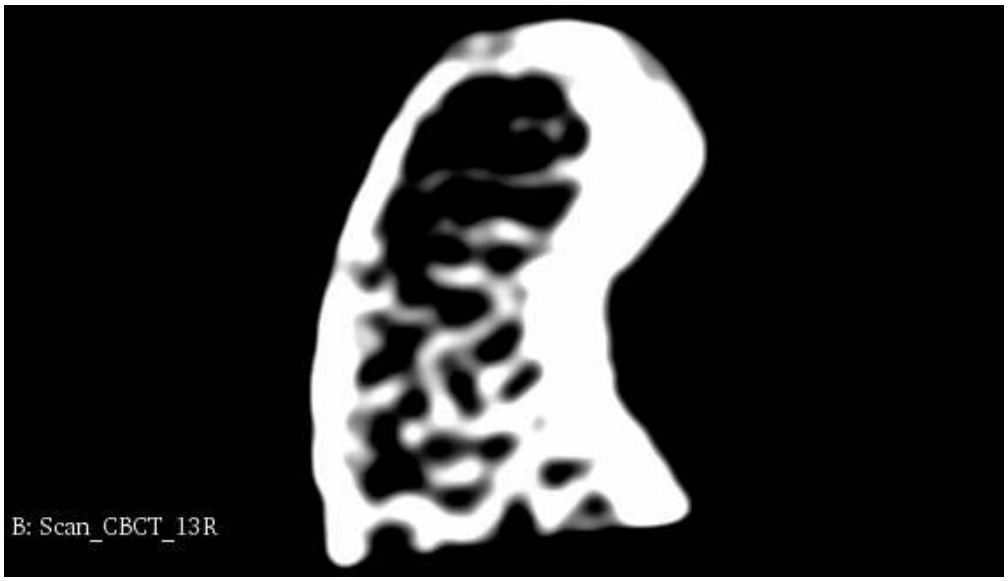

Figure 2: CBCT of the test condyle: this condyle suffers of a lack of trabecula in the upper part

The results exposed in this part were obtained by specifying the following parameters (the default parameters were used for the other ones):

- Input data: Scan_CBCT_13R.nrrd
- Input mask: SegmC_CBCT_13R.nrrd
- Threshold: 1100
- Neighborhood Radius: 6

Distributed under Creative Commons Attribution License

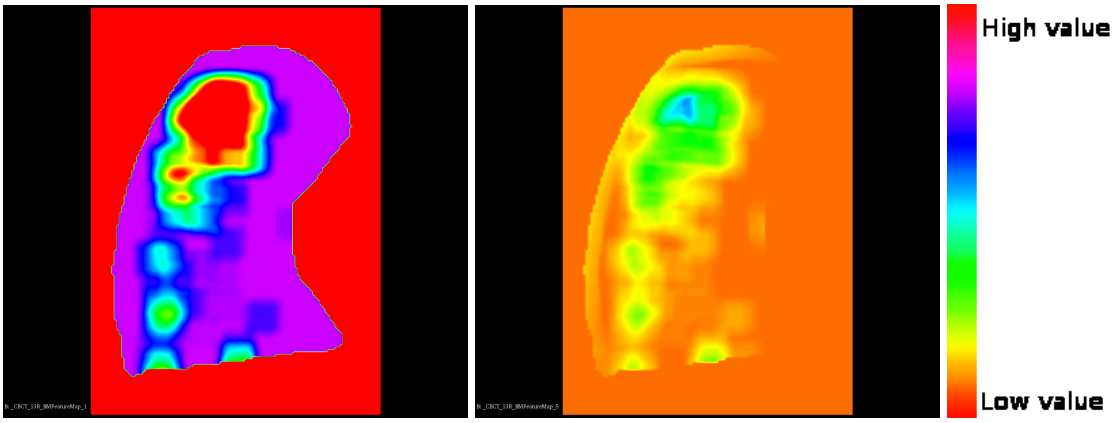

Figure 3: BSBV (left) and BVTV (right)

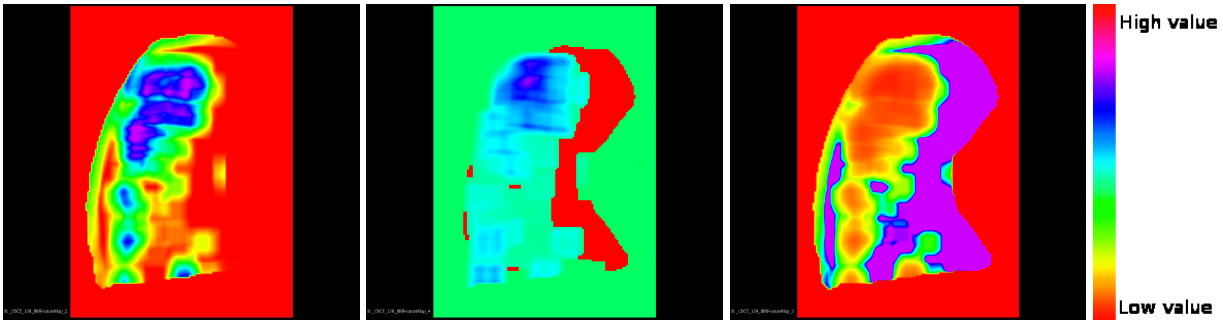

Figure 4: TbN (left), TbSp (center) and TbTh (right)

All the different bone morphometry feature maps computed for this case (figure 3 and 4) seem to discriminate unaffected areas from affected areas in the TMJ trabecular bone. It is highly probable that those features can help in an automatic detection of TM JOA.

Distributed under Creative Commons Attribution License

6 Conclusion

This document presented a new, fast, and efficient tool to compute bone morphometry features in N-Dimensional images. The described features are correlated to each other, so they do not provide independent discrimination. The usage with a combination with other types of image feature descriptors (such as some textural features[1]) will allow the detection of a larger number of disease variations. This method is currently used in conjunction with other biomarkers in a large study aimed to create a new method to detect TMJ OA at early stages using subchondral bone structure as a biomarker.

Acknowledgements

This work was supported by the National Institute of Health (NIH) National Institute for Dental and Craniofacial Research (NIDCR) grant R01EB021391 (Textural Biomarkers of Arthritis for the Subchondral Bone in the Temporomandibular Joint), NIDCR grant R01DE024450 (Quantification of 3D bony Changes in Temporomandibular Joint Osteoarthritis) and National Institute of Biomedical Imaging and Bioengineering (NIBIB) grant R01EB021391 (Shape Analysis Toolbox for Medical Image Computing Projects).

We would like to thank Dr. Larry Wolford from the Baylor University Medical Center for kindly providing the bone specimens from which we obtained the scans used in the paper. We would like to thank Drs. Lucia Cevidanes, Erika Benavides and Antonio Ruellas at the University of Michigan School of Dentistry as well, for generating the CBCT scans that were processed with the filters presented in the paper.

We are also grateful for the support received from the ITK community.

References

[1] F. Budin B. Paniagua J. Vimort, M. McCormick. Computing textural feature maps for n-dimensional images. *Insight Journal*, 2017. 6

[2] W. Jee. The past, present, and future of bone morphometry: its contribution to an improved understanding of bone biology. *Journal of Bone and Mineral Metabolism*, 23(1):1–10, 2005. 1

[3] R. Müller R. Voide, G. van Lenthe. Bone morphometry strongly predicts cortical bone stiffness and strength, but not toughness, in inbred mouse models of high and low bone mass. *Journal of Bone and Mineral Metabolism*, 23(8):203, 2008. 1

Distributed under Creative Commons Attribution License