MedCV

Release 1.0.0

Zhang Hongyuan

Apr 26, 2021

CONTENTS

CHAPTER

ONE

MEDCV: OPENCV FOR MEDICAL

1.1 You have just found MedCV

MedCV is an advanced medical image visualization library, which is written by Numpy and OpenCV to complete image visualization. MedCV was born to solve the compatibility between OpenCV and medical images. It would help you to quickly visualize your results.

If you have the following requirements, please choose MedCV:

- Simple and fast medical visualization design (MedCV is highly modular, minimalist, and expandable)
- Establish a medical image interactive platform

1.2 Guiding principles

- **Friendly**. MedCV is an API designed for humans. MedCV follows best practices for reducing cognitive difficulties: MedCV provides a consistent and concise API to quickly complete visualization tasks.
- **Modularity**. MedCV has three independent modules: visualize, tools, and GUI controls. You can use them to customize your own visualization operations or build an interactive platform.
- **Work with Python**. No separate configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

1.3 Version and update

This document is the English document of MedCV, including all the contents of MedCV, as well as more examples, explanations and suggestions. If you have any comments, suggestions or questions during the use process, please send email to moyan_work@foxmail.com to get in touch with me.

Any contribution you made to the document, including document translation, checking for deficiencies, conceptual explanations, finding and modifying problems, contributing sample programs, etc., will be recorded in the acknowledgment. Thank you very much for your contribution to MedCV!

1.4 Installation

Before installing MedCV, please install the following requirements:

- Numpy (used for image matrix operations)
- OpenCV (Compatible call completion visualization)
- PyQt5 (design control for customized medical image version)

Install MedCV in two ways:

- **Use PyPI to install MedCV(recommended)**

```
sudo pip install medcv
```

If you use virtual environment, you can avoid using sudo:

```
pip install medcv
```

- **Oruse Github source code to install MedCV**

First, use`git`to clone MedCV:

```
git clone https://github.com/szuboy/medcv.git
```

Then, `cd`to the MedCV directory and run the installation command:

```
cd medcv
sudo python setup.py install
```

1.5 Support

You can ask questions or join the MedCV development discussion at the following URL:

- MedCV Google group
- You can also ask questions or request new features in Github issues (please make sure you read our guide before asking questions, and I will answer it frequently for you)

CHAPTER

TWO

VISUALIZE API

2.1 Classification

2.1.1 image_with_heatmap

```
medcv.visualize.cls.image_with_heatmap(image, heatmap, alpha=0.8, beta=0.3,␣
↪colormap=cv2.COLORMAP_JET, width=None, level=None)
```

Visualize the attention heatmap on the medical image. For example, the Grad-CAM heatmap of the deep learning classification network is superimposed on the medical image for visualization.

This function is implemented by the `cv2.addWdighted` function, which is an image fusion method. The fusion is performed according to the following equation:

$$dst = \alpha \cdot image + \beta \cdot heatmap + \gamma$$

Parameter

- **image** (np.ndarray): medical image with 2-dim
- **heatmap** (np.ndarray): heatmap corresponding to `image`
- **alpha** (float): weight corresponding to `image` , the value range is `0 ~ 1` , default is `0.8`
- **beta** (float): weight corresponding to `heatmap` , the value range is `0 ~ 1` , default is `0.3`
- **colormap**: default is `cv2.COLORMAP_JET` , refer to this document for details
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image)`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`

Usage

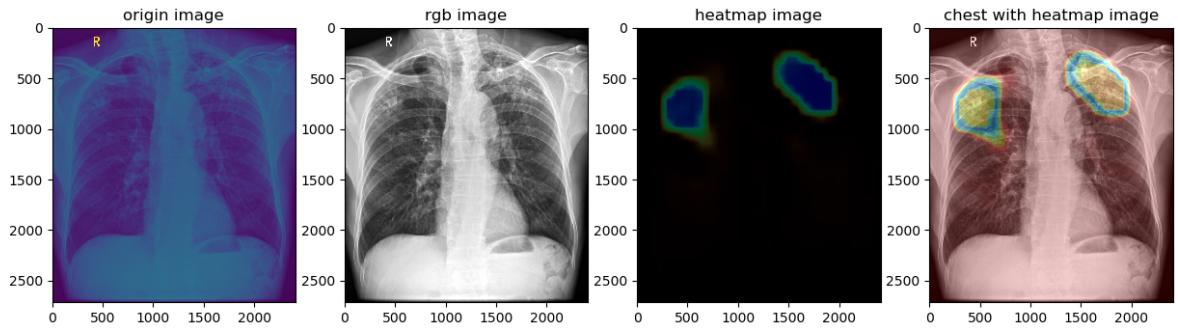
```
import numpy as np
import matplotlib.pyplot as plt
from medcv.tools.transform import med2rgb
from medcv.data import chest_dcm, chest_heatmap
from medcv.visualize.cls import image_with_heatmap

# load chest dcm image and its heatmap
chest_dcm_image = chest_dcm()
```

(continues on next page)

(continued from previous page)

```
chest_heatmap_image = chest_heatmap()

# dcm to rgb image with image window width and level
chest_rgb_image = med2rgb(chest_dcm_image, width=22135, level=12209)

# dcm with heatmap
chest_image_with_heatmap = image_with_heatmap(chest_dcm_image, chest_heatmap_image,␣
↪width=22135, level=12209)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('origin image')
plt.imshow(np.squeeze(chest_dcm_image))
plt.subplot(1, 4, 2)
plt.title('rgb image')
plt.imshow(chest_rgb_image)
plt.subplot(1, 4, 3)
plt.title('heatmap image')
plt.imshow(chest_heatmap_image)
plt.subplot(1, 4, 4)
plt.title('chest with heatmap image')
plt.imshow(chest_image_with_heatmap)
plt.show()
```

Result



2.2 Detection

2.2.1 image_with_bbox

```
medcv.visualize.det.imag_with_bbox(image, bbox, bbox_ids=None, colors=None,␣
↪thickness=1, width=None, level=None)
```

Visualize the target detection bounding box on the medical image, and support the visualization of multiple different object frames by specifying the `bbox_ids` parameter

Parameter

- **image** (np.ndarray): medical image with 2-dim
- **bbox** (list): bbox list, the form is `[(x, y, w, h), (x, y, w, h), ...]`
- **bbox_ids** (list): id list of bbox, each box corresponds to an id to distinguish different category, default is `None`, which is each box corresponds to a category
- **colors** (list): color list for different category, the form is `[(r, g, b), (r, g, b), ...]` . If is `None`, the color will be random
- **thickness** (int)thickness for boundary, default is `thickness=1`
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image))`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`

Usage

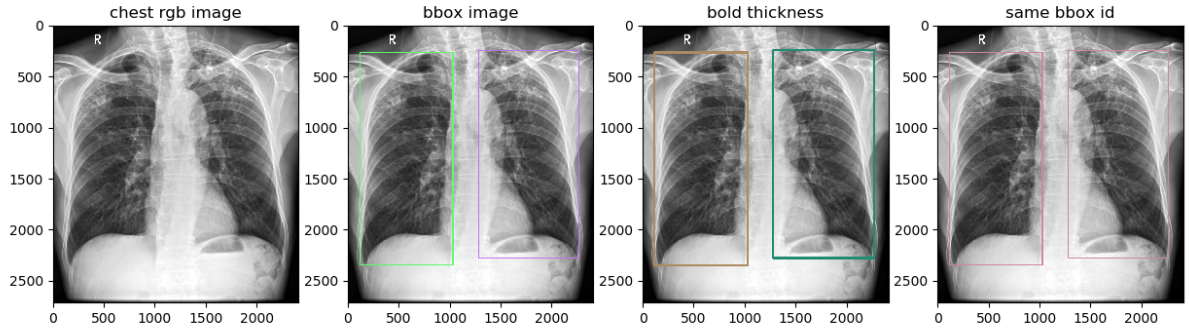
```
import matplotlib.pyplot as plt
from medcv.tools.transform import med2rgb
from medcv.data import chest_dcm, chest_bbox
from medcv.visualize.det import image_with_bbox

# load chest dcm image and its heatmap
chest_dcm_image = chest_dcm()
chest_bbox_dict = chest_bbox()

# bbox list, because chest_bbox_dict has left-lung and right-lung bbox
bbox_list = chest_bbox_dict.values()

# dcm to rgb image with image window width and level
chest_rgb_image = med2rgb(chest_dcm_image, width=22135, level=12209)

# dcm with bbox, default is different bbox id.
chest_image_with_bbox = image_with_bbox(chest_dcm_image, bbox_list, width=22135,␣
↪level=12209, thickness=10)

# dcm with bbox with thickness=20
chest_image_with_bold_bbox = image_with_bbox(chest_dcm_image, bbox_list, width=22135,␣
↪level=12209, thickness=20)

# dcm with same bbox id
chest_image_with_same_bbox = image_with_bbox(chest_dcm_image, bbox_list, bbox_ids=[0,␣
↪0], width=22135, level=12209, thickness=10)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('chest rgb image')
plt.imshow(chest_rgb_image)
plt.subplot(1, 4, 2)
plt.title('bbox image')
plt.imshow(chest_image_with_bbox)
plt.subplot(1, 4, 3)
plt.title('bold thickness')
plt.imshow(chest_image_with_bold_bbox)
```

(continues on next page)

(continued from previous page)

```
plt.subplot(1, 4, 4)
plt.title('same bbox id')
plt.imshow(chest_image_with_same_bbox)
plt.show()
```

Result



2.3 Segmentation

2.3.1 imag_with_mask

```
medcv.visualize.seg.imag_with_mask(image, mask, alpha=0.5, colors=None, width=None,␣
↪level=None)
```

Visualize the labeled Mask on the medical image. For example: visualize the degree of overlap between the segmentation result and the ground truth(GT).

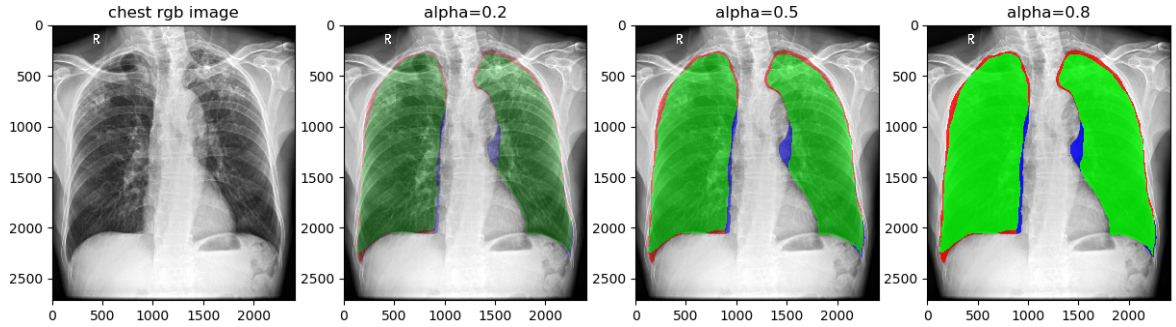
Parameter

- **image** (np.ndarray): medical image with 2-dim
- **mask** (np.ndarray): mask corresponding to `image` , default zero is background, non-zeros is region of interest(ROI)
- **alpha** (float): weight corresponding to `mask` , the value range is `0 ~ 1` , default is `0.5`
- **colors** (list): color list for different ROI, the form is `[(r, g, b), (r, g, b), ...]` . If is `None`, the color will be random
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image))`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`

Usage

```
import numpy as np
import matplotlib.pyplot as plt
from medcv.tools.transform import med2rgb
from medcv.data import chest_dcm, chest_mask
from medcv.visualize.seg import image_with_mask

# load chest dcm image and its mask
chest_dcm_image = chest_dcm()
chest_mask_image = chest_mask()

# dcm to rgb image with image window width and level
chest_rgb_image = med2rgb(chest_dcm_image, width=22135, level=12209)

# visualize colors
colors = [(0, 255, 0), (255, 0, 0), (0, 0, 255)]

# alpha=0.2
chest_image_with_mask1 = image_with_mask(chest_dcm_image, chest_mask_image,␣
↪colors=colors, alpha=0.2, width=22135, level=12209)
# alpha=0.5 (default alpha=0.5)
chest_image_with_mask2 = image_with_mask(chest_dcm_image, chest_mask_image,␣
↪colors=colors, alpha=0.5, width=22135, level=12209)
# alpha=0.8
chest_image_with_mask3 = image_with_mask(chest_dcm_image, chest_mask_image,␣
↪colors=colors, alpha=0.8, width=22135, level=12209)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('chest rgb image')
plt.imshow(chest_rgb_image)
plt.subplot(1, 4, 2)
plt.title('alpha=0.2')
plt.imshow(chest_image_with_mask1)
plt.subplot(1, 4, 3)
plt.title('alpha=0.5')
plt.imshow(chest_image_with_mask2)
plt.subplot(1, 4, 4)
plt.title('alpha=0.8')
plt.imshow(chest_image_with_mask3)
plt.show()
```

Result



Remarks: green is the overlap between the GT and the segmentation result, red+green=GT, blue+green=segmentation result.

2.3.2 image_with_contours

```
medcv.visualize.seg.image_with_contours(image, mask, colors=None, thickness=1,␣
↪width=None, level=None)
```

This function is implemented by the `cv2.drawContours` function to visualize the edge contour of the mask on the medical image. For example, comparing the segmentation result with the GT edge information.

Parameter

- **image** (np.ndarray): medical image with 2-dim
- **mask** (np.ndarray): mask corresponding to `image` , default zero is background, non-zeros is region of interest(ROI)
- **colors** (list): color list for different ROI, the form is `[(r, g, b), (r, g, b), ...]` . If is `None`, the color will be random
- **thickness** (int)thickness for boundary, default is `thickness=1`
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image))`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`

Usage

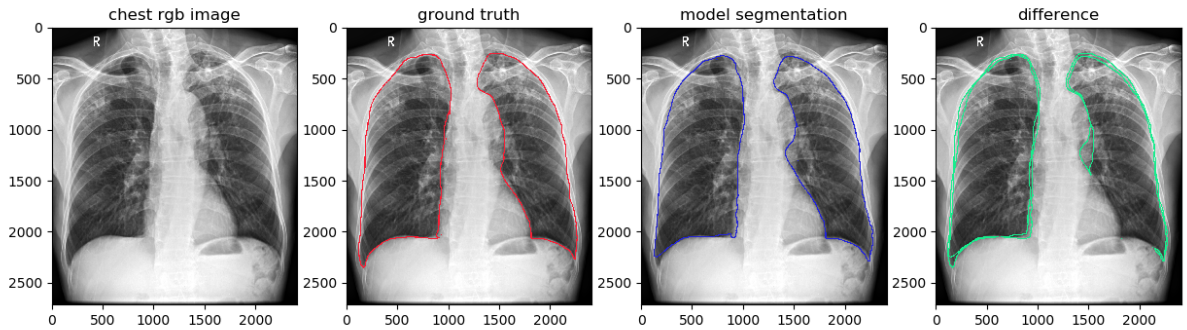
```
import numpy as np
import matplotlib.pyplot as plt
from medcv.tools.transform import med2rgb
from medcv.data import chest_dcm, chest_mask
from medcv.visualize.seg import image_with_contours

# load chest dcm image and its mask
chest_dcm_image = chest_dcm()
chest_mask_image = chest_mask()
```

(continues on next page)

(continued from previous page)

```
# dcm to rgb image with image window width and level
chest_rgb_image = med2rgb(chest_dcm_image, width=22135, level=12209)

# ground truth
gt_mask = np.zeros_like(chest_mask_image)
gt_mask[np.logical_or(chest_mask_image == 1, chest_mask_image == 2)] = 1
chest_image_with_gt = image_with_contours(chest_dcm_image, gt_mask, thickness=10,␣
↪width=22135, level=12209)
# model segmentation
seg_mask = np.zeros_like(chest_mask_image)
seg_mask[np.logical_or(chest_mask_image == 1, chest_mask_image == 3)] = 1
chest_image_with_seg = image_with_contours(chest_dcm_image, seg_mask, thickness=10,␣
↪width=22135, level=12209)
# difference between ground truth and model segmentation
diff_mask = np.zeros_like(chest_mask_image)
diff_mask[np.logical_or(chest_mask_image == 2, chest_mask_image == 3)] = 1
chest_image_with_diff = image_with_contours(chest_dcm_image, diff_mask, thickness=10,␣
↪width=22135, level=12209)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('chest rgb image')
plt.imshow(chest_rgb_image)
plt.subplot(1, 4, 2)
plt.title('ground truth')
plt.imshow(chest_image_with_gt)
plt.subplot(1, 4, 3)
plt.title('model segmentation')
plt.imshow(chest_image_with_seg)
plt.subplot(1, 4, 4)
plt.title('difference')
plt.imshow(chest_image_with_diff)
plt.show()
```

Result



CHAPTER

THREE

TOOLS API

3.1 Transform

Image transform is to map the medical image to output format by establishing a corresponding lookup table(LUT), which can be used to adjust the medical image window and convert the medical image to the natural image format (mapping between 0-255) and other scenes.

3.1.1 med2med

```
medcv.tools.med2med(med_image, width=None, level=None, dtype=np.float64, invert=False)
```

This function is mainly used to efficiently adjust image window, return the medical image after window adjustment, and support two-dimensional or three-dimensional input.

Parameter

- **med_image** (np.ndarray): medical image, support 2-dim or 3-dim
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image))`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`
- **dtype**: output's data type, default is `np.float64`
- **invert**: whether to reverse mapping, default is `False`

Usage

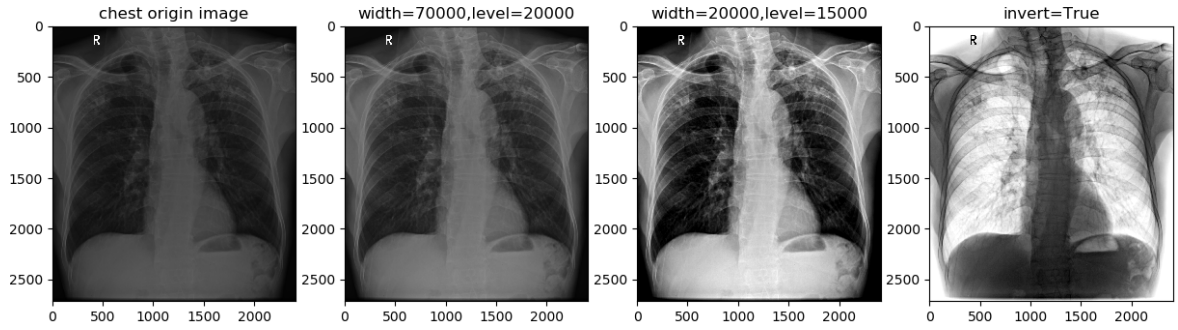
```
import matplotlib.pyplot as plt
from medcv.data import chest_dcm
from medcv.tools.transform import med2med, med2rgb

# load chest dcm image
chest_dcm_image = chest_dcm()

# width=50000, level=20000
adjust_image1 = med2med(chest_dcm_image, width=50000, level=20000)

# width=20000, level=15000
adjust_image2 = med2med(chest_dcm_image, width=20000, level=15000)
```

(continues on next page)

(continued from previous page)

```
# width=20000, level=15000, invert=True
adjust_image3 = med2med(chest_dcm_image, width=20000, level=15000, invert=True)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('chest origin image')
plt.imshow(med2rgb(chest_dcm_image))
plt.subplot(1, 4, 2)
plt.title('width=70000,level=20000')
plt.imshow(med2rgb(adjust_image1))
plt.subplot(1, 4, 3)
plt.title('width=20000,level=15000')
plt.imshow(med2rgb(adjust_image2))
plt.subplot(1, 4, 4)
plt.title('invert=True')
plt.imshow(med2rgb(adjust_image3))
plt.show()
```

Result



3.1.2 med2grey

```
medcv.tools.med2grey(med_image, width=None, level=None, invert=False)
```

The function is to convert a two-dimensional medical image into a grayscale image, which supports window adjust and returns the converted grayscale image to be compatible with the OpenCV input format.

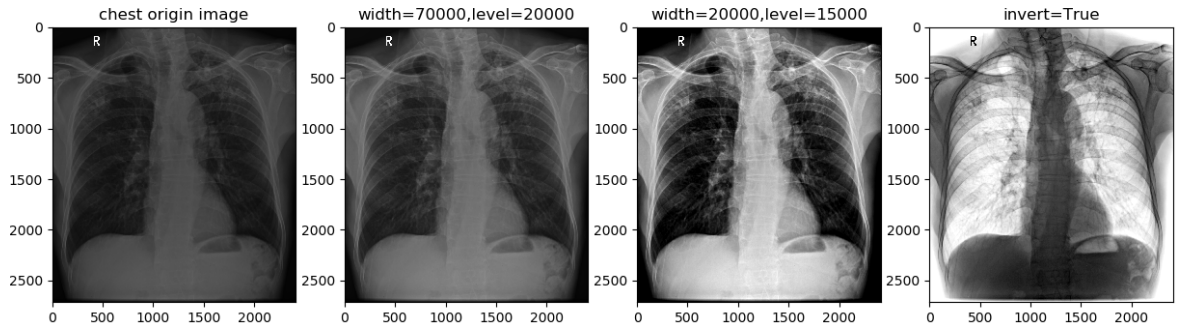
Parameter

- **med_image** (np.ndarray): medical image with 2-dim
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image))`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`
- **invert**: whether to reverse mapping, default is `False`

Usage

```
import matplotlib.pyplot as plt
from medcv.data import chest_dcm
from medcv.tools.transform import med2grey

# load chest dcm image
chest_dcm_image = chest_dcm()

# width=50000, level=20000
grey_image1 = med2grey(chest_dcm_image, width=50000, level=20000)

# width=20000, level=15000
grey_image2 = med2grey(chest_dcm_image, width=20000, level=15000)

# width=20000, level=15000, invert=True
grey_image3 = med2grey(chest_dcm_image, width=20000, level=15000, invert=True)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('chest origin image')
plt.imshow(med2grey(chest_dcm_image), cmap='gray')
plt.subplot(1, 4, 2)
plt.title('width=70000,level=20000')
plt.imshow(grey_image1, cmap='gray')
plt.subplot(1, 4, 3)
plt.title('width=20000,level=15000')
plt.imshow(grey_image2, cmap='gray')
plt.subplot(1, 4, 4)
plt.title('invert=True')
plt.imshow(grey_image3, cmap='gray')
plt.show()
```

Result



3.1.3 med2rgb

```
medcv.tools.med2rgb(med_image, width=None, level=None, invert=False)
```

The function is to convert a two-dimensional medical image into a rgb image, which supports window adjust and returns the converted rgb image to be compatible with the OpenCV input format.

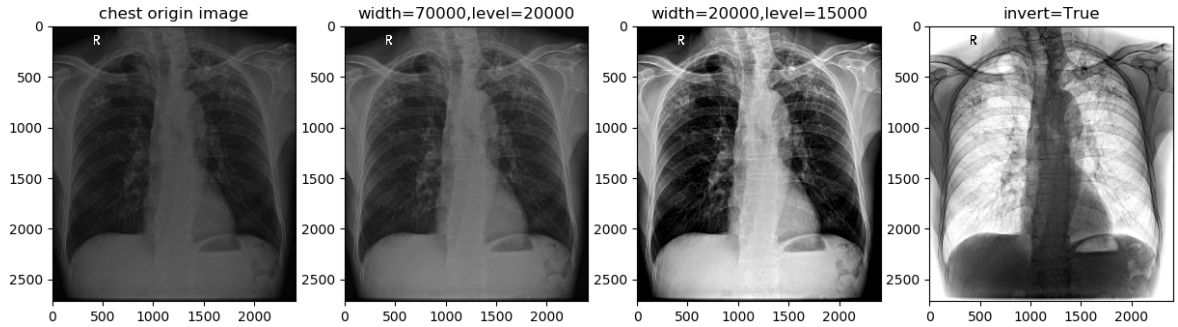
Parameter

- **med_image** (np.ndarray): medical image with 2-dim
- **width** (int): window width for image, default is `None`, which is `width=max(image)-min(image))`
- **level** (int): window level for image, default is `None`, which is `level=((image)+min(image))/2`
- **invert**: whether to reverse mapping, default is `False`

Usage

```
import matplotlib.pyplot as plt
from medcv.data import chest_dcm
from medcv.tools.transform import med2rgb

# load chest dcm image
chest_dcm_image = chest_dcm()

# width=50000, level=20000
rgb_image1 = med2rgb(chest_dcm_image, width=50000, level=20000)

# width=20000, level=15000
rgb_image2 = med2rgb(chest_dcm_image, width=20000, level=15000)

# width=20000, level=15000, invert=True
rgb_image3 = med2rgb(chest_dcm_image, width=20000, level=15000, invert=True)

# plot visualize
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.title('chest origin image')
plt.imshow(med2rgb(chest_dcm_image))
plt.subplot(1, 4, 2)
plt.title('width=70000,level=20000')
plt.imshow(rgb_image1)
plt.subplot(1, 4, 3)
plt.title('width=20000,level=15000')
plt.imshow(rgb_image2)
plt.subplot(1, 4, 4)
plt.title('invert=True')
plt.imshow(rgb_image3)
plt.show()
```

Result


chest origin image
width=70000,level=20000
width=20000,level=15000
invert=True
R
0
500
1000
1500
2000
2500

CHAPTER

FOUR

MEDQT API

4.1 QMedManger

```
medcv.gui.QMedManager()
```

Medical image manager, used to manage medical image information and data format conversion. This class is also the built-in manager of other GUI controls, completing the efficient collaboration between medical images and PyQt5.

4.1.1 API

set

- `set_invert(invert)`: sets invert mode to the invert parameter
- `set_medical_image(medical_image)`: sets medical image
- `set_image_window_width(window_width)`: sets medical image window width
- `set_image_window_level(window_level)`: sets medical image window level
- `set_image_window(window_width, window_level)`: sets medical image window

get

- `is_invert()`: gets invert mode
- `get_medical_image()`: gets medical image
- `get_image_window_width()`: gets medical image window width
- `get_image_window_level()`: gets medical image window level
- `get_image_window()`: gets medical image window of the manager

other

- `pixmap()`: return`pixmap` format image
- `update_lut_array()`: update mapping lookup table

4.2 QMedLabel

```
medcv.gui.QMedLabel()
```

This class inherits from the `QLabel` class of PyQt5, and is encapsulated as a display panel for medical images, which can quickly complete the display of medical images.

4.2.1 API

set

- `setInvert(invert)`: sets invert mode to the invert parameter
- `setMedicalImage(medicalImage)`: sets medical image
- `setImageWindowWidth(windowWidth)`: sets medical image window width
- `setImageWindowLevel(windowLevel)`: sets medical image window level
- `setImageWindow(windowWidth, windowLevel)`: sets medical image window

get

- `isInvert()`: gets invert mode
- `getMedicalImage()`: gets medical image
- `getImageWindowWidth()`: gets medical image window width
- `getImageWindowLevel()`: gets medical image window level
- `getImageWindow()`: gets medical image window of the manager

4.2.2 Usage

```
import sys
from medcv.gui import QMedLabel
from medcv.data import chest_dcm
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QSlider


application = QApplication(sys.argv)

main_widget = QWidget()
layout = QVBoxLayout(main_widget)

med_label = QMedLabel()
```

(continues on next page)

(continued from previous page)

```
med_label.setMaximumSize(600, 600)
med_label.setScaledContents(True)
med_label.setMedicalImage(chest_dcm())


width_slider = QSlider(Qt.Horizontal)
level_slider = QSlider(Qt.Horizontal)
width_slider.setRange(1, 50000)
width_slider.valueChanged.connect(med_label.setImageWindowWidth)
level_slider.setRange(0, 50000)
level_slider.valueChanged.connect(med_label.setImageWindowLevel)

layout.addWidget(med_label)
layout.addWidget(width_slider)
layout.addWidget(level_slider)

main_widget.show()

sys.exit(application.exec_())
```

CHAPTER

FIVE

UTILS API

5.1 LUT

Lookup Table(LUT) is an array that replaces complex operations by indexing, and converts input data into a more ideal output format, thereby saving a lot of processing time. Especially in the interactive interface, the fluency of the interaction directly determines the user's experience.

For the processing of medical images(always with a three-dimensional format), the image window adjustment and image format conversion are completed efficiently by establishing of LUT, which provide a guarantee for the smoothness of the interaction of the GUI controls.

5.1.1 Mapping function

Medical images usually have a high dynamic range (for example, the dynamic range of CT is generally -1024~1024), and the image display can be completed through the mapping function. According to DICOM , three mapping functions are defined: `LINEARLINEAR_EXACTSIGMOID`(MedCV uses`LINEAR`by default). Each function is related to dual variables of window level and window width, please refer to the code for details.

5.1.2 LUT building

```
medcv.utils.generate_lut_array(image, width, center, y_min=0, y_max=255, dtype=np.
↪uint8, invert=False, mode=medcv.LINEAR_MODE)
```

This function generates the LUT array based on the incoming image data, and the return value is: `offset` and `lut_array`, please refer to the code for details.

Parameter

- **image** (np.ndarray): medical image array, support 2-dim and 3-dim
- **width** (int): window width for medical image
- **center** (int): window center(level) for medical image
- **y_min** (number): minimum value corresponding to the mapping function, the default is `0`
- **y_max** (number): maximum value corresponding to the mapping function, the default is `255`
- **dtype**: `lut_array` data type, default is `np.uint8`
- **invert** (bool): reverse mapping mode, default is `False`
- **mode**mapping function selection, the options are:`LINEARLINEAR_EXACT` and `SIGMOID`, default is `LINEAR`

5.2 Argument validate

```
medcv.utils.expected_type(*type_args, **type_kwargs)
```

Based on the Python closure and decorator, combined with the `inspect` library `signature` method, the inspection and verification of the function parameter types are elegantly and concisely completed, please refer to the code for details.

CHAPTER

SIX

UPDATE

- 2021.04.22
  - Release the official version v1.0
  - Determine the structure of MedCV package

CHAPTER

SEVEN

FAQS

updating…

CHAPTER

EIGHT

APPENDIX

updating…

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search