OCTseg

Mohammad Haft-Javaherian

Aug 27, 2019

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER TWO

TRAINING AND TESTING

`train.main()`

Train or test a U-Net model to analyze OCT images.

Parameters

- **`***Note**`** – **All arguments are bash arguments**
- **`exp_def`** – experiment definition
- **`models_path`** – path for saving models
- **`lr`** – learning rate
- **`lr_decay`** – learning rate step for decay
- **`data_pat`** – data folder path
- **`nEpoch`** – number of epochs
- **`nBatch`** – batch size
- **`outCh`** – size of output channel
- **`inCh`** – size of input channel
- **`nZ`** – size of input depth
- **`w`** – size of input width (number of columns)
- **`l`** – size of input Length (number of rows)
- **`loss_w`** – loss wights
- **`isAug`** – Is data augmentation
- **`isCarts`** – whether images should be converted into Cartesian
- **`isTest`** – Is test run instead of train
- **`testEpoch`** – epoch of the saved model for testing
- **`saveEpoch`** – epoch interval to save the model
- **`logEpoch`** – epoch interval to save the log")
- **`nFeature`** – number of features in the first layer
- **`nLayer`** – number of layers in the U-Nnet model
- **`gpu_id`** – ID of GPUs to be used
- **`optimizer`** – keras optimizer. see `keras.optimizers()`

See also:

- *unet.unet.unet_model()*
- *unet.loss.multi_loss_fun()*
- *util.load_data.load_train_data()*
- *util.load_batch.load_batch_parallel()*
- `keras.utils.multi_gpu_model()`
- `keras.optimizers()`

CHAPTER

THREE

U-NET

3.1 loss

CNN related loss functions

`unet.loss.dice_loss(label, target)`

Soft Dice coefficient loss

TP, FP, and FN are true positive, false positive, and false negative.

$$dice = \frac{2 \times TP}{2 \times TP + FN + FP}$$

$$dice = \frac{2 \times TP}{(TP + FN) + (TP + FP)}$$

objective is to maximize the dice, thus the loss is negate of dice for numerical stability (+1 in denominator) and fixing the loss range (+1 in numerator and +1 to the negated dice).

The final Dice loss is formulated as

$$dice\ loss = 1 - \frac{2 \times TP + 1}{(TP + FN) + (TP + FP) + 1}$$

it is soft as each components of the confusion matrix (TP, FP, and FN) are estimated by dot product of probability instead of hard classification

Parameters

- **label** – 4D or 5D label tensor
- **target** – 4D or 5D target tensor

Returns dice loss

`unet.loss.multi_loss_fun(loss_weight)`

Semantic loss function based on the weighted cross entropy and dice and wighted by the loss weights in the input argument

Parameters **loss_weight** – a list with two weights for weighted cross entropy and dice losses, respectively.

Returns

function, which similar to *weighted_cross_entropy()* **and** *dice_loss()* has label and target arguments

See also:

- *weighted_cross_entropy()*

- *dice_loss()*

`unet.loss.`**`weighted_cross_entropy`**`(`*label*, *target*`)`

Weighted cross entropy with foreground pixels having ten times higher weights

Parameters

- **`label`** – 4D or 5D label tensor
- **`target`** – 4D or 5D target tensor

Returns weighted cross entropy value

`unet.loss.`**`weighted_cross_entropy_with_boundary`**`(`*label*, *target*`)`

Weighted cross entropy with foreground pixels having ten times higher weights

Parameters

- **`label`** – 4D or 5D label tensor
- **`target`** – 4D or 5D target tensor

Returns weighted cross entropy value

3.2 ops

CNN related operations

`unet.ops.`**`MaxPoolingND`**`(`*x*`)`

Maxpooling in x and y direction for 2D and 3D inputs

Parameters **`x`** – input 4D or 5D tensor

Returns downscaled of *x* in x and y direction

See also:

- *up_conv()*
- `keras.layers.MaxPooling2D()`
- `keras.layers.MaxPooling3D()`

`unet.ops.`**`accuracy`**`(`*labels*, *logits*`)`

Measure accuracy metrics. The code calculate the prediction based on the input logits. Metrics are:

- accuracy: The ratio of correctly labeled voxels to the total number of voxels.
- Jaccard Index: ratio of number of foreground voxels in the intersection of *labels* and *logits* divided by total number of foreground voxels in the union of *labels* and *logits*

$$accuracy = \frac{1}{N \times M \times L} \sum_{i\in[[N]],j\in[[M]],k\in[[L]]} (label_{i,j,k} == predict_{i,j,k})$$

$$Jaccard = \frac{\sum_{i\in[[N]],j\in[[M]],k\in[[L]]} (label_{i,j,k} \;\&\&\; predict_{i,j,k})}{\sum_{i\in[[N]],j\in[[M]],k\in[[L]]} (label_{i,j,k} \parallel predict_{i,j,k})}$$

Parameters

- **`labels`** – 4D or 5D tensor of labels
- **`logits`** – 4D or 5D tensor of prediction logits.

Returns accuracy and Jaccard Index

`unet.ops.`**`conv_layer`**(*x*, *ChOut*)

Multi-layer convolution operators consists of three convolutions (2D or 3D based on the input shape) followed by LeakyReLY.

Parameters

- **`x`** – input 4D or 5D tensor to the layers
- **`ChOut`** – number of features of outputs of all convolutions

Returns output of the final layer with same size as *x*

See also:

- `keras.layers.LeakyReLU()`
- `keras.layers.Conv2D()`
- `keras.layers.Conv3D()`

`unet.ops.`**`placeholder_inputs`**(*im_shape*, *outCh*)

Generate placeholder variables to represent the input tensors.

Parameters

- **`im_shape`** – shape of the input tensor
- **`outCh`** – number of channels in the output

Returns image and label placeholders

`unet.ops.`**`up_conv`**(*x*)

upscaling of input tensor in x and y direction using transpose convolution in 2D or 3D.

Parameters **`x`** – input 4D or 5D tensor

Returns unscaled of *x* in x and y direction

See also:

- *`MaxPoolingND()`*
- `keras.layers..Conv2DTranspose()`
- `keras.layers..Conv3DTranspose()`

3.3 unet

Build U-Net model

`unet.unet.`**`unet_model`**(*im_shape*, *nFeature=32*, *outCh=2*, *nLayer=3*)

Build U-Net model.

Parameters

- **`x`** – input placeholder
- **`outCh`** – number of output channels

Returns keras model

CHAPTER

FOUR

UTILITY

4.1 confusion matrix

calculate confusion matrix. Confusion matrix contains

- TP: True positive
- TN: True negative
- FP: False positive
- FN: False Negative
- TPR: True positive ratio or sensitivity
- TNR: True Negative ratio or specificity
- Dice Index

$$Dice = \frac{2 \times \sum_{i\in[[N]],j\in[[M]],k\in[[L]]}(label_{i,j,k} \; \&\& \; predict_{i,j,k})}{\sum_{i\in[[N]],j\in[[M]],k\in[[L]]} label_{i,j,k} \; + \sum_{i\in[[N]],j\in[[M]],k\in[[L]]} predict_{i,j,k}}$$

Notes

Arguments are bash arguments.

param exp_def experiment definition

param models_path experiment definition

param epoch model saved at this epoch

param useMask use guide wire and nonIEL masks

returns add a line to the *../model/confusion_matrix.csv* file, which contains confusion matrix of testing and vali

4.2 load batch

Load a batch of data.

Creates batches of data randomly in serial or multi-thread parallel fashion.

class `util.load_batch.`**`LoadBatchGen`**(*im*, *datasetID*, *nBatch*, *label=None*, *isAug=False*, *coord_sys='carts'*)

data generator class, a sub-class of Keras' Sequence class `util.load_batch.`**`img_aug`**(*im*, *l*, *coord_sys*, *prob_lim=0.5*)

Image augmentation manager.

Based on the coordinate system (*Polar* vs. *Cartesian*), it selects the corresponding method.

Parameters

- **im** – input image 4D or 5D tensor
- **l** – input label 4D or 5D tensor
- **coord_sys** – coordinate system. 'polar' or 'carts' for Polar and Cartesian, respectively.
- **prob_lim** – probability limit for applying each augmentation case.

Returns augmented im and l

See also:

- *img_aug_carts()*
- *img_aug_polar()*

`util.load_batch.`**`img_aug_carts`**(*image*, *L*, *prob_lim=0.5*)

Data augmentation in Cartesian coordinate system.

Applies different image augmentation procedures:

- mirroring the image along 45 degree (y=x line)
- mirroring the image along the x axis
- mirroring the image along the y axis
- mirroring the image along the z axis for 3D images
- multiple 90 degree rotations
- image intensity scaling by multplying the intensity values with close to one scale value
- image scaling. See *img_rand_scale()*

based on the input probability limit probabilistically applies different augmentation cases.

Parameters

- **image** – input image 4D or 5D tensor
- **L** – input label 4D or 5D tensor
- **prob_lim** – probability limit for applying each augmentation case.

Returns augmented image and L

See also:

- *img_aug()*

`util.load_batch.`**`img_aug_polar`**(*image*, *label*, *prob_lim=0.5*)

Data augmentation in Polar coordinate.

Applies different image augmentation procedures:

- random rotations
- image intensity scaling by multplying the intensity valuse with close to one scale value

- image scaling, which randomly crops or add pads and scale the image to the original size

based on the input probability limit probabilistically applies different augmentation cases.

Args; image: input image 4D or 5D tensor label: input label 4D or 5D tensor prob_lim: probability limit for applying each augmentation case.

Returns augmented image and l

See also:

- *img_aug()*

`util.load_batch.`**`img_rand_scale`**(*im*, *scale*, *order*)

Scale one image or label batch in Cartesian coordinate system.

scale the image based on the input scale value and interpolation order followed by cropping or padding to maintain the original image shape. For interpolation close to the boundaries, the reflection mode is used.

Parameters

- **`im`** – 3D or 4D image or label tensor
- **`scale`** – scalar scale values for x and y direction
- **`order`** – interpolation order

Returns same size image with the scale image in the center of it.

`util.load_batch.`**`load_batch`**(*im*, *datasetID*, *nBatch*, *label=None*, *isAug=False*, *coord_sys='carts'*)

load a batch of data from im and/or label based on dataset (e.g. test).

This function handel different coordinate system and image augmentation.

Parameters

- **`im`** – 4D or 5D image tensor
- **`datasetID`** – index of images in im and/or label along the first axis, which belong to this dataset (e.g. test)
- **`nBatch`** – batch size
- **`label`** – 4D or 5D label tensor
- **`isAug`** – whether to apply data augmentation. See *img_aug()*
- **`coord_sys`** – coordinate system {'polar' or 'carts}

Returns a batch of data as tuple of (image, label)

See also:

- *load_batch_parallel()*

`util.load_batch.`**`load_batch_parallel`**(*im*, *datasetID*, *nBatch*, *label=None*, *isAug=False*, *coord_sys='carts'*)

load a batch of data from im and/or label based on dataset (e.g. test) using multi-thread.

This function handel different coordinate system and image augmentation.

Parameters

- **`im`** – 4D or 5D image tensor

- **datasetID** – index of images in im and/or label along the first axis, which belong to this dataset (e.g. test)
- **nBatch** – batch size
- **label** – 4D or 5D label tensor
- **isAug** – whether to apply data augmentation. See *img_aug()*
- **coord_sys** – coordinate system {'polar' or 'carts}

Returns a batch of data as tuple of (image, label)

See also:

- *load_batch()*

4.3 load data

Convert an 2D or 3D image from polar or cylindrical coordinate to the cartesian coordinate.

util.load_data.**im_fix_width**(*im*, *w*)
pad or crop the 3D image to have width and length equal to the input width in Cartesian coordinate system.

Parameters

- **im** – input image
- **w** – output width size

Returns image with the *w* width and length

util.load_data.**load_train_data**(*folder_path*, *im_shape*, *coord_sys*)
loading the training data.

Parameters

- **folder_path** – the input folder path containing the data
- **im_shape** – shape of the images in the dataset in (depth,width,length,channel) format
- **coord_sys** – coordinate system (*polar* or *carts*)

Returns

- **im**: image tensor of dataset with first row is sample ID
- **label**: label tensor similar to *im*
- **train_data_id**: row IDs of training samples
- **test_data_id**: row IDs of testing samples
- **valid_data_id**: row IDs of validation samples
- **sample_caseID**: caseID of each row

See also:

make_dataset()

util.load_data.**make_dataset**(*folder_path*, *im_shape*, *coord_sys*, *carts_w=512*)
Produce dataset based oon the results of *util.process_oct_folder()*

Parameters

- **folder_path** – the path to the folder that contains the images
- **im_shape** – shape of the images in the dataset in (depth,width,length,channel) format
- **coord_sys** – coordinate system (*polar* or *carts*)
- **carts_w** – width of the image in case *coord_sys* == *carts*

Returns image and label as the 4D or 5D tensors. sample_caseID contains the case ID for each row of image and label

See also:

- *util.process_oct_folder()*

4.4 plot log file

plot the log file within the save model folder. 111

plots the train and validation loss values over last 100 recorded performance evaluations and update the plot every 5 second. The figure has two subplots: top one has all the results and bottom one has last 100 log records.

Notes

Arguments are bash arguments.

param exp_def the experiment definition used for saving the model.

param models_path the path that model folder for *exp_def*

returns PyPlot figure with two subplots.

See also:

- `train()`

4.5 polar to cartesian

Convert an 2D or 3D image from polar or cylindrical coordinate to the cartesian coordinate.

util.polar2cartesian.**polar2cartesian**(*im*, *r0=0*, *full=True*, *deg=1*, *scale=1*)

Convert the input image from polar to *Cartesian* coordinate system.

A rectangle image in the Polar coordinate system is a circle in the Cartesian coordinate system. The output rectangle frame can inscribe the circle or be inscribed by the circle. The Radius dimension is along the rows and the zero radius can be the first row or can be a any other row, which crop the rows above that zero-radius row. The final image can be scale by a factor and the interpolation can be done in zero or one order.

Parameters

- **im** – input polar 2D or 3D image. The 3D is in cylindrical coordinate system.
- **r0** – the row index of zero radius. The rows with lower index will be removed. Default is 0.
- **full** – True if the output boundary inscribes the resulted circular boundary or be inscribed by the circular boundary. Default is True.

- **deg** – degree of interpolation. 0 for nearest neighbor and 1 for linear interpolation. Default is 1.
- **scale** – the scaling ratio of the final result. Default is 1.

Returns The converted version of im in Cartesian coordinate system. The final size depends on all the arguments.

`util.polar2cartesian.`**`polar2cartesian_large_3d_file`**`(`*im*, *r0=0*, *full=True*, *deg=1*, *scale=1*`)`

polar to Cartesian conversion for big files.

Similar to `polar2cartesian()` but convert chunk by chunk to handle very large images.

Parameters

- **im** – input polar 2D or 3D image. The 3D is in cylindrical coordinate system.
- **r0** – the row index of zero radius. The rows with lower index will be removed. Default is 0.
- **full** – True if the output boundary inscribes the resulted circular boundary or be inscribed by the circular boundary. Default is True.
- **deg** – degree of interpolation. 0 for nearest neighbor and 1 for linear interpolation. Default is 1.
- **scale** – the scaling ratio of the final result. Default is 1.

Returns The converted version of im in Cartesian coordinate system. The final size depends on all the arguments.

4.6 process oct folder

process OCT folder to generate the segmentation labels of cases. Each case has all these three files

- * .PSTIF
- * .INI
- * ROI.txt

`util.process_oct_folder.`**`process_oct_folder`**`(`*folder_path*, *scale=0.25*`)`

process OCT folder to generate the segmentation labels of cases.

The *Cartesian* output file can be scale. Each case should have all these three files

- * .PSTIF
- * .INI
- * ROI.txt

Parameters

- **folder_path** –
- **scale** –

Returns

It saves three files in the *folder_path* for each case with a suffix:

- **-im.tif**: the image in cartesian, possibly scaled.

- **-SegP.tif**: The segmentation results in polar coordinate system.
- **-SegP.tif**: The segmentation results in *Cartesian* coordinate system.

See also:

- *util.polar2cartesian.polar2cartesian_large_3d_file()*
- *util.read_oct_roi_file.read_oct_roi_file()*

4.7 read oct roi file

Read ROI file generated based on the and generate segmentation results.

`util.read_oct_roi_file.`**`lumen_iel_mask`**(*obj_list*, *im_shape*)

generate lumen or IEL mask based on the point list.

Based on the periodic nature of polar coordinate system, the boundary within [0, 2 pi] is copied to [- 2 pi, 0] and [2 pi, 4 pi]. The interpolation happened along the path for x and y independently. Number of points interpolated between each two consecutive points is based on the largest arc length between all pairs of consecutive points measured in the cartesian coordinate system.

Parameters

- **`obj_list`** – list of a single boundary in a single plane
- **`im_shape`** – the original image shape

Returns A mask image based on the *obj_list* and size of *im_obj_list*.

See also:

- *read_oct_roi_file()*

`util.read_oct_roi_file.`**`read_oct_roi_file`**(*file_path*, *im_shape*)

generate a label tensor based on a **ROI.txt* file. The label tensor is a 8-bit integer, which each bit encode one the classes:

- bit 1 (2**0) encode *gw* (Guide Wire, where guide wire has shadow)
- bit 2 (2**1) encode *noniel* (NonIEL, where IEL is not visible)
- bit 2 (2**2) encode *lumen* area
- bit 3 (2**3) encode *iel* area (the correct term is *intima* layer), which is the between *lumen* and *iel* boundaries.
- other bits are not utilized and can be used for other classes in future.

Parameters

- **`file_path`** – file path to **ROI.txt* file for a case
- **`im_shape`** – the output image shape.

Returns the output label image

Return type uint8

See also:

- *lumen_iel_mask()*
- *roi_file_parser()*

`util.read_oct_roi_file.roi_file_parser(file_path)`

Parse roi file and output the lists of objects.

The object list is a dictionary that has a list for each keys. Each class of object has a key. Keys are:

- **lumen**: vessel lumen boundary
- **iel**: IEL boundary
- **gw**: guide wire arc defined by three points.
- **noniel**: None IEL arc, which is the places that IEL is not visible, defined by three points

Each key's value is a nested list, which first index represents a complete boundary or an arc whitin a plane. The second index represents a point within the boundary or arc. The third index represnets the x, y, z coordinates of the point.

Notes

In **ROI.txt* files:

- Each file has a header line as *ROIformat*.
- The first record of a boundary or an arc section has a final field that contain the classification label.
- Each boundary record section finishes with the keyword *closed*.
- Boundary records start with keyword *Snake*.
- Arc records start with keyword *Angle*.
- **Lumen class can have one of the following classification labels:**
  1. *lumen*
  2. *fibro-fatty*
  3. *fibrous*
  4. *fc*
  5. *fibrous*
  6. *fa*
  7. *normal*
- There are some classifications that are ignore in this function such as *calcification*

Parameters **`file_path`** – the path to **ROI.txt* file

Returns The object_list dictionary.

See also:

- *read_oct_roi_file()*

PYTHON MODULE INDEX

u

INDEX