Datasets#

The Dataset to automatically download MedNIST data and generate items for training, validation or test. It’s based on CacheDataset to accelerate the training process. root_dir (Union[str, PathLike]) – target directory to download and load MedNIST dataset. section (str) – expected data section, can be: training, validation or test. transform (Union[Sequence[Callable], Callable]) – transforms to execute operations on input data. download (bool) – whether to download and extract the MedNIST from resource link, default is False. if expected file already exists, skip downloading even set it to True. user can manually copy MedNIST.tar.gz file or MedNIST folder to root directory. seed (int) – random seed to randomly split training, validation and test datasets, default is 0. val_frac (float) – percentage of validation fraction in the whole dataset, default is 0.1. test_frac (float) – percentage of test fraction in the whole dataset, default is 0.1. cache_num (int) – number of items to be cached. Default is sys.maxsize. will take the minimum of (cache_num, data_length x cache_rate, data_length). cache_rate (float) – percentage of cached data in total, default is 1.0 (cache all). will take the minimum of (cache_num, data_length x cache_rate, data_length). num_workers (Optional[int]) – the number of worker threads if computing cache in the initialization. If num_workers is None then the number returned by os.cpu_count() is used. If a value less than 1 is specified, 1 will be used instead. progress (bool) – whether to display a progress bar when downloading dataset and computing the transform cache content. copy_cache (bool) – whether to deepcopy the cache content before applying the random transforms, default to True. if the random transforms don’t modify the cached content (for example, randomly crop from the cached image and deepcopy the crop region) or if every cache item is only used once in a multi-processing environment, may set copy=False for better performance. as_contiguous (bool) – whether to convert the cached NumPy array or PyTorch tensor to be contiguous. it may help improve the performance of following logic. runtime_cache – whether to compute cache at the runtime, default to False to prepare the cache content at initialization. See: monai.data.CacheDataset. ValueError – When root_dir is not a directory. RuntimeError – When dataset_dir doesn’t exist and downloading is not selected (download=False). Get number of classes. int Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None The Dataset to automatically download the data of Medical Segmentation Decathlon challenge (http://medicaldecathlon.com/) and generate items for training, validation or test. It will also load these properties from the JSON config file of dataset. user can call get_properties() to get specified properties or all the properties loaded. It’s based on monai.data.CacheDataset to accelerate the training process. root_dir (Union[str, PathLike]) – user’s local directory for caching and loading the MSD datasets. task (str) – which task to download and execute: one of list (“Task01_BrainTumour”, “Task02_Heart”, “Task03_Liver”, “Task04_Hippocampus”, “Task05_Prostate”, “Task06_Lung”, “Task07_Pancreas”, “Task08_HepaticVessel”, “Task09_Spleen”, “Task10_Colon”). section (str) – expected data section, can be: training, validation or test. transform (Union[Sequence[Callable], Callable]) – transforms to execute operations on input data. for further usage, use EnsureChannelFirstd to convert the shape to [C, H, W, D]. download (bool) – whether to download and extract the Decathlon from resource link, default is False. if expected file already exists, skip downloading even set it to True. user can manually copy tar file or dataset folder to the root directory. val_frac (float) – percentage of validation fraction in the whole dataset, default is 0.2. seed (int) – random seed to randomly

shuffle the datalist before splitting into training and validation, default is 0. note to set same seed for training and validation sections. cache_num (int) – number of items to be cached. Default is sys.maxsize. will take the minimum of (cache_num, data_length x cache_rate, data_length). cache_rate (float) – percentage of cached data in total, default is 1.0 (cache all). will take the minimum of (cache_num, data_length x cache_rate, data_length). num_workers (int) – the number of worker threads if computing cache in the initialization. If num_workers is None then the number returned by os.cpu_count() is used. If a value less than 1 is specified, 1 will be used instead. progress (bool) – whether to display a progress bar when downloading dataset and computing the transform cache content. copy_cache (bool) – whether to deepcopy the cache content before applying the random transforms, default to True. if the random transforms don't modify the cached content (for example, randomly crop from the cached image and deepcopy the crop region) or if every cache item is only used once in a multi-processing environment, may set copy=False for better performance. as_contiguous (bool) – whether to convert the cached NumPy array or PyTorch tensor to be contiguous. it may help improve the performance of following logic. runtime_cache – whether to compute cache at the runtime, default to False to prepare the cache content at initialization. See: monai.data.CacheDataset. ValueError – When root_dir is not a directory. ValueError – When task is not one of ["Task01_BrainTumour", "Task02_Heart", "Task03_Liver", "Task04_Hippocampus", "Task05_Prostate", "Task06_Lung", "Task07_Pancreas", "Task08_HepaticVessel", "Task09_Spleen", "Task10_Colon"]. RuntimeError – When dataset_dir doesn't exist and downloading is not selected (download=False). Example: Get the indices of datalist used in this dataset. ndarray Get the loaded properties of dataset with specified keys. If no keys specified, return all the loaded properties. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None The Dataset to automatically download the data from a public The Cancer Imaging Archive (TCIA) dataset and generate items for training, validation or test. The Highdicom library is used to load dicom data with modality "SEG", but only a part of collections are supported, such as: "C4KC-KiTS", "NSCLC-Radiomics", "NSCLC-Radiomics-Interobserver1", " QIN-PROSTATE-Repeatability" and "PROSTATEx". Therefore, if "seg" is included in keys of the LoadImaged transform and loading some other collections, errors may be raised. For supported collections, the original "SEG" information may not always be consistent for each dicom file. Therefore, to avoid creating different format of labels, please use the label_dict argument of PydicomReader when calling the LoadImaged transform. The prepared label dicts of collections that are mentioned above is also saved in: monai.apps.tcia.TCIA_LABEL_DICT. You can also refer to the second example bellow. This class is based on monai.data.CacheDataset to accelerate the training process. root_dir (Union[str, PathLike]) – user's local directory for caching and loading the TCIA dataset. collection (str) – name of a TCIA collection. a TCIA dataset is defined as a collection. Please check the following list to browse the collection list (only public collections can be downloaded): https://www.cancerimagingarchive.net/collections/ section (str) – expected data section, can be: training, validation or test. transform (Union[Sequence[Callable], Callable]) – transforms to execute operations on input data. for further usage, use EnsureChannelFirstd to convert the shape to [C, H, W, D]. If not specified, LoadImaged(reader="PydicomReader", keys=["image"]) will be used as the default transform. In addition, we suggest to set the argument labels for PydicomReader if segmentations are needed to be loaded. The original labels for each dicom series

may be different, using this argument is able to unify the format of labels. download (bool) – whether to download and extract the dataset, default is False. if expected file already exists, skip downloading even set it to True. user can manually copy tar file or dataset folder to the root directory. download_len (int) – number of series that will be downloaded, the value should be larger than 0 or -1, where -1 means all series will be downloaded. Default is -1. seg_type (str) – modality type of segmentation that is used to do the first step download. Default is “SEG”. modality_tag (Tuple) – tag of modality. Default is (0x0008, 0x0060). ref_series_uid_tag (Tuple) – tag of referenced Series Instance UID. Default is (0x0020, 0x000e). ref_sop_uid_tag (Tuple) – tag of referenced SOP Instance UID. Default is (0x0008, 0x1155). specific_tags (Tuple) – tags that will be loaded for “SEG” series. This argument will be used in monai.data.PydicomReader. Default is [(0x0008, 0x1115), (0x0008,0x1140), (0x3006, 0x0010), (0x0020,0x000D), (0x0010,0x0010), (0x0010,0x0020), (0x0020,0x0011), (0x0020,0x0012)]. val_frac (float) – percentage of validation fraction in the whole dataset, default is 0.2. seed (int) – random seed to randomly shuffle the datalist before splitting into training and validation, default is 0. note to set same seed for training and validation sections. cache_num (int) – number of items to be cached. Default is sys.maxsize. will take the minimum of (cache_num, data_length x cache_rate, data_length). cache_rate (float) – percentage of cached data in total, default is 0.0 (no cache). will take the minimum of (cache_num, data_length x cache_rate, data_length). num_workers (int) – the number of worker threads if computing cache in the initialization. If num_workers is None then the number returned by os.cpu_count() is used. If a value less than 1 is specified, 1 will be used instead. progress (bool) – whether to display a progress bar when downloading dataset and computing the transform cache content. copy_cache (bool) – whether to deepcopy the cache content before applying the random transforms, default to True. if the random transforms don’t modify the cached content (for example, randomly crop from the cached image and deepcopy the crop region) or if every cache item is only used once in a multi-processing environment, may set copy=False for better performance. as_contiguous (bool) – whether to convert the cached NumPy array or PyTorch tensor to be contiguous. it may help improve the performance of following logic. runtime_cache – whether to compute cache at the runtime, default to False to prepare the cache content at initialization. See: monai.data.CacheDataset. Example: Get the indices of datalist used in this dataset. ndarray Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Cross validation dataset based on the general dataset which must have _split_datalist API. dataset_cls – dataset class to be used to create the cross validation partitions. It must have _split_datalist API. nfolds (int) – number of folds to split the data for cross validation. seed (int) – random seed to randomly shuffle the datalist before splitting into N folds, default is 0. dataset_params – other additional parameters for the dataset_cls base class. Example of 5 folds cross validation training: Generate dataset based on the specified fold indices in the cross validation group. folds (Union[Sequence[int], int]) – index of folds for training or validation, if a list of values, concatenate the data. dataset_params – other additional parameters for the dataset_cls base class, will override the same parameters in self.dataset_params.

Clara MMARs#

Download and extract Medical Model Archive (MMAR) from Nvidia Clara Train. See also https://docs.nvidia.com/clara/ Nvidia NGC Registry CLI https://docs.nvidia.com/clara/clara-train-sdk/pt/mmar.html item – the corresponding model item from MODEL_DESC. Or when api is True, the substring to query NGC's model name field. mmar_dir (Union[str, PathLike, None]) – target directory to store the MMAR, default is mmars subfolder under torch.hub get_dir(). progress (bool) – whether to display a progress bar. api (bool) – whether to query NGC and download via api version (int) – which version of MMAR to download. -1 means the latest from ngc. The local directory of the downloaded model. If api is True, a list of local directories of downloaded models. Download and extract Medical Model Archive (MMAR) model weights from Nvidia Clara Train. item – the corresponding model item from MODEL_DESC. mmar_dir (Union[str, PathLike, None]) – : target directory to store the MMAR, default is mmars subfolder under torch.hub get_dir(). progress (bool) – whether to display a progress bar when downloading the content. version (int) – version number of the MMAR. Set it to -1 to use item[Keys.VERSION]. map_location – pytorch API parameter for torch.load or torch.jit.load. pretrained – whether to load the pretrained weights after initializing a network module. weights_only – whether to load only the weights instead of initializing the network module and assign weights. model_key (str) – a key to search in the model file or config file for the model dictionary. Currently this function assumes that the model dictionary has {"[name|path]": "test.module", "args": {'kw': 'test'}}. api (bool) – whether to query NGC API to get model infomation. model_file – the relative path to the model file within an MMAR. See also https://docs.nvidia.com/clara/ Built-in immutable sequence. If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items. If the argument is a tuple, the return value is the same object.

Utilities#

Verify hash signature of specified file. filepath (Union[str, PathLike]) – path of source file to verify hash value. val (Optional[str]) – expected hash value of the file. hash_type (str) – type of hash algorithm to use, default is "md5". The supported hash types are "md5", "sha1", "sha256", "sha512". See also: monai.apps.utils.SUPPORTED_HASH_TYPES. bool Download file from specified URL link, support process bar and hash check. url (str) – source URL link to download file. filepath (Union[str, PathLike]) – target filepath to save the downloaded file (including the filename). If undefined, os.path.basename(url) will be used. hash_val (Optional[str]) – expected hash value to validate the downloaded file. if None, skip hash validation. hash_type (str) – 'md5' or 'sha1', defaults to 'md5'. progress (bool) – whether to display a progress bar. gdown_kwargs – other args for gdown except for the url, output and quiet. these args will only be used if download from google drive. details of the args of it: wkentaro/gdown RuntimeError – When the hash validation of the filepath existing file fails. RuntimeError – When a network issue or denied permission prevents the file download from url to filepath. URLError – See urllib.request.urlretrieve. HTTPError – See urllib.request.urlretrieve. ContentTooShortError – See urllib.request.urlretrieve. IOError – See urllib.request.urlretrieve. RuntimeError – When the hash validation of the url downloaded file fails. None Extract file to the output directory. Expected file types are: zip, tar.gz and tar. filepath (Union[str, PathLike]) – the file path of compressed file. output_dir (Union[str, PathLike]) – target directory to save extracted files. hash_val (Optional[str]) – expected hash value to validate the compressed file. if None, skip hash validation. hash_type (str) – 'md5' or 'sha1', defaults to 'md5'. file_type (str) –

string of file type for decompressing. Leave it empty to infer the type from the filepath basename. has_base (bool) – whether the extracted files have a base folder. This flag is used when checking if the existing folder is a result of extractall, if it is, the extraction is skipped. For example, if A.zip is unzipped to folder structure A/*.png, this flag should be True; if B.zip is unzipped to *.png, this flag should be False. RuntimeError – When the hash validation of the filepath compressed file fails. NotImplementedError – When the filepath file extension is not one of [zip", "tar.gz", "tar"]. None Download file from URL and extract it to the output directory. url (str) – source URL link to download file. filepath (Union[str, PathLike]) – the file path of the downloaded compressed file. use this option to keep the directly downloaded compressed file, to avoid further repeated downloads. output_dir (Union[str, PathLike]) – target directory to save extracted files. default is the current directory. hash_val (Optional[str]) – expected hash value to validate the downloaded file. if None, skip hash validation. hash_type (str) – 'md5' or 'sha1', defaults to 'md5'. file_type (str) – string of file type for decompressing. Leave it empty to infer the type from url's base file name. has_base (bool) – whether the extracted files have a base folder. This flag is used when checking if the existing folder is a result of extractall, if it is, the extraction is skipped. For example, if A.zip is unzipped to folder structure A/*.png, this flag should be True; if B.zip is unzipped to *.png, this flag should be False. progress (bool) – whether to display progress bar. None

Deepgrow#

Utility to pre-process and create dataset list for Deepgrow training over on existing one. The input data list is normally a list of images and labels (3D volume) that needs pre-processing for Deepgrow training pipeline. datalist – A list of data dictionary. Each entry should at least contain 'image_key': . For example, typical input data can be a list of dictionaries: [{'image': , 'label': }] A list of data dictionary. Each entry should at least contain 'image_key': . For example, typical input data can be a list of dictionaries: output_dir (str) – target directory to store the training data for Deepgrow Training pixdim – output voxel spacing. dimension (int) – dimension for Deepgrow training. It can be 2 or 3. image_key (str) – image key in input datalist. Defaults to 'image'. label_key (str) – label key in input datalist. Defaults to 'label'. base_dir – base directory in case related path is used for the keys in datalist. Defaults to None. limit (int) – limit number of inputs for pre-processing. Defaults to 0 (no limit). relative_path (bool) – output keys values should be based on relative path. Defaults to False. transforms – explicit transforms to execute operations on input data. ValueError – When dimension is not one of [2, 3] ValueError – When datalist is Empty List[Dict] A new datalist that contains path to the images/labels after pre-processing. Example: Ignite process_function used to introduce interactions (simulation of clicks) for Deepgrow Training/Evaluation. For more details please refer to: https://pytorch.org/ignite/generated/ignite.engine.engine.Engine.html. This implementation is based on: Sakinis et al., Interactive segmentation of medical images through fully convolutional neural networks. (2019) https://arxiv.org/abs/1903.08205 transforms (Union[Sequence[Callable], Callable]) – execute additional transformation during every iteration (before train). Typically, several Tensor based transforms composed by Compose. max_interactions (int) – maximum number of interactions per iteration train (bool) – training or evaluation key_probability (str) – field name to fill probability for every interaction Add random guidance as initial seed point for a given label. Note that the label is of size (C, D, H, W) or (C, H, W) The guidance is of size (2, N, # of dims) where N is number of guidance added. # of dims = 4 when C, D, H, W; # of dims = 3 when (C, H, W) label

(str) – label source. guidance (str) – key to store guidance. sids (str) – key that represents list of valid slice indices for the given label. sid (str) – key that represents the slice to add initial seed point. If not present, random sid will be chosen. connected_regions (int) – maximum connected regions to use for adding initial points. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. Add Guidance signal for input image. Based on the “guidance” points, apply gaussian to them and add them as new channel for input image. image (str) – key to the image source. guidance (str) – key to store guidance. sigma (int) – standard deviation for Gaussian kernel. number_intensity_ch (int) – channel index. Add random guidance based on discrepancies that were found between label and prediction. input shape is as below: Guidance is of shape (2, N, # of dim) Discrepancy is of shape (2, C, D, H, W) or (2, C, H, W) Probability is of shape (1) guidance (str) – key to guidance source. discrepancy (str) – key that represents discrepancies found between label and prediction. probability (str) – key that represents click/interaction probability. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. Add guidance based on user clicks. We assume the input is loaded by LoadImaged and has the shape of (H, W, D) originally. Clicks always specify the coordinates in (H, W, D) If depth_first is True: Input is now of shape (D, H, W), will return guidance that specifies the coordinates in (D, H, W) else: Input is now of shape (H, W, D), will return guidance that specifies the coordinates in (H, W, D) ref_image – key to reference image to fetch current and original image details. guidance (str) – output key to store guidance. foreground (str) – key that represents user foreground (+ve) clicks. background (str) – key that represents user background (-ve) clicks. axis (int) – axis that represents slices in 3D volume. (axis to Depth) depth_first (bool) – if depth (slices) is positioned at first dimension. spatial_dims (int) – dimensions based on model used for deepgrow (2D vs 3D). slice_key (str) – key that represents applicable slice to add guidance. meta_keys (Optional[str]) – explicitly indicate the key of the metadata dictionary of ref_image. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. if None, will try to construct meta_keys by {ref_image}_{meta_key_postfix}. meta_key_postfix (str) – if meta_key is None, use {ref_image}_{meta_key_postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary’s affine field. Deprecated since version 0.6.0: dimensions is deprecated, use spatial_dims instead. Crop only the foreground object of the expected images. Difference VS monai.transforms.CropForegroundd: If the bounding box is smaller than spatial size in all dimensions then this transform will crop the object using box’s center and spatial_size. This transform will set “start_coord_key”, “end_coord_key”, “original_shape_key” and “cropped_shape_key” in data[{key}_{meta_key_postfix}] The typical usage is to help training and evaluation if the valid part is small in the whole medical image. The valid part can be determined by any field in the data with source_key, for example: Select values > 0 in image field as the foreground and crop on all fields specified by keys. Select label = 3 in label field as the foreground to crop on all fields specified by keys. Select label > 0 in the third channel of a One-Hot label field as the foreground to crop all keys fields. Users can define arbitrary function to select expected foreground from the whole source image

or specified channels. And it can also add margin to every dim of the bounding box of foreground object. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform source_key (str) – data source to generate the bounding box of foreground, can be image or label, etc. spatial_size (Union[Sequence[int], ndarray]) – minimal spatial size of the image patch e.g. [128, 128, 128] to fit in. select_fn (Callable) – function to select expected foreground, default is to select values > 0. channel_indices (Union[Iterable[int], int, None]) – if defined, select foreground only on the specified channels of image. if None, select foreground on the whole image. margin (int) – add margin value to spatial dims of the bounding box, if only 1 value provided, use it for all dims. allow_smaller (bool) – when computing box size with margin, whether allow the image size to be smaller than box size, default to True. if the margined size is bigger than image size, will pad with specified mode. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix – if meta_keys is None, use {key}_{meta_key_postfix} to fetch/store the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary's affine field. start_coord_key (str) – key to record the start coordinate of spatial bounding box for foreground. end_coord_key (str) – key to record the end coordinate of spatial bounding box for foreground. original_shape_key (str) – key to record original shape for foreground. cropped_shape_key (str) – key to record cropped shape for foreground. allow_missing_keys (bool) – don't raise exception if key is missing. Crop image based on guidance with minimal spatial size. If the bounding box is smaller than spatial size in all dimensions then this transform will crop the object using box's center and spatial_size. This transform will set "start_coord_key", "end_coord_key", "original_shape_key" and "cropped_shape_key" in data[{key}_{meta_key_postfix}] Input data is of shape (C, spatial_1, [spatial_2, …]) keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. guidance (str) – key to the guidance. It is used to generate the bounding box of foreground spatial_size – minimal spatial size of the image patch e.g. [128, 128, 128] to fit in. margin – add margin value to spatial dims of the bounding box, if only 1 value provided, use it for all dims. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix – if meta_keys is None, use key_{postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary's affine field. start_coord_key (str) – key to record the start coordinate of spatial bounding box for foreground. end_coord_key (str) – key to record the end coordinate of spatial bounding box for foreground. original_shape_key (str) – key to record original shape for foreground. cropped_shape_key (str) – key to record cropped shape for foreground. allow_missing_keys (bool) – don't raise exception if key is missing. Restores label based on the ref image. The ref_image is assumed that it went through the following transforms: Fetch2DSliced (If 2D) Spacingd SpatialCropGuidanced Resized And its shape is assumed to be (C, D, H, W) This transform tries to undo these operation so that the result label can be

overlapped with original volume. It does the following operation: Undo Resized Undo SpatialCropGuidanced Undo Spacingd Undo Fetch2DSliced The resulting label is of shape (D, H, W) keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. ref_image (str) – reference image to fetch current and original image details slice_only (bool) – apply only to an applicable slice, in case of 2D model/prediction mode (Union[Sequence[Union[InterpolateMode, str]], InterpolateMode, str]) – {"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} One of the listed string values or a user supplied function for padding. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html align_corners (Union[Sequence[Optional[bool]], bool, None]) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of bool, each element corresponds to a key in keys. meta_keys (Optional[str]) – explicitly indicate the key of the corresponding metadata dictionary. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_key is None, use key_{meta_key_postfix} to fetch the metadata according to the key data, default is `meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary's affine field. start_coord_key (str) – key that records the start coordinate of spatial bounding box for foreground. end_coord_key (str) – key that records the end coordinate of spatial bounding box for foreground. original_shape_key (str) – key that records original shape for foreground. cropped_shape_key (str) – key that records cropped shape for foreground. allow_missing_keys (bool) – don't raise exception if key is missing. Resize the guidance based on cropped vs resized image. This transform assumes that the images have been cropped and resized. And the shape after cropped is store inside the meta dict of ref image. guidance (str) – key to guidance ref_image (str) – key to reference image to fetch current and original image details meta_keys (Optional[str]) – explicitly indicate the key of the metadata dictionary of ref_image. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. if None, will try to construct meta_keys by {ref_image}_{meta_key_postfix}. meta_key_postfix (str) – if meta_key is None, use {ref_image}_{meta_key_postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary's affine field. cropped_shape_key (str) – key that records cropped shape for foreground. Find discrepancy between prediction and actual during click interactions during training. label (str) – key to label source. pred (str) – key to prediction source. discrepancy (str) – key to store discrepancies found between label and prediction. Find/List all valid slices in the label. Label is assumed to be a 4D Volume with shape CDHW, where C=1. label (str) – key to the label source. sids (str) – key to store slices indices having valid label map. Fetch one slice in case of a 3D volume. The volume only contains spatial coordinates. keys – keys of the corresponding items to be transformed. guidance – key that represents guidance. axis (int) – axis that represents slice in 3D volume. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}.

meta_key_postfix (str) – use key_{meta_key_postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary's affine field. allow_missing_keys (bool) – don't raise exception if key is missing.

Pathology#

This dataset reads whole slide images, extracts regions, and creates patches. It also reads labels for each patch and provides each patch with its associated class labels. data (List) – the list of input samples including image, location, and label (see the note below for more details). region_size (Union[int, Tuple[int, int]]) – the size of regions to be extracted from the whole slide image. grid_shape (Union[int, Tuple[int, int]]) – the grid shape on which the patches should be extracted. patch_size (Union[int, Tuple[int, int]]) – the size of patches extracted from the region on the grid. transform (Optional[Callable]) – transforms to be executed on input data. image_reader_name (str) – the name of library to be used for loading whole slide imaging, either CuCIM or OpenSlide. Defaults to CuCIM. kwargs – additional parameters for WSIReader Note The input data has the following form as an example: [{"image": "path/to/image1.tiff", "location": [200, 500], "label": [0,0,0,1]}]. This means from "image1.tiff" extract a region centered at the given location location with the size of region_size, and then extract patches with the size of patch_size from a grid with the shape of grid_shape. Be aware the grid_shape should construct a grid with the same number of element as labels, so for this example the grid_shape should be (2, 2). Add SmartCache functionality to PatchWSIDataset. data (List) – the list of input samples including image, location, and label (see PatchWSIDataset for more details) region_size (Union[int, Tuple[int, int]]) – the region to be extracted from the whole slide image. grid_shape (Union[int, Tuple[int, int]]) – the grid shape on which the patches should be extracted. patch_size (Union[int, Tuple[int, int]]) – the size of patches extracted from the region on the grid. image_reader_name (str) – the name of library to be used for loading whole slide imaging, either CuCIM or OpenSlide. Defaults to CuCIM. transform (Union[Sequence[Callable], Callable]) – transforms to be executed on input data. replace_rate (float) – percentage of the cached items to be replaced in every epoch. cache_num (int) – number of items to be cached. Default is sys.maxsize. will take the minimum of (cache_num, data_length x cache_rate, data_length). cache_rate (float) – percentage of cached data in total, default is 1.0 (cache all). will take the minimum of (cache_num, data_length x cache_rate, data_length). num_init_workers (Optional[int]) – the number of worker threads to initialize the cache for first epoch. If num_init_workers is None then the number returned by os.cpu_count() is used. If a value less than 1 is specified, 1 will be used instead. num_replace_workers (Optional[int]) – the number of worker threads to prepare the replacement cache for every epoch. If num_replace_workers is None then the number returned by os.cpu_count() is used. If a value less than 1 is specified, 1 will be used instead. progress (bool) – whether to display a progress bar when caching for the first epoch. copy_cache (bool) – whether to deepcopy the cache content before applying the random transforms, default to True. if the random transforms don't modify the cache content or every cache item is only used once in a multi-processing environment, may set copy=False for better performance. as_contiguous (bool) – whether to convert the cached NumPy array or PyTorch tensor to be contiguous. it may help improve the performance of following logic. kwargs – additional parameters for WSIReader This dataset load the provided foreground masks at an arbitrary resolution level, and extract patches based on that mask from the associated whole

slide image. data (List[Dict[str, str]]) – a list of sample including the path to the whole slide image and the path to the mask. Like this: [{“image”: “path/to/image1.tiff”, “mask”: “path/to/mask1.npy}, …]”. patch_size (Union[int, Tuple[int, int]]) – the size of patches to be extracted from the whole slide image for inference. transform (Optional[Callable]) – transforms to be executed on extracted patches. image_reader_name (str) – the name of library to be used for loading whole slide imaging, either CuCIM or OpenSlide. Defaults to CuCIM. kwargs – additional parameters for WSIReader Note supposed to be the same size as the foreground mask and not the original wsi image size. Event handler triggered on completing every iteration to save the probability map output_dir (str) – output directory to save probability maps. output_postfix (str) – a string appended to all output file names. dtype (Union[dtype, type, str, None]) – the data type in which the probability map is stored. Default np.float64. name (Optional[str]) – identifier of logging.logger to use, defaulting to engine.logger. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None This method save the probability map for an image, when its inference is finished, and delete that probability map from memory. name (str) – the name of image to be saved. None Sliding window method for HoVerNet model inference, with sw_batch_size windows for every model.forward(). Usage example can be found in the monai.inferers.Inferer base class. roi_size (Union[Sequence[int], int]) – the window size to execute SlidingWindow evaluation. If it has non-positive components, the corresponding inputs size will be used. if the components of the roi_size are non-positive values, the transform will use the corresponding components of img size. For example, roi_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. sw_batch_size (int) – the batch size to run window slices. overlap (float) – Amount of overlap between scans. mode (Union[BlendMode, str]) – {"constant", "gaussian"} How to blend output of overlapping windows. Defaults to "constant". "constant": gives equal weight to all predictions. "gaussian”: gives less weight to predictions on edges of windows. {"constant", "gaussian"} How to blend output of overlapping windows. Defaults to "constant". "constant": gives equal weight to all predictions. "gaussian”: gives less weight to predictions on edges of windows. sigma_scale (Union[Sequence[float], float]) – the standard deviation coefficient of the Gaussian window when mode is "gaussian". Default: 0.125. Actual window sigma is sigma_scale * dim_size. When sigma_scale is a sequence of floats, the values denote sigma_scale at the corresponding spatial dimensions. padding_mode (Union[PytorchPadMode, str]) – {"constant", "reflect", "replicate", "circular"} Padding mode when roi_size is larger than inputs. Defaults to "constant" See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html cval (float) – fill value for ‘constant’ padding mode. Default: 0 sw_device (Union[str, device, None]) – device for the window data. By default the device (and accordingly the memory) of the inputs is used. Normally sw_device should be consistent with the device where predictor is defined. device (Union[str, device, None]) – device for the stitched output prediction. By default the device (and accordingly the memory) of the inputs is used. If for example set to device=torch.device(‘cpu’) the gpu memory consumption is less and independent of the inputs and roi_size. Output is on the device. progress (bool) – whether to print a tqdm progress bar. cache_roi_weight_map (bool) – whether to pre-compute the ROI weight map. cpu_thresh (Optional[int]) – when provided, dynamically switch to stitching on cpu (to save gpu memory) when input image volume is larger than this threshold (in pixels/voxels). Otherwise use "device". Thus, the output may end-up on either cpu or gpu. extra_input_padding (Optional[Tuple[int]]) – the amount of padding for the input image, which is a tuple of even number of pads. Refer to to the pad argument of torch.nn.functional.pad for more details. Note sw_batch_size denotes the max number of windows per network inference iteration, not the batch size of inputs. Loss function for HoVerNet pipeline, which is combination

of losses across the three branches. The NP (nucleus prediction) branch uses Dice + CrossEntropy. The HV (Horizontal and Vertical) distance from centroid branch uses MSE + MSE of the gradient. The NC (Nuclear Class prediction) branch uses Dice + CrossEntropy The result is a weighted sum of these losses. lambda_hv_mse (float) – Weight factor to apply to the HV regression MSE part of the overall loss lambda_hv_mse_grad (float) – Weight factor to apply to the MSE of the HV gradient part of the overall loss lambda_np_ce (float) – Weight factor to apply to the nuclei prediction CrossEntropyLoss part of the overall loss lambda_np_dice (float) – Weight factor to apply to the nuclei prediction DiceLoss part of overall loss lambda_nc_ce (float) – Weight factor to apply to the nuclei class prediction CrossEntropyLoss part of the overall loss lambda_nc_dice (float) – Weight factor to apply to the nuclei class prediction DiceLoss part of the overall loss prediction (Dict[str, Tensor]) – dictionary of predicted outputs for three branches, each of which should have the shape of BNHW. target (Dict[str, Tensor]) – dictionary of ground truths for three branches, each of which should have the shape of BNHW. Tensor Evaluate with Free Response Operating Characteristic (FROC) score. data (List[Dict]) – either the list of dictionaries containing probability maps (inference result) and tumor mask (ground truth), as below, or the path to a json file containing such list. { “prob_map”: “path/to/prob_map_1.npy”, “tumor_mask”: “path/to/ground_truth_1.tiff”, “level”: 6, “pixel_spacing”: 0.243 } grow_distance (int) – Euclidean distance (in micrometer) by which to grow the label the ground truth’s tumors. Defaults to 75, which is the equivalent size of 5 tumor cells. itc_diameter (int) – the maximum diameter of a region (in micrometer) to be considered as an isolated tumor cell. Defaults to 200. eval_thresholds (Tuple) – the false positive rates for calculating the average sensitivity. Defaults to (0.25, 0.5, 1, 2, 4, 8) which is the same as the CAMELYON 16 Challenge. nms_sigma (float) – the standard deviation for gaussian filter of non-maximal suppression. Defaults to 0.0. nms_prob_threshold (float) – the probability threshold of non-maximal suppression. Defaults to 0.5. nms_box_size (int) – the box size (in pixel) to be removed around the pixel for non-maximal suppression. image_reader_name (str) – the name of library to be used for loading whole slide imaging, either CuCIM or OpenSlide. Defaults to CuCIM. Note For more info on nms_* parameters look at monai.utils.prob_nms.ProbNMS`. Compute false positive and true positive probabilities for tumor detection, by comparing the model outputs with the prepared ground truths for all samples Evaluate the detection performance of a model based on the model probability map output, the ground truth tumor mask, and their associated metadata (e.g., pixel_spacing, level) Prepare the ground truth for evaluation based on the binary tumor mask Prepare the probability map for detection evaluation. This method computes the segmentation mask according to the binary tumor mask. mask (ndarray) – the binary mask array threshold (float) – the threshold to fill holes This method computes identifies Isolated Tumor Cells (ITC) and return their labels. tumor_mask (ndarray) – the tumor mask. threshold (float) – the threshold (at the mask level) to define an isolated tumor cell (ITC). A region with the longest diameter less than this threshold is considered as an ITC. List[int] This class extends monai.utils.ProbNMS and add the resolution option for Pathology. Class to extract a target stain from an image, using stain deconvolution (see Note). tli (float) – transmitted light intensity. Defaults to 240. alpha (float) – tolerance in percentile for the pseudo-min (alpha percentile) and pseudo-max (100 - alpha percentile). Defaults to 1. beta (float) – absorbance threshold for transparent pixels. Defaults to 0.15 max_cref (Union[tuple, ndarray]) – reference maximum stain concentrations for Hematoxylin & Eosin (H&E;). Defaults to (1.9705, 1.0308). Note For more information refer to: - the original paper: Macenko et al., 2009 http://wwwx.cs.unc.edu/~mn/sites/default/files/macenko2009.pdf - the previous implementations: MATLAB: mitkovetta/staining-normalization Python:

schaugf/HEnorm_python Class to normalize patches/images to a reference or target image stain (see Note). Performs stain deconvolution of the source image using the ExtractHEStains class, to obtain the stain matrix and calculate the stain concentration matrix for the image. Then, performs the inverse Beer-Lambert transform to recreate the patch using the target H&E; stain matrix provided. If no target stain provided, a default reference stain is used. Similarly, if no maximum stain concentrations are provided, a reference maximum stain concentrations matrix is used. tli (float) – transmitted light intensity. Defaults to 240. alpha (float) – tolerance in percentile for the pseudo-min (alpha percentile) and pseudo-max (100 - alpha percentile). Defaults to 1. beta (float) – absorbance threshold for transparent pixels. Defaults to 0.15. target_he (Union[tuple, ndarray]) – target stain matrix. Defaults to ((0.5626, 0.2159), (0.7201, 0.8012), (0.4062, 0.5581)). max_cref (Union[tuple, ndarray]) – reference maximum stain concentrations for Hematoxylin & Eosin (H&E;). Defaults to [1.9705, 1.0308]. Note For more information refer to: - the original paper: Macenko et al., 2009 http://wwwx.cs.unc.edu/~mn/sites/default/files/macenko2009.pdf - the previous implementations: MATLAB: mitkovetta/staining-normalization Python: schaugf/HEnorm_python A collection of dictionary-based wrappers around the pathology transforms defined in monai.apps.pathology.transforms.array. Class names are ended with ‘d’ to denote dictionary-based transforms. Dictionary-based wrapper of monai.apps.pathology.transforms.ExtractHEStains. Class to extract a target stain from an image, using stain deconvolution. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform tli (float) – transmitted light intensity. Defaults to 240. alpha (float) – tolerance in percentile for the pseudo-min (alpha percentile) and pseudo-max (100 - alpha percentile). Defaults to 1. beta (float) – absorbance threshold for transparent pixels. Defaults to 0.15 max_cref (Union[tuple, ndarray]) – reference maximum stain concentrations for Hematoxylin & Eosin (H&E;). Defaults to (1.9705, 1.0308). allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.apps.pathology.transforms.NormalizeHEStains. Class to normalize patches/images to a reference or target image stain. Performs stain deconvolution of the source image using the ExtractHEStains class, to obtain the stain matrix and calculate the stain concentration matrix for the image. Then, performs the inverse Beer-Lambert transform to recreate the patch using the target H&E; stain matrix provided. If no target stain provided, a default reference stain is used. Similarly, if no maximum stain concentrations are provided, a reference maximum stain concentrations matrix is used. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform tli (float) – transmitted light intensity. Defaults to 240. alpha (float) – tolerance in percentile for the pseudo-min (alpha percentile) and pseudo-max (100 - alpha percentile). Defaults to 1. beta (float) – absorbance threshold for transparent pixels. Defaults to 0.15. target_he (Union[tuple, ndarray]) – target stain matrix. Defaults to None. max_cref (Union[tuple, ndarray]) – reference maximum stain concentrations for Hematoxylin & Eosin (H&E;). Defaults to None. allow_missing_keys (bool) – don’t raise exception if key is missing. Split the image into patches based on the provided grid shape. This transform works only with torch.Tensor inputs. grid_size (Union[int, Tuple[int, int]]) – a tuple or an integer define the shape of the grid upon which to extract patches. If it’s an integer, the value will be repeated for each dimension. Default is 2x2 patch_size (Union[int, Tuple[int, int], None]) – a tuple or an integer that defines the output patch sizes. If it’s an integer, the value will be repeated for each dimension. The default is (0, 0), where the patch size will be inferred from the grid shape. Note: the shape of the input image is inferred based on the first image used. Tile the 2D image into patches on a grid and maintain a

subset of it. This transform works only with np.ndarray inputs for 2D images. tile_count (Optional[int]) – number of tiles to extract, if None extracts all non-background tiles Defaults to None. tile_size (int) – size of the square tile Defaults to 256. step (Optional[int]) – step size Defaults to None (same as tile_size) random_offset (bool) – Randomize position of the grid, instead of starting from the top-left corner Defaults to False. pad_full (bool) – pad image to the size evenly divisible by tile_size Defaults to False. background_val (int) – the background constant (e.g. 255 for white background) Defaults to 255. filter_mode (str) – mode must be in [“min”, “max”, “random”]. If total number of tiles is more than tile_size, then sort by intensity sum, and take the smallest (for min), largest (for max) or random (for random) subset Defaults to min (which assumes background is high value) Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Split the image into patches based on the provided grid shape. This transform works only with torch.Tensor inputs. grid_size (Union[int, Tuple[int, int]]) – a tuple or an integer define the shape of the grid upon which to extract patches. If it’s an integer, the value will be repeated for each dimension. Default is 2x2 patch_size (Union[int, Tuple[int, int], None]) – a tuple or an integer that defines the output patch sizes. If it’s an integer, the value will be repeated for each dimension. The default is (0, 0), where the patch size will be inferred from the grid shape. Note: the shape of the input image is inferred based on the first image used. Tile the 2D image into patches on a grid and maintain a subset of it. This transform works only with np.ndarray inputs for 2D images. tile_count (Optional[int]) – number of tiles to extract, if None extracts all non-background tiles Defaults to None. tile_size (int) – size of the square tile Defaults to 256. step (Optional[int]) – step size Defaults to None (same as tile_size) random_offset (bool) – Randomize position of the grid, instead of starting from the top-left corner Defaults to False. pad_full (bool) – pad image to the size evenly divisible by tile_size Defaults to False. background_val (int) – the background constant (e.g. 255 for white background) Defaults to 255. filter_mode (str) – mode must be in [“min”, “max”, “random”]. If total number of tiles is more than tile_size, then sort by intensity sum, and take the smallest (for min), largest (for max) or random (for random) subset Defaults to min (which assumes background is high value) Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Converts SciPy-style contours (generated by skimage.measure.find_contours) to a more succinct version which only includes the pixels to which lines need to be drawn (i.e. not the intervening pixels along each line). height (int) – height of bounding box, used to detect direction of line segment. width (int) – width of bounding box, used to detect direction of line segment. the pixels that need to be joined by straight lines to describe the outmost pixels of the foreground similar toOpenCV’s cv.CHAIN_APPROX_SIMPLE (counterclockwise) OpenCV’s cv.CHAIN_APPROX_SIMPLE (counterclockwise) Generate contour for each instance in a 2D array. Use GenerateSuccinctContour to only include the pixels to which lines need to be drawn min_num_points (int) – assumed that the created contour does not form a contour if it does not contain more points than the specified value. Defaults to 3. contour_level (Optional[float]) – an optional value for skimage.measure.find_contours to find contours in the array. If not provided, the level is set to (max(image) + min(image)) / 2. Generate instance centroid using skimage.measure.centroid. dtype (Union[dtype, type, str, None]) – the data type of

output centroid. Generate instance type and probability for each instance. Use skimage.segmentation.watershed to get instance segmentation results from images. See: https://scikit-image.org/docs/stable/api/skimage.segmentation.html#skimage.segmentation.watershed. connectivity (Optional[int]) – an array with the same number of dimensions as image whose non-zero elements indicate neighbors for connection. Following the scipy convention, default is a one-connected array of the dimension of the image. dtype (Union[dtype, type, str, None]) – target data content type to convert, default is np.int64. generate mask used in watershed. Only points at which mask == True will be labeled. activation (Union[str, Callable]) – the activation layer to be applied on the input probability map. It can be “softmax” or “sigmoid” string, or any callable. Defaults to “softmax”. threshold (Optional[float]) – an optional float value to threshold to binarize probability map. If not provided, defaults to 0.5 when activation is not “softmax”, otherwise None. min_object_size (int) – objects smaller than this size (in pixel) are removed. Defaults to 10. dtype (Union[dtype, type, str, None]) – target data content type to convert, default is np.uint8. Generate instance border by hover map. The more parts of the image that cannot be identified as foreground areas, the larger the grey scale value. The grey value of the instance’s border will be larger. kernel_size (int) – the size of the Sobel kernel. Defaults to 5. dtype (Union[dtype, type, str, None]) – target data type to convert to. Defaults to np.float32. ValueError – when the mask shape is not [1, H, W]. ValueError – when the hover_map shape is not [2, H, W]. Generate distance map. In general, the instance map is calculated from the distance to the background. Here, we use 1 - “instance border map” to generate the distance map. Nuclei values form mountains so invert them to get basins. smooth_fn (Optional[Callable]) – smoothing function for distance map, which can be any callable object. If not provided monai.transforms.GaussianSmooth() is used. dtype (Union[dtype, type, str, None]) – target data type to convert to. Defaults to np.float32. Generate markers to be used in watershed. The watershed algorithm treats pixels values as a local topography (elevation). The algorithm floods basins from the markers until basins attributed to different markers meet on watershed lines. Generally, markers are chosen as local minima of the image, from which basins are flooded. Here is the implementation from HoVerNet paper. For more details refer to papers: https://arxiv.org/abs/1812.06499. threshold (float) – a float value to threshold to binarize instance border map. It turns uncertain area to 1 and other area to 0. Defaults to 0.4. radius (int) – the radius of the disk-shaped footprint used in opening. Defaults to 2. min_object_size (int) – objects smaller than this size (in pixel) are removed. Defaults to 10. postprocess_fn (Optional[Callable]) – additional post-process function on the markers. If not provided, monai.transforms.post.FillHoles() will be used. dtype (Union[dtype, type, str, None]) – target data type to convert to. Defaults to np.int64. The post-processing transform for HoVerNet model to generate nuclear type information. It updates the input instance info dictionary with information about types of the nuclei (value and probability). Also if requested (return_type_map=True), it generates a pixel-level type map. activation (Union[str, Callable]) – the activation layer to be applied on nuclear type branch. It can be “softmax” or “sigmoid” string, or any callable. Defaults to “softmax”. threshold (Optional[float]) – an optional float value to threshold to binarize probability map. If not provided, defaults to 0.5 when activation is not “softmax”, otherwise None. return_type_map (bool) – whether to calculate and return pixel-level type map. The post-processing transform for HoVerNet model to generate instance segmentation map. It generates an instance segmentation map as well as a dictionary containing centroids, bounding boxes, and contours for each instance. activation (Union[str, Callable]) – the activation layer to be applied on the input probability map. It can be “softmax” or “sigmoid” string, or any callable. Defaults to “softmax”. mask_threshold (Optional[float]) – a float value to threshold to binarize probability map to generate

mask. min_object_size (int) – objects smaller than this size (in pixel) are removed. Defaults to 10. sobel_kernel_size (int) – the size of the Sobel kernel used in GenerateInstanceBorder. Defaults to 5. distance_smooth_fn (Optional[Callable]) – smoothing function for distance map. If not provided, monai.transforms.intensity.GaussianSmooth() will be used. marker_threshold (float) – a float value to threshold to binarize instance border map for markers. It turns uncertain area to 1 and other area to 0. Defaults to 0.4. marker_radius (int) – the radius of the disk-shaped footprint used in opening of markers. Defaults to 2. marker_postprocess_fn (Optional[Callable]) – post-process function for watershed markers. If not provided, monai.transforms.post.FillHoles() will be used. watershed_connectivity (Optional[int]) – connectivity argument of skimage.segmentation.watershed. min_num_points (int) – minimum number of points to be considered as a contour. Defaults to 3. contour_level (Optional[float]) – an optional value for skimage.measure.find_contours to find contours in the array. If not provided, the level is set to (max(image) + min(image)) / 2. Dictionary-based wrapper of monai.apps.pathology.transforms.post.array.GenerateSuccinctContour. Converts SciPy-style contours (generated by skimage.measure.find_contours) to a more succinct version which only includes the pixels to which lines need to be drawn (i.e. not the intervening pixels along each line). keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. height (int) – height of bounding box, used to detect direction of line segment. width (int) – width of bounding box, used to detect direction of line segment. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.apps.pathology.transforms.post.array.GenerateInstanceContour. Generate contour for each instance in a 2D array. Use GenerateSuccinctContour to only include the pixels to which lines need to be drawn keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. contour_key_postfix (str) – the output contour coordinates will be written to the value of {key}_{contour_key_postfix}. offset_key (Optional[str]) – keys of offset used in GenerateInstanceContour. min_num_points (int) – assumed that the created contour does not form a contour if it does not contain more points than the specified value. Defaults to 3. level (Optional[float]) – optional. Value along which to find contours in the array. By default, the level is set to (max(image) + min(image)) / 2. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.apps.pathology.transforms.post.array.GenerateInstanceCentroid. Generate instance centroid using skimage.measure.centroid. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. centroid_key_postfix (str) – the output centroid coordinates will be written to the value of {key}_{centroid_key_postfix}. offset_key (Optional[str]) – keys of offset used in GenerateInstanceCentroid. dtype (Union[dtype, type, str, None]) – the data type of output centroid. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.apps.pathology.transforms.post.array.GenerateInstanceType. Generate instance type and probability for each instance. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. type_info_key (str) – the output instance type and probability will be written to the value of {type_info_key}. bbox_key (str) – keys of bounding box. seg_pred_key (str) – keys of segmentation prediction map. instance_id_key (str) – keys of instance id. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.apps.pathology.transforms.array.Watershed. Use skimage.segmentation.watershed to get instance segmentation results from images. See: https://scikit-image.org/docs/stable/api/skimage.segmentation.html#skimage.segmentation.watershed. keys (Union[Collection[Hashable], Hashable]) – keys of the

corresponding items to be transformed. See also: monai.transforms.MapTransform mask_key (Optional[str]) – keys of mask used in watershed. Only points at which mask == True will be labeled. markers_key (Optional[str]) – keys of markers used in watershed. If None (no markers given), the local minima of the image are used as markers. connectivity (Optional[int]) – An array with the same number of dimensions as image whose non-zero elements indicate neighbors for connection. Following the scipy convention, default is a one-connected array of the dimension of the image. dtype (Union[dtype, type, str, None]) – target data content type to convert. Defaults to np.uint8. allow_missing_keys (bool) – don't raise exception if key is missing. ValueError – when the image shape is not [1, H, W]. ValueError – when the mask shape is not [1, H, W]. Dictionary-based wrapper of monai.apps.pathology.transforms.array.GenerateWatershedMask. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. mask_key (str) – the mask will be written to the value of {mask_key}. activation (Union[str, Callable]) – the activation layer to be applied on nuclear type branch. It can be "softmax" or "sigmoid" string, or any callable. Defaults to "softmax". threshold (Optional[float]) – if not None, threshold the float values to int number 0 or 1 with specified threshold. min_object_size (int) – objects smaller than this size are removed. Defaults to 10. dtype (Union[dtype, type, str, None]) – target data content type to convert, default is np.uint8. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.apps.pathology.transforms.array.GenerateInstanceBorder. mask_key (str) – the input key where the watershed mask is stored. Defaults to "mask". hover_map_key (str) – the input key where hover map is stored. Defaults to "hover_map". border_key (str) – the output key where instance border map is written. Defaults to "border". kernel_size (int) – the size of the Sobel kernel. Defaults to 21. dtype (Union[dtype, type, str, None]) – target data content type to convert, default is np.float32. allow_missing_keys – don't raise exception if key is missing. ValueError – when the hover_map has only one value. ValueError – when the sobel gradient map has only one value. Dictionary-based wrapper of monai.apps.pathology.transforms.array.GenerateDistanceMap. mask_key (str) – the input key where the watershed mask is stored. Defaults to "mask". border_key (str) – the input key where instance border map is stored. Defaults to "border". dist_map_key (str) – the output key where distance map is written. Defaults to "dist_map". smooth_fn (Optional[Callable]) – smoothing function for distance map, which can be any callable object. If not provided monai.transforms.GaussianSmooth() is used. dtype (Union[dtype, type, str, None]) – target data content type to convert, default is np.float32. Dictionary-based wrapper of monai.apps.pathology.transforms.array.GenerateWatershedMarkers. mask_key (str) – the input key where the watershed mask is stored. Defaults to "mask". border_key (str) – the input key where instance border map is stored. Defaults to "border". markers_key (str) – the output key where markers is written. Defaults to "markers". threshold (float) – threshold the float values of instance border map to int 0 or 1 with specified threshold. It turns uncertain area to 1 and other area to 0. Defaults to 0.4. radius (int) – the radius of the disk-shaped footprint used in opening. Defaults to 2. min_object_size (int) – objects smaller than this size are removed. Defaults to 10. postprocess_fn (Optional[Callable]) – execute additional post transformation on marker. Defaults to None. dtype (Union[dtype, type, str, None]) – target data content type to convert, default is np.uint8. allow_missing_keys – don't raise exception if key is missing. Dictionary-based wrapper for monai.apps.pathology.transforms.post.array.HoVerNetInstanceMapPostProcessing. The post-processing transform for HoVerNet model to generate instance segmentation map. It generates an instance segmentation map as well as a dictionary

containing centroids, bounding boxes, and contours for each instance. nuclear_prediction_key (str) – the key for HoVerNet NP (nuclear prediction) branch. Defaults to HoVerNetBranch.NP. hover_map_key (str) – the key for HoVerNet NC (nuclear prediction) branch. Defaults to HoVerNetBranch.HV. instance_info_key (str) – the output key where instance information (contour, bounding boxes, and centroids) is written. Defaults to “instance_info”. instance_map_key (str) – the output key where instance map is written. Defaults to “instance_map”. activation (Union[str, Callable]) – the activation layer to be applied on the input probability map. It can be “softmax” or “sigmoid” string, or any callable. Defaults to “softmax”. mask_threshold (Optional[float]) – a float value to threshold to binarize probability map to generate mask. min_object_size (int) – objects smaller than this size are removed. Defaults to 10. sobel_kernel_size (int) – the size of the Sobel kernel used in GenerateInstanceBorder. Defaults to 5. distance_smooth_fn (Optional[Callable]) – smoothing function for distance map. If not provided, monai.transforms.intensity.GaussianSmooth() will be used. marker_threshold (float) – a float value to threshold to binarize instance border map for markers. It turns uncertain area to 1 and other area to 0. Defaults to 0.4. marker_radius (int) – the radius of the disk-shaped footprint used in opening of markers. Defaults to 2. marker_postprocess_fn (Optional[Callable]) – post-process function for watershed markers. If not provided, monai.transforms.post.FillHoles() will be used. watershed_connectivity (Optional[int]) – connectivity argument of skimage.segmentation.watershed. min_num_points (int) – minimum number of points to be considered as a contour. Defaults to 3. contour_level (Optional[float]) – an optional value for skimage.measure.find_contours to find contours in the array. If not provided, the level is set to (max(image) + min(image)) / 2. Dictionary-based wrapper for monai.apps.pathology.transforms.post.array.HoVerNetNuclearTypePostProcessing. It updates the input instance info dictionary with information about types of the nuclei (value and probability). Also if requested (return_type_map=True), it generates a pixel-level type map. type_prediction_key (str) – the key for HoVerNet NC (type prediction) branch. Defaults to HoVerNetBranch.NC. instance_info_key (str) – the key where instance information (contour, bounding boxes, and centroids) is stored. Defaults to “instance_info”. instance_map_key (str) – the key where instance map is stored. Defaults to “instance_map”. type_map_key (str) – the output key where type map is written. Defaults to “type_map”.

Hard Negative Sampler#

The functions in this script are adapted from nnDetection, MIC-DKFZ/nnDetection HardNegativeSampler is used to suppress false positive rate in classification tasks. During training, it selects negative samples with high prediction scores. The training workflow is described as the follows: 1) forward network and get prediction scores (classification prob/logits) for all the samples; 2) use hard negative sampler to choose negative samples with high prediction scores and some positive samples; 3) compute classification loss for the selected samples; 4) do back propagation. batch_size_per_image (int) – number of training samples to be randomly selected per image positive_fraction (float) – percentage of positive elements in the selected samples min_neg (int) – minimum number of negative samples to select if possible. pool_size (float) – when we need num_neg hard negative samples, they will be randomly selected from num_neg * pool_size negative samples with the highest prediction scores. Larger pool_size gives more randomness, yet selects negative samples that are less ‘hard’, i.e., negative samples with lower prediction scores.

Sample enough negatives to fill up self.batch_size_per_image negative (Tensor) – indices of positive samples num_pos (int) – number of positive samples to draw int number of negative samples Number of positive samples to draw positive (Tensor) – indices of positive samples int number of positive sample Select positive samples positive (Tensor) – indices of positive samples, sized (P,), where P is the number of positive samples num_pos (int) – number of positive samples to sample labels (Tensor) – labels for all samples, sized (A,), where A is the number of samples. Tensor binary mask of positive samples to choose, sized (A,),where A is the number of samples in one image where A is the number of samples in one image Select positives and hard negatives from list samples per image. Hard negative sampler will be applied to each image independently. target_labels (List[Tensor]) – list of labels per image. For image i in the batch, target_labels[i] is a Tensor sized (A_i,), where A_i is the number of samples in image i. Positive samples have positive labels, negative samples have label 0. fg_probs (List[Tensor]) – list of maximum foreground probability per images, For image i in the batch, target_labels[i] is a Tensor sized (A_i,), where A_i is the number of samples in image i. Tuple[List[Tensor], List[Tensor]] list of binary mask for positive samples list binary mask for negative samples list of binary mask for positive samples list binary mask for negative samples Example Select positives and hard negatives from samples. labels_per_img (Tensor) – labels, sized (A,). Positive samples have positive labels, negative samples have label 0. fg_probs_per_img (Tensor) – maximum foreground probability, sized (A,) Tuple[Tensor, Tensor] binary mask for positive samples, sized (A,) binary mask for negative samples, sized (A,) binary mask for positive samples, sized (A,) binary mask for negative samples, sized (A,) Example Base class of hard negative sampler. Hard negative sampler is used to suppress false positive rate in classification tasks. During training, it select negative samples with high prediction scores. The training workflow is described as the follows: 1) forward network and get prediction scores (classification prob/logits) for all the samples; 2) use hard negative sampler to choose negative samples with high prediction scores and some positive samples; 3) compute classification loss for the selected samples; 4) do back propagation. pool_size (float) – when we need num_neg hard negative samples, they will be randomly selected from num_neg * pool_size negative samples with the highest prediction scores. Larger pool_size gives more randomness, yet selects negative samples that are less ‘hard’, i.e., negative samples with lower prediction scores. Select hard negative samples. negative (Tensor) – indices of all the negative samples, sized (P,), where P is the number of negative samples num_neg (int) – number of negative samples to sample fg_probs (Tensor) – maximum foreground prediction scores (probability) across all the classes for each sample, sized (A,), where A is the number of samples. Tensor binary mask of negative samples to choose, sized (A,),where A is the number of samples in one image where A is the number of samples in one image

RetinaNet Network#

Part of this script is adapted from pytorch/vision The network used in RetinaNet. It takes an image tensor as inputs, and outputs a dictionary head_outputs. head_outputs[self.cls_key] is the predicted classification maps, a list of Tensor. head_outputs[self.box_reg_key] is the predicted box regression maps, a list of Tensor. spatial_dims (int) – number of spatial dimensions of the images. We support both 2D and 3D images. num_classes (int) – number of output classes of the model (excluding the background). num_anchors (int) – number of anchors at each location. feature_extractor – a network that outputs feature maps from the input images, each feature map corresponds to a different resolution. Its output can have a format of

Tensor, Dict[Any, Tensor], or Sequence[Tensor]. It can be the output of resnet_fpn_feature_extractor(*args, **kwargs). size_divisible (Union[Sequence[int], int]) – the spatial size of the network input should be divisible by size_divisible, decided by the feature_extractor. Example It takes an image tensor as inputs, and outputs a dictionary head_outputs. head_outputs[self.cls_key] is the predicted classification maps, a list of Tensor. head_outputs[self.box_reg_key] is the predicted box regression maps, a list of Tensor. images (Tensor) – input images, sized (B, img_channels, H, W) or (B, img_channels, H, W, D). Dict[str, List[Tensor]] a dictionary head_outputs with keys including self.cls_key and self.box_reg_key. head_outputs[self.cls_key] is the predicted classification maps, a list of Tensor. head_outputs[self.box_reg_key] is the predicted box regression maps, a list of Tensor. A classification head for use in RetinaNet. This head takes a list of feature maps as inputs, and outputs a list of classification maps. Each output map has same spatial size with the corresponding input feature map, and the number of output channel is num_anchors * num_classes. in_channels (int) – number of channels of the input feature num_anchors (int) – number of anchors to be predicted num_classes (int) – number of classes to be predicted spatial_dims (int) – spatial dimension of the network, should be 2 or 3. prior_probability (float) – prior probability to initialize classification convolutional layers. It takes a list of feature maps as inputs, and outputs a list of classification maps. Each output classification map has same spatial size with the corresponding input feature map, and the number of output channel is num_anchors * num_classes. x (List[Tensor]) – list of feature map, x[i] is a (B, in_channels, H_i, W_i) or (B, in_channels, H_i, W_i, D_i) Tensor. List[Tensor] cls_logits_maps, list of classification map. cls_logits_maps[i] is a (B, num_anchors * num_classes, H_i, W_i) or (B, num_anchors * num_classes, H_i, W_i, D_i) Tensor. A regression head for use in RetinaNet. This head takes a list of feature maps as inputs, and outputs a list of box regression maps. Each output box regression map has same spatial size with the corresponding input feature map, and the number of output channel is num_anchors * 2 * spatial_dims. in_channels (int) – number of channels of the input feature num_anchors (int) – number of anchors to be predicted spatial_dims (int) – spatial dimension of the network, should be 2 or 3. It takes a list of feature maps as inputs, and outputs a list of box regression maps. Each output box regression map has same spatial size with the corresponding input feature map, and the number of output channel is num_anchors * 2 * spatial_dims. x (List[Tensor]) – list of feature map, x[i] is a (B, in_channels, H_i, W_i) or (B, in_channels, H_i, W_i, D_i) Tensor. List[Tensor] box_regression_maps, list of box regression map. cls_logits_maps[i] is a (B, num_anchors * 2 * spatial_dims, H_i, W_i) or (B, num_anchors * 2 * spatial_dims, H_i, W_i, D_i) Tensor. Constructs a feature extractor network with a ResNet-FPN backbone, used as feature_extractor in RetinaNet. Reference: “Focal Loss for Dense Object Detection”. The returned feature_extractor network takes an image tensor as inputs, and outputs a dictionary that maps string to the extracted feature maps (Tensor). The input to the returned feature_extractor is expected to be a list of tensors, each of shape [C, H, W] or [C, H, W, D], one for each image. Different images can have different sizes. backbone (ResNet) – a ResNet model, used as backbone. spatial_dims (int) – number of spatial dimensions of the images. We support both 2D and 3D images. pretrained_backbone (bool) – whether the backbone has been pre-trained. returned_layers (Sequence[int]) – returned layers to extract feature maps. Each returned layer should be in the range [1,4]. len(returned_layers)+1 will be the number of extracted feature maps. There is an extra maxpooling layer LastLevelMaxPool() appended. trainable_backbone_layers (Optional[int]) – number of trainable (not frozen) resnet layers starting from final block. Valid values are between 0 and 5, with 5 meaning all backbone layers are trainable. When pretrained_backbone is False, this value is set to be 5. When pretrained_backbone is True, if None is

passed (the default) this value is set to 3. Example

RetinaNet Detector#

Part of this script is adapted from pytorch/vision Retinanet detector, expandable to other one stage anchor based box detectors in the future. An example of construction can found in the source code of retinanet_resnet50_fpn_detector() . The input to the model is expected to be a list of tensors, each of shape (C, H, W) or (C, H, W, D), one for each image, and should be in 0-1 range. Different images can have different sizes. Or it can also be a Tensor sized (B, C, H, W) or (B, C, H, W, D). In this case, all images have same size. The behavior of the model changes depending if it is in training or evaluation mode. During training, the model expects both the input tensors, as well as a targets (list of dictionary), containing: boxes (FloatTensor[N, 4] or FloatTensor[N, 6]): the ground-truth boxes in StandardMode, i.e., [xmin, ymin, xmax, ymax] or [xmin, ymin, zmin, xmax, ymax, zmax] format, with 0 <= xmin < xmax <= H, 0 <= ymin < ymax <= W, 0 <= zmin < zmax <= D. labels: the class label for each ground-truth box The model returns a Dict[str, Tensor] during training, containing the classification and regression losses. When saving the model, only self.network contains trainable parameters and needs to be saved. During inference, the model requires only the input tensors, and returns the post-processed predictions as a List[Dict[Tensor]], one for each input image. The fields of the Dict are as follows: boxes (FloatTensor[N, 4] or FloatTensor[N, 6]): the predicted boxes in StandardMode, i.e., [xmin, ymin, xmax, ymax] or [xmin, ymin, zmin, xmax, ymax, zmax] format, with 0 <= xmin < xmax <= H, 0 <= ymin < ymax <= W, 0 <= zmin < zmax <= D. labels (Int64Tensor[N]): the predicted labels for each image labels_scores (Tensor[N]): the scores for each prediction network – a network that takes an image Tensor sized (B, C, H, W) or (B, C, H, W, D) as input and outputs a dictionary Dict[str, List[Tensor]] or Dict[str, Tensor]. anchor_generator (AnchorGenerator) – anchor generator. box_overlap_metric (Callable) – func that compute overlap between two sets of boxes, default is Intersection over Union (IoU). debug (bool) – whether to print out internal parameters, used for debugging and parameter tuning. Notes Input argument network can be a monai.apps.detection.networks.retinanet_network.RetinaNet(*) object, but any network that meets the following rules is a valid input network. It should have attributes including spatial_dims, num_classes, cls_key, box_reg_key, num_anchors, size_divisible. spatial_dims (int) is the spatial dimension of the network, we support both 2D and 3D. num_classes (int) is the number of classes, excluding the background. size_divisible (int or Sequence[int]) is the expectation on the input image shape. The network needs the input spatial_size to be divisible by size_divisible, length should be 2 or 3. cls_key (str) is the key to represent classification in the output dict. box_reg_key (str) is the key to represent box regression in the output dict. num_anchors (int) is the number of anchor shapes at each location. it should equal to self.anchor_generator.num_anchors_per_location()[0]. Its input should be an image Tensor sized (B, C, H, W) or (B, C, H, W, D). About its output head_outputs: It should be a dictionary with at least two keys: network.cls_key and network.box_reg_key. head_outputs[network.cls_key] should be List[Tensor] or Tensor. Each Tensor represents classification logits map at one resolution level, sized (B, num_classes*num_anchors, H_i, W_i) or (B, num_classes*num_anchors, H_i, W_i, D_i). head_outputs[network.box_reg_key] should be List[Tensor] or Tensor. Each Tensor represents box regression map at one resolution level, sized (B, 2*spatial_dims*num_anchors, H_i, W_i)or (B, 2*spatial_dims*num_anchors, H_i, W_i, D_i). len(head_outputs[network.cls_key]) == len(head_outputs[network.box_reg_key]). Example Compute the matched indices

between anchors and ground truth (gt) boxes in targets. output[k][i] represents the matched gt index for anchor[i] in image k. Suppose there are M gt boxes for image k. The range of it output[k][i] value is [-2, -1, 0, …, M-1]. [0, M - 1] indicates this anchor is matched with a gt box, while a negative value indicating that it is not matched. anchors (List[Tensor]) – a list of Tensor. Each Tensor represents anchors for each image, sized (sum(HWA), 2*spatial_dims) or (sum(HWDA), 2*spatial_dims). A = self.num_anchors_per_loc. targets (List[Dict[str, Tensor]]) – a list of dict. Each dict with two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. num_anchor_locs_per_level (Sequence[int]) – each element represents HW or HWD at this level. List[Tensor] a list of matched index matched_idxs_per_image (Tensor[int64]), Tensor sized (sum(HWA),) or (sum(HWDA),). Suppose there are M gt boxes. matched_idxs_per_image[i] is a matched gt index in [0, M - 1] or a negative value indicating that anchor i could not be matched. BELOW_LOW_THRESHOLD = -1, BETWEEN_THRESHOLDS = -2 Compute box regression losses. box_regression (Tensor) – box regression results, sized (B, sum(HWA), 2*self.spatial_dims) targets (List[Dict[str, Tensor]]) – a list of dict. Each dict with two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. anchors (List[Tensor]) – a list of Tensor. Each Tensor represents anchors for each image, sized (sum(HWA), 2*spatial_dims) or (sum(HWDA), 2*spatial_dims). A = self.num_anchors_per_loc. matched_idxs (List[Tensor]) – a list of matched index. each element is sized (sum(HWA),) or (sum(HWDA),) Tensor box regression losses. Compute classification losses. cls_logits (Tensor) – classification logits, sized (B, sum(HW(D)A), self.num_classes) targets (List[Dict[str, Tensor]]) – a list of dict. Each dict with two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. matched_idxs (List[Tensor]) – a list of matched index. each element is sized (sum(HWA),) or (sum(HWDA),) Tensor classification losses. Compute losses. head_outputs_reshape (Dict[str, Tensor]) – reshaped head_outputs. head_output_reshape[self.cls_key] is a Tensor sized (B, sum(HW(D)A), self.num_classes). head_output_reshape[self.box_reg_key] is a Tensor sized (B, sum(HW(D)A), 2*self.spatial_dims) targets (List[Dict[str, Tensor]]) – a list of dict. Each dict with two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. anchors (List[Tensor]) – a list of Tensor. Each Tensor represents anchors for each image, sized (sum(HWA), 2*spatial_dims) or (sum(HWDA), 2*spatial_dims). A = self.num_anchors_per_loc. Dict[str, Tensor] a dict of several kinds of losses. Returns a dict of losses during training, or a list predicted dict of boxes and labels during inference. input_images (Union[List[Tensor], Tensor]) – The input to the model is expected to be a list of tensors, each of shape (C, H, W) or (C, H, W, D), one for each image, and should be in 0-1 range. Different images can have different sizes. Or it can also be a Tensor sized (B, C, H, W) or (B, C, H, W, D). In this case, all images have same size. targets (Optional[List[Dict[str, Tensor]]]) – a list of dict. Each dict with two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image (optional). use_inferer (bool) – whether to use self.inferer, a sliding window inferer, to do the inference. If False, will simply forward the network. If True, will use self.inferer, and requires self.set_sliding_window_inferer(*args) to have been called before. Union[Dict[str, Tensor], List[Dict[str, Tensor]]] If training mode, will return a dict with at least two keys, including self.cls_key and self.box_reg_key, representing classification loss and box regression loss. If evaluation mode, will return a list of detection results. Each element corresponds to an images in input_images, is a dict with at least three keys, including self.target_box_key, self.target_label_key, self.pred_score_key, representing predicted boxes, classification labels, and classification scores. If training mode, will return a dict with at least two keys, including self.cls_key and self.box_reg_key,

representing classification loss and box regression loss. If evaluation mode, will return a list of detection results. Each element corresponds to an images in input_images, is a dict with at least three keys, including self.target_box_key, self.target_label_key, self.pred_score_key, representing predicted boxes, classification labels, and classification scores. Generate anchors and store it in self.anchors: List[Tensor]. We generate anchors only when there is no stored anchors, or the new coming images has different shape with self.previous_image_shape images (Tensor) – input images, a (B, C, H, W) or (B, C, H, W, D) Tensor. head_outputs (Dict[str, List[Tensor]]) – head_outputs. head_output_reshape[self.cls_key] is a Tensor sized (B, sum(HW(D)A), self.num_classes). head_output_reshape[self.box_reg_key] is a Tensor sized (B, sum(HW(D)A), 2*self.spatial_dims) Get samples from one image for box regression losses computation. box_regression_per_image (Tensor) – box regression result for one image, (sum(HWA), 2*self.spatial_dims) targets_per_image (Dict[str, Tensor]) – a dict with at least two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. anchors_per_image (Tensor) – anchors of one image, sized (sum(HWA), 2*spatial_dims) or (sum(HWDA), 2*spatial_dims). A = self.num_anchors_per_loc. matched_idxs_per_image (Tensor) – matched index, sized (sum(HWA),) or (sum(HWDA),) Tuple[Tensor, Tensor] paired predicted and GT samples from one image for box regression losses computation Get samples from one image for classification losses computation. cls_logits_per_image (Tensor) – classification logits for one image, (sum(HWA), self.num_classes) targets_per_image (Dict[str, Tensor]) – a dict with at least two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. matched_idxs_per_image (Tensor) – matched index, Tensor sized (sum(HWA),) or (sum(HWDA),) Suppose there are M gt boxes. matched_idxs_per_image[i] is a matched gt index in [0, M - 1] or a negative value indicating that anchor i could not be matched. BELOW_LOW_THRESHOLD = -1, BETWEEN_THRESHOLDS = -2 Tuple[Tensor, Tensor] paired predicted and GT samples from one image for classification losses computation Postprocessing to generate detection result from classification logits and box regression. Use self.box_selector to select the final output boxes for each image. head_outputs_reshape (Dict[str, Tensor]) – reshaped head_outputs. head_output_reshape[self.cls_key] is a Tensor sized (B, sum(HW(D)A), self.num_classes). head_output_reshape[self.box_reg_key] is a Tensor sized (B, sum(HW(D)A), 2*self.spatial_dims) targets – a list of dict. Each dict with two keys: self.target_box_key and self.target_label_key, ground-truth boxes present in the image. anchors (List[Tensor]) – a list of Tensor. Each Tensor represents anchors for each image, sized (sum(HWA), 2*spatial_dims) or (sum(HWDA), 2*spatial_dims). A = self.num_anchors_per_loc. List[Dict[str, Tensor]] a list of dict, each dict corresponds to detection result on image. Using for training. Set ATSS matcher that matches anchors with ground truth boxes num_candidates (int) – number of positions to select candidates from. Smaller value will result in a higher matcher threshold and less matched candidates. center_in_gt (bool) – If False (default), matched anchor center points do not need to lie withing the ground truth box. Recommend False for small objects. If True, will result in a strict matcher and less matched candidates. None Using for training. Set torchvision balanced sampler that samples part of the anchors for training. batch_size_per_image (int) – number of elements to be selected per image positive_fraction (float) – percentage of positive elements per batch Set the weights for box coder. weights (Tuple[float]) – a list/tuple with length of 2*self.spatial_dims Using for training. Set loss for box regression. box_loss (Module) – loss module for box regression encode_gt (bool) – if True, will encode ground truth boxes to target box regression before computing the losses. Should be True for L1 loss and False for GIoU loss. decode_pred (bool) – if True, will decode predicted box

regression into predicted boxes before computing losses. Should be False for L1 loss and True for GIoU loss. Example None Using for inference. Set the parameters that are used for box selection during inference. The box selection is performed with the following steps: For each level, discard boxes with scores less than self.score_thresh. For each level, keep boxes with top self.topk_candidates_per_level scores. For the whole image, perform non-maximum suppression (NMS) on boxes, with overlapping threshold nms_thresh. For the whole image, keep boxes with top self.detections_per_img scores. score_thresh (float) – no box with scores less than score_thresh will be kept topk_candidates_per_level (int) – max number of boxes to keep for each level nms_thresh (float) – box overlapping threshold for NMS detections_per_img (int) – max number of boxes to keep for each image Using for training. Set loss for classification that takes logits as inputs, make sure sigmoid/softmax is built in. cls_loss (Module) – loss module for classification Example None Using for training. Set hard negative sampler that samples part of the anchors for training. HardNegativeSampler is used to suppress false positive rate in classification tasks. During training, it select negative samples with high prediction scores. batch_size_per_image (int) – number of elements to be selected per image positive_fraction (float) – percentage of positive elements in the selected samples min_neg (int) – minimum number of negative samples to select if possible. pool_size (float) – when we need num_neg hard negative samples, they will be randomly selected from num_neg * pool_size negative samples with the highest prediction scores. Larger pool_size gives more randomness, yet selects negative samples that are less ‘hard’, i.e., negative samples with lower prediction scores. Using for training. Set torchvision matcher that matches anchors with ground truth boxes. fg_iou_thresh (float) – foreground IoU threshold for Matcher, considered as matched if IoU > fg_iou_thresh bg_iou_thresh (float) – background IoU threshold for Matcher, considered as not matched if IoU < bg_iou_thresh allow_low_quality_matches – if True, produce additional matches for predictions that have only low-quality match candidates. None Define sliding window inferer and store it to self.inferer. Set keys for the training targets and inference outputs. During training, both box_key and label_key should be keys in the targets when performing self.forward(input_images, targets). During inference, they will be the keys in the output dict of self.forward(input_images)`. Returns a RetinaNet detector using a ResNet-50 as backbone, which can be pretrained from Med3D: Transfer Learning for 3D Medical Image Analysis _. num_classes (int) – number of output classes of the model (excluding the background). anchor_generator (AnchorGenerator) – AnchorGenerator, returned_layers (Sequence[int]) – returned layers to extract feature maps. Each returned layer should be in the range [1,4]. len(returned_layers)+1 will be the number of extracted feature maps. There is an extra maxpooling layer LastLevelMaxPool() appended. pretrained (bool) – If True, returns a backbone pre-trained on 23 medical datasets progress (bool) – If True, displays a progress bar of the download to stderr RetinaNetDetector A RetinaNetDetector object with resnet50 as backbone Example

Transforms#

This function applies affine matrices to the boxes boxes (Union[ndarray, Tensor]) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode affine (Union[ndarray, Tensor]) – affine matrix to be applied to the box coordinates, sized (spatial_dims+1,spatial_dims+1) Union[ndarray, Tensor] returned affine transformed boxes, with same data type as boxes, does not share memory with boxes Convert box to int16 mask image, which has the same size with the input

image. boxes (Union[ndarray, Tensor]) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode. labels (Union[ndarray, Tensor]) – classification foreground(fg) labels corresponding to boxes, dtype should be int, sized (N,). spatial_size (Union[Sequence[int], int]) – image spatial size. bg_label (int) – background labels for the output mask image, make sure it is smaller than any fg labels. ellipse_mask (bool) – bool. If True, it assumes the object shape is close to ellipse or ellipsoid. If False, it assumes the object shape is close to rectangle or cube and well occupies the bounding box. If the users are going to apply random rotation as data augmentation, we suggest setting ellipse_mask=True See also Kalra et al. “Towards Rotation Invariance in Object Detection”, ICCV 2021. bool. If True, it assumes the object shape is close to ellipse or ellipsoid. If False, it assumes the object shape is close to rectangle or cube and well occupies the bounding box. If the users are going to apply random rotation as data augmentation, we suggest setting ellipse_mask=True See also Kalra et al. “Towards Rotation Invariance in Object Detection”, ICCV 2021. Union[ndarray, Tensor] int16 array, sized (num_box, H, W). Each channel represents a box.The foreground region in channel c has intensity of labels[c]. The background intensity is bg_label. The foreground region in channel c has intensity of labels[c]. The background intensity is bg_label. Convert int16 mask image to box, which has the same size with the input image boxes_mask (Union[ndarray, Tensor]) – int16 array, sized (num_box, H, W). Each channel represents a box. The foreground region in channel c has intensity of labels[c]. The background intensity is bg_label. bg_label (int) – background labels for the boxes_mask box_dtype – output dtype for boxes label_dtype – output dtype for labels Tuple[Union[ndarray, Tensor], Union[ndarray, Tensor]] bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode. classification foreground(fg) labels, dtype should be int, sized (N,). bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode. classification foreground(fg) labels, dtype should be int, sized (N,). Flip boxes when the corresponding image is flipped boxes (Union[ndarray, Tensor]) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode spatial_size (Union[Sequence[int], int]) – image spatial size. flip_axes (Union[Sequence[int], int, None]) – spatial axes along which to flip over. Default is None. The default axis=None will flip over all of the axes of the input array. If axis is negative it counts from the last to the first axis. If axis is a tuple of ints, flipping is performed on all of the axes specified in the tuple. flipped boxes, with same data type as boxes, does not share memory with boxes Resize boxes when the corresponding image is resized boxes (Union[ndarray, Tensor]) – source bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode src_spatial_size (Union[Sequence[int], int]) – source image spatial size. dst_spatial_size (Union[Sequence[int], int]) – target image spatial size. resized boxes, with same data type as boxes, does not share memory with boxes Example Rotate boxes by 90 degrees in the plane specified by axes. Rotation direction is from the first towards the second axis. boxes (Union[ndarray, Tensor]) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode spatial_size (Union[Sequence[int], int]) – image spatial size. k (int) – number of times the array is rotated by 90 degrees. axes (Tuple[int, int]) – (2,) array_like The array is rotated in the plane defined by the axes. Axes must be different. A rotated view of boxes. Notes rot90_boxes(boxes, spatial_size, k=1, axes=(1,0)) is the reverse of rot90_boxes(boxes, spatial_size, k=1, axes=(0,1)) rot90_boxes(boxes, spatial_size, k=1, axes=(1,0)) is equivalent to rot90_boxes(boxes, spatial_size, k=-1, axes=(0,1)) For element in labels, select indices keep from it. labels (Union[Sequence[Union[ndarray, Tensor]], ndarray, Tensor]) – Sequence of array. Each element represents classification labels or scores corresponding to boxes, sized

(N,). keep (Union[ndarray, Tensor]) – the indices to keep, same length with each element in labels. Union[Tuple, ndarray, Tensor] selected labels, does not share memory with original labels. Interchange two axes of boxes. boxes (Union[ndarray, Tensor]) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode axis1 (int) – First axis. axis2 (int) – Second axis. boxes with two axes interchanged. Zoom boxes boxes (Union[ndarray, Tensor]) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode zoom (Union[Sequence[float], float]) – The zoom factor along the spatial axes. If a float, zoom is the same for each spatial axis. If a sequence, zoom should contain one value for each spatial axis. zoomed boxes, with same data type as boxes, does not share memory with boxes Example A collection of “vanilla” transforms for box operations Project-MONAI/MONAI Applies affine matrix to the boxes Convert box to int16 mask image, which has the same size with the input image. bg_label (int) – background labels for the output mask image, make sure it is smaller than any foreground(fg) labels. ellipse_mask (bool) – bool. If True, it assumes the object shape is close to ellipse or ellipsoid. If False, it assumes the object shape is close to rectangle or cube and well occupies the bounding box. If the users are going to apply random rotation as data augmentation, we suggest setting ellipse_mask=True See also Kalra et al. “Towards Rotation Invariance in Object Detection”, ICCV 2021. bool. If True, it assumes the object shape is close to ellipse or ellipsoid. If False, it assumes the object shape is close to rectangle or cube and well occupies the bounding box. If the users are going to apply random rotation as data augmentation, we suggest setting ellipse_mask=True See also Kalra et al. “Towards Rotation Invariance in Object Detection”, ICCV 2021. Clip the bounding boxes and the associated labels/scores to make sure they are within the image. There might be multiple arrays of labels/scores associated with one array of boxes. remove_empty (bool) – whether to remove the boxes and corresponding labels that are actually empty This transform converts the boxes in src_mode to the dst_mode. src_mode (Union[str, BoxMode, Type[BoxMode], None]) – source box mode. If it is not given, this func will assume it is StandardMode(). dst_mode (Union[str, BoxMode, Type[BoxMode], None]) – target box mode. If it is not given, this func will assume it is StandardMode(). Note StandardMode = CornerCornerModeTypeA, also represented as “xyxy” for 2D and “xyzxyz” for 3D. “xyxy”: boxes has format [xmin, ymin, xmax, ymax] “xyzxyz”: boxes has format [xmin, ymin, zmin, xmax, ymax, zmax] “xxyy”: boxes has format [xmin, xmax, ymin, ymax] “xxyyzz”: boxes has format [xmin, xmax, ymin, ymax, zmin, zmax] “xyxyzz”: boxes has format [xmin, ymin, xmax, ymax, zmin, zmax] “xywh”: boxes has format [xmin, ymin, xsize, ysize] “xyzwhd”: boxes has format [xmin, ymin, zmin, xsize, ysize, zsize] “ccwh”: boxes has format [xcenter, ycenter, xsize, ysize] “cccwhd”: boxes has format [xcenter, ycenter, zcenter, xsize, ysize, zsize] CornerCornerModeTypeA: equivalent to “xyxy” or “xyzxyz” CornerCornerModeTypeB: equivalent to “xxyy” or “xxyyzz” CornerCornerModeTypeC: equivalent to “xyxy” or “xyxyzz” CornerSizeMode: equivalent to “xywh” or “xyzwhd” CenterSizeMode: equivalent to “ccwh” or “cccwhd” CornerCornerModeTypeA(): equivalent to “xyxy” or “xyzxyz” CornerCornerModeTypeB(): equivalent to “xxyy” or “xxyyzz” CornerCornerModeTypeC(): equivalent to “xyxy” or “xyxyzz” CornerSizeMode(): equivalent to “xywh” or “xyzwhd” CenterSizeMode(): equivalent to “ccwh” or “cccwhd” None: will assume mode is StandardMode() Example Convert given boxes to standard mode. Standard mode is “xyxy” or “xyzxyz”, representing box format of [xmin, ymin, xmax, ymax] or [xmin, ymin, zmin, xmax, ymax, zmax]. mode (Union[str, BoxMode, Type[BoxMode], None]) – source box mode. If it is not given, this func will assume it is StandardMode(). It follows the same format with src_mode in ConvertBoxMode . Example Reverses the box coordinates along the given spatial axis. Preserves shape. spatial_axis (Union[Sequence[int], int, None]) – spatial axes

along which to flip over. Default is None. The default axis=None will flip over all of the axes of the input array. If axis is negative it counts from the last to the first axis. If axis is a tuple of ints, flipping is performed on all of the axes specified in the tuple. Convert int16 mask image to box, which has the same size with the input image. Pairs with monai.apps.detection.transforms.array.BoxToMask. Please make sure the same min_fg_label is used when using the two transforms in pairs. bg_label (int) – background labels for the output mask image, make sure it is smaller than any foreground(fg) labels. box_dtype – output dtype for boxes label_dtype – output dtype for labels Resize the input boxes when the corresponding image is resized to given spatial size (with scaling, not cropping/padding). spatial_size (Union[Sequence[int], int]) – expected shape of spatial dimensions after resize operation. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. size_mode (str) – should be “all” or “longest”, if “all”, will use spatial_size for all the spatial dims, if “longest”, rescale the image so that only the longest side is equal to specified spatial_size, which must be an int number in this case, keeping the aspect ratio of the initial image, refer to:
https://albumentations.ai/docs/api_reference/augmentations/geometric/resize/
#albumentations.augmentations.geometric.resize.LongestMaxSize. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Rotate a boxes by 90 degrees in the plane specified by axes. See box_ops.rot90_boxes for additional details k (int) – number of times to rotate by 90 degrees. spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default: (0, 1), this is the first two axis in spatial dimensions. If axis is negative it counts from the last to the first axis. General purpose box cropper when the corresponding image is cropped by SpatialCrop(*) with the same ROI. The difference is that we do not support negative indexing for roi_slices. If a dimension of the expected ROI size is bigger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. It can support to crop ND spatial boxes. a list of slices for each spatial dimension (do not allow for use of negative indexing) a spatial center and size the start and end coordinates of the ROI roi_center (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for center of the crop ROI. roi_size (Union[Sequence[int], ndarray, Tensor, None]) – size of the crop ROI, if a dimension of ROI size is bigger than image size, will not crop that dimension of the image. roi_start (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for start of the crop ROI. roi_end (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for end of the crop ROI, if a coordinate is out of image, use the end coordinate of image. roi_slices (Optional[Sequence[slice]]) – list of slices for each of the spatial dimensions. Zooms an ND Box with same padding or slicing setting with Zoom(). zoom (Union[Sequence[float], float]) – The zoom factor along the spatial axes. If a float, zoom is the same for each spatial axis. If a sequence, zoom should contain one value for each spatial axis. keep_size (bool) – Should keep original size (padding/slicing if needed), default is True. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. A collection of dictionary-based wrappers around the “vanilla” transforms for box operations defined in monai.apps.detection.transforms.array. Class names are ended with ‘d’ to denote dictionary-based transforms. alias of AffineBoxToImageCoordinated alias of AffineBoxToImageCoordinated Dictionary-based transform that converts box in world coordinate to image coordinate. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be

StandardMode. box_ref_image_keys (str) – The single key that represents the reference image to which box_keys are attached. remove_empty – whether to remove the boxes that are actually empty allow_missing_keys (bool) – don’t raise exception if key is missing. image_meta_key (Optional[str]) – explicitly indicate the key of the corresponding metadata dictionary. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, affine, original_shape, etc. it is a string, map to the box_ref_image_key. if None, will try to construct meta_keys by box_ref_image_key_{meta_key_postfix}. image_meta_key_postfix (Optional[str]) – if image_meta_keys=None, use box_ref_image_key_{postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. For example, to handle key image, read/write affine matrices from the metadata image_meta_dict dictionary’s affine field. affine_lps_to_ras – default False. Yet if 1) the image is read by ITKReader, and 2) the ITKReader has affine_lps_to_ras=True, and 3) the box is in world coordinate, then set affine_lps_to_ras=True. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] alias of BoxToMaskd alias of BoxToMaskd Dictionary-based wrapper of monai.apps.detection.transforms.array.BoxToMask. Pairs with monai.apps.detection.transforms.dictionary.MaskToBoxd . Please make sure the same min_fg_label is used when using the two transforms in pairs. The output d[box_mask_key] will have background intensity 0, since the following operations may pad 0 on the border. This is the general solution for transforms that need to be applied on images and boxes simultaneously. It is performed with the following steps. use BoxToMaskd to covert boxes and labels to box_masks; do transforms, e.g., rotation or cropping, on images and box_masks together; use MaskToBoxd to convert box_masks back to boxes and labels. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_mask_keys (Union[Collection[Hashable], Hashable]) – Keys to store output box mask results for transformation. Same length with box_keys. label_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the labels corresponding to the box_keys. Same length with box_keys. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. min_fg_label (int) – min foreground box label. ellipse_mask (bool) – bool. If True, it assumes the object shape is close to ellipse or ellipsoid. If False, it assumes the object shape is close to rectangle or cube and well occupies the bounding box. If the users are going to apply random rotation as data augmentation, we suggest setting ellipse_mask=True See also Kalra et al. “Towards Rotation Invariance in Object Detection”, ICCV 2021. bool. If True, it assumes the object shape is close to ellipse or ellipsoid. If False, it assumes the object shape is close to rectangle or cube and well occupies the bounding box. If the users are going to apply random rotation as data augmentation, we suggest setting ellipse_mask=True See also Kalra et al. “Towards Rotation Invariance in Object Detection”, ICCV 2021. allow_missing_keys (bool) – don’t raise exception if key is missing. Example alias of ClipBoxToImaged alias of ClipBoxToImaged Dictionary-based wrapper of monai.apps.detection.transforms.array.ClipBoxToImage. Clip the bounding boxes and the associated labels/scores to makes sure they are within the image. There might be multiple keys of labels/scores associated with one key of boxes. box_keys (Union[Collection[Hashable], Hashable]) – The single key to pick box data for transformation. The box mode is assumed to be StandardMode. label_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the labels corresponding to the box_keys. Multiple keys are allowed. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – The single key that represents the

reference image to which box_keys and label_keys are attached. remove_empty (bool) – whether to remove the boxes that are actually empty allow_missing_keys (bool) – don't raise exception if key is missing. Example alias of ConvertBoxModed alias of ConvertBoxModed Dictionary-based wrapper of monai.apps.detection.transforms.array.ConvertBoxMode. This transform converts the boxes in src_mode to the dst_mode. Example box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. src_mode (Union[str, BoxMode, Type[BoxMode], None]) – source box mode. If it is not given, this func will assume it is StandardMode(). It follows the same format with src_mode in ConvertBoxMode . dst_mode (Union[str, BoxMode, Type[BoxMode], None]) – target box mode. If it is not given, this func will assume it is StandardMode(). It follows the same format with src_mode in ConvertBoxMode . allow_missing_keys (bool) – don't raise exception if key is missing. See also monai.apps.detection,transforms.array.ConvertBoxMode Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] alias of ConvertBoxToStandardModed alias of ConvertBoxToStandardModed Dictionary-based wrapper of monai.apps.detection.transforms.array.ConvertBoxToStandardMode. Convert given boxes to standard mode. Standard mode is "xyxy" or "xyzxyz", representing box format of [xmin, ymin, xmax, ymax] or [xmin, ymin, zmin, xmax, ymax, zmax]. Example box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. mode (Union[str, BoxMode, Type[BoxMode], None]) – source box mode. If it is not given, this func will assume it is StandardMode(). It follows the same format with src_mode in ConvertBoxMode . allow_missing_keys (bool) – don't raise exception if key is missing. See also monai.apps.detection,transforms.array.ConvertBoxToStandardMode Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] alias of FlipBoxd alias of FlipBoxd Dictionary-based transform that flip boxes and images. image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. spatial_axis (Union[Sequence[int], int, None]) – Spatial axes along which to flip over. Default is None. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] alias of MaskToBoxd alias of MaskToBoxd Dictionary-based wrapper of monai.apps.detection.transforms.array.MaskToBox. Pairs with monai.apps.detection.transforms.dictionary.BoxToMaskd . Please make sure the same min_fg_label is used when using the two transforms in pairs. This is the general solution for transforms that need to be applied on images and boxes simultaneously. It is performed with the following steps. use BoxToMaskd to covert boxes and labels to box_masks; do transforms, e.g., rotation or cropping, on images and box_masks together; use MaskToBoxd to convert box_masks back to boxes and labels. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_mask_keys (Union[Collection[Hashable], Hashable]) – Keys to store output box mask results for transformation. Same length with box_keys. label_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the labels corresponding to the box_keys. Same length with box_keys. min_fg_label (int) – min foreground box label. box_dtype – output dtype for box_keys label_dtype – output dtype for label_keys allow_missing_keys (bool) – don't raise exception if key is missing. Example alias of

RandCropBoxByPosNegLabeld alias of RandCropBoxByPosNegLabeld Crop random fixed sized regions that contains foreground boxes. Suppose all the expected fields specified by image_keys have same shape, and add patch_index to the corresponding meta data. And will return a list of dictionaries for all the cropped images. If a dimension of the expected spatial size is bigger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected size, and the cropped results of several images may not have exactly the same shape. image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. They need to have the same spatial size. box_keys (str) – The single key to pick box data for transformation. The box mode is assumed to be StandardMode. label_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the labels corresponding to the box_keys. Multiple keys are allowed. spatial_size (Union[Sequence[int], int]) – the spatial size of the crop region e.g. [224, 224, 128]. if a dimension of ROI size is bigger than image size, will not crop that dimension of the image. if its components have non-positive values, the corresponding size of data[label_key] will be used. for example: if the spatial size of input data is [40, 40, 40] and spatial_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. pos (float) – used with neg together to calculate the ratio pos / (pos + neg) for the probability to pick a foreground voxel as a center rather than a background voxel. neg (float) – used with pos together to calculate the ratio pos / (pos + neg) for the probability to pick a foreground voxel as a center rather than a background voxel. num_samples (int) – number of samples (crop regions) to take in each list. whole_box (bool) – Bool, default True, whether we prefer to contain at least one whole box in the cropped foreground patch. Even if True, it is still possible to get partial box if there are multiple boxes in the image. thresh_image_key (Optional[str]) – if thresh_image_key is not None, use label == 0 & thresh_image > image_threshold to select the negative sample(background) center. so the crop center will only exist on valid image area. image_threshold (float) – if enabled thresh_image_key, use thresh_image > image_threshold to determine the valid image content area. fg_indices_key (Optional[str]) – if provided pre-computed foreground indices of label, will ignore above image_key and image_threshold, and randomly select crop centers based on them, need to provide fg_indices_key and bg_indices_key together, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call FgBgToIndicesd transform first and cache the results. bg_indices_key (Optional[str]) – if provided pre-computed background indices of label, will ignore above image_key and image_threshold, and randomly select crop centers based on them, need to provide fg_indices_key and bg_indices_key together, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call FgBgToIndicesd transform first and cache the results. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. used to add patch_index to the meta dict. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_keys is None, use key_{postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. used to add patch_index to the meta dict. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will be set to match the cropped size (i.e., no cropping in that dimension). allow_missing_keys (bool) – don’t raise exception if key is missing. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of

the input data. NotImplementedError – When the subclass does not override this method. None alias of RandFlipBoxd alias of RandFlipBoxd Dictionary-based transform that randomly flip boxes and images with the given probabilities. image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. prob (float) – Probability of flipping. spatial_axis (Union[Sequence[int], int, None]) – Spatial axes along which to flip over. Default is None. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandFlipBoxd a Randomizable instance. alias of RandRotateBox90d alias of RandRotateBox90d With probability prob, input boxes and images are rotated by 90 degrees in the plane specified by spatial_axes. image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. prob (float) – probability of rotating. (Default 0.1, with 10% probability it returns a rotated array.) max_k (int) – number of rotations will be sampled from np.random.randint(max_k) + 1. (Default 3) spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default: (0, 1), this is the first two axis in spatial dimensions. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] alias of RandZoomBoxd alias of RandZoomBoxd Dictionary-based transform that randomly zooms input boxes and images with given probability within given zoom range. image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. prob (float) – Probability of zooming. min_zoom (Union[Sequence[float], float]) – Min zoom factor. Can be float or sequence same size as image. If a float, select a random factor from [min_zoom, max_zoom] then apply to all spatial dims to keep the original spatial shape ratio. If a sequence, min_zoom should contain one value for each spatial axis. If 2 values provided for 3D data, use the first value for both H & W dims to keep the same zoom ratio. max_zoom (Union[Sequence[float], float]) – Max zoom factor. Can be float or sequence same size as image. If a float, select a random factor from [min_zoom, max_zoom] then apply to all spatial dims to keep the original spatial shape ratio. If a sequence, max_zoom should contain one value for each spatial axis. If 2 values provided for 3D data, use the first value for both H & W dims to keep the same zoom ratio. mode (Union[Sequence[str], str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of string, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – available modes for numpy

array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Union[Sequence[Optional[bool]], bool, None]) – This only has an effect when mode is ‘linear’, ‘bilinear’, ‘bicubic’ or ‘trilinear’. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of bool or None, each element corresponds to a key in keys. keep_size (bool) – Should keep original size (pad if needed), default is True. allow_missing_keys (bool) – don’t raise exception if key is missing. kwargs – other args for np.pad API, note that np.pad treats channel dimension as the first dimension. more details: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandZoomBoxd a Randomizable instance. alias of RotateBox90d alias of RotateBox90d Input boxes and images are rotated by 90 degrees in the plane specified by spatial_axes for k times image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. k (int) – number of times to rotate by 90 degrees. spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default (0, 1), this is the first two axis in spatial dimensions. allow_missing_keys (bool) – don’t raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] alias of ZoomBoxd alias of ZoomBoxd Dictionary-based transform that zooms input boxes and images with the given zoom scale. image_keys (Union[Collection[Hashable], Hashable]) – Keys to pick image data for transformation. box_keys (Union[Collection[Hashable], Hashable]) – Keys to pick box data for transformation. The box mode is assumed to be StandardMode. box_ref_image_keys (Union[Collection[Hashable], Hashable]) – Keys that represent the reference images to which box_keys are attached. zoom (Union[Sequence[float], float]) – The zoom factor along the spatial axes. If a float, zoom is the same for each spatial axis. If a sequence, zoom should contain one value for each spatial axis. mode (Union[Sequence[str], str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of string, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Union[Sequence[Optional[bool]], bool, None]) – This only has an effect when mode is

‘linear’, ‘bilinear’, ‘bicubic’ or ‘trilinear’. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of bool or None, each element corresponds to a key in keys. keep_size (bool) – Should keep original size (pad if needed), default is True. allow_missing_keys (bool) – don’t raise exception if key is missing. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor]

Anchor#

This script is adapted from pytorch/vision This module is modified from torchvision to support both 2D and 3D images. Module that generates anchors for a set of feature maps and image sizes. The module support computing anchors at multiple sizes and aspect ratios per feature map. sizes and aspect_ratios should have the same number of elements, and it should correspond to the number of feature maps. sizes[i] and aspect_ratios[i] can have an arbitrary number of elements. For 2D images, anchor width and height w:h = 1:aspect_ratios[i,j] For 3D images, anchor width, height, and depth w:h:d = 1:aspect_ratios[i,j,0]:aspect_ratios[i,j,1] AnchorGenerator will output a set of sizes[i] * aspect_ratios[i] anchors per spatial location for feature map i. sizes (Sequence[Sequence[int]]) – base size of each anchor. len(sizes) is the number of feature maps, i.e., the number of output levels for the feature pyramid network (FPN). Each element of sizes is a Sequence which represents several anchor sizes for each feature map. aspect_ratios (Sequence) – the aspect ratios of anchors. len(aspect_ratios) = len(sizes). For 2D images, each element of aspect_ratios[i] is a Sequence of float. For 3D images, each element of aspect_ratios[i] is a Sequence of 2 value Sequence. indexing (str) – choose from {'ij', 'xy'}, optional, Matrix ('ij', default and recommended) or Cartesian ('xy') indexing of output. Matrix ('ij', default and recommended) indexing keeps the original axis not changed. To use other monai detection components, please set indexing = 'ij'. Cartesian ('xy') indexing swaps axis 0 and 1. For 2D cases, monai AnchorGenerator(sizes, aspect_ratios, indexing='xy') and torchvision.models.detection.anchor_utils.AnchorGenerator(sizes, aspect_ratios) are equivalent. choose from {'ij', 'xy'}, optional, Matrix ('ij', default and recommended) or Cartesian ('xy') indexing of output. Matrix ('ij', default and recommended) indexing keeps the original axis not changed. To use other monai detection components, please set indexing = 'ij'. Cartesian ('xy') indexing swaps axis 0 and 1. For 2D cases, monai AnchorGenerator(sizes, aspect_ratios, indexing='xy') and torchvision.models.detection.anchor_utils.AnchorGenerator(sizes, aspect_ratios) are equivalent. pytorch/vision Example Generate anchor boxes for each image. images (Tensor) – sized (B, C, W, H) or (B, C, W, H, D) feature_maps (List[Tensor]) – for FPN level i, feature_maps[i] is sized (B, C_i, W_i, H_i) or (B, C_i, W_i, H_i, D_i). This input argument does not have to be the actual feature maps. Any list variable with the same (C_i, W_i, H_i) or (C_i, W_i, H_i, D_i) as feature maps works. List[Tensor] A list with length of B. Each element represents the anchors for this image. The B elements are identical. Example Compute cell anchor shapes at multiple sizes and aspect ratios for the current feature map. scales (Sequence) – a sequence which represents several anchor sizes for the current feature map. aspect_ratios (Sequence) – a sequence which represents several aspect_ratios for the current feature map. For 2D images, it is a Sequence of float aspect_ratios[j], anchor width and height w:h = 1:aspect_ratios[j]. For 3D images, it is a Sequence of 2 value Sequence aspect_ratios[j,0] and aspect_ratios[j,1], anchor width, height, and depth w:h:d = 1:aspect_ratios[j,0]:aspect_ratios[j,1] dtype (dtype) – target data type of the output

Tensor. device (Optional[device]) – target device to put the output Tensor data. Returns – For each s in scales, returns [s, s*aspect_ratios[j]] for 2D images, and [s, s*aspect_ratios[j,0],s*aspect_ratios[j,1]] for 3D images. Tensor Every combination of (a, (g, s), i) in (self.cell_anchors, zip(grid_sizes, strides), 0:spatial_dims) corresponds to a feature map. It outputs g[i] anchors that are s[i] distance apart in direction i, with the same dimensions as a. grid_sizes (List[List[int]]) – spatial size of the feature maps strides (List[List[Tensor]]) – strides of the feature maps regarding to the original image Example List[Tensor] Return number of anchor shapes for each feature map. Convert each element in self.cell_anchors to dtype and send to device. Module that generates anchors for a set of feature maps and image sizes, inherited from AnchorGenerator The module support computing anchors at multiple base anchor shapes per feature map. feature_map_scales should have the same number of elements with the number of feature maps. base_anchor_shapes can have an arbitrary number of elements. For 2D images, each element represents anchor width and height [w,h]. For 2D images, each element represents anchor width, height, and depth [w,h,d]. AnchorGenerator will output a set of len(base_anchor_shapes) anchors per spatial location for feature map i. feature_map_scales (Union[Sequence[int], Sequence[float]]) – scale of anchors for each feature map, i.e., each output level of the feature pyramid network (FPN). len(feature_map_scales) is the number of feature maps. scale[i]*base_anchor_shapes represents the anchor shapes for feature map i. base_anchor_shapes (Union[Sequence[Sequence[int]], Sequence[Sequence[float]]]) – a sequence which represents several anchor shapes for one feature map. For N-D images, it is a Sequence of N value Sequence. indexing (str) – choose from {‘xy’, ‘ij’}, optional Cartesian (‘xy’) or matrix (‘ij’, default) indexing of output. Cartesian (‘xy’) indexing swaps axis 0 and 1, which is the setting inside torchvision. matrix (‘ij’, default) indexing keeps the original axis not changed. See also indexing in https://pytorch.org/docs/stable/generated/torch.meshgrid.html Example Compute cell anchor shapes at multiple sizes and aspect ratios for the current feature map. anchor_shapes (Tensor) – [w, h] or [w, h, d], sized (N, spatial_dims), represents N anchor shapes for the current feature map. dtype (dtype) – target data type of the output Tensor. device (Optional[device]) – target device to put the output Tensor data. Tensor For 2D images, returns [-w/2, -h/2, w/2, h/2]; For 3D images, returns [-w/2, -h/2, -d/2, w/2, h/2, d/2]

Matcher#

The functions in this script are adapted from nnDetection, MIC-DKFZ/nnDetection which is adapted from torchvision. These are the changes compared with nndetection: 1) comments and docstrings; 2) reformat; 3) add a debug option to ATSSMatcher to help the users to tune parameters; 4) add a corner case return in ATSSMatcher.compute_matches; 5) add support for float16 cpu Compute matching based on ATSS https://arxiv.org/abs/1912.02424 Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection num_candidates (int) – number of positions to select candidates from. Smaller value will result in a higher matcher threshold and less matched candidates. similarity_fn (Callable[[Tensor, Tensor], Tensor]) – function for similarity computation between boxes and anchors center_in_gt (bool) – If False (default), matched anchor center points do not need to lie withing the ground truth box. Recommend False for small objects. If True, will result in a strict matcher and less matched candidates. debug (bool) – if True, will print the matcher threshold in order to tune num_candidates and center_in_gt. Compute matches according to ATTS for a single image Adapted from (sfzhang15/ATSS) boxes (Tensor) – bounding boxes, Nx4 or Nx6 torch tensor or

ndarray. The box mode is assumed to be StandardMode anchors (Tensor) – anchors to match Mx4 or Mx6, also assumed to be StandardMode. num_anchors_per_level (Sequence[int]) – number of anchors per feature pyramid level num_anchors_per_loc (int) – number of anchors per position Tuple[Tensor, Tensor] matrix which contains the similarity from each boxes to each anchor [N, M] vector which contains the matched box index for all anchors (if background BELOW_LOW_THRESHOLD is used and if it should be ignored BETWEEN_THRESHOLDS is used) [M] matrix which contains the similarity from each boxes to each anchor [N, M] vector which contains the matched box index for all anchors (if background BELOW_LOW_THRESHOLD is used and if it should be ignored BETWEEN_THRESHOLDS is used) [M] Note StandardMode = CornerCornerModeTypeA, also represented as “xyxy” ([xmin, ymin, xmax, ymax]) for 2D and “xyzxyz” ([xmin, ymin, zmin, xmax, ymax, zmax]) for 3D. Base class of Matcher, which matches boxes and anchors to each other similarity_fn (Callable[[Tensor, Tensor], Tensor]) – function for similarity computation between boxes and anchors Compute matches boxes (Tensor) – bounding boxes, Nx4 or Nx6 torch tensor or ndarray. The box mode is assumed to be StandardMode anchors (Tensor) – anchors to match Mx4 or Mx6, also assumed to be StandardMode. num_anchors_per_level (Sequence[int]) – number of anchors per feature pyramid level num_anchors_per_loc (int) – number of anchors per position Tuple[Tensor, Tensor] matrix which contains the similarity from each boxes to each anchor [N, M] vector which contains the matched box index for all anchors (if background BELOW_LOW_THRESHOLD is used and if it should be ignored BETWEEN_THRESHOLDS is used) [M] matrix which contains the similarity from each boxes to each anchor [N, M] vector which contains the matched box index for all anchors (if background BELOW_LOW_THRESHOLD is used and if it should be ignored BETWEEN_THRESHOLDS is used) [M]

Box coder#

This script is modified from torchvision to support N-D images, pytorch/vision This class encodes and decodes a set of bounding boxes into the representation used for training the regressors. weights (Tuple[float]) – 4-element tuple or 6-element tuple boxes_xform_clip (Optional[float]) – high threshold to prevent sending too large values into torch.exp() Example From a set of original reference_boxes and encoded relative box offsets, rel_codes (Tensor) – encoded boxes, Nx4 or Nx6 torch tensor. reference_boxes (Sequence[Tensor]) – a list of reference boxes, each element is Mx4 or Mx6 torch tensor. The box mode is assumed to be StandardMode Tensor decoded boxes, Nx1x4 or Nx1x6 torch tensor. The box mode will be StandardMode From a set of original boxes and encoded relative box offsets, rel_codes (Tensor) – encoded boxes, Nx(4*num_box_reg) or Nx(6*num_box_reg) torch tensor. reference_boxes (Tensor) – reference boxes, Nx4 or Nx6 torch tensor. The box mode is assumed to be StandardMode Tensor decoded boxes, Nx(4*num_box_reg) or Nx(6*num_box_reg) torch tensor. The box mode will to be StandardMode Encode a set of proposals with respect to some ground truth (gt) boxes. gt_boxes (Sequence[Tensor]) – list of gt boxes, Nx4 or Nx6 torch tensor. The box mode is assumed to be StandardMode proposals (Sequence[Tensor]) – list of boxes to be encoded, each element is Mx4 or Mx6 torch tensor. The box mode is assumed to be StandardMode Tuple[Tensor] A tuple of encoded gt, target of box regression that is used toconvert proposals into gt_boxes, Nx4 or Nx6 torch tensor. convert proposals into gt_boxes, Nx4 or Nx6 torch tensor. Encode proposals with respect to ground truth (gt) boxes. gt_boxes (Tensor) – gt boxes, Nx4 or Nx6 torch tensor. The box mode is assumed to be StandardMode proposals (Tensor) – boxes to be encoded, Nx4 or Nx6 torch tensor. The box mode is

assumed to be StandardMode Tensor encoded gt, target of box regression that is used to convert proposals into gt_boxes, Nx4 or Nx6 torch tensor. Encode a set of proposals with respect to some reference ground truth (gt) boxes. gt_boxes (Tensor) – gt boxes, Nx4 or Nx6 torch tensor. The box mode is assumed to be StandardMode proposals (Tensor) – boxes to be encoded, Nx4 or Nx6 torch tensor. The box mode is assumed to be StandardMode weights (Tensor) – the weights for (cx, cy, w, h) or (cx,cy,cz, w,h,d) Tensor encoded gt, target of box regression that is used to convert proposals into gt_boxes, Nx4 or Nx6 torch tensor.

Detection Utilities#

Validate the input dimensionality (raise a ValueError if invalid). input_images (Union[List[Tensor], Tensor]) – It can be 1) a tensor sized (B, C, H, W) or (B, C, H, W, D), or 2) a list of image tensors, each image i may have different size (C, H_i, W_i) or (C, H_i, W_i, D_i). spatial_dims (int) – number of spatial dimensions of the images, 2 or 3. None Validate the input images/targets during training (raise a ValueError if invalid). input_images (Union[List[Tensor], Tensor]) – It can be 1) a tensor sized (B, C, H, W) or (B, C, H, W, D), or 2) a list of image tensors, each image i may have different size (C, H_i, W_i) or (C, H_i, W_i, D_i). targets (Optional[List[Dict[str, Tensor]]]) – a list of dict. Each dict with two keys: target_box_key and target_label_key, ground-truth boxes present in the image. spatial_dims (int) – number of spatial dimensions of the images, 2 or 3. target_label_key (str) – the expected key of target labels. target_box_key (str) – the expected key of target boxes. None Pad the input images, so that the output spatial sizes are divisible by size_divisible. It pads them at the end to create a (B, C, H, W) or (B, C, H, W, D) Tensor. Padded size (H, W) or (H, W, D) is divisible by size_divisible. Default padding uses constant padding with value 0.0 input_images (Union[List[Tensor], Tensor]) – It can be 1) a tensor sized (B, C, H, W) or (B, C, H, W, D), or 2) a list of image tensors, each image i may have different size (C, H_i, W_i) or (C, H_i, W_i, D_i). spatial_dims (int) – number of spatial dimensions of the images, 2D or 3D. size_divisible (Union[int, Sequence[int]]) – int or Sequence[int], is the expected pattern on the input image shape. If an int, the same size_divisible will be applied to all the input spatial dimensions. mode (Union[PytorchPadMode, str]) – available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html kwargs – other arguments for torch.pad function. Tuple[Tensor, List[List[int]]] images, a (B, C, H, W) or (B, C, H, W, D) Tensor image_sizes, the original spatial size of each image images, a (B, C, H, W) or (B, C, H, W, D) Tensor image_sizes, the original spatial size of each image Preprocess the input images, including validate of the inputs pad the inputs so that the output spatial sizes are divisible by size_divisible. It pads them at the end to create a (B, C, H, W) or (B, C, H, W, D) Tensor. Padded size (H, W) or (H, W, D) is divisible by size_divisible. Default padding uses constant padding with value 0.0 input_images (Union[List[Tensor], Tensor]) – It can be 1) a tensor sized (B, C, H, W) or (B, C, H, W, D), or 2) a list of image tensors, each image i may have different size (C, H_i, W_i) or (C, H_i, W_i, D_i). spatial_dims (int) – number of spatial dimensions of the images, 2 or 3. size_divisible (Union[int, Sequence[int]]) – int or Sequence[int], is the expected pattern on the input image shape. If an int, the same size_divisible will be applied to all the input spatial dimensions. mode (Union[PytorchPadMode, str]) – available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html kwargs – other

arguments for torch.pad function. Tuple[Tensor, List[List[int]]] images, a (B, C, H, W) or (B, C, H, W, D) Tensor image_sizes, the original spatial size of each image images, a (B, C, H, W) or (B, C, H, W, D) Tensor image_sizes, the original spatial size of each image We expect the values in head_outputs: Dict[str, List[Tensor]] to have the same length. Will raise ValueError if not. head_outputs (Dict[str, List[Tensor]]) – a Dict[str, List[Tensor]] or Dict[str, Tensor] keys (Optional[List[str]]) – the keys in head_output that need to have values (List) with same length. If not provided, will use head_outputs.keys(). None An in-place function. We expect head_outputs to be Dict[str, List[Tensor]]. Yet if it is Dict[str, Tensor], this func converts it to Dict[str, List[Tensor]]. It will be modified in-place. head_outputs (Dict[str, List[Tensor]]) – a Dict[str, List[Tensor]] or Dict[str, Tensor], will be modifier in-place keys (Optional[List[str]]) – the keys in head_output that need to have value type List[Tensor]. If not provided, will use head_outputs.keys(). None Predict network dict output with an inferer. Compared with directly output network(images), it enables a sliding window inferer that can be used to handle large inputs. images (Tensor) – input of the network, Tensor sized (B, C, H, W) or (B, C, H, W, D) network – a network that takes an image Tensor sized (B, C, H, W) or (B, C, H, W, D) as input and outputs a dictionary Dict[str, List[Tensor]] or Dict[str, Tensor]. keys (List[str]) – the keys in the output dict, should be network output keys or a subset of them. inferer (Optional[SlidingWindowInferer]) – a SlidingWindowInferer to handle large inputs. Dict[str, List[Tensor]] The predicted head_output from network, a Dict[str, List[Tensor]] Example

Inference box selector#

Part of this script is adapted from pytorch/vision Box selector which selects the predicted boxes. The box selection is performed with the following steps: For each level, discard boxes with scores less than self.score_thresh. For each level, keep boxes with top self.topk_candidates_per_level scores. For the whole image, perform non-maximum suppression (NMS) on boxes, with overlapping threshold nms_thresh. For the whole image, keep boxes with top self.detections_per_img scores. apply_sigmoid (bool) – whether to apply sigmoid to get scores from classification logits score_thresh (float) – no box with scores less than score_thresh will be kept topk_candidates_per_level (int) – max number of boxes to keep for each level nms_thresh (float) – box overlapping threshold for NMS detections_per_img (int) – max number of boxes to keep for each image Example Postprocessing to generate detection result from classification logits and boxes. The box selection is performed with the following steps: For each level, discard boxes with scores less than self.score_thresh. For each level, keep boxes with top self.topk_candidates_per_level scores. For the whole image, perform non-maximum suppression (NMS) on boxes, with overlapping threshold nms_thresh. For the whole image, keep boxes with top self.detections_per_img scores. boxes_list (List[Tensor]) – list of predicted boxes from a single image, each element i is a Tensor sized (N_i, 2*spatial_dims) logits_list (List[Tensor]) – list of predicted classification logits from a single image, each element i is a Tensor sized (N_i, num_classes) spatial_size (Union[List[int], Tuple[int]]) – spatial size of the image Tuple[Tensor, Tensor, Tensor] selected boxes, Tensor sized (P, 2*spatial_dims) selected_scores, Tensor sized (P,) selected_labels, Tensor sized (P,) selected boxes, Tensor sized (P, 2*spatial_dims) selected_scores, Tensor sized (P,) selected_labels, Tensor sized (P,) Select indices with highest scores. The indices selection is performed with the following steps: If self.apply_sigmoid, get scores by applying sigmoid to logits. Otherwise, use logits as scores. Discard indices with scores less than self.score_thresh Keep indices with top

self.topk_candidates_per_level scores logits (Tensor) – predicted classification logits, Tensor sized (N, num_classes) selected M indices, Tensor sized (M,) - selected_scores: selected M scores, Tensor sized (M,) - selected_labels: selected M labels, Tensor sized (M,) topk_idxs topk_idxs

Detection metrics#

This script is almost same with MIC-DKFZ/nnDetection The changes include 1) code reformatting, 2) docstrings. This script is almost same with MIC-DKFZ/nnDetection The changes include 1) code reformatting, 2) docstrings, 3) allow input args gt_ignore to be optional. (If so, no GT boxes will be ignored.) Match boxes of a batch to corresponding ground truth for each category independently. iou_fn (Callable[[ndarray, ndarray], ndarray]) – compute overlap for each pair iou_thresholds (Sequence[float]) – defined which IoU thresholds should be evaluated pred_boxes (Sequence[ndarray]) – predicted boxes from single batch; List[[D, dim * 2]], D number of predictions pred_classes (Sequence[ndarray]) – predicted classes from a single batch; List[[D]], D number of predictions pred_scores (Sequence[ndarray]) – predicted score for each bounding box; List[[D]], D number of predictions gt_boxes (Sequence[ndarray]) – ground truth boxes; List[[G, dim * 2]], G number of ground truth gt_classes (Sequence[ndarray]) – ground truth classes; List[[G]], G number of ground truth gt_ignore (Union[Sequence[Sequence[bool]], Sequence[ndarray], None]) – specified if which ground truth boxes are not counted as true positives. If not given, when use all the gt_boxes. (detections which match theses boxes are not counted as false positives either); List[[G]], G number of ground truth max_detections (int) – maximum number of detections which should be evaluated List[Dict[int, Dict[str, ndarray]]] List[Dict[int, Dict[str, np.ndarray]]], each Dict[str, np.ndarray] corresponds to an image. Dict has the following keys. dtMatches: matched detections [T, D], where T = number of thresholds, D = number of detections gtMatches: matched ground truth boxes [T, G], where T = number of thresholds, G = number of ground truth dtScores: prediction scores [D] detection scores gtIgnore: ground truth boxes which should be ignored [G] indicate whether ground truth should be ignored dtIgnore: detections which should be ignored [T, D], indicate which detections should be ignored List[Dict[int, Dict[str, np.ndarray]]], each Dict[str, np.ndarray] corresponds to an image. Dict has the following keys. dtMatches: matched detections [T, D], where T = number of thresholds, D = number of detections gtMatches: matched ground truth boxes [T, G], where T = number of thresholds, G = number of ground truth dtScores: prediction scores [D] detection scores gtIgnore: ground truth boxes which should be ignored [G] indicate whether ground truth should be ignored dtIgnore: detections which should be ignored [T, D], indicate which detections should be ignored Example

FastMRIReader#

Load fastMRI files with '.h5' suffix. fastMRI files, when loaded with "h5py", are HDF5 dictionary-like datasets. The keys are: kspace: contains the fully-sampled kspace is the ground-truth image. It also has several attributes with the following keys: acquisition (str): acquisition mode of the data (e.g., AXT2 denotes T2 brain MRI scans) max (float): dynamic range of the data norm (float): norm of the kspace patient_id (str): the patient's id whose measurements were recorded Extract data array and metadata from the loaded data and return them. This function returns two objects, first is numpy array of image data, second is dict of metadata. dat (Dict) – a dictionary loaded from an h5 file Tuple[ndarray, dict] Read data from specified h5 file.

Note that the returned object is a dictionary. data (Union[Sequence[Union[str, PathLike]], str, PathLike]) – file name to read. Dict Verify whether the specified file format is supported by h5py reader. filename (Union[Sequence[Union[str, PathLike]], str, PathLike]) – file name bool

ConvertToTensorComplex#

Convert complex-valued data to a 2-channel PyTorch tensor. The real and imaginary parts are stacked along the last dimension. This function relies on 'monai.utils.type_conversion.convert_to_tensor' data – input data can be PyTorch Tensor, numpy array, list, int, and float. will convert Tensor, Numpy array, float, int, bool to Tensor, strings and objects keep the original. for list, convert every item to a Tensor if applicable. dtype (Optional[dtype]) – target data type to when converting to Tensor. device (Optional[device]) – target device to put the converted Tensor data. wrap_sequence (bool) – if False, then lists will recursively call this function. E.g., [1, 2] -> [tensor(1), tensor(2)]. If True, then [1, 2] -> tensor([1, 2]). track_meta (bool) – whether to track the meta information, if True, will convert to MetaTensor. default to False. Tensor PyTorch version of the data Example

ComplexAbs#

Compute the absolute value of a complex array. x (Union[ndarray, Tensor]) – Input array/tensor with 2 channels in the last dimension representing real and imaginary parts. Union[ndarray, Tensor] Absolute value along the last dimension Example

RootSumOfSquares#

Compute the root sum of squares (rss) of the data (typically done for multi-coil MRI samples) x (Union[ndarray, Tensor]) – Input array/tensor spatial_dim (int) – dimension along which rss is applied Union[ndarray, Tensor] rss of x along spatial_dim Example

ComplexMul#

Compute complex-valued multiplication. Supports Ndim inputs with last dim equal to 2 (real/imaginary channels) x (Union[ndarray, Tensor]) – Input array/tensor with 2 channels in the last dimension representing real and imaginary parts. y (Union[ndarray, Tensor]) – Input array/tensor with 2 channels in the last dimension representing real and imaginary parts. Union[ndarray, Tensor] Complex multiplication of x and y Example

ComplexConj#

Compute complex conjugate of an/a array/tensor. Supports Ndim inputs with last dim equal to 2 (real/imaginary channels) x (Union[ndarray, Tensor]) – Input array/tensor with 2 channels in the last dimension representing real and imaginary parts. Union[ndarray, Tensor] Complex conjugate of x Example

auto3dseg#

The base class of Ensemble methods ensemble the results using either “mean” or “vote” method preds – a list of probability prediction in Tensor-Like format. sigmoid – use the sigmoid function to threshold probability one-hot map, otherwise argmax is used. Defaults to False a tensor which is the ensembled prediction. Get a model by identifier. identifier – the name of the bundleAlgo Get the algo ensemble after ranking or a empty list if ranking was not started. A list of Algo Register model in the ensemble Set the files to perform model inference. dataroot (str) – the path of the files data_src_cfg_file – the data source file path Ensemble method that select the best models that are the tops in each fold. n_fold (int) – number of cross-validation folds used in training Rank the algos by finding the best model in each cross-validation fold None Ensemble method that select N model out of all using the models’ best_metric scores n_best (int) – number of models to pick for ensemble (N). Rank the algos by finding the top N (n_best) validation scores. Sort the best_metrics Build ensemble workflow from configs and arguments. history (Sequence[Dict]) – a collection of trained bundleAlgo algorithms. data_src_cfg_filename (Optional[str]) – filename of the data source. Examples Add model inferer to the builder. identifier (str) – name of the bundleAlgo. gen_algo (BundleAlgo) – a trained BundleAlgo model object. best_metric (Optional[float]) – the best metric in validation of the trained model. Get the ensemble Set the ensemble method. ensemble (AlgoEnsemble) – the AlgoEnsemble to build. An interface for handling Auto3Dseg with minimal inputs and understanding of the internal states in Auto3Dseg. The users can run the Auto3Dseg with default settings in one line of code. They can also customize the advanced features Auto3Dseg in a few additional lines. Examples of customization include change cross-validation folds change training/prediction parameters change ensemble methods automatic hyperparameter optimization. The output of the interface is a directory that contains data statistics analysis report algorithm definition files (scripts, configs, pickle objects) and training results (checkpoints, accuracies) the predictions on the testing datasets from the final algorithm ensemble a copy of the input arguments in form of YAML cached intermediate results work_dir (str) – working directory to save the intermediate and final results. input (Union[Dict[str, Any], str, None]) – the configuration dictionary or the file path to the configuration in form of YAML. The configuration should contain datalist, dataroot, modality, multigpu, and class_names info. algos (Union[Dict, List, str, None]) – optionally specify algorithms to use. If a dictionary, must be in the form {“algname”: dict(_target_=”algname.scripts.algo.AlgnameAlgo”, template_path=”algname”), …} If a list or a string, defines a subset of names of the algorithms to use, e.g. ‘segresnet’ or [‘segresnet’, ‘dints’] out of the full set of algorithm templates provided by templates_path_or_url. Defaults to None, to use all available algorithms. analyze (Optional[bool]) – on/off switch to run DataAnalyzer and generate a datastats report. Defaults to None, to automatically decide based on cache, and run data analysis only if we have not completed this step yet. algo_gen (Optional[bool]) – on/off switch to run AlgoGen and generate templated BundleAlgos. Defaults to None, to automatically decide based on cache, and run algorithm folders generation only if we have not completed this step yet. train (Optional[bool]) – on/off switch to run training and generate algorithm checkpoints. Defaults to None, to automatically decide based on cache, and run training only if we have not completed this step yet. hpo (bool) – use hyperparameter optimization (HPO) in the training phase. Users can provide a list of hyper-parameter and a search will be performed to investigate the algorithm performances. hpo_backend (str) – a string that indicates the backend of the HPO. Currently, only NNI Grid-search mode is supported ensemble (bool) – on/off switch to run model ensemble and use the ensemble to predict outputs in testing datasets. not_use_cache (bool) – if the value is True, it will ignore all cached results in

data analysis, algorithm generation, or training, and start the pipeline from scratch. templates_path_or_url (Optional[str]) – the folder with the algorithm templates or a url. If None provided, the default template zip url will be downloaded and extracted into the work_dir. kwargs – image writing parameters for the ensemble inference. The kwargs format follows the SaveImage transform. For more information, check https://docs.monai.io/en/stable/transforms.html#saveimage. Examples User can use the one-liner to start the Auto3Dseg workflow User can also save the input dictionary as a input YAML file and use the following one-liner User can specify work_dir and data source config input and run AutoRunner: User can specify a subset of algorithms to use and run AutoRunner: User can specify a a local folder with algorithms templates and run AutoRunner: User can specify training parameters by: User can specify the fold number of cross validation User can specify the prediction parameters during algo ensemble inference: User can define a grid search space and use the HPO during training. Notes Expected results in the work_dir as below: Save the cache state as cache.yaml in the working directory Check if the intermediate result is cached after each step in the current working directory a dict of cache results. If not_use_cache is set to True, or there is no cache file in the working directory, the result will be empty_cache in which all has_cache keys are set to False. Run the AutoRunner pipeline Set the data analysis extra params. params (Optional[Dict[str, Any]]) – a dict that defines the overriding key-value pairs during training. The overriding method is defined by the algo class. Examples {“num_iterations”: 8, “num_iterations_per_validation”: 4} Set the bundle ensemble method ensemble_method_name (str) – the name of the ensemble method. Only two methods are supported “AlgoEnsembleBestN” and “AlgoEnsembleBestByFold”. kwargs – the keyword arguments used to define the ensemble method. Currently only n_best for AlgoEnsembleBestN is supported. Set options for GPU-based parameter customization/optimization. gpu_customization (bool) – the switch to determine automatically customize/optimize bundle script/config parameters for each bundleAlgo based on gpus. Custom parameters are obtained through dummy training to simulate the actual model training process and hyperparameter optimization (HPO) experiments. gpu_customization_specs (optinal) – the dictionary to enable users overwrite the HPO settings. user can overwrite part of variables as follows or all of them. The structure is as follows. gpu_customization_specs = { 'ALOG': { 'num_trials': 6, 'range_num_images_per_batch': [1, 20], 'range_num_sw_batch_size': [1, 20] } } the dictionary to enable users overwrite the HPO settings. user can overwrite part of variables as follows or all of them. The structure is as follows. ALGO – the name of algorithm. It could be one of algorithm names (e.g., ‘dints’) or ‘unversal’ which would apply changes to all algorithms. Possible options are {"unversal", "dints", "segresnet", "segresnet2d", "swinunetr"}. the name of algorithm. It could be one of algorithm names (e.g., ‘dints’) or ‘unversal’ which would apply changes to all algorithms. Possible options are {"unversal", "dints", "segresnet", "segresnet2d", "swinunetr"}. num_trials – the number of HPO trials/experiments to run. range_num_images_per_batch – the range of number of images per mini-batch. range_num_sw_batch_size – the range of batch size in sliding-window inferer. Set parameters for the HPO module and the algos before the training. It will attempt to (1) override bundle templates with the key-value pairs in params (2) change the config of the HPO module (e.g. NNI) if the key is found to be one of: “trialCodeDirectory” “trialGpuNumber” “trialConcurrency” “maxTrialNumber” “maxExperimentDuration” “tuner” “trainingService” and (3) enable the dry-run mode if the user would generate the NNI configs without starting the NNI service. params (Optional[Dict[str, Any]]) – a dict that defines the overriding key-value pairs during instantiation of the algo. For BundleAlgo, it will override the template config filling. Notes Users can set nni_dry_run to True in the params to enable the dry-run mode for the NNI backend. Set the

ensemble output transform. kwargs – image writing parameters for the ensemble inference. The kwargs format follows SaveImage transform. For more information, check https://docs.monai.io/en/stable/transforms.html#saveimage . Set the search space for NNI parameter search. search_space – hyper parameter search space in the form of dict. For more information, please check NNI documentation: https://nni.readthedocs.io/en/v2.2/Tutorial/SearchSpaceSpec.html . Set the number of cross validation folds for all algos. num_fold (int) – a positive integer to define the number of folds. Notes If the ensemble method is AlgoEnsembleBestByFold, this function automatically updates the n_fold parameter in the ensemble_method to avoid inconsistency between the training and the ensemble. Set the prediction params for all algos. params (Optional[Dict[str, Any]]) – a dict that defines the overriding key-value pairs during prediction. The overriding method is defined by the algo class. Examples two files in the testing datalist {“file_slices”: slice(0, 2)} Set the training params for all algos. params (Optional[Dict[str, Any]]) – a dict that defines the overriding key-value pairs during training. The overriding method is defined by the algo class. Examples {“num_iterations”: 8, “num_iterations_per_validation”: 4} An algorithm represented by a set of bundle configurations and scripts. BundleAlgo.cfg is a monai.bundle.ConfigParser instance. This class creates MONAI bundles from a directory of ‘bundle template’. Different from the regular MONAI bundle format, the bundle template may contain placeholders that must be filled using fill_template_config during export_to_disk. Then created bundle keeps the same file structure as the template. Create an Algo instance based on the predefined Algo template. template_path (str) – path to the root of the algo template. Fill the configuration templates, write the bundle (configs + scripts) to folder output_path/algo_name. output_path (str) – Path to export the ‘scripts’ and ‘configs’ directories. algo_name (str) – the identifier of the algorithm (usually contains the name and extra info like fold ID). kwargs – other parameters, including: “copy_dirs=True/False” means whether to copy the template as output instead of inplace operation, “fill_template=True/False” means whether to fill the placeholders in the template. other parameters are for fill_template_config function. The configuration files defined when constructing this Algo instance might not have a complete training and validation pipelines. Some configuration components and hyperparameters of the pipelines depend on the training data and other factors. This API is provided to allow the creation of fully functioning config files. Return the records of filling template config: {“”: {“”: value, …}, …}. data_stats_filename (str) – filename of the data stats report (generated by DataAnalyzer) Notes Template filling is optional. The user can construct a set of pre-filled configs without replacing values by using the data analysis results. It is also intended to be re-implemented in subclasses of BundleAlgo if the user wants their own way of auto-configured template filling. dict Load the InferClass from the infer.py. The InferClass should be defined in the template under the path of “scripts/infer.py”. It is required to define the “InferClass” (name is fixed) with two functions at least (__init__ and infer). The init class has an override kwargs that can be used to override parameters in the run-time optionally. Examples: Returns the algo output paths to find the algo scripts and configs. Returns validation scores of the model trained by the current Algo. Use the trained model to predict the outputs with a given input image. predict_files (list) – a list of paths to files to run inference on [“path_to_image_1”, “path_to_image_2”] predict_params – a dict to override the parameters in the bundle config (including the files to predict). Set the data source configuration file data_src_cfg (str) – path to a configuration file (yaml) that contains datalist, dataroot, and other params. The config will be in a form of {“modality”: “ct”, “datalist”: “path_to_json_datalist”, “dataroot”: “path_dir_data”} Set the data analysis report (generated by DataAnalyzer). data_stats_files (str) – path to the datastats yaml file Load the run function in the training script of each model. Training parameter is

predefined by the algo_config.yaml file, which is pre-filled by the fill_template_config function in the same instance. train_params – to specify the devices using a list of integers: {"CUDA_VISIBLE_DEVICES": [1,2,3]}. This class generates a set of bundles according to the cross-validation folds, each of them can run independently. algo_path (str) – the directory path to save the algorithm templates. Default is the current working dir. algos (Union[Dict, List, str, None]) – If dictionary, it outlines the algorithm to use. If a list or a string, defines a subset of names of the algorithms to use, e.g. ('segresnet', 'dints') out of the full set of algorithm templates provided by templates_path_or_url. Defaults to None - to use all available algorithms. templates_path_or_url (Optional[str]) – the folder with the algorithm templates or a url. If None provided, the default template zip url will be downloaded and extracted into the algo_path. The current default options are released at: Project-MONAI/research-contributions data_stats_filename (Optional[str]) – the path to the data stats file (generated by DataAnalyzer) data_src_cfg_name (Optional[str]) – the path to the data source config YAML file. The config will be in a form of {"modality": "ct", "datalist": "path_to_json_datalist", "dataroot": "path_dir_data"} Generate the bundle scripts/configs for each bundleAlgo output_folder – the output folder to save each algorithm. num_fold (int) – the number of cross validation fold. gpu_customization (bool) – the switch to determine automatically customize/optimize bundle script/config parameters for each bundleAlgo based on gpus. Custom parameters are obtained through dummy training to simulate the actual model training process and hyperparameter optimization (HPO) experiments. gpu_customization_specs (optinal) – the dictionary to enable users overwrite the HPO settings. user can overwrite part of variables as follows or all of them. The structure is as follows. gpu_customization_specs = { 'ALOG': { 'num_trials': 6, 'range_num_images_per_batch': [1, 20], 'range_num_sw_batch_size': [1, 20] } } the dictionary to enable users overwrite the HPO settings. user can overwrite part of variables as follows or all of them. The structure is as follows. ALGO – the name of algorithm. It could be one of algorithm names (e.g., 'dints') or 'unversal' which would apply changes to all algorithms. Possible options are {"unversal", "dints", "segresnet", "segresnet2d", "swinunetr"}. the name of algorithm. It could be one of algorithm names (e.g., 'dints') or 'unversal' which would apply changes to all algorithms. Possible options are {"unversal", "dints", "segresnet", "segresnet2d", "swinunetr"}. num_trials – the number of HPO trials/experiments to run. range_num_images_per_batch – the range of number of images per mini-batch. range_num_sw_batch_size – the range of batch size in sliding-window inferer. Get the data source filename Get the filename of the data stats get the history of the bundleAlgo object with their names/identifiers List Set the data source filename data_src_cfg_filename – filename of data_source file Set the data stats filename data_stats_filename (str) – filename of datastats The DataAnalyzer automatically analyzes given medical image dataset and reports the statistics. The module expects file paths to the image data and utilizes the LoadImaged transform to read the files, which supports nii, nii.gz, png, jpg, bmp, npz, npy, and dcm formats. Currently, only segmentation task is supported, so the user needs to provide paths to the image and label files (if have). Also, label data format is preferred to be (1,H,W,D), with the label index in the first dimension. If it is in onehot format, it will be converted to the preferred format. datalist (Union[str, Dict]) – a Python dictionary storing group, fold, and other information of the medical image dataset, or a string to the JSON file storing the dictionary. dataroot (str) – user's local directory containing the datasets. output_path (str) – path to save the analysis result. average (bool) – whether to average the statistical value across different image modalities. do_ccp (bool) – apply the connected component algorithm to process the labels/images device (Union[str, device]) – a string specifying hardware (CUDA/CPU) utilized for the operations.

worker (int) – number of workers to use for parallel processing. If device is cuda/GPU, worker has to be 0. image_key (str) – a string that user specify for the image. The DataAnalyzer will look it up in the datalist to locate the image files of the dataset. label_key (Optional[str]) – a string that user specify for the label. The DataAnalyzer will look it up in the datalist to locate the label files of the dataset. If label_key is NoneType or “None”, the DataAnalyzer will skip looking for labels and all label-related operations. hist_bins (Union[list, int, None]) – bins to compute histogram for each image channel. hist_range (Optional[list]) – ranges to compute histogram for each image channel. fmt (Optional[str]) – format used to save the analysis results. Defaults to “yaml”. histogram_only (bool) – whether to only compute histograms. Defaults to False. extra_params – other optional arguments. Currently supported arguments are : ‘allowed_shape_difference’ (default 5) can be used to change the default tolerance of the allowed shape differences between the image and label items. In case of shape mismatch below the tolerance, the label image will be resized to match the image using nearest interpolation. Examples Notes The module can also be called from the command line interface (CLI). For example: Get all case stats. Caller of the DataAnalyser class. The function iterates datalist and call get_case_stats to generate stats. Then get_case_summary is called to combine results. key – dataset key transform_list – option list of transforms before SegSummarizer A data statistics dictionary containing”stats_summary” (summary statistics of the entire datasets). Within stats_summary there are “image_stats” (summarizing info of shape, channel, spacing, and etc using operations_summary), “image_foreground_stats” (info of the intensity for the non-zero labeled voxels), and “label_stats” (info of the labels, pixel percentage, image_intensity, and each individual label in a list) “stats_by_cases” (List type value. Each element of the list is statistics of an image-label info. Within each element, there are: “image” (value is the path to an image), “label” (value is the path to the corresponding label), “image_stats” (summarizing info of shape, channel, spacing, and etc using operations), “image_foreground_stats” (similar to the previous one but one foreground image), and “label_stats” (stats of the individual labels) ”stats_summary” (summary statistics of the entire datasets). Within stats_summary there are “image_stats” (summarizing info of shape, channel, spacing, and etc using operations_summary), “image_foreground_stats” (info of the intensity for the non-zero labeled voxels), and “label_stats” (info of the labels, pixel percentage, image_intensity, and each individual label in a list) “stats_by_cases” (List type value. Each element of the list is statistics of an image-label info. Within each element, there are: “image” (value is the path to an image), “label” (value is the path to the corresponding label), “image_stats” (summarizing info of shape, channel, spacing, and etc using operations), “image_foreground_stats” (similar to the previous one but one foreground image), and “label_stats” (stats of the individual labels) Notes Since the backend of the statistics computation are torch/numpy, nan/inf value may be generated and carried over in the computation. In such cases, the output dictionary will include .nan/.inf in the statistics. Generate algorithms for the NNI to automate hyperparameter tuning. The module has two major interfaces: __init__ which prints out how to set up the NNI, and a trialCommand function run_algo for the NNI library to start the trial of the algo. More about trialCommand function can be found in trail code section in NNI webpage https://nni.readthedocs.io/en/latest/tutorials/hpo_quickstart_pytorch/main.html . algo (Optional[Algo]) – an Algo object (e.g. BundleAlgo) with defined methods: get_output_path and train and supports saving to and loading from pickle files via algo_from_pickle and algo_to_pickle. params – a set of parameter to override the algo if override is supported by Algo subclass. Examples: Notes The NNIGen will prepare the algorithms in a folder and suggest a command to replace trialCommand in the experiment config. However, NNIGen will not trigger NNI. User needs to write their

NNI experiment configs, and then run the NNI command manually. Generate the record for each Algo. If it is a BundleAlgo, it will generate the config files. output_folder (str) – the directory nni will save the results to. None Get parameter for next round of training from NNI server. Return the filename of the dumped pickle algo object. Get the identifier of the current experiment. In the format of listing the searching parameter name and values connected by underscore in the file name. Print how to write the trial commands for Bundle Algo. The python interface for NNI to run. obj_filename (str) – the pickle-exported Algo object. output_folder (str) – the root path of the algorithms templates. template_path – the algorithm_template. It must contain algo.py in the follow path: {algorithm_templates_dir}/{network}/scripts/algo.py None Report the acc to NNI server. Translate the parameter from monai bundle to meet NNI requirements. params (dict) – a dict of parameters. Generate algorithms for the Optuna to automate hyperparameter tuning. Please refer to NNI and Optuna (https://optuna.readthedocs.io/en/stable/) for more information. Optuna has different running scheme compared to NNI. The hyperparameter samples come from a trial object (trial.suggest…) created by Optuna, so OptunaGen needs to accept this trial object as input. Meanwhile, Optuna calls OptunaGen, thus OptunaGen.__call__() should return the accuracy. Use functools.partial to wrap OptunaGen for addition input arguments. algo (Optional[Algo]) – an Algo object (e.g. BundleAlgo). The object must at least define two methods: get_output_path and train and supports saving to and loading from pickle files via algo_from_pickle and algo_to_pickle. params – a set of parameter to override the algo if override is supported by Algo subclass. Examples: Notes Different from NNI and NNIGen, OptunaGen and Optuna can be ran within the Python process. Generate the record for each Algo. If it is a BundleAlgo, it will generate the config files. output_folder (str) – the directory nni will save the results to. None Get parameter for next round of training from optuna trial object. This function requires user rewrite during usage for different search space. Return the dumped pickle object of algo. Get the identifier of the current experiment. In the format of listing the searching parameter name and values connected by underscore in the file name. The python interface for NNI to run. obj_filename (str) – the pickle-exported Algo object. output_folder (str) – the root path of the algorithms templates. template_path – the algorithm_template. It must contain algo.py in the follow path: {algorithm_templates_dir}/{network}/scripts/algo.py None Set the accuracy score Set the Optuna trial Translate the parameter from monai bundle. params (dict) – a dict of parameters. Save all the BundleAlgo in the history to algo_object.pkl in each individual folder history (List[Dict[str, BundleAlgo]]) – a List of Bundle. Typically, the history can be obtained from BundleGen get_history method import the history of the bundleAlgo object with their names/identifiers output_folder (str) – the root path of the algorithms templates. template_path (Optional[str]) – the algorithm_template. It must contain algo.py in the follow path: {algorithm_templates_dir}/{network}/scripts/algo.py. only_trained (bool) – only read the algo history if the algo is trained. List previous API Reference next Auto3dseg