FROC#

This function is modified from the official evaluation code of CAMELYON 16 Challenge, and used to distinguish true positive and false positive predictions. A true positive prediction is defined when the detection point is within the annotated ground truth region. probs (Union[ndarray, Tensor]) – an array with shape (n,) that represents the probabilities of the detections. Where, n is the number of predicted detections. y_coord (Union[ndarray, Tensor]) – an array with shape (n,) that represents the Y-coordinates of the detections. x_coord (Union[ndarray, Tensor]) – an array with shape (n,) that represents the X-coordinates of the detections. evaluation_mask (Union[ndarray, Tensor]) – the ground truth mask for evaluation. labels_to_exclude (Optional[List]) – labels in this list will not be counted for metric calculation. resolution_level (int) – the level at which the evaluation mask is made. an array that contains the probabilities of the false positive detections. tp_probs: an array that contains the probabilities of the True positive detections. num_targets: the total number of targets (excluding labels_to_exclude) for all images under evaluation. fp_probs This function is modified from the official evaluation code of CAMELYON 16 Challenge, and used to compute the required data for plotting the Free Response Operating Characteristic (FROC) curve. fp_probs (Union[ndarray, Tensor]) – an array that contains the probabilities of the false positive detections for all images under evaluation. tp_probs (Union[ndarray, Tensor]) – an array that contains the probabilities of the True positive detections for all images under evaluation. num_targets (int) – the total number of targets (excluding labels_to_exclude) for all images under evaluation. num_images (int) – the number of images under evaluation. This function is modified from the official evaluation code of CAMELYON 16 Challenge, and used to compute the challenge’s second evaluation metric, which is defined as the average sensitivity at the predefined false positive rates per whole slide image. fps_per_image (ndarray) – the average number of false positives per image for different thresholds. total_sensitivity (ndarray) – sensitivities (true positive rates) for different thresholds. 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.

Metric#

Base class for metric computation for evaluating the performance of a model. __call__ is designed to execute the computation.

Variance#

y_pred (Tensor) – [N, C, H, W, D] or [N, C, H, W] or [N, C, H] where N is repeats, C is channels and H, W, D stand for Height, Width & Depth include_background (bool) – Whether to include the background of the spatial image or channel 0 of the 1-D vector spatial_map (bool) – Boolean, if set to True, spatial map of variance will be returned corresponding to i/p image dimensions scalar_reduction (str) – reduction type of the metric, either ‘sum’ or ‘mean’ can be used threshold (float) – To avoid NaN’s a threshold is used to replace zero’s A single scalar uncertainty/variance value or the spatial map of uncertainty/variance Compute the Variance of a given T-repeats N-dimensional array/tensor. The primary usage is as an uncertainty based metric for Active Learning. It can return the spatial variance/uncertainty map based on user choice or a single scalar value via mean/sum of the variance for scoring purposes include_background (bool) – Whether to include the background of the spatial image or channel 0 of the 1-D vector spatial_map (bool) – Boolean, if set to True, spatial map of variance will be returned corresponding to i/p image dimensions scalar_reduction (str) – reduction type of the metric, either ‘sum’ or ‘mean’ can be used threshold (float) – To avoid NaN’s a threshold is used to replace zero’s

LabelQualityScore#

The assumption is that the DL model makes better predictions than the provided label quality, hence the difference can be treated as a label quality score y_pred (Tensor) – Input data of dimension [B, C, H, W, D] or [B, C, H, W] or [B, C, H] where B is Batch-size, C is channels and H, W, D stand for Height, Width & Depth y (Tensor) – Ground Truth of dimension [B, C, H, W, D] or [B, C, H, W] or [B, C, H] where B is Batch-size, C is channels and H, W, D stand for Height, Width & Depth include_background (bool) – Whether to include the background of the spatial image or channel 0 of the 1-D vector scalar_reduction (str) – reduction type of the metric, either ‘sum’ or ‘mean’ can be used to retrieve a single scalar value, if set to ‘none’ a spatial map will be returned A single scalar absolute difference value as score with a reduction based on sum/mean or the spatial map of absolute difference The assumption is that the DL model makes better predictions than the provided label quality, hence the difference can be treated as a label quality score It can be combined with variance/uncertainty for active learning frameworks to factor in the quality of label along with uncertainty :type include_background: bool :param include_background: Whether to include the background of the spatial image or channel 0 of the 1-D vector :param spatial_map: Boolean, if set to True, spatial map of variance will be returned corresponding to i/p image :param dimensions: :type scalar_reduction: str :param scalar_reduction: reduction type of the metric, either ‘sum’ or ‘mean’ can be used

IterationMetric#

Base class for metrics computation at the iteration level, that is, on a min-batch of samples usually using the model outcome of one iteration. __call__ is designed to handle y_pred and y (optional) in torch tensors or a list/tuple of tensors. Subclasses typically implement the _compute_tensor function for the actual tensor computation logic.

Cumulative#

Utility class for the typical cumulative computation process based on PyTorch Tensors. It provides interfaces to accumulate values in the local buffers, synchronize buffers across distributed nodes, and aggregate the buffered values. In multi-processing, PyTorch programs usually distribute data to multiple nodes. Each node runs with a subset of the data, adds values to its local buffers. Calling get_buffer could gather all the results and aggregate can further handle the results to generate the final outcomes. Users can implement their own aggregate method to handle the results, using get_buffer to get the buffered contents. Note: the data list should have the same length every time calling add() in a round, it will automatically create buffers according to the length of data list. Typically, this class is expected to execute the following steps: The following is an example of maintaining two internal buffers: The following is an example of extending with variable length data: Initialize the internal buffers. self._buffers are local buffers, they are not usually used directly. self._sync_buffers are the buffers with all the results across all the nodes. Aggregate final results based on the gathered buffers. This method is expected to use get_buffer to gather the local buffer contents. Add samples to the local cumulative buffers. A buffer will be allocated for each data item. Compared with self.extend, this method adds a single sample (instead of a “batch”) to the local buffers. data – each item will be converted into a torch tensor. they will be stacked at the 0-th dim with a new dimension when get_buffer() is called. None Extend the local buffers with new (“batch-first”) data. A buffer will be allocated for each data item. Compared with self.append, this method adds a “batch” of data to the local buffers. data – each item can be a “batch-first” tensor or a list of “channel-first” tensors. they will be concatenated at the 0-th dimension when get_buffer() is called. None Get the synchronized list of buffers. A typical usage is to generate the metrics report based on the raw metric details. Each buffer is a PyTorch Tensor. Reset the buffers for cumulative tensors and the synced results.

CumulativeIterationMetric#

Base class of cumulative metric which collects metrics on each mini-batch data at the iteration level. Typically, it computes some intermediate results for each iteration, adds them to the buffers, then the buffer contents could be gathered and aggregated for the final result when epoch completed. Currently,``Cumulative.aggregate()`` and IterationMetric._compute_tensor() are expected to be implemented. For example, MeanDice inherits this class and the usage is as follows: And to load predictions and labels from files, then compute metrics with multi-processing, please refer to: Project-MONAI/tutorials.

LossMetric#

A wrapper to make loss_fn available as a cumulative metric. That is, the loss values computed from mini-batches can be combined in the reduction mode across multiple iterations, as a quantitative measurement of a model. Example: loss_fn (_Loss) – a callable function that takes y_pred and optionally y as input (in the “batch-first” format), returns a “batch-first” tensor of loss values. reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans). Here not_nans count the number of not nans for the metric, thus its shape equals to the shape of the metric. Returns the aggregated loss value across multiple iterations. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction.

Mean Dice#

Compute average Dice score between two tensors. It can support both multi-classes and multi-labels tasks. Input y_pred is compared with ground truth y. y_preds is expected to have binarized predictions and y should be in one-hot format. You can use suitable transforms in monai.transforms.post first to achieve binarized values. The include_background parameter can be set to False to exclude the first category (channel index 0) which is by convention assumed to be background. If the non-background segmentations are small compared to the total image size they can get overwhelmed by the signal from the background. y_preds and y can be a list of channel-first Tensor (CHW[D]) or a batch-first Tensor (BCHW[D]). Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. include_background (bool) – whether to skip Dice computation on the first channel of the predicted output. Defaults to True. reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans). Here not_nans count the number of not nans for the metric, thus its shape equals to the shape of the metric. ignore_empty (bool) – whether to ignore empty ground truth cases during calculation. If True, NaN value will be set for empty ground truth cases. If False, 1 will be set if the predictions of empty ground truth cases are also empty. Execute reduction logic for the output of compute_meandice. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction.

Mean IoU#

Compute average Intersection over Union (IoU) score between two tensors. It supports both multi-classes and multi-labels tasks. Input y_pred is compared with ground truth y. y_pred is expected to have binarized predictions and y should be in one-hot format. You can use suitable transforms in monai.transforms.post first to achieve binarized values. The include_background parameter can be set to False to exclude the first category (channel index 0) which is by convention assumed to be background. If the non-background segmentations are small compared to the total image size they can get overwhelmed by the signal from the background. y_pred and y can be a list of channel-first Tensor (CHW[D]) or a batch-first Tensor (BCHW[D]). Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. include_background (bool) – whether to skip IoU computation on the first channel of the predicted output. Defaults to True. reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans). Here not_nans count the number of not nans for the metric, thus its shape equals to the shape of the metric. ignore_empty (bool) – whether to ignore empty ground truth cases during calculation. If True, NaN value will be set for empty ground truth cases. If False, 1 will be set if the predictions of empty ground truth cases are also empty. Execute reduction logic for the output of compute_meaniou. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction.

Generalized Dice Score#

Computes the Generalized Dice Score and returns a tensor with its per image values. y_pred (torch.Tensor) – binarized segmentation model output. It should be binarized, in one-hot format and in the NCHW[D] format, where N is the batch dimension, C is the channel dimension, and the remaining are the spatial dimensions. y (torch.Tensor) – binarized ground-truth. It should be binarized, in one-hot format and have the same shape as y_pred. include_background (bool, optional) – whether to skip score computation on the first channel of the predicted output. Defaults to True. weight_type (Union[Weight, str], optional) – {"square", "simple", "uniform"}. Type of function to transform ground truth volume into a weight factor. Defaults to "square". per batch and per class Generalized Dice Score, i.e., with the shape [batch_size, num_classes]. torch.Tensor ValueError – if y_pred or y are not PyTorch tensors, if y_pred and y have less than three dimensions, or y_pred and y don’t have the same shape. Compute the Generalized Dice Score metric between tensors, as the complement of the Generalized Dice Loss defined in: loss function for highly unbalanced segmentations. DLMIA 2017. The inputs y_pred and y are expected to be one-hot, binarized channel-first or batch-first tensors, i.e., CHW[D] or BCHW[D]. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. include_background (bool, optional) – whether to include the background class (assumed to be in channel 0), in the score computation. Defaults to True. reduction (str, optional) – define mode of reduction to the metrics. Available reduction modes: {"none", "mean_batch", "sum_batch"}. Default to "mean_batch". If “none”, will not do reduction. weight_type (Union[Weight, str], optional) – {"square", "simple", "uniform"}. Type of function to transform ground truth volume into a weight factor. Defaults to "square". ValueError – when the weight_type is not one of {"none", "mean", "sum"}. Execute reduction logic for the output of compute_generalized_dice. reduction (Union[MetricReduction, str, None], optional) – define mode of reduction to the metrics. Available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch"}. Defaults to "mean". If “none”, will not do reduction.

Area under the ROC curve#

Computes Area Under the Receiver Operating Characteristic Curve (ROC AUC). Referring to: sklearn.metrics.roc_auc_score. y_pred (Tensor) – input data to compute, typical classification model output. the first dim must be batch, if multi-classes, it must be in One-Hot format. for example: shape [16] or [16, 1] for a binary data, shape [16, 2] for 2 classes data. y (Tensor) – ground truth to compute ROC AUC metric, the first dim must be batch. if multi-classes, it must be in One-Hot format. for example: shape [16] or [16, 1] for a binary data, shape [16, 2] for 2 classes data. average (Union[Average, str]) – {"macro", "weighted", "micro", "none"} Type of averaging performed if not binary classification. Defaults to "macro". "macro": calculate metrics for each label, and find their unweighted mean.This does not take label imbalance into account. "weighted": calculate metrics for each label, and find their average,weighted by support (the number of true instances for each label). "micro": calculate metrics globally by considering each element of the labelindicator matrix as a label. "none": the scores for each class are returned. {"macro", "weighted", "micro", "none"} Type of averaging performed if not binary classification. Defaults to "macro". This does not take label imbalance into account. weighted by support (the number of true instances for each label). indicator matrix as a label. "none": the scores for each class are returned. ValueError – When y_pred dimension is not one of [1, 2]. ValueError – When y dimension is not one of [1, 2]. ValueError – When average is not one of [“macro”, “weighted”, “micro”, “none”]. Note ROCAUC expects y to be comprised of 0’s and 1’s. y_pred must be either prob. estimates or confidence values. Computes Area Under the Receiver Operating Characteristic Curve (ROC AUC). Referring to: sklearn.metrics.roc_auc_score. The input y_pred and y can be a list of channel-first Tensor or a batch-first Tensor. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. average (Union[Average, str]) – {"macro", "weighted", "micro", "none"} Type of averaging performed if not binary classification. Defaults to "macro". "macro": calculate metrics for each label, and find their unweighted mean.This does not take label imbalance into account. "weighted": calculate metrics for each label, and find their average,weighted by support (the number of true instances for each label). "micro": calculate metrics globally by considering each element of the labelindicator matrix as a label. "none": the scores for each class are returned. {"macro", "weighted", "micro", "none"} Type of averaging performed if not binary classification. Defaults to "macro". This does not take label imbalance into account. weighted by support (the number of true instances for each label). indicator matrix as a label. "none": the scores for each class are returned. Typically y_pred and y are stored in the cumulative buffers at each iteration, This function reads the buffers and computes the area under the ROC. average (Union[Average, str, None]) – {"macro", "weighted", "micro", "none"} Type of averaging performed if not binary classification. Defaults to self.average.

Confusion matrix#

Compute confusion matrix. A tensor with the shape [BC4] will be returned. Where, the third dimension represents the number of true positive, false positive, true negative and false negative values for each channel of each sample within the input batch. Where, B equals to the batch size and C equals to the number of classes that need to be computed. y_pred (Tensor) – input data to compute. It must be one-hot format and first dim is batch. The values should be binarized. y (Tensor) – ground truth to compute the metric. It must be one-hot format and first dim is batch. The values should be binarized. include_background (bool) – whether to skip metric computation on the first channel of the predicted output. Defaults to True. ValueError – when y_pred and y have different shapes. This function is used to compute confusion matrix related metric. metric_name (str) – ["sensitivity", "specificity", "precision", "negative predictive value", "miss rate", "fall out", "false discovery rate", "false omission rate", "prevalence threshold", "threat score", "accuracy", "balanced accuracy", "f1 score", "matthews correlation coefficient", "fowlkes mallows index", "informedness", "markedness"] Some of the metrics have multiple aliases (as shown in the wikipedia page aforementioned), and you can also input those names instead. confusion_matrix (Tensor) – Please see the doc string of the function get_confusion_matrix for more details. ValueError – when the size of the last dimension of confusion_matrix is not 4. NotImplementedError – when specify a not implemented metric_name. Compute confusion matrix related metrics. This function supports to calculate all metrics mentioned in: Confusion matrix. It can support both multi-classes and multi-labels classification and segmentation tasks. y_preds is expected to have binarized predictions and y should be in one-hot format. You can use suitable transforms in monai.transforms.post first to achieve binarized values. The include_background parameter can be set to False for an instance to exclude the first category (channel index 0) which is by convention assumed to be background. If the non-background segmentations are small compared to the total image size they can get overwhelmed by the signal from the background. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. include_background (bool) – whether to skip metric computation on the first channel of the predicted output. Defaults to True. metric_name (Union[Sequence[str], str]) – ["sensitivity", "specificity", "precision", "negative predictive value", "miss rate", "fall out", "false discovery rate", "false omission rate", "prevalence threshold", "threat score", "accuracy", "balanced accuracy", "f1 score", "matthews correlation coefficient", "fowlkes mallows index", "informedness", "markedness"] Some of the metrics have multiple aliases (as shown in the wikipedia page aforementioned), and you can also input those names instead. Except for input only one metric, multiple metrics are also supported via input a sequence of metric names, such as (“sensitivity”, “precision”, “recall”), if compute_sample is True, multiple f and not_nans will be returned with the same order as input names when calling the class. compute_sample (bool) – when reducing, if True, each sample’s metric will be computed based on each confusion matrix first. if False, compute reduction on the confusion matrices first, defaults to False. reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns [(metric, not_nans), …]. If False, aggregate() returns [metric, …]. Here not_nans count the number of not nans for True Positive, False Positive, True Negative and False Negative. Its shape depends on the shape of the metric, and it has one more dimension with size 4. For example, if the shape of the metric is [3, 3], not_nans has the shape [3, 3, 4]. Execute reduction for the confusion matrix values. compute_sample (bool) – when reducing, if True, each sample’s metric will be computed based on each confusion matrix first. if False, compute reduction on the confusion matrices first, defaults to False. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction.

Hausdorff distance#

Compute the Hausdorff distance. y_pred (Union[ndarray, Tensor]) – input data to compute, typical segmentation model output. It must be one-hot format and first dim is batch, example shape: [16, 3, 32, 32]. The values should be binarized. y (Union[ndarray, Tensor]) – ground truth to compute mean the distance. It must be one-hot format and first dim is batch. The values should be binarized. include_background (bool) – whether to skip distance computation on the first channel of the predicted output. Defaults to False. distance_metric (str) – : ["euclidean", "chessboard", "taxicab"] the metric used to compute surface distance. Defaults to "euclidean". percentile (Optional[float]) – an optional float number between 0 and 100. If specified, the corresponding percentile of the Hausdorff Distance rather than the maximum result will be achieved. Defaults to None. directed (bool) – whether to calculate directed Hausdorff distance. Defaults to False. This function is used to compute the directed Hausdorff distance. Compute Hausdorff Distance between two tensors. It can support both multi-classes and multi-labels tasks. It supports both directed and non-directed Hausdorff distance calculation. In addition, specify the percentile parameter can get the percentile of the distance. Input y_pred is compared with ground truth y. y_preds is expected to have binarized predictions and y should be in one-hot format. You can use suitable transforms in monai.transforms.post first to achieve binarized values. y_preds and y can be a list of channel-first Tensor (CHW[D]) or a batch-first Tensor (BCHW[D]). The implementation refers to DeepMind's implementation. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. include_background (bool) – whether to include distance computation on the first channel of the predicted output. Defaults to False. distance_metric (str) – : ["euclidean", "chessboard", "taxicab"] the metric used to compute surface distance. Defaults to "euclidean". percentile (Optional[float]) – an optional float number between 0 and 100. If specified, the corresponding percentile of the Hausdorff Distance rather than the maximum result will be achieved. Defaults to None. directed (bool) – whether to calculate directed Hausdorff distance. Defaults to False. reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if "none", will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans). Here not_nans count the number of not nans for the metric, thus its shape equals to the shape of the metric. Execute reduction logic for the output of compute_hausdorff_distance. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if "none", will not do reduction.

Average surface distance#

This function is used to compute the Average Surface Distance from y_pred to y under the default setting. In addition, if sets symmetric = True, the average symmetric surface distance between these two inputs will be returned. The implementation refers to DeepMind's implementation. y_pred (Union[ndarray, Tensor]) – input data to compute, typical segmentation model output. It must be one-hot format and first dim is batch, example shape: [16, 3, 32, 32]. The values should be binarized. y (Union[ndarray, Tensor]) – ground truth to compute mean the distance. It must be one-hot format and first dim is batch. The values should be binarized. include_background (bool) – whether to skip distance computation on the first channel of the predicted output. Defaults to False. symmetric (bool) – whether to calculate the symmetric average surface distance between seg_pred and seg_gt. Defaults to False. distance_metric (str) – : ["euclidean", "chessboard", "taxicab"] the metric used to compute surface distance. Defaults to "euclidean". Compute Surface Distance between two tensors. It can support both multi-classes and multi-labels tasks. It supports both symmetric and asymmetric surface distance calculation. Input y_pred is compared with ground truth y. y_preds is expected to have binarized predictions and y should be in one-hot format. You can use suitable transforms in monai.transforms.post first to achieve binarized values. y_preds and y can be a list of channel-first Tensor (CHW[D]) or a batch-first Tensor (BCHW[D]). Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. include_background (bool) – whether to skip distance computation on the first channel of the predicted output. Defaults to False. symmetric (bool) – whether to calculate the symmetric average surface distance between seg_pred and seg_gt. Defaults to False. distance_metric (str) – : ["euclidean", "chessboard", "taxicab"] the metric used to compute surface distance. Defaults to "euclidean". reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans). Here not_nans count the number of not nans for the metric, thus its shape equals to the shape of the metric. Execute reduction logic for the output of compute_average_surface_distance. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction.

Surface dice#

This function computes the (Normalized) Surface Dice (NSD) between the two tensors y_pred (referred to as \(\hat{Y}\)) and y (referred to as \(Y\)). This metric determines which fraction of a segmentation boundary is correctly predicted. A boundary element is considered correctly predicted if the closest distance to the reference boundary is smaller than or equal to the specified threshold related to the acceptable amount of deviation in pixels. The NSD is bounded between 0 and 1. This implementation supports multi-class tasks with an individual threshold \(\tau_c\) for each class \(c\). The class-specific NSD for batch index \(b\), \(\operatorname {NSD}_{b,c}\), is computed using the function: with \(\mathcal{D}_{Y_{b,c}}\) and \(\mathcal{D}_{\hat{Y}_{b,c}}\) being two sets of nearest-neighbor distances. \(\mathcal{D}_{Y_{b,c}}\) is computed from the predicted segmentation boundary towards the reference segmentation boundary and vice-versa for \(\mathcal{D}_{\hat{Y}_{b,c}}\). \(\mathcal{D}_{Y_{b,c}}^{'}\) and \(\mathcal{D}_{\hat{Y}_{b,c}}^{'}\) refer to the subsets of distances that are smaller or equal to the acceptable distance \(\tau_c\): In the case of a class neither being present in the predicted segmentation, nor in the reference segmentation, a nan value will be returned for this class. In the case of a class being present in only one of predicted segmentation or reference segmentation, the class NSD will be 0. This implementation is based on https://arxiv.org/abs/2111.05408 and supports 2D images. Be aware that the computation of boundaries is different from DeepMind’s implementation deepmind/surface-distance. In this implementation, the length of a segmentation boundary is interpreted as the number of its edge pixels. In DeepMind’s implementation, the length of a segmentation boundary depends on the local neighborhood (cf. https://arxiv.org/abs/1809.04430). y_pred (Tensor) – Predicted segmentation, typically segmentation model output. It must be a one-hot encoded, batch-first tensor [B,C,H,W]. y (Tensor) – Reference segmentation. It must be a one-hot encoded, batch-first tensor [B,C,H,W]. class_thresholds (List[float]) – List of class-specific thresholds. The thresholds relate to the acceptable amount of deviation in the segmentation boundary in pixels. Each threshold needs to be a finite, non-negative number. include_background (bool) – Whether to skip the surface dice computation on the first channel of the predicted output. Defaults to False. distance_metric (str) – The metric used to compute surface distances. One of ["euclidean", "chessboard", "taxicab"]. Defaults to "euclidean". ValueError – If y_pred and/or y are not PyTorch tensors. ValueError – If y_pred and/or y do not have four dimensions. ValueError – If y_pred and/or y have different shapes. ValueError – If y_pred and/or y are not one-hot encoded ValueError – If the number of channels of y_pred and/or y is different from the number of class thresholds. ValueError – If any class threshold is not finite. ValueError – If any class threshold is negative. Pytorch Tensor of shape [B,C], containing the NSD values \(\operatorname {NSD}_{b,c}\) for each batch index \(b\) and class \(c\). Computes the Normalized Surface Distance (NSD) for each batch sample and class of predicted segmentations y_pred and corresponding reference segmentations y according to equation (1). This implementation supports 2D images. For 3D images, please refer to DeepMind’s implementation deepmind/surface-distance. The class- and batch sample-wise NSD values can be aggregated with the function aggregate. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. class_thresholds (List[float]) – List of class-specific thresholds. The thresholds relate to the acceptable amount of deviation in the segmentation boundary in pixels. Each threshold needs to be a finite, non-negative number. include_background (bool) – Whether to skip NSD computation on the first channel of the predicted output. Defaults to False. distance_metric (str) – The metric used to compute surface distances. One of ["euclidean", "chessboard", "taxicab"]. Defaults to "euclidean". reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count. Defaults to False. not_nans is the number of batch samples for which not all class-specific NSD values were nan values. If set to True, the function aggregate will return both the aggregated NSD and the not_nans count. If set to False, aggregate will only return the aggregated NSD. Aggregates the output of _compute_tensor. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction. If get_not_nans is set to True, this function returns the aggregated NSD and the not_nans count. If get_not_nans is set to False, this function returns only the aggregated NSD.

PanopticQualityMetric#

Computes Panoptic Quality (PQ). If specifying metric_name to “SQ” or “RQ”, Segmentation Quality (SQ) or Recognition Quality (RQ) will be returned instead. In addition, if output_confusion_matrix is True, the function will return a tensor with shape 4, which represents the true positive, false positive, false negative and the sum of iou. These four values are used to calculate PQ, and returning them directly enables further calculation over all images. pred (Tensor) – input data to compute, it must be in the form of HW and have integer type. gt (Tensor) – ground truth. It must have the same shape as pred and have integer type. metric_name (str) – output metric. The value can be “pq”, “sq” or “rq”. remap (bool) – whether to remap pred and gt to ensure contiguous ordering of instance id. match_iou_threshold (float) – IOU threshold to determine the pairing between pred and gt. Usually, it should >= 0.5, the pairing between instances of pred and gt are identical. If set match_iou_threshold < 0.5, this function uses Munkres assignment to find the maximal amount of unique pairing. smooth_numerator (float) – a small constant added to the numerator to avoid zero. ValueError – when pred and gt have different shapes. ValueError – when match_iou_threshold <= 0.0 or > 1.0. Compute Panoptic Quality between two instance segmentation masks. If specifying metric_name to “SQ” or “RQ”, Segmentation Quality (SQ) or Recognition Quality (RQ) will be returned instead. Panoptic Quality is a metric used in panoptic segmentation tasks. This task unifies the typically distinct tasks of semantic segmentation (assign a class label to each pixel) and instance segmentation (detect and segment each object instance). Compared with semantic segmentation, panoptic segmentation distinguish different instances that belong to same class. Compared with instance segmentation, panoptic segmentation does not allow overlap and only one semantic label and one instance id can be assigned to each pixel. Please refer to the following paper for more details: https://openaccess.thecvf.com/content_CVPR_2019/papers/Kirillov_Panoptic_Segmentation_CVPR_2019_paper.pdf This class also refers to the following implementation: TissueImageAnalytics/CoNIC num_classes (int) – number of classes. The number should not count the background. metric_name (Union[Sequence[str], str]) – output metric. The value can be “pq”, “sq” or “rq”. Except for input only one metric, multiple metrics are also supported via input a sequence of metric names such as (“pq”, “sq”, “rq”). If input a sequence, a list of results with the same order as the input names will be returned. reduction (Union[MetricReduction, str]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction. match_iou_threshold (float) – IOU threshold to determine the pairing between y_pred and y. Usually, it should >= 0.5, the pairing between instances of y_pred and y are identical. If set match_iou_threshold < 0.5, this function uses Munkres assignment to find the maximal amount of unique pairing. smooth_numerator (float) – a small constant added to the numerator to avoid zero. Execute reduction logic for the output of compute_panoptic_quality. reduction (Union[MetricReduction, str, None]) – define mode of reduction to the metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to self.reduction. if “none”, will not do reduction.

Mean squared error#

Compute Mean Squared Error between two tensors using function: More info: https://en.wikipedia.org/wiki/Mean_squared_error Input y_pred is compared with ground truth y. Both y_pred and y are expected to be real-valued, where y_pred is output from a regression model. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. reduction (Union[MetricReduction, str]) – define the mode to reduce metrics, will only execute reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans).

Mean absolute error#

Compute Mean Absolute Error between two tensors using function: More info: https://en.wikipedia.org/wiki/Mean_absolute_error Input y_pred is compared with ground truth y. Both y_pred and y are expected to be real-valued, where y_pred is output from a regression model. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. reduction (Union[MetricReduction, str]) – define the mode to reduce metrics, will only execute reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction.

get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans).

Root mean squared error#

Compute Root Mean Squared Error between two tensors using function: More info: https://en.wikipedia.org/wiki/Root-mean-square_deviation Input y_pred is compared with ground truth y. Both y_pred and y are expected to be real-valued, where y_pred is output from a regression model. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. reduction (Union[MetricReduction, str]) – define the mode to reduce metrics, will only execute reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans).

Peak signal to noise ratio#

Compute Peak Signal To Noise Ratio between two tensors using function: More info: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio Help taken from: tensorflow/tensorflow line 4139 Input y_pred is compared with ground truth y. Both y_pred and y are expected to be real-valued, where y_pred is output from a regression model. Example of the typical execution steps of this metric class follows monai.metrics.metric.Cumulative. max_val (Union[int, float]) – The dynamic range of the images/volumes (i.e., the difference between the maximum and the minimum allowed values e.g. 255 for a uint8 image). reduction (Union[MetricReduction, str]) – define the mode to reduce metrics, will only execute reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction. get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans).

Structural similarity index measure#

Build a Pytorch version of the SSIM metric based on the original formula of SSIM https://vicuesoft.com/glossary/term/ssim-ms-ssim/ facebookresearch/fastMRI Wang, Zhou, et al. “Image quality assessment: from error visibility to structural similarity.” IEEE transactions on image processing 13.4 (2004): 600-612. data_range (Tensor) – dynamic range of the data win_size (int) – gaussian weighting window size k1 (float) – stability constant used in the luminance denominator k2 (float) – stability constant used in the contrast denominator spatial_dims (int) – if 2, input shape is expected to be (B,C,W,H). if 3, it is expected to be (B,C,W,H,D) reduction (Union[MetricReduction, str]) – define the mode to reduce metrics, will only execute reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, will not do reduction get_not_nans (bool) – whether to return the not_nans count, if True, aggregate() returns (metric, not_nans)

Cumulative average#

A utility class to keep track of average values. For example during training/validation loop, we need to accumulate the per-batch metrics and calculate the final average value for the whole dataset. When training in multi-gpu environment, with DistributedDataParallel, it will average across the processes. Example: returns the total average value (averaged across processes) to_numpy (bool) – whether to convert to numpy array. Defaults to True Union[ndarray, Tensor] with torch.as_tensor() e.g. number, list, numpy array, or Tensor. val (Any) – value (e.g. number, list, numpy array or Tensor) to keep track of count (Optional[Any]) – count (e.g. number, list, numpy array or Tensor), to update the contribution count # a simple constant tracking avg = CumulativeAverage() avg.append(0.6) avg.append(0.8) print(avg.aggregate()) #prints 0.7 # an array tracking, e.g. metrics from 3 classes avg= CumulativeAverage() avg.append([0.2, 0.4, 0.4]) avg.append([0.4, 0.6, 0.4]) print(avg.aggregate()) #prints [0.3, 0.5. 0.4] # different contributions / counts avg= CumulativeAverage() avg.append(1, count=4) #avg metric 1 coming from a batch of 4 avg.append(2, count=6) #avg metric 2 coming from a batch of 6 print(avg.aggregate()) #prints 1.6 == (1*4 +2*6)/(4+6) # different contributions / counts avg= CumulativeAverage() avg.append([0.5, 0.5, 0], count=[1, 1, 0]) # last elements count is zero to ignore it avg.append([0.5, 0.5, 0.5], count=[1, 1, 1]) # print(avg.aggregate()) #prints [0.5, 0.5, 0,5] == ([0.5, 0.5, 0] + [0.5, 0.5, 0.5]) / ([1, 1, 0] + [1, 1, 1]) None returns the most recent value (averaged across processes) to_numpy (bool) – whether to convert to numpy array. Defaults to True Union[ndarray, Tensor] Reset all stats None

Utilities#

This function is to do the metric reduction for calculated not-nan metrics of each sample's each class. The function also returns not_nans, which counts the number of not nans for the metric. f (Tensor) – a tensor that contains the calculated metric scores per batch and per class. The first two dims should be batch and class. reduction (Union[MetricReduction, str]) – define the mode to reduce metrics, will only apply reduction on not-nan values, available reduction modes: {"none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"}, default to "mean". if “none”, return the input f tensor and not_nans. ValueError – When reduction is not one of [“mean”, “sum”, “mean_batch”, “sum_batch”, “mean_channel”, “sum_channel” “none”]. Do binary erosion and use XOR for input to get the edges. This function is helpful to further calculate metrics such as Average Surface Distance and Hausdorff Distance. The input images can be binary or labelfield images. If labelfield images are supplied, they are converted to binary images using label_idx. scipy's binary erosion is used to calculate the edges of the binary labelfield. In order to improve the computing efficiency, before getting the edges, the images can be cropped and only keep the foreground if not specifies crop = False. We require that images are the same size, and assume that they occupy the same space (spacing, orientation, etc.). seg_pred – the predicted binary or labelfield image. seg_gt – the actual binary or labelfield image. label_idx (int) – for labelfield images, convert to binary with seg_pred = seg_pred == label_idx. crop (bool) – crop input images and only keep the foregrounds. In order to maintain two inputs' shapes, here the bounding box is achieved by (seg_pred | seg_gt) which represents the union set of two images. Defaults to True. Tuple[ndarray, ndarray] This function is used to compute the surface distances from seg_pred to seg_gt. seg_pred (ndarray) – the edge of the predictions. seg_gt (ndarray) – the edge of the ground truth. distance_metric (str) – : ["euclidean", "chessboard", "taxicab"] the metric used to compute surface distance. Defaults to "euclidean". "euclidean", uses Exact Euclidean distance transform. "chessboard", uses chessboard metric in chamfer type of transform. "taxicab", uses taxicab metric in chamfer type of transform. : ["euclidean", "chessboard", "taxicab"] the metric used to compute surface distance. Defaults to "euclidean". "euclidean", uses Exact Euclidean distance transform. "chessboard", uses chessboard metric in chamfer type of transform. "taxicab", uses taxicab metric in chamfer type of transform. Note If seg_pred or seg_gt is all 0, may result in nan/inf distance. ndarray This function is used to remove background (the first channel) for y_pred and y. y_pred (Union[ndarray, Tensor]) – predictions. As for classification tasks, y_pred should has the shape [BN] where N is larger than 1. As for segmentation tasks, the shape should be [BNHW] or [BNHWD]. y (Union[ndarray, Tensor]) – ground truth, the first dim is batch. Determines whether the input tensor is torch binary tensor or not. input (torch.Tensor) – tensor to validate. name (str) – name of the tensor being checked. ValueError – if input is not a PyTorch Tensor. warning message, if the tensor is not binary. Otherwise, None. Union[str, None] previous Network architectures next Optimizers