Model checkpoint loader#

CheckpointLoader acts as an Ignite handler to load checkpoint data from file. It can load variables for network, optimizer, lr_scheduler, etc. If saving checkpoint after torch.nn.DataParallel, need to save model.module instead as PyTorch recommended and then use this loader to load the model. load_path (str) – the file path of checkpoint, it should be a PyTorch pth file. load_dict (Dict) – target objects that load checkpoint to. examples: {'network': net, 'optimizer': optimizer, 'lr_scheduler': lr_scheduler} target objects that load checkpoint to. examples: name (Optional[str]) – identifier of logging.logger to use, if None, defaulting to engine.logger. map_location (Optional[Dict]) – when loading the module for distributed training/evaluation, need to provide an appropriate map_location argument to prevent a process to step into others' devices. If map_location is missing, torch.load will first load the module to CPU and then copy each parameter to where it was saved, which would result in all processes on the same machine using the same set of devices. strict (bool) – whether to strictly enforce that the keys and data shape in the state_dict of every item of load_dict match the state_dict of the corresponding items of checkpoint, default to True. strict_shape (bool) – whether to enforce the data shape of the matched layers in the checkpoint, if `False, it will skip the layers that have different data shape with checkpoint content, and ignore the strict arg. this can be useful advanced feature for transfer learning. users should totally understand which layers will have different shape. default to True. items in the load_dict. For example, if the shape of some layers in current model can't match the checkpoint, the parameter_group of current optimizer may also can't match the checkpoint, so skip loading checkpoint for optimizer. For more details about loading checkpoint, please refer to: https://pytorch.org/ignite/v0.4.5/generated/ignite.handlers.checkpoint.Checkpoint.html#ignite.handlers.checkpoint.Checkpoint.load_objects. https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module.load_state_dict. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Model checkpoint saver#

CheckpointSaver acts as an Ignite handler to save checkpoint data into files. It supports to save according to metrics result, epoch number, iteration number and last model or exception. save_dir (str) – the target directory to save the checkpoints. save_dict (Dict) – source objects that save to the checkpoint. examples: {'network': net, 'optimizer': optimizer, 'lr_scheduler': lr_scheduler} source objects that save to the checkpoint. examples: name (Optional[str]) – identifier of logging.logger to use, if None, defaulting to engine.logger. file_prefix (str) – prefix for the filenames to which objects will be saved. save_final (bool) – whether to save checkpoint or session at final iteration or exception. If checkpoints are to be saved when an exception is raised, put this handler before StatsHandler in the handler list, because the logic with Ignite can only trigger the first attached handler for EXCEPTION_RAISED event. final_filename (Optional[str]) – set a fixed filename to save the final model if save_final=True. If None, default to checkpoint_final_iteration=N.pt. save_key_metric (bool) – whether to save checkpoint or session when the value of key_metric is higher than all the previous values during training.keep 4 decimal places of metric, checkpoint name is: {file_prefix}_key_metric=0.XXXX.pth. key_metric_name (Optional[str]) – the name of key_metric in ignite metrics dictionary. If None, use engine.state.key_metric instead. key_metric_n_saved (int) – save top N checkpoints or sessions, sorted by the value of key metric in descending order. key_metric_filename (Optional[str]) – set a fixed filename to set the best metric model,

if not None, key_metric_n_saved should be 1 and only keep the best metric model. key_metric_save_state (bool) – whether to save the tracking list of key metric in the checkpoint file. if True, then will save an object in the checkpoint file with key checkpointer to be consistent with the include_self arg of Checkpoint in ignite: https://pytorch.org/ignite/v0.4.5/generated/ignite.handlers.checkpoint.Checkpoint.html. typically, it's used to resume training and compare current metric with previous N values. key_metric_greater_or_equal (bool) – if True, the latest equally scored model is stored. Otherwise, save the first equally scored model. default to False. key_metric_negative_sign (bool) – whether adding a negative sign to the metric score to compare metrics, because for error-like metrics, smaller is better(objects with larger score are retained). default to False. epoch_level (bool) – save checkpoint during training for every N epochs or every N iterations. True is epoch level, False is iteration level. save_interval (int) – save checkpoint every N epochs, default is 0 to save no checkpoint. n_saved (Optional[int]) – save latest N checkpoints of epoch level or iteration level, 'None' is to save all. Note CheckpointHandler can be used during training, validation or evaluation. example of saved files: checkpoint_iteration=400.pt checkpoint_iteration=800.pt checkpoint_epoch=1.pt checkpoint_final_iteration=1000.pt checkpoint_key_metric=0.9387.pt engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Callback for train or validation/evaluation completed Event. Save final checkpoint if configure save_final is True. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Callback for train or validation/evaluation exception raised Event. Save current data as final checkpoint if configure save_final is True. This callback may be skipped because the logic with Ignite can only trigger the first attached handler for EXCEPTION_RAISED event. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. e (Exception) – the exception caught in Ignite during engine.run(). None Callback for train epoch/iteration completed Event. Save checkpoint if configure save_interval = N engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Utility to resume the internal state of key metric tracking list if configured to save checkpoints based on the key metric value. Note to set key_metric_save_state=True when saving the previous checkpoint. Example: None Callback to compare metrics and save models in train or validation when epoch completed. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Metrics saver#

ignite handler to save metrics values and details into expected files. save_dir (str) – directory to save the metrics and metric details. metrics (Union[str, Sequence[str], None]) – expected final metrics to save into files, can be: None, "*" or list of strings. None - don't save any metrics into files. "*" - save all the existing metrics in engine.state.metrics dict into separate files. list of strings - specify the expected metrics to save. default to "*" to save all the metrics into metrics.csv. metric_details (Union[str, Sequence[str], None]) – expected metric details to save into files, the data comes from engine.state.metric_details, which should be provided by different Metrics, typically, it's some intermediate values in metric computation. for example: mean dice of every channel of every image in the validation dataset. it must contain at least 2 dims: (batch, classes, …), if not, will unsqueeze to 2 dims. this arg can be: None, "*" or list of strings. None - don't save any metric_details into files. "*" - save all the existing metric_details in engine.state.metric_details dict into separate files. list of strings - specify the metric_details of expected metrics to save. if not None, every metric_details array will save a separate {metric name}_raw.csv file. batch_transform

(Callable) – a callable that is used to extract the meta_data dictionary of the input images from ignite.engine.state.batch if saving metric details. the purpose is to get the input filenames from the meta_data and store with metric details together. engine.state and batch_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. summary_ops (Union[str, Sequence[str], None]) – expected computation operations to generate the summary report. it can be: None, “*” or list of strings, default to None. None - don’t generate summary report for every expected metric_details. “*” - generate summary report for every metric_details with all the supported operations. list of strings - generate summary report for every metric_details with specified operations, they should be within list: [“mean”, “median”, “max”, “min”, “percentile”, “std”, “notnans”]. the number in “percentile” should be [0, 100], like: “15percentile”. default: “90percentile”. for more details, please check: https://numpy.org/doc/stable/reference/generated/numpy.nanpercentile.html. note that: for the overall summary, it computes nanmean of all classes for each image first, then compute summary. example of the generated summary report: class mean median max 5percentile 95percentile notnans class0 6.0000 6.0000 7.0000 5.1000 6.9000 2.0000 class1 6.0000 6.0000 6.0000 6.0000 6.0000 1.0000 mean 6.2500 6.2500 7.0000 5.5750 6.9250 2.0000 expected computation operations to generate the summary report. it can be: None, “*” or list of strings, default to None. None - don’t generate summary report for every expected metric_details. “*” - generate summary report for every metric_details with all the supported operations. list of strings - generate summary report for every metric_details with specified operations, they should be within list: [“mean”, “median”, “max”, “min”, “percentile”, “std”, “notnans”]. the number in “percentile” should be [0, 100], like: “15percentile”. default: “90percentile”. for more details, please check: https://numpy.org/doc/stable/reference/generated/numpy.nanpercentile.html. note that: for the overall summary, it computes nanmean of all classes for each image first, then compute summary. example of the generated summary report: save_rank (int) – only the handler on specified rank will save to files in multi-gpus validation, default to 0. delimiter (str) – the delimiter character in the saved file, default to “,” as the default output type is csv. to be consistent with: https://docs.python.org/3/library/csv.html#csv.Dialect.delimiter. output_type (str) – expected output file type, supported types: [“csv”], default to “csv”. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

CSV saver#

Event handler triggered on completing every iteration to save the classification predictions as CSV file. If running in distributed data parallel, only saves CSV file in the specified rank. output_dir (str) – if saver=None, output CSV file directory. filename (str) – if saver=None, name of the saved CSV file name. delimiter (str) – the delimiter character in the saved file, default to “,” as the default output type is csv. to be consistent with: https://docs.python.org/3/library/csv.html#csv.Dialect.delimiter. overwrite (bool) – if saver=None, whether to overwriting existing file content, if True, will clear the file before saving. otherwise, will append new content to the file. batch_transform (Callable) – a callable that is used to extract the meta_data dictionary of the input images from ignite.engine.state.batch. the purpose is to get the input filenames from the meta_data and store with classification results together. engine.state and batch_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. output_transform (Callable) – a callable that is

used to extract the model prediction data from ignite.engine.state.output. the first dimension of its output will be treated as the batch dimension. each item in the batch will be saved individually. engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. name (Optional[str]) – identifier of logging.logger to use, defaulting to engine.logger. save_rank (int) – only the handler on specified rank will save to CSV file in multi-gpus validation, default to 0. saver (Optional[CSVSaver]) – the saver instance to save classification results, if None, create a CSVSaver internally. the saver must provide save_batch(batch_data, meta_data) and finalize() APIs. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Ignite Metric#

Base Metric class based on ignite event handler mechanism. The input prediction or label data can be a PyTorch Tensor or numpy array with batch dim and channel dim, or a list of PyTorch Tensor or numpy array without batch dim. metric_fn (CumulativeIterationMetric) – callable function or class to compute raw metric results after every iteration. expect to return a Tensor with shape (batch, channel, …) or tuple (Tensor, not_nans). output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: mean_dice of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. Attaches current metric to provided engine. On the end of engine’s run, engine.state.metrics dictionary will contain computed metric’s value under provided name. engine (Engine) – the engine to which the metric must be attached. name (str) – the name of the metric to attach. None NotComputableError – When compute is called before an update occurs. Any Resets the metric to it’s initial state. By default, this is called at the start of each epoch. None output (Sequence[Tensor]) – sequence with contents [y_pred, y]. ValueError – When output length is not 2. metric_fn can only support y_pred and y. None

Mean Dice metrics handler#

Computes Dice score metric from full size Tensor and collects average over batch, class-channels, iterations. include_background (bool) – whether to include dice computation on the first channel of the predicted output. Defaults to True. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: mean dice of every image.

default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.meandice.compute_meandice()

Mean IoU metric handler#

Computes IoU score metric from full size Tensor and collects average over batch, class-channels, iterations. include_background (bool) – whether to include iou computation on the first channel of the predicted output. Defaults to True. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: mean iou of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.meaniou.compute_meaniou()

ROC AUC metrics handler#

Computes Area Under the Receiver Operating Characteristic Curve (ROC AUC). accumulating predictions and the ground-truth during an epoch and applying compute_roc_auc. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. Note ROCAUC expects y to be comprised of 0’s and 1’s. y_pred must either be probability estimates or confidence values.

Confusion matrix metrics handler#

Compute confusion matrix related metrics from full size Tensor and collects average over batch, class-channels, iterations. include_background (bool) – whether to skip metric computation on the first channel of the predicted output. Defaults to True. 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. 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 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: TP/TN/FP/FN of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.confusion_matrix()

Hausdorff distance metrics handler#

Computes Hausdorff distance from full size Tensor and collects average over batch, class-channels, iterations. 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 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: hausdorff distance of every image. default to True, will save to engine.state.metric_details dict with the metric name as key.

Surface distance metrics handler#

Computes surface distance from full size Tensor and collects average over batch, class-channels, iterations. include_background (bool) – whether to include 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 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: surface dice of every image. default to True, will save to engine.state.metric_details dict with the metric name as key.

Panoptic Quality metrics handler#

Computes Panoptic quality from full size Tensor and collects average over batch, class-channels, iterations. num_classes (int) – number of classes. The number should not count the background. metric_name (str) – output metric. The value can be “pq”, “sq” or “rq”. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: panoptic quality of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.panoptic_quality.compute_panoptic_quality()

Mean squared error metrics handler#

Computes Mean Squared Error from full size Tensor and collects average over batch, iterations. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric

computation details per image, for example: mean squared error of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.MSEMetric

Mean absolute error metrics handler#

Computes Mean Absolute Error from full size Tensor and collects average over batch, iterations. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: mean squared error of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.MAEMetric

Root mean squared error metrics handler#

Computes Root Mean Squared Error from full size Tensor and collects average over batch, iterations. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: mean squared error of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. See also monai.metrics.RMSEMetric

Peak signal to noise ratio metrics handler#

Computes Peak Signal to Noise Ratio from full size Tensor and collects average over batch, iterations. 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. output_transform (Callable) – callable to extract y_pred and y from ignite.engine.state.output then construct (y_pred, y) pair, where y_pred and y can be batch-first Tensors or lists of channel-first Tensors. the form of (y_pred, y) is required

by the update(). engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. save_details (bool) – whether to save metric computation details per image, for example: mean squared error of every image. default to True, will save to engine.state.metric_details dict with the metric name as key. reduction – {"none", "mean", "sum", "mean_batch", "sum_batch", See also monai.metrics.PSNRMetric

Metric logger#

Collect per-iteration metrics and loss value from the attached trainer. This will also collect metric values from a given evaluator object which is expected to perform evaluation at the end of training epochs. This class is useful for collecting loss and metric values in one place for storage with checkpoint savers (state_dict and load_state_dict methods provided as expected by Pytorch and Ignite) and for graphing during training. # construct an evaluator saving mean dice metric values in the key “val_mean_dice” evaluator = SupervisedEvaluator(…, key_val_metric={“val_mean_dice”: MeanDice(…)}) # construct the logger and associate with evaluator to extract metric values from logger = MetricLogger(evaluator=evaluator) # construct the trainer with the logger passed in as a handler so that it logs loss values trainer = SupervisedTrainer(…, train_handlers=[logger, ValidationHandler(1, evaluator)]) # run training, logger.loss will be a list of (iteration, loss) values, logger.metrics a dict with key # “val_mean_dice” storing a list of (iteration, metric) values trainer.run() loss_transform (Callable) – Converts the output value from the trainer’s state into a loss value engine.state and loss_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. metric_transform (Callable) – Converts the metric value coming from the trainer/evaluator’s state into a storable value evaluator (Optional[Engine]) – Optional evaluator to consume metric results from at the end of its evaluation run engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Attach event handlers to the given evaluator to log metric values from it. evaluator (Engine) – Ignite Engine implementing network evaluation None Log metrics from the given Engine’s state member. engine (Engine) – Ignite Engine to log from None

Logfile handler#

Adds a logging.FileHandler to the attached engine’s logger when the start event occurs and removes it again when then completed event occurs. A handler is needed to remove FileHandler object when the complete event occurs so that further runs of different engines write only to the log files they should, rather than previous files. Multiple handlers can write to the same file which allows output from train and evaluation engine objects to be condensed in one file. If the given output directory doesn’t exist it will by default be created when the start event occurs. This can be used in conjunction with CheckpointSaver to save a log file to the same destination as the saved checkpoints. Since the handler is added possibly after other logging events during initialisation, not all logging data will be retained. output_dir (str) – directory to save the log file to filename (str) – name of the file to save log to loglevel (int) – log level for the handler formatter (str) – format string for the logging.Formatter set for the handler create_dir (bool) – if True, create output_dir if it doesn’t exist

Training stats handler#

StatsHandler defines a set of Ignite Event-handlers for all the log printing logics. It can be used for any Ignite Engine(trainer, validator and evaluator). And it can support logging for epoch level and iteration level with pre-defined loggers. Note that if name arg is None, will leverage engine.logger as default logger directly, otherwise, get logger from logging.getLogger(name), we can setup a logger outside first with the same name. As the default log level of RootLogger is WARNING, may need to call logging.basicConfig(stream=sys.stdout, level=logging.INFO) before running this handler to enable the stats logging. When EPOCH_COMPLETED, logs engine.state.metrics using self.logger. When ITERATION_COMPLETED, logs self.output_transform(engine.state.output) using self.logger. Usage example: More details of example is available in the tutorial: Project-MONAI/tutorials. iteration_log (bool) – whether to log data when iteration completed, default to True. epoch_log (bool) – whether to log data when epoch completed, default to True. epoch_print_logger (Optional[Callable[[Engine], Any]]) – customized callable printer for epoch level logging. Must accept parameter “engine”, use default printer if None. iteration_print_logger (Optional[Callable[[Engine], Any]]) – customized callable printer for iteration level logging. Must accept parameter “engine”, use default printer if None. output_transform (Callable) – a callable that is used to transform the ignite.engine.state.output into a scalar to print, or a dictionary of {key: scalar}. In the latter case, the output string will be formatted as key: value. By default this value logging happens when every iteration completed. The default behavior is to print loss from output[0] as output is a decollated list and we replicated loss value for every item of the decollated list. engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. global_epoch_transform (Callable) – a callable that is used to customize global epoch number. For example, in evaluation, the evaluator engine might want to print synced epoch number with the trainer engine. state_attributes (Optional[Sequence[str]]) – expected attributes from engine.state, if provided, will extract them when epoch completed. name (Optional[str]) – identifier of logging.logger to use, if None, defaulting to engine.logger. tag_name (str) – when iteration output is a scalar, tag_name is used to print tag_name: scalar_value to logger. Defaults to 'Loss'. key_var_format (str) – a formatting string to control the output string format of key: value. Register a set of Ignite Event-Handlers to a specified Ignite engine. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Handler for train or validation/evaluation epoch completed Event. Print epoch level log, default values are from Ignite engine.state.metrics dict. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Handler for train or validation/evaluation exception raised Event. Print the exception information and traceback. This callback may be skipped because the logic with Ignite can only trigger the first attached handler for EXCEPTION_RAISED event. _engine (Engine) – Ignite Engine, unused argument. e (Exception) – the exception caught in Ignite during engine.run(). None Handler for train or validation/evaluation iteration completed Event. Print iteration level log, default values are from Ignite engine.state.output. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Tensorboard handlers#

Base class for the handlers to write data into TensorBoard. summary_writer – user can specify TensorBoard or TensorBoardX SummaryWriter, default to create a new TensorBoard writer. log_dir (str) – if using default SummaryWriter, write logs to this directory, default is ./runs. Close the summary writer if created in this TensorBoard handler. TensorBoardStatsHandler defines a set of Ignite Event-handlers for all the TensorBoard logics. It can be used for any Ignite Engine(trainer, validator and evaluator). And it can support both epoch level and iteration level with pre-defined TensorBoard event writer. The expected data source is Ignite engine.state.output and engine.state.metrics. When EPOCH_COMPLETED, write each dictionary item in engine.state.metrics to TensorBoard. When ITERATION_COMPLETED, write each dictionary item in self.output_transform(engine.state.output) to TensorBoard. Usage example is available in the tutorial: Project-MONAI/tutorials. summary_writer – user can specify TensorBoard or TensorBoardX SummaryWriter, default to create a new TensorBoard writer. log_dir (str) – if using default SummaryWriter, write logs to this directory, default is ./runs. iteration_log (bool) – whether to write data to TensorBoard when iteration completed, default to True. epoch_log (bool) – whether to write data to TensorBoard when epoch completed, default to True. epoch_event_writer (Optional[Callable[[Engine, Any], Any]]) – customized callable TensorBoard writer for epoch level. Must accept parameter “engine” and “summary_writer”, use default event writer if None. epoch_interval (int) – the epoch interval at which the epoch_event_writer is called. Defaults to 1. iteration_event_writer (Optional[Callable[[Engine, Any], Any]]) – customized callable TensorBoard writer for iteration level. Must accept parameter “engine” and “summary_writer”, use default event writer if None. iteration_interval (int) – the iteration interval at which the iteration_event_writer is called. Defaults to 1. output_transform (Callable) – a callable that is used to transform the ignite.engine.state.output into a scalar to plot, or a dictionary of {key: scalar}. In the latter case, the output string will be formatted as key: value. By default this value plotting happens when every iteration completed. The default behavior is to print loss from output[0] as output is a decollated list and we replicated loss value for every item of the decollated list. engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. global_epoch_transform (Callable) – a callable that is used to customize global epoch number. For example, in evaluation, the evaluator engine might want to use trainer engines epoch number when plotting epoch vs metric curves. state_attributes (Optional[Sequence[str]]) – expected attributes from engine.state, if provided, will extract them when epoch completed. tag_name (str) – when iteration output is a scalar, tag_name is used to plot, defaults to 'Loss'. Register a set of Ignite Event-Handlers to a specified Ignite engine. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Handler for train or validation/evaluation epoch completed Event. Write epoch level events, default values are from Ignite engine.state.metrics dict. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Handler for train or validation/evaluation iteration completed Event. Write iteration level events, default values are from Ignite engine.state.output. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None TensorBoardImageHandler is an Ignite Event handler that can visualize images, labels and outputs as 2D/3D images. 2D output (shape in Batch, channel, H, W) will be shown as simple image using the first element in the batch, for 3D to ND output (shape in Batch, channel, H, W, D) input, each of self.max_channels number of images’ last three dimensions will be shown as animated GIF along the last axis (typically Depth). And if writer is from TensorBoardX, data has 3 channels and max_channels=3, will plot as RGB video. It can be used for any Ignite Engine (trainer, validator and evaluator). User can easily add it to engine for

any expected Event, for example: EPOCH_COMPLETED, ITERATION_COMPLETED. The expected data source is ignite's engine.state.batch and engine.state.output. Show y_pred as images (GIF for 3D) on TensorBoard when Event triggered, Need to use batch_transform and output_transform to specify how many images to show and show which channel. Expects batch_transform(engine.state.batch) to return data format: (image[N, channel, …], label[N, channel, …]). Expects output_transform(engine.state.output) to return a torch tensor in format (y_pred[N, channel, …], loss). Usage example is available in the tutorial: Project-MONAI/tutorials. summary_writer – user can specify TensorBoard or TensorBoardX SummaryWriter, default to create a new TensorBoard writer. log_dir (str) – if using default SummaryWriter, write logs to this directory, default is ./runs. interval (int) – plot content from engine.state every N epochs or every N iterations, default is 1. epoch_level (bool) – plot content from engine.state every N epochs or N iterations. True is epoch level, False is iteration level. batch_transform (Callable) – a callable that is used to extract image and label from ignite.engine.state.batch, then construct (image, label) pair. for example: if ignite.engine.state.batch is {"image": xxx, "label": xxx, "other": xxx}, batch_transform can be lambda x: (x["image"], x["label"]). will use the result to plot image from result[0][index] and plot label from result[1][index]. engine.state and batch_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. output_transform (Callable) – a callable that is used to extract the predictions data from ignite.engine.state.output, will use the result to plot output from result[index]. engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. global_iter_transform (Callable) – a callable that is used to customize global step number for TensorBoard. For example, in evaluation, the evaluator engine needs to know current epoch from trainer. index (int) – plot which element in a data batch, default is the first element. max_channels (int) – number of channels to plot. frame_dim (int) – if plotting 3D image as GIF, specify the dimension used as frames, expect input data shape as NCHWD, default to -3 (the first spatial dim) max_frames (int) – if plot 3D RGB image as video in TensorBoardX, set the FPS to max_frames. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

LR Schedule handler#

Ignite handler to update the Learning Rate based on PyTorch LR scheduler. lr_scheduler (Union[_LRScheduler, ReduceLROnPlateau]) – typically, lr_scheduler should be PyTorch lr_scheduler object. If customized version, must have step and get_last_lr methods. print_lr (bool) – whether to print out the latest learning rate with logging. name (Optional[str]) – identifier of logging.logger to use, if None, defaulting to engine.logger. epoch_level (bool) – execute lr_scheduler.step() after every epoch or every iteration. True is epoch level, False is iteration level. step_transform (Callable[[Engine], Any]) – a callable that is used to transform the information from engine to expected input data of lr_scheduler.step() function if necessary. TypeError – When step_transform is not callable. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Validation handler#

Attach validator to the trainer engine in Ignite. It can support to execute validation every N epochs or every N iterations. interval (int) – do validation every N epochs or every N iterations during training. validator (Optional[Evaluator]) – run the validator when trigger validation, suppose to be Evaluator. if None, should call set_validator() before training. epoch_level (bool) – execute validation every N epochs or N iterations. True is epoch level, False is iteration level. TypeError – When validator is not a monai.engines.evaluator.Evaluator. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Set validator if not setting in the __init__().

SmartCache handler#

Attach SmartCache logic to the engine in Ignite. Mainly include the start, update_cache, and shutdown functions of SmartCacheDataset. smartcacher (SmartCacheDataset) – predefined SmartCacheDataset, will attach it to the engine. TypeError – When smartcacher is not a monai.data.SmartCacheDataset. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Callback for train or validation/evaluation completed Event. Stop the replacement thread of SmartCacheDataset. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Callback for train or validation/evaluation epoch completed Event. Update cache content with replacement data. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Callback for train or validation/evaluation started Event. Start the replacement thread of SmartCacheDataset. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Parameter Scheduler handler#

General purpose scheduler for parameters values. By default it can schedule in a linear, exponential, step or multistep function. One can also pass Callables to have customized scheduling logic. parameter_setter (Callable) – Function that sets the required parameter value_calculator (Union[str,Callable]) – Either a string (‘linear’, ‘exponential’, ‘step’ or ‘multistep’) or Callable for custom logic. vc_kwargs (Dict) – Dictionary that stores the required parameters for the value_calculator. epoch_level (bool) – Whether the step is based on epoch or iteration. Defaults to False. name (Optional[str]) – Identifier of logging.logger to use, if None, defaulting to engine.logger. event (Optional[str]) – Event to which the handler attaches. Defaults to Events.ITERATION_COMPLETED. engine (Engine) – Ignite Engine that is used for training. None

EarlyStop handler#

EarlyStopHandler acts as an Ignite handler to stop training if no improvement after a given number of events. It‘s based on the EarlyStopping handler in ignite. patience (int) – number of events to wait if no improvement and then stop the training. score_function (Callable) – It should be a function taking a single argument, an Engine object that the handler attached, can be a trainer or validator, and return a score float. an improvement is considered if the score is higher. trainer (Optional[Engine]) – trainer engine to stop the run if no improvement, if None, must call set_trainer() before training. min_delta (float) – a minimum increase in the score to qualify as an improvement, i.e. an increase of less than or equal to min_delta, will count as no improvement. cumulative_delta (bool) – if True, min_delta defines an

increase since the last patience reset, otherwise, it defines an increase after the last event, default to False. epoch_level (bool) – check early stopping for every epoch or every iteration of the attached engine, True is epoch level, False is iteration level, default to epoch level. Note If in distributed training and uses loss value of every iteration to detect early stopping, the values may be different in different ranks. User may attach this handler to validator engine to detect validation metrics and stop the training, in this case, the score_function is executed on validator engine and trainer is the trainer engine. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Set trainer to execute early stop if not setting properly in __init__().

GarbageCollector handler#

Run garbage collector after each epoch trigger_event (str) – the event that trigger a call to this handler. - “epoch”, after completion of each epoch (equivalent of ignite.engine.Events.EPOCH_COMPLETED) - “iteration”, after completion of each iteration (equivalent of ignite.engine.Events.ITERATION_COMPLETED) - any ignite built-in event from ignite.engine.Events. Defaults to “epoch”. log_level (int) – log level (integer) for some garbage collection information as below. Defaults to 10 (DEBUG). - 50 (CRITICAL) - 40 (ERROR) - 30 (WARNING) - 20 (INFO) - 10 (DEBUG) - 0 (NOTSET)

Post processing#

Ignite handler to execute additional post processing after the post processing in engines. So users can insert other handlers between engine postprocessing and this post processing handler. If using components from monai.transforms as the transform, recommend to decollate engine.state.batch and engine.state.batch in the engine(set decollate=True) or in the DecollateBatch handler first. transform (Callable) – callable function to execute on the engine.state.batch and engine.state.output. can also be composed transforms. event (str) – expected EVENT to attach the handler, should be “MODEL_COMPLETED” or “ITERATION_COMPLETED”. default to “MODEL_COMPLETED”. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

Decollate batch#

Ignite handler to execute the decollate batch logic for engine.state.batch and engine.state.output. Typical usage is to set decollate=False in the engine and execute some postprocessing logic first then decollate the batch, otherwise, engine will decollate batch before the postprocessing. event (str) – expected EVENT to attach the handler, should be “MODEL_COMPLETED” or “ITERATION_COMPLETED”. default to “MODEL_COMPLETED”. detach (bool) – whether to detach the tensors. scalars tensors will be detached into number types instead of torch tensors. decollate_batch (bool) – whether to decollate engine.state.batch of ignite engine. batch_keys (Union[Collection[Hashable], Hashable, None]) – if decollate_batch=True, specify the keys of the corresponding items to decollate in engine.state.batch, note that it will delete other keys not specified. if None, will decollate all the keys. it replicates the scalar values to every item of the decollated list. decollate_output (bool) – whether to decollate engine.state.output of ignite engine. output_keys (Union[Collection[Hashable], Hashable, None]) – if decollate_output=True, specify the

keys of the corresponding items to decollate in engine.state.output, note that it will delete other keys not specified. if None, will decollate all the keys. it replicates the scalar values to every item of the decollated list. allow_missing_keys (bool) – don't raise exception if key is missing. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None

MLFlow handler#

MLFlowHandler defines a set of Ignite Event-handlers for the MLFlow tracking logics. It can be used for any Ignite Engine(trainer, validator and evaluator). And it can track both epoch level and iteration level logging, then MLFlow can store the data and visualize. The expected data source is Ignite engine.state.output and engine.state.metrics. When EPOCH_COMPLETED, track each dictionary item in engine.state.metrics in MLFlow. When ITERATION_COMPLETED, track expected item in self.output_transform(engine.state.output) in MLFlow, default to Loss. Usage example is available in the tutorial: Project-MONAI/tutorials. tracking_uri (Optional[str]) – connects to a tracking URI. can also set the MLFLOW_TRACKING_URI environment variable to have MLflow find a URI from there. in both cases, the URI can either be an HTTP/HTTPS URI for a remote server, a database connection string, or a local path to log data to a directory. The URI defaults to path mlruns. for more details: https://mlflow.org/docs/latest/python_api/mlflow.html#mlflow.set_tracking_uri. iteration_log (bool) – whether to log data to MLFlow when iteration completed, default to True. epoch_log (bool) – whether to log data to MLFlow when epoch completed, default to True. epoch_logger (Optional[Callable[[Engine], Any]]) – customized callable logger for epoch level logging with MLFlow. Must accept parameter "engine", use default logger if None. iteration_logger (Optional[Callable[[Engine], Any]]) – customized callable logger for iteration level logging with MLFlow. Must accept parameter "engine", use default logger if None. output_transform (Callable) – a callable that is used to transform the ignite.engine.state.output into a scalar to track, or a dictionary of {key: scalar}. By default this value logging happens when every iteration completed. The default behavior is to track loss from output[0] as output is a decollated list and we replicated loss value for every item of the decollated list. engine.state and output_transform inherit from the ignite concept: https://pytorch.org/ignite/concepts.html#state, explanation and usage example are in the tutorial: Project-MONAI/tutorials. global_epoch_transform (Callable) – a callable that is used to customize global epoch number. For example, in evaluation, the evaluator engine might want to track synced epoch number with the trainer engine. state_attributes (Optional[Sequence[str]]) – expected attributes from engine.state, if provided, will extract them when epoch completed. tag_name (str) – when iteration output is a scalar, tag_name is used to track, defaults to 'Loss'. experiment_name (str) – name for an experiment, defaults to default_experiment. run_name (Optional[str]) – name for run in an experiment. experiment_param (Optional[Dict]) – a dict recording parameters which will not change through whole experiment, like torch version, cuda version and so on. artifacts (Union[str, Sequence[Path], None]) – paths to images that need to be recorded after a whole run. optimizer_param_names (Union[str, Sequence[str]]) – parameters' name in optimizer that need to be record during running, defaults to "lr". close_on_complete (bool) – whether to close the mlflow run in complete phase in workflow, default to False. For more details of MLFlow usage, please refer to: https://mlflow.org/docs/latest/index.html. Register a set of Ignite Event-Handlers to a specified Ignite engine. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Stop current running logger of

MLFlow. None Handler for train or validation/evaluation completed Event. None Handler for train or validation/evaluation epoch completed Event. Track epoch level log, default values are from Ignite engine.state.metrics dict. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Handler for train or validation/evaluation iteration completed Event. Track iteration level log. engine (Engine) – Ignite Engine, it can be a trainer, validator or evaluator. None Check MLFlow status and start if not active. None

NVTX Handlers#

Wrapper around NVIDIA Tools Extension for profiling MONAI ignite workflow Mark an instantaneous event that occurred at some point. msg (Optional[str]) – ASCII message to associate with range Add an NVTX mark to a specific Ignite event :type engine: Engine :param engine: Ignite Engine, it can be a trainer, validator or evaluator. None Attach a NVTX range to a pair of Ignite events. It pushes an NVTX range at the first event and pops it at the second event. Stores zero-based depth of the range that is started. events (Union[str, Tuple[Union[str, Events], Union[str, Events]]]) – a string, pair of Ignite events, pair of Ignite event literals, or pair of Ignite events and literals. If a single string is provided, it should describe the base name of a pair of default Ignite events with _STARTED and _COMPLETED postfix (like “EPOCH” for Events.EPOCH_STARTED and Events.EPOCH_COMPLETED). The accepted events are: BATCH, ITERATION, EPOCH, and ENGINE. If pair of literals, each should be the literal equivalent of an Ignite event, fo instance: (“EPOCH_STARTED” and “EPOCH_COMPLETED”). One can combine events and literals, like (Events.EPOCH_STARTED and “EPOCH_COMPLETED”). For the complete list of Events, check https://pytorch.org/ignite/generated/ignite.engine.events.Events.html. msg (Optional[str]) – ASCII message to associate with range. If not provided, the name of first event will be assigned to the NVTX range. Attach an NVTX Range to specific Ignite events :type engine: Engine :param engine: Ignite Engine, it can be a trainer, validator or evaluator. None Create pair of Ignite events from a event prefix name Tuple[Events, Events] Resolve the input events to create a pair of Ignite events Tuple[Events, Events] At a specific event, pop a previously pushed range. Stores zero-based depth of the range that is started. msg – ASCII message to associate with range Pop an NVTX range at a specific Ignite event :type engine: Engine :param engine: Ignite Engine, it can be a trainer, validator or evaluator. None At a specific event, pushes a range onto a stack of nested range span. Stores zero-based depth of the range that is started. msg (Optional[str]) – ASCII message to associate with range Push an NVTX range at a specific Ignite event :type engine: Engine :param engine: Ignite Engine, it can be a trainer, validator or evaluator. None

Utilities#

Utility function to simplify the batch_transform or output_transform args of ignite components when handling dictionary or list of dictionaries(for example: engine.state.batch or engine.state.output). Users only need to set the expected keys, then it will return a callable function to extract data from dictionary and construct a tuple respectively. If data is a list of dictionaries after decollating, extract expected keys and construct lists respectively, for example, if data is [{“A”: 1, “B”: 2}, {“A”: 3, “B”: 4}], from_engine([“A”, “B”]): ([1, 3], [2, 4]). It can help avoid a complicated lambda function and make the arg of metrics more straight-forward. For example, set the first

key as the prediction and the second key as label to get the expected data from engine.state.output for a metric: keys (Union[Collection[Hashable], Hashable]) – specified keys to extract data from dictionary or decollated list of dictionaries. first (bool) – whether only extract specified keys from the first item if input data is a list of dictionaries, it's used to extract the scalar data which doesn't have batch dim and was replicated into every dictionary when decollating, like loss, etc. Returns a stopping function for ignite.handlers.EarlyStopping using the loss value. Returns a stopping function for ignite.handlers.EarlyStopping using the given metric name. Utility function to write the metrics into files, contains 3 parts: 1. if metrics dict is not None, write overall metrics into file, every line is a metric name and value pair. 2. if metric_details dict is not None, write raw metric data of every image into file, every line for 1 image. 3. if summary_ops is not None, compute summary based on operations on metric_details and write to file. save_dir (Union[str, PathLike]) – directory to save all the metrics reports. images (Optional[Sequence[str]]) – name or path of every input image corresponding to the metric_details data. if None, will use index number as the filename of every input image. metrics (Optional[Dict[str, Union[Tensor, ndarray]]]) – a dictionary of (metric name, metric value) pairs. metric_details (Optional[Dict[str, Union[Tensor, ndarray]]]) – a dictionary of (metric name, metric raw values) pairs, usually, it comes from metrics computation, for example, the raw value can be the mean_dice of every channel of every input image. summary_ops (Union[str, Sequence[str], None]) – expected computation operations to generate the summary report. it can be: None, "*" or list of strings, default to None. None - don't generate summary report for every expected metric_details. "*" - generate summary report for every metric_details with all the supported operations. list of strings - generate summary report for every metric_details with specified operations, they should be within list: ["mean", "median", "max", "min", "percentile", "std", "notnans"]. the number in "percentile" should be [0, 100], like: "15percentile". default: "90percentile". for more details, please check: https://numpy.org/doc/stable/reference/generated/numpy.nanpercentile.html. note that: for the overall summary, it computes nanmean of all classes for each image first, then compute summary. example of the generated summary report: class mean median max 5percentile 95percentile notnans class0 6.0000 6.0000 7.0000 5.1000 6.9000 2.0000 class1 6.0000 6.0000 6.0000 6.0000 6.0000 1.0000 mean 6.2500 6.2500 7.0000 5.5750 6.9250 2.0000 expected computation operations to generate the summary report. it can be: None, "*" or list of strings, default to None. None - don't generate summary report for every expected metric_details. "*" - generate summary report for every metric_details with all the supported operations. list of strings - generate summary report for every metric_details with specified operations, they should be within list: ["mean", "median", "max", "min", "percentile", "std", "notnans"]. the number in "percentile" should be [0, 100], like: "15percentile". default: "90percentile". for more details, please check: https://numpy.org/doc/stable/reference/generated/numpy.nanpercentile.html. note that: for the overall summary, it computes nanmean of all classes for each image first, then compute summary. example of the generated summary report: deli (str) – the delimiter character in the saved file, default to "," as the default output type is csv. to be consistent with: https://docs.python.org/3/library/csv.html#csv.Dialect.delimiter. output_type (str) – expected output file type, supported types: ["csv"], default to "csv".

Probability Map Handlers#

Event handler triggered on completing every iteration to calculate and save the probability map. This handler use metadata from MetaTensor to create the probability

map. This can be simply achieved by using monai.data.SlidingPatchWSIDataset or monai.data.MaskedPatchWSIDataset as the dataset. output_dir (str) – output directory to save probability maps. output_postfix (str) – a string appended to all output file names. prob_key (str) – the key associated to the probability output of the model 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 previous Inference methods next Visualizations