Configurations#

Config information of the PyTorch ignite package. Get system info as an ordered dictionary. OrderedDict Print the package versions to file. file – print() text stream file. Defaults to sys.stdout. Print config (installed dependencies, etc.) and system info for debugging. file (TextIO) – print() text stream file. Defaults to sys.stdout. None Print GPU info to file. file (TextIO) – print() text stream file. Defaults to sys.stdout. None Print system info to file. Requires the optional library, psutil. file (TextIO) – print() text stream file. Defaults to sys.stdout. None

Module utils#

Raised when called function or method requires a more recent PyTorch version than that installed. Could not import APIs from an optional dependency. Calculates the Damerau–Levenshtein distance between two strings for spelling correction. https://en.wikipedia.org/wiki/Damerau–Levenshtein_distance int Returns True if the module's __version__ matches version_str bool Make the decorated object a member of the named module. This will also add the object under its aliases if it has a __aliases__ member, thus this decorator should be before the alias decorator to pick up those names. Alias names which conflict with package names or existing members will be ignored. Utility to get the full path name of a class or object type. Try to load package and get version. If not found, return default. tuple of ints represents the pytorch major/minor version. Create an object instance or call a callable object from a class or function represented by _path. kwargs will be part of the input arguments to the class constructor or function. The target component must be a class or a function, if not, return the component directly. __path (str) – if a string is provided, it's interpreted as the full path of the target class or function component. If a callable is provided, __path(**kwargs) or functools.partial(__path, **kwargs) will be returned. __mode (str) – the operating mode for invoking the (callable) component represented by __path: "default": returns component(**kwargs) "partial": returns functools.partial(component, **kwargs) "debug": returns pdb.runcall(component, **kwargs) the operating mode for invoking the (callable) component represented by __path: "default": returns component(**kwargs) "partial": returns functools.partial(component, **kwargs) "debug": returns pdb.runcall(component, **kwargs) kwargs (Any) – keyword arguments to the callable represented by __path. Any Traverse the source of the module structure starting with module basemod, loading all packages plus all files if load_all is True, excluding anything whose name matches exclude_pattern. tuple[list[module], list[str]] Look up the option in the supported collection and return the matched item. Raise a value error possibly with a guess of the closest match. opt_str – The option string or Enum to look up. supported – The collection of supported options, it can be list, tuple, set, dict, or Enum. default – If it is given, this method will return default when opt_str is not found, instead of raising a ValueError. Otherwise, it defaults to "no_default", so that the method may raise a ValueError. print_all_options – whether to print all available options when opt_str is not found. Defaults to True Examples: Adapted from NifTK/NiftyNet Convert version strings into tuples of int and compare them. Returns True if the module's version is greater or equal to the 'min_version'. When min_version_str is not provided, it always returns True. bool Imports an optional module specified by module string. Any importing related exceptions will be stored, and exceptions raise lazily when attempting to use the failed-to-import module. module (str) – name of the module to be imported. version (str) – version string used by the version_checker. version_checker (Callable[…, bool]) – a callable to check the module version, Defaults to monai.utils.min_version. name (str) – a non-module attribute (such as method/class) to import from the imported module. descriptor (str) – a format string for the final error message when using a not imported module. version_args (Optional[Any]) – additional parameters to the version checker. allow_namespace_pkg (bool) – whether importing a namespace package is allowed. Defaults to False. as_type (str) – there are cases where the optionally imported object is used as a base class, or a decorator, the exceptions should raise accordingly. The current supported values are “default” (call once to raise), “decorator” (call the constructor and the second call to raise), and anything else will return a lazy class that can be used as a base class (call the constructor to raise). tuple[Any, bool] The imported module and a boolean flag indicating whether the import is successful. Examples: Compute whether the current pytorch version is after or equal to the specified version. The current system pytorch version is determined by torch.__version__ or via system environment variable PYTORCH_VER. major – major version number to be compared with minor – minor version number to be compared with patch – patch version number to be compared with current_ver_string – if None, torch.__version__ will be used. True if the current pytorch version is greater than or equal to the specified version. Decorator function to check the required package installation. pkg_name (str) – required package name, like: “itk”, “nibabel”, etc. version (str) – required version string used by the version_checker. version_checker (Callable[…, bool]) – a callable to check the module version, defaults to monai.utils.min_version. raise_error (bool) – if True, raise OptionalImportError error if the required package is not installed or the version doesn’t match requirement, if False, print the error in a warning. Callable Returns True if version lhs is earlier or equal to rhs. lhs (str) – version name to compare with rhs, return True if earlier or equal to rhs. rhs (str) – version name to compare with lhs, return True if later or equal to lhs. bool

Aliases#

This module is written for configurable workflow, not currently in use. Stores the decorated function or class in the global aliases table under the given names and as the __aliases__ member of the decorated object. This new member will contain all alias names declared for that object. Search for the declaration (function or class) with the given name. This will first search the list of aliases to see if it was declared with this aliased name, then search treating name as a fully qualified name, then search the loaded modules for one having a declaration with the given name. If no declaration is found, raise ValueError. ValueError – When the module is not found. ValueError – When the module does not have the specified member. ValueError – When multiple modules with the declaration name are found. ValueError – When no module with the specified member is found.

Misc#

Convert the values from input unit to the target unit input_unit (str) – the unit of the input quantity target_unit (str) – the unit of the target quantity Common key names in the metadata header of images Environment variables used by MONAI. Checks if there is a duplicated key in the sequence of ordered_pairs. If there is - it will log a warning or raise ValueError (if configured by environmental var MONAI_FAIL_ON_DUPLICATE_CONFIG==1) Otherwise, it returns the dict made from this sequence. Satisfies a format for an object_pairs_hook in json.load ordered_pairs (Sequence[tuple[Any, Any]]) – sequence of (key, value) dict[Any, Any]

Check if the all keys in kwargs exist in the __init__ method of the class. cls – the class to check. kwargs – kwargs to examine. a boolean indicating if all keys exist. a set of extra keys that are not used in the __init__. Utility to check whether the parent directory of the path exists. path (Union[str, PathLike]) – input path to check the parent directory. create_dir (bool) – if True, when the parent directory doesn't exist, create the directory, otherwise, raise exception. None Copy object or tuple/list/dictionary of objects to device. obj – object or tuple/list/dictionary of objects to move to device. device – move obj to this device. Can be a string (e.g., cpu, cuda, cuda:0, etc.) or of type torch.device. non_blocking – when True, moves data to device asynchronously if possible, e.g., moving CPU Tensors with pinned memory to CUDA devices. verbose – when True, will print a warning for any elements of incompatible type not copied to device. Same as input, copied to device where possible. Original input will beunchanged. unchanged. Returns a tuple of vals. vals (Any) – input data to convert to a tuple. wrap_array (bool) – if True, treat the input numerical array (ndarray/tensor) as one item of the tuple. if False, try to convert the array with tuple(vals), default to False. tuple Returns a copy of tup with dim values by either shortened or duplicated input. ValueError – When tup is a sequence and tup length is not dim. Examples: tuple[Any, …] Returns a copy of tup with dim values by either shortened or padded with pad_val as necessary. tuple Refine user_provided according to the default, and returns as a validated tuple. The validation is done for each element in user_provided using func. If func(user_provided[idx]) returns False, the corresponding default[idx] will be used as the fallback. Typically used when user_provided is a tuple of window size provided by the user, default is defined by data, this function returns an updated user_provided with its non-positive components replaced by the corresponding components from default. user_provided – item to be validated. default – a sequence used to provided the fallbacks. func – a Callable to validate every components of user_provided. Examples: Returns the first item in the given iterable or default if empty, meaningful mostly with 'for' expressions. Return a boolean indicating whether the given callable obj has the keywords in its signature. Determine if the object is an immutable object. see also python/cpython bool Determine if a module's version is at least equal to the given value. module – imported module's name, e.g., np or torch. version – required version, given as a tuple, e.g., (1, 8, 0). True if module is the given version or newer. Determine if the object is an iterable sequence and is not a string. bool Compute the union of class IDs in label and generate a list to include all class IDs :param x: a list of numbers (for example, class_IDs) a list showing the union (the union the class IDs) To convert a list of "key=value" pairs into a dictionary. For examples: items: ["a=1", "b=2", "c=3"], return: {"a": "1", "b": "2", "c": "3"}. If no "=" in the pair, use None as the value, for example: ["a"], return: {"a": None}. Note that it will remove the blanks around keys and values. Convert a file path to URI. if not absolute path, will convert to absolute path first. path (Union[str, PathLike]) – input file path to convert, can be a string or Path object. str Pretty print the head and tail n_lines of val, and omit the middle part if the part has more than 3 lines. Returns: the formatted string. str print a progress bar to track some time consuming task. index – current status in progress. count – total steps of the progress. desc – description of the progress bar, if not None, show before the progress bar. bar_len – the total length of the bar on screen, default is 30 char. newline – whether to print in a new line for every index. sample several slices of input numpy array or Tensor on specified dim. data (Union[ndarray, Tensor]) – input data to sample slices, can be numpy array or PyTorch Tensor. dim (int) – expected dimension index to sample slices, default to 1. as_indices (bool) – if True, slicevals arg will be treated as the expected indices of slice, like: 1, 3, 5 means data[…, [1, 3, 5], …], if False, slicevals arg will be treated as args for slice func, like: 1, None means data[…, [1:], …], 1, 5 means data[…, [1: 5], …]. slicevals (int) – indices of slices or start and end indices of expected slices, depends on as_indices flag. Union[ndarray, Tensor] Save an object to file with specified path. Support to serialize to a temporary file first, then move to final destination, so that files are guaranteed to not be damaged if exception occurs. obj – input object data to save. path – target file path to save the input object. create_dir – whether to create dictionary of the path if not existing, default to True. atomic – if True, state is serialized to a temporary file first, then move to final destination. so that files are guaranteed to not be damaged if exception occurs. default to True. func – the function to save file, if None, default to torch.save. kwargs – other args for the save func except for the checkpoint and filename. default func is torch.save(), details of other args: https://pytorch.org/docs/stable/generated/torch.save.html. Set random seed for modules to enable or disable deterministic training. seed – the random seed to use, default is np.iinfo(np.int32).max. It is recommended to set a large seed, i.e. a number that has a good balance of 0 and 1 bits. Avoid having many 0 bits in the seed. if set to None, will disable deterministic training. use_deterministic_algorithms – Set whether PyTorch operations must use "deterministic" algorithms. additional_settings – additional settings that need to set random seed. Note This function will not affect the randomizable objects in monai.transforms.Randomizable, which have independent random states. For those objects, the set_random_state() method should be used to ensure the deterministic behavior (alternatively, monai.data.DataLoader by default sets the seeds according to the global random state, please see also: monai.data.utils.worker_init_fn and monai.data.utils.set_rnd). Use starmap as the mapping function in zipWith. Convert a string to a boolean. Case insensitive. True: yes, true, t, y, 1. False: no, false, f, n, 0. value – string to be converted to a boolean. If value is a bool already, simply return it. raise_exc – if value not in tuples of expected true or false inputs, should we raise an exception? If not, return default. raise_exc is True. parser.add_argument("–convert", default=False, type=str2bool) python mycode.py –convert=True parser.add_argument("–blocks", default=[1,2,3], type=str2list) python mycode.py –blocks=1,2,2,4 value – string (comma separated) to be converted to a list raise_exc – if not possible to convert to a list, raise an exception ValueError: value not a string or list or not possible to convert Given a dictionary whose values contain scalars or tuples (with the same length as keys), Create a dictionary for each key containing the scalar values mapping to that key. dictionary_of_tuples (dict) – a dictionary whose values are scalars or tuples whose length is the length of keys keys (Any) – a tuple of string values representing the keys in question tuple[dict[Any, Any], …] a tuple of dictionaries that contain scalar values, one dictionary for each key ValueError – when values in the dictionary are tuples but not the same length as the length of keys – Examples Map op, using mapfunc, to each tuple derived from zipping the iterables in vals.

NVTX Annotations#

Decorators and context managers for NVIDIA Tools Extension to profile MONAI components A decorator and context manager for NVIDIA Tools Extension (NVTX) Range for profiling. When used as a decorator it encloses a specific method of the object with an NVTX Range. When used as a context manager, it encloses the runtime context (created by with statement) with an NVTX Range. name – the name to be associated to the range methods – (only when used as decorator) the name of a method (or a list of the name of the methods) to be wrapped by NVTX range. If None (default), the method(s) will be inferred based on the object's type for various MONAI components, such as Networks, Losses, Functions, Transforms, and Datasets. Otherwise, it look up predefined methods: "forward", "__call__", "__next__", "__getitem__" append_method_name – if append the name of the methods to be decorated to the range’s name If None (default), it appends the method’s name only if we are annotating more than one method. recursive – if set to True, it will recursively annotate every individual module in a list or in a chain of modules (chained using Compose). Default to False.

Profiling#

Context manager for tracking how much time is spent within context blocks. This uses time.perf_counter to accumulate the total amount of time in seconds in the attribute total_time over however many context blocks the object is used in. Handler for Ignite Engine classes which measures the time from a start event ton an end event. This can be used to profile epoch, iteration, and other events as defined in ignite.engine.Events. This class should be used only within the context of a profiler object. name – name of event to profile profiler – instance of WorkflowProfiler used by the handler, should be within the context of this object start_event – item in ignite.engine.Events stating event at which to start timing end_event – item in ignite.engine.Events stating event at which to stop timing Profiler for timing all aspects of a workflow. This includes using stack tracing to capture call times for all selected calls (by default calls to Transform.__call__ methods), times within context blocks, times to generate items from iterables, and times to execute decorated functions. This profiler must be used only within its context because it uses an internal thread to read results from a multiprocessing queue. This allows the profiler to function across multiple threads and processes, though the multiprocess tracing is at times unreliable and not available in Windows at all. The profiler uses sys.settrace and threading.settrace to find all calls to profile, this will be set when the context enters and cleared when it exits so proper use of the context is essential to prevent excessive tracing. Note that tracing has a high overhead so times will not accurately reflect real world performance but give an idea of relative share of time spent. The tracing functionality uses a selector to choose which calls to trace, since tracing all calls induces infinite loops and would be terribly slow even if not. This selector is a callable accepting a call trace frame and returns True if the call should be traced. The default is select_transform_call which will return True for Transform.__call__ calls only. Example showing use of all profiling functions: call_selector – selector to determine which calls to trace, use None to disable tracing Add a result in a thread-safe manner to the internal results dictionary. None Save all results to a csv file. Get a fresh results dictionary containing fresh tuples of ProfileResult objects. Returns a dictionary mapping results entries to tuples containing the number of items, time sum, time average, time std dev, time min, and time max. Returns the same information as get_times_summary but in a Pandas DataFrame. Decorator which can be applied to a function which profiles any calls to it. All calls to decorated callables must be done within the context of the profiler. Creates a context to profile, placing a timing result onto the queue when it exits. Wrapper around anything iterable to profile how long it takes to generate items. Returns True if frame is a call to a Transform object’s _call__ method. A decorator which will run the torch profiler for the decorated function, printing the results in full. Note: Enforces a gpu sync point which could slow down pipelines. A decorator which measures the execution time of both the CPU and GPU components of the decorated function, printing both results. Note: Enforces a gpu sync point which could slow down pipelines. A decorator which measures the total execution time from when the decorated function is called to when the last cuda operation finishes, printing the result. Note: Enforces a gpu sync point which could slow down pipelines.

Deprecated#

Marks a function or class as deprecated. If since is given this should be a version at or earlier than the current version and states at what version of the definition was marked as deprecated. If removed is given this can be any version and marks when the definition was removed. When the decorated definition is called, that is when the function is called or the class instantiated, a warning_category is issued if since is given and the current version is at or later than that given. a DeprecatedError exception is instead raised if removed is given and the current version is at or later than that, or if neither since nor removed is provided. The relevant docstring of the deprecating function should also be updated accordingly, using the Sphinx directives such as .. versionchanged:: version and .. deprecated:: version. https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-versionadded since – version at which the definition was marked deprecated but not removed. removed – version at which the definition was/will be removed and no longer usable. msg_suffix – message appended to warning/exception detailing reasons for deprecation and what to use instead. version_val – (used for testing) version to compare since and removed against, default is MONAI version. warning_category – a warning category class, defaults to FutureWarning. Decorated definition which warns or raises exception when used Marks a particular named argument of a callable as deprecated. The same conditions for since and removed as described in the deprecated decorator. When the decorated definition is called, that is when the function is called or the class instantiated with args, a warning_category is issued if since is given and the current version is at or later than that given. a DeprecatedError exception is instead raised if removed is given and the current version is at or later than that, or if neither since nor removed is provided. The relevant docstring of the deprecating function should also be updated accordingly, using the Sphinx directives such as .. versionchanged:: version and .. deprecated:: version. https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-versionadded name – name of position or keyword argument to mark as deprecated. since – version at which the argument was marked deprecated but not removed. removed – version at which the argument was/will be removed and no longer usable. msg_suffix – message appended to warning/exception detailing reasons for deprecation and what to use instead. version_val – (used for testing) version to compare since and removed against, default is MONAI version. new_name – name of position or keyword argument to replace the deprecated argument. if it is specified and the signature of the decorated function has a kwargs, the value to the deprecated argument name will be removed. warning_category – a warning category class, defaults to FutureWarning. Decorated callable which warns or raises exception when deprecated argument used. Marks a particular arguments default of a callable as deprecated. It is changed from old_default to new_default in version changed. When the decorated definition is called, a warning_category is issued if since is given, the default is not explicitly set by the caller and the current version is at or later than that given. Another warning with the same category is issued if changed is given and the current version is at or later. The relevant docstring of the deprecating function should also be updated accordingly, using the Sphinx directives such as .. versionchanged:: version and .. deprecated:: version. https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-versionadded name – name of position or keyword argument where the default is deprecated/changed. old_default – name of the old default. This is only for the warning message, it will not be validated. new_default – name of the new default. It is validated that this value is not present as the default before version replaced. This means, that you can also use this if the actual default value is None and set later in the function. You can also set this to any string representation, e.g. “calculate_default_value()” if the default is calculated from another function. since – version at which the argument default was marked deprecated but not replaced. replaced – version at which the argument default was/will be replaced. msg_suffix – message appended to warning/exception detailing reasons for deprecation. version_val – (used for testing) version to compare since and removed against, default is MONAI version. warning_category – a warning category class, defaults to FutureWarning. Decorated callable which warns when deprecated default argument is not explicitly specified.

Type conversion#

Convert to MetaTensor, torch.Tensor or np.ndarray from MetaTensor, torch.Tensor, np.ndarray, float, int, etc. data – data to be converted output_type – monai.data.MetaTensor, torch.Tensor, or np.ndarray (if None, unchanged) device – if output is MetaTensor or torch.Tensor, select device (if None, unchanged) dtype – dtype of output data. Converted to correct library type (e.g., np.float32 is converted to torch.float32 if output type is torch.Tensor). If left blank, it remains unchanged. wrap_sequence – if False, then lists will recursively call this function. E.g., [1, 2] -> [array(1), array(2)]. If True, then [1, 2] -> array([1, 2]). safe – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. modified data, orig_type, orig_device Note When both output_type and dtype are specified with different backend (e.g., torch.Tensor and np.float32), the output_type will be used as the primary type, for example: Utility to convert the input data to a cupy array. If passing a dictionary, list or tuple, recursively check every item and convert it to cupy array. data – input data can be PyTorch Tensor, numpy array, cupy array, list, dictionary, int, float, bool, str, etc. Tensor, numpy array, cupy array, float, int, bool are converted to cupy arrays, for dictionary, list or tuple, convert every item to a numpy array if applicable. dtype – target data type when converting to Cupy array, tt must be an argument of numpy.dtype, for more details: https://docs.cupy.dev/en/stable/reference/generated/cupy.array.html. wrap_sequence – if False, then lists will recursively call this function. E.g., [1, 2] -> [array(1), array(2)]. If True, then [1, 2] -> array([1, 2]). safe – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. Convert source data to the same data type and device as the destination data. If dst is an instance of torch.Tensor or its subclass, convert src to torch.Tensor with the same data type as dst, if dst is an instance of numpy.ndarray or its subclass, convert to numpy.ndarray with the same data type as dst, otherwise, convert to the type of dst directly. src – source data to convert type. dst – destination data that convert to the same data type as it. dtype – an optional argument if the target dtype is different from the original dst’s data type. wrap_sequence – if False, then lists will recursively call this function. E.g., [1, 2] -> [array(1), array(2)]. If True, then [1, 2] -> array([1, 2]). device – target device to put the converted Tensor data. If unspecified, dst.device will be used if possible. safe – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. See also convert_data_type() Utility to convert the input data to a numpy array. If passing a dictionary, list or tuple, recursively check every item and convert it to numpy array. data (Any) – input data can be PyTorch Tensor, numpy array, list, dictionary, int, float, bool, str, etc. will convert Tensor, Numpy array, float, int, bool to numpy arrays, strings and objects keep the original. for dictionary, list or tuple, convert every item to a numpy array if applicable. dtype (Union[dtype, type, str, None]) – target data type when converting to numpy array. wrap_sequence (bool) – if False, then lists will recursively call this function. E.g., [1, 2] -> [array(1), array(2)]. If True, then [1, 2] -> array([1, 2]). safe (bool) – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. Any Utility to convert the input data to a PyTorch Tensor, if track_meta is True, the output will be a MetaTensor, otherwise, the output will be a regular torch Tensor. If passing a dictionary, list or tuple, recursively check every item and convert it to PyTorch Tensor. data – input data can be PyTorch Tensor, numpy array, list, dictionary, int, float, bool, str, etc. will convert Tensor, Numpy array, float, int, bool to Tensor, strings and objects keep the original. for dictionary, list or tuple, convert every item to a Tensor if applicable. dtype – target data type to when converting to Tensor. device – target device to put the converted Tensor data. wrap_sequence – if False, then lists will recursively call this function. E.g., [1, 2] -> [tensor(1), tensor(2)]. If True, then [1, 2] -> tensor([1, 2]). track_meta – whether to track the meta information, if True, will convert to MetaTensor. default to False. safe – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [tensor(0), tensor(244)]. If True, then [256, -12] -> [tensor(255), tensor(0)]. Convert a numpy dtype to its torch equivalent. dtype Convert a torch dtype to its numpy equivalent. dtype Get the dtype of an image, or if there is a sequence, recursively call the method on the 0th element. This therefore assumes that in a Sequence, all types are the same. Convert to the dtype that corresponds to data_type. The input dtype can also be a string. e.g., “float32” becomes torch.float32 or np.float32 as necessary. Example: Get a numpy dtype (e.g., np.float32) from its string (e.g., “float32”). dtype Get a torch dtype (e.g., torch.float32) from its string (e.g., “float32”). dtype

Decorators#

Base class for method decorators which can be used to replace methods pass to replace_method() with wrapped versions. Return a new method to replace meth in the instantiated object, or meth to do nothing. Wraps a generator callable which will be called whenever this class is iterated and its result returned. This is used to create an iterator which can start iteration over the given generator multiple times.

Distributed Data Parallel#

The RankFilter class is a convenient filter that extends the Filter class in the Python logging module. The purpose is to control which log records are processed based on the rank in a distributed environment. rank – the rank of the process in the torch.distributed. Default is None and then it will use dist.get_rank(). filter_fn – an optional lambda function used as the filtering criteria. The default function logs only if the rank of the process is 0, but the user can define their own function to implement custom filtering logic. Determine if the specified record is to be logged. Returns True if the record should be logged, or False otherwise. If deemed appropriate, the record may be modified in-place. Utility function for distributed data parallel to pad at first dim to make it evenly divisible and all_gather. The input data of every rank should have the same number of dimensions, only the first dim can be different. Note: If has ignite installed, will execute based on ignite distributed APIs, otherwise, if the native PyTorch distributed group initialized, will execute based on native PyTorch distributed APIs. data – source tensor to pad and execute all_gather in distributed data parallel.

concat – whether to concat the gathered list to be a Tensor, if False, return a list of Tensors, similar behavior as torch.distributed.all_gather(). default to True. Note The input data on different ranks must have exactly same dtype. Get the expected target device in the native PyTorch distributed data parallel. For NCCL backend, return GPU device of current process. For GLOO backend, return CPU. For any other backends, return None as the default, tensor.to(None) will not change the device. Utility function for distributed data parallel to all gather a list of strings. Refer to the idea of ignite all_gather(string): https://pytorch.org/ignite/v0.4.5/distributed.html#ignite.distributed.utils.all_gather. Note: If has ignite installed, will execute based on ignite distributed APIs, otherwise, if the native PyTorch distributed group initialized, will execute based on native PyTorch distributed APIs. strings (list[str]) – a list of strings to all gather. delimiter (str) – use the delimiter to join the string list to be a long string, then all gather across ranks and split to a list. default to “ “. list[str]

Enums#

Default keys for Mixed Ensemble Default keys for templated Auto3DSeg Algo. ID is the identifier of the algorithm. The string has the format of __. ALGO is the Auto3DSeg Algo instance. IS_TRAINED is the status that shows if the Algo has been trained. SCORE is the score the Algo has achieved after training. See also: monai.metrics.rocauc.compute_roc_auc See also: monai.data.utils.compute_importance_map Box mode names. Bundle property fields: DESC is the description of the property. REQUIRED is flag to indicate whether the property is required or optional. additional bundle property fields for config based bundle workflow: ID is the config item ID of the property. REF_ID is the ID of config item which is supposed to refer to this property. this field is only useful to check the optional property ID. See also: monai.networks.nets.HighResBlock Enums for color order. Expand as necessary. A set of common keys for dictionary based supervised training process. IMAGE is the input image data. LABEL is the training or evaluation label of segmentation or classification task. PRED is the prediction data of model output. LOSS is the loss value of current iteration. INFO is some useful information during training or evaluation, like loss value, etc. Mode names for instantiating a class or calling a callable. See also: monai.utils.module.instantiate() Defaults keys for dataset statistical analysis modules See also monai.losses.dice.DiceCELoss Default keys for the statistics of trainer and evaluator engines. The keys to be used for extracting data from the fastMRI dataset See also: monai.transforms.engines.evaluator.Evaluator A set of common keys for generative adversarial networks. The sorting method for the generated patches in GridPatch See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html interpolation mode of torch.nn.functional.grid_sample Note (documentation from torch.nn.functional.grid_sample) mode=’bicubic’ supports only 4-D input. When mode=’bilinear’ and the input is 5-D, the interpolation mode used internally will actually be trilinear. However, when the input is 4-D, the interpolation mode will legitimately be bilinear. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html Three branches of HoVerNet model, which results in three outputs: HV is horizontal and vertical gradient map of each nucleus (regression), NP is the pixel prediction of all nuclei (segmentation), and NC is the type of each nucleus (classification). Modes for HoVerNet model: FAST: a faster implementation (than original) ORIGINAL: the original implementation Defaults keys for dataset statistical analysis image modules See also:

https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html Defaults keys for dataset statistical analysis label modules MetaTensor with pending operations requires some key attributes tracked especially when the primary array is not up-to-date due to lazy evaluation. This class specifies the set of key attributes to be tracked for each MetaTensor. See also: monai.transforms.lazy.utils.resample() for more details. See also monai.losses.dice.DiceLoss monai.losses.dice.GeneralizedDiceLoss monai.losses.focal_loss.FocalLoss monai.losses.tversky.TverskyLoss Typical keys for MetaObj.meta See also: monai.transforms.croppad.array.SpatialPad See also: monai.metrics.utils.do_metric_reduction() The available options determine how the input array is extended beyond its boundaries when interpolating. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html Post-fixes. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html See also: monai.networks.layers.SkipConnection The coordinate system keys, for example, Nifti1 uses Right-Anterior-Superior or “RAS”, DICOM (0020,0032) uses Left-Posterior-Superior or “LPS”. This type does not distinguish spatial 1/2/3D. Order of spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html Enum subclass that converts its value to a string. Extra metadata keys used for traceable transforms. Enumerable status keys for the TraceKeys.STATUS flag Transform backends. Most of monai.transforms components first converts the input data into torch.Tensor or monai.data.MetaTensor. Internally, some transforms are made by converting the data into numpy.array or cupy.array and use the underlying transform backend API to achieve the actual output array and converting back to Tensor/MetaTensor. Transforms with more than one backend indicate the that they may convert the input data types to accommodate the underlying API. See also: monai.networks.blocks.UpSample See also: monai.losses.dice.GeneralizedDiceLoss

Jupyter Utilities#

This set of utility function is meant to make using Jupyter notebooks easier with MONAI. Plotting functions using Matplotlib produce common plots for metrics and images. Named members of the status dictionary, others may be present for named metric values. Contains a running Engine object within a separate thread from main thread in a Jupyter notebook. This allows an engine to begin a run in the background and allow the starting notebook cell to complete. A user can thus start a run and then navigate away from the notebook without concern for loosing connection with the running cell. All output is acquired through methods which synchronize with the running engine using an internal lock member, acquiring this lock allows the engine to be inspected while it’s prevented from starting the next iteration. engine (Engine) – wrapped Engine object, when the container is started its run method is called loss_transform (Callable) – callable to convert an output dict into a single numeric value metric_transform (Callable) – callable to convert a named metric value into a single numeric value status_format (str) – format string for status key-value pairs. Generate a plot of the current status of the contained engine whose loss and metrics were tracked by logger. The function plot_func must accept arguments title, engine, logger, and fig which are the plot title, self.engine, logger, and self.fig respectively. The return value must be a figure object (stored in self.fig) and a list of Axes objects for the plots in the figure. Only the figure is returned by this method, which holds the internal lock during the plot generation. Figure Calls the run method of the wrapped engine. Returns a status string for the current state of the engine. str A dictionary containing status information, current loss, and current metric values. dict[str, str] Stop the engine and join the thread. Plot the status of the given Engine with its logger. The plot will consist of a graph of loss values and metrics taken from the logger, and images taken from the output and batch members of engine.state. The images are converted to Numpy arrays suitable for input to Axes.imshow using image_fn, if this is None then no image plotting is done. engine – Engine to extract images from logger – MetricLogger to extract loss and metric data from title – graph title yscale – for metric plot, scale for y-axis compatible with Axes.set_yscale avg_keys – for metric plot, tuple of keys in graphmap to provide running average plots for window_fraction – for metric plot, what fraction of the graph value length to use as the running average window image_fn – callable converting tensors keyed to a name in the Engine to a tuple of images to plot fig – Figure object to plot into, reuse from previous plotting for flicker-free refreshing selected_inst – index of the instance to show in the image plot Figure object (or fig if given), list of Axes objects for graph and images Plot metrics on a single graph with running averages plotted for selected keys. The values in graphmap should be lists of (timepoint, value) pairs as stored in MetricLogger objects. ax – Axes object to plot into title – graph title graphmap – dictionary of named graph values, which are lists of values or (index, value) pairs yscale – scale for y-axis compatible with Axes.set_yscale avg_keys – tuple of keys in graphmap to provide running average plots for window_fraction – what fraction of the graph value length to use as the running average window Plot metric graph data with images below into figure fig. The intended use is for the graph data to be metrics from a training run and the images to be the batch and output from the last iteration. This uses plot_metric_graph to plot the metric graph. fig – Figure object to plot into, reuse from previous plotting for flicker-free refreshing title – graph title graphmap – dictionary of named graph values, which are lists of values or (index, value) pairs imagemap – dictionary of named images to show with metric plot yscale – for metric plot, scale for y-axis compatible with Axes.set_yscale avg_keys – for metric plot, tuple of keys in graphmap to provide running average plots for window_fraction – for metric plot, what fraction of the graph value length to use as the running average window list of Axes objects for graph followed by images Return an tuple of images derived from the given tensor. The name value indices which key from the output or batch value the tensor was stored as, or is “Batch” or “Output” if these were single tensors instead of dictionaries. Returns a tuple of 2D images of shape HW, or 3D images of shape CHW where C is color channels RGB or RGBA. This allows multiple images to be created from a single tensor, ie. to show each channel separately.

State Cacher#

Class to cache and retrieve the state of an object. Objects can either be stored in memory or on disk. If stored on disk, they can be stored in a given directory, or alternatively a temporary location will be used. If necessary/possible, restored objects will be returned to their original device. Example: Constructor. in_memory – boolean to determine if the object will be cached in memory or on disk. cache_dir – directory for data to be cached if in_memory==False. Defaults to using a temporary directory. Any created files will be deleted during the StateCacher’s destructor. allow_overwrite – allow the cache to be overwritten. If set to False, an error will be thrown if a matching already exists in the list of cached objects. pickle_module – module used for pickling metadata and objects, default to pickle. this arg is used by torch.save, for more details, please check: https://pytorch.org/docs/stable/generated/torch.save.html#torch.save. pickle_protocol – can be specified to override the default protocol, default to 2. this arg is used by torch.save, for more details, please check: https://pytorch.org/docs/stable/generated/torch.save.html#torch.save. Retrieve the object stored under a given key name. Any Store a given object with the given key name. key – key of the data object to store. data_obj – data object to store. pickle_module – module used for pickling metadata and objects, default to self.pickle_module. this arg is used by torch.save, for more details, please check: https://pytorch.org/docs/stable/generated/torch.save.html#torch.save. pickle_protocol – can be specified to override the default protocol, default to self.pickle_protocol. this arg is used by torch.save, for more details, please check: https://pytorch.org/docs/stable/generated/torch.save.html#torch.save. previous Visualizations next Installation Guide