# fastreid Documentation

*Release 1.0.0*

**fastreid contributors**

**Jan 23, 2021**

# CONTENTS

CHAPTER

ONE

# API DOCUMENTATION

## 1.1 fastreid.checkpoint

## 1.2 fastreid.config

Related tutorials: ../tutorials/configs, ../tutorials/extend.

@author: l1aoxingyu @contact: sherlockliao01@gmail.com

### 1.2.1 Config References

```
# Convention about Training / Test specific parameters
# -----------------------------------------------------------------------------
# Whenever an argument can be either used for training or for testing, the
# corresponding name will be post-fixed by a _TRAIN for a training parameter,
# or _TEST for a test-specific parameter.
# For example, the number of images during training will be
# IMAGES_PER_BATCH_TRAIN, while the number of images for testing will be
# IMAGES_PER_BATCH_TEST

# -----------------------------------------------------------------------------
# Config definition
# -----------------------------------------------------------------------------

_C = CN()

# -----------------------------------------------------------------------------
# MODEL
# -----------------------------------------------------------------------------
_C.MODEL = CN()
_C.MODEL.DEVICE = "cuda"
_C.MODEL.META_ARCHITECTURE = "Baseline"

_C.MODEL.FREEZE_LAYERS = ['']

# MoCo memory size
_C.MODEL.QUEUE_SIZE = 8192

# ---------------------------------------------------------------------------- #
# Backbone options
# ---------------------------------------------------------------------------- #
```

(continues on next page)

(continued from previous page)

```
_C.MODEL.BACKBONE = CN()

_C.MODEL.BACKBONE.NAME = "build_resnet_backbone"
_C.MODEL.BACKBONE.DEPTH = "50x"
_C.MODEL.BACKBONE.LAST_STRIDE = 1
# Backbone feature dimension
_C.MODEL.BACKBONE.FEAT_DIM = 2048
# Normalization method for the convolution layers.
_C.MODEL.BACKBONE.NORM = "BN"
# If use IBN block in backbone
_C.MODEL.BACKBONE.WITH_IBN = False
# If use SE block in backbone
_C.MODEL.BACKBONE.WITH_SE = False
# If use Non-local block in backbone
_C.MODEL.BACKBONE.WITH_NL = False
# If use ImageNet pretrain model
_C.MODEL.BACKBONE.PRETRAIN = True
# Pretrain model path
_C.MODEL.BACKBONE.PRETRAIN_PATH = ''

# ---------------------------------------------------------------------------- #
# REID HEADS options
# ---------------------------------------------------------------------------- #
_C.MODEL.HEADS = CN()
_C.MODEL.HEADS.NAME = "EmbeddingHead"
# Normalization method for the convolution layers.
_C.MODEL.HEADS.NORM = "BN"
# Number of identity
_C.MODEL.HEADS.NUM_CLASSES = 0
# Embedding dimension in head
_C.MODEL.HEADS.EMBEDDING_DIM = 0
# If use BNneck in embedding
_C.MODEL.HEADS.WITH_BNNECK = True
# Triplet feature using feature before(after) bnneck
_C.MODEL.HEADS.NECK_FEAT = "before"  # options: before, after
# Pooling layer type
_C.MODEL.HEADS.POOL_LAYER = "avgpool"

# Classification layer type
_C.MODEL.HEADS.CLS_LAYER = "linear"  # "arcSoftmax" or "circleSoftmax"

# Margin and Scale for margin-based classification layer
_C.MODEL.HEADS.MARGIN = 0.15
_C.MODEL.HEADS.SCALE = 128

# ---------------------------------------------------------------------------- #
# REID LOSSES options
# ---------------------------------------------------------------------------- #
_C.MODEL.LOSSES = CN()
_C.MODEL.LOSSES.NAME = ("CrossEntropyLoss",)

# Cross Entropy Loss options
_C.MODEL.LOSSES.CE = CN()
# if epsilon == 0, it means no label smooth regularization,
# if epsilon == -1, it means adaptive label smooth regularization
_C.MODEL.LOSSES.CE.EPSILON = 0.0
_C.MODEL.LOSSES.CE.ALPHA = 0.2
```

(continues on next page)

(continued from previous page)

```
_C.MODEL.LOSSES.CE.SCALE = 1.0

# Focal Loss options
_C.MODEL.LOSSES.FL = CN()
_C.MODEL.LOSSES.FL.ALPHA = 0.25
_C.MODEL.LOSSES.FL.GAMMA = 2
_C.MODEL.LOSSES.FL.SCALE = 1.0

# Triplet Loss options
_C.MODEL.LOSSES.TRI = CN()
_C.MODEL.LOSSES.TRI.MARGIN = 0.3
_C.MODEL.LOSSES.TRI.NORM_FEAT = False
_C.MODEL.LOSSES.TRI.HARD_MINING = True
_C.MODEL.LOSSES.TRI.SCALE = 1.0

# Circle Loss options
_C.MODEL.LOSSES.CIRCLE = CN()
_C.MODEL.LOSSES.CIRCLE.MARGIN = 0.25
_C.MODEL.LOSSES.CIRCLE.GAMMA = 128
_C.MODEL.LOSSES.CIRCLE.SCALE = 1.0

# Cosface Loss options
_C.MODEL.LOSSES.COSFACE = CN()
_C.MODEL.LOSSES.COSFACE.MARGIN = 0.25
_C.MODEL.LOSSES.COSFACE.GAMMA = 128
_C.MODEL.LOSSES.COSFACE.SCALE = 1.0

# Path to a checkpoint file to be loaded to the model. You can find available models␣
↪in the model zoo.
_C.MODEL.WEIGHTS = ""

# Values to be used for image normalization
_C.MODEL.PIXEL_MEAN = [0.485*255, 0.456*255, 0.406*255]
# Values to be used for image normalization
_C.MODEL.PIXEL_STD = [0.229*255, 0.224*255, 0.225*255]

# -----------------------------------------------------------------------------
# KNOWLEDGE DISTILLATION
# -----------------------------------------------------------------------------

_C.KD = CN()
_C.KD.MODEL_CONFIG = ""
_C.KD.MODEL_WEIGHTS = ""

# -----------------------------------------------------------------------------
# INPUT
# -----------------------------------------------------------------------------
_C.INPUT = CN()
# Size of the image during training
_C.INPUT.SIZE_TRAIN = [256, 128]
# Size of the image during test
_C.INPUT.SIZE_TEST = [256, 128]

# Random probability for image horizontal flip
_C.INPUT.DO_FLIP = True
_C.INPUT.FLIP_PROB = 0.5

```

(continues on next page)

(continued from previous page)

```
# Value of padding size
_C.INPUT.DO_PAD = True
_C.INPUT.PADDING_MODE = 'constant'
_C.INPUT.PADDING = 10

# Random color jitter
_C.INPUT.CJ = CN()
_C.INPUT.CJ.ENABLED = False
_C.INPUT.CJ.PROB = 0.5
_C.INPUT.CJ.BRIGHTNESS = 0.15
_C.INPUT.CJ.CONTRAST = 0.15
_C.INPUT.CJ.SATURATION = 0.1
_C.INPUT.CJ.HUE = 0.1

# Random Affine
_C.INPUT.DO_AFFINE = False

# Auto augmentation
_C.INPUT.DO_AUTOAUG = False
_C.INPUT.AUTOAUG_PROB = 0.0

# Augmix augmentation
_C.INPUT.DO_AUGMIX = False
_C.INPUT.AUGMIX_PROB = 0.0

# Random Erasing
_C.INPUT.REA = CN()
_C.INPUT.REA.ENABLED = False
_C.INPUT.REA.PROB = 0.5
_C.INPUT.REA.VALUE = [0.485*255, 0.456*255, 0.406*255]
# Random Patch
_C.INPUT.RPT = CN()
_C.INPUT.RPT.ENABLED = False
_C.INPUT.RPT.PROB = 0.5

# -----------------------------------------------------------------------------
# Dataset
# -----------------------------------------------------------------------------
_C.DATASETS = CN()
# List of the dataset names for training
_C.DATASETS.NAMES = ("Market1501",)
# List of the dataset names for testing
_C.DATASETS.TESTS = ("Market1501",)
# Combine trainset and testset joint training
_C.DATASETS.COMBINEALL = False

# -----------------------------------------------------------------------------
# DataLoader
# -----------------------------------------------------------------------------
_C.DATALOADER = CN()
# P/K Sampler for data loading
_C.DATALOADER.PK_SAMPLER = True
# Naive sampler which don't consider balanced identity sampling
_C.DATALOADER.NAIVE_WAY = True
# Number of instance for each person
_C.DATALOADER.NUM_INSTANCE = 4
_C.DATALOADER.NUM_WORKERS = 8
```

(continues on next page)

(continued from previous page)

```

# ---------------------------------------------------------------------------- #
# Solver
# ---------------------------------------------------------------------------- #
_C.SOLVER = CN()

# AUTOMATIC MIXED PRECISION
_C.SOLVER.FP16_ENABLED = False

# Optimizer
_C.SOLVER.OPT = "Adam"

_C.SOLVER.MAX_EPOCH = 120

_C.SOLVER.BASE_LR = 3e-4
_C.SOLVER.BIAS_LR_FACTOR = 1.
_C.SOLVER.HEADS_LR_FACTOR = 1.

_C.SOLVER.MOMENTUM = 0.9
_C.SOLVER.NESTEROV = True

_C.SOLVER.WEIGHT_DECAY = 0.0005
_C.SOLVER.WEIGHT_DECAY_BIAS = 0.

# Multi-step learning rate options
_C.SOLVER.SCHED = "MultiStepLR"

_C.SOLVER.DELAY_EPOCHS = 0

_C.SOLVER.GAMMA = 0.1
_C.SOLVER.STEPS = [30, 55]

# Cosine annealing learning rate options
_C.SOLVER.ETA_MIN_LR = 1e-7

# Warmup options
_C.SOLVER.WARMUP_FACTOR = 0.1
_C.SOLVER.WARMUP_ITERS = 1000
_C.SOLVER.WARMUP_METHOD = "linear"

# Backbone freeze iters
_C.SOLVER.FREEZE_ITERS = 0

# FC freeze iters
_C.SOLVER.FREEZE_FC_ITERS = 0


# SWA options
# _C.SOLVER.SWA = CN()
# _C.SOLVER.SWA.ENABLED = False
# _C.SOLVER.SWA.ITER = 10
# _C.SOLVER.SWA.PERIOD = 2
# _C.SOLVER.SWA.LR_FACTOR = 10.
# _C.SOLVER.SWA.ETA_MIN_LR = 3.5e-6
# _C.SOLVER.SWA.LR_SCHED = False

_C.SOLVER.CHECKPOINT_PERIOD = 20
```

(continues on next page)

(continued from previous page)

```

# Number of images per batch across all machines.
# This is global, so if we have 8 GPUs and IMS_PER_BATCH = 16, each GPU will
# see 2 images per batch
_C.SOLVER.IMS_PER_BATCH = 64

# This is global, so if we have 8 GPUs and IMS_PER_BATCH = 16, each GPU will
# see 2 images per batch
_C.TEST = CN()

_C.TEST.EVAL_PERIOD = 20

# Number of images per batch in one process.
_C.TEST.IMS_PER_BATCH = 64
_C.TEST.METRIC = "cosine"
_C.TEST.ROC_ENABLED = False
_C.TEST.FLIP_ENABLED = False

# Average query expansion
_C.TEST.AQE = CN()
_C.TEST.AQE.ENABLED = False
_C.TEST.AQE.ALPHA = 3.0
_C.TEST.AQE.QE_TIME = 1
_C.TEST.AQE.QE_K = 5

# Re-rank
_C.TEST.RERANK = CN()
_C.TEST.RERANK.ENABLED = False
_C.TEST.RERANK.K1 = 20
_C.TEST.RERANK.K2 = 6
_C.TEST.RERANK.LAMBDA = 0.3

# Precise batchnorm
_C.TEST.PRECISE_BN = CN()
_C.TEST.PRECISE_BN.ENABLED = False
_C.TEST.PRECISE_BN.DATASET = 'Market1501'
_C.TEST.PRECISE_BN.NUM_ITER = 300

# ---------------------------------------------------------------------------- #
# Misc options
# ---------------------------------------------------------------------------- #
_C.OUTPUT_DIR = "logs/"

# Benchmark different cudnn algorithms.
# If input images have very different sizes, this option will have large overhead
# for about 10k iterations. It usually hurts total time, but can benefit for certain␣
↪models.
# If input images have the same or similar sizes, benchmark is often helpful.
_C.CUDNN_BENCHMARK = False
```

## 1.3 fastreid.data

### 1.3.1 fastreid.data.data_utils module

### 1.3.2 fastreid.data.datasets module

### 1.3.3 fastreid.data.samplers module

### 1.3.4 fastreid.data.transforms module

## 1.4 fastreid.data.transforms

## 1.5 fastreid.engine

### 1.5.1 fastreid.engine.defaults module

### 1.5.2 fastreid.engine.hooks module

## 1.6 fastreid.evaluation

## 1.7 fastreid.layers

## 1.8 fastreid.modeling

### 1.8.1 Model Registries

These are different registries provided in modeling. Each registry provide you the ability to replace it with your customized component, without having to modify fastreid's code.

Note that it is impossible to allow users to customize any line of code directly. Even just to add one line at some place, you'll likely need to find out the smallest registry which contains that line, and register your component to that registry.

## 1.9 fastreid.solver

## 1.10 fastreid.utils

### 1.10.1 fastreid.utils.colormap module

### 1.10.2 fastreid.utils.comm module

This file contains primitives for multi-gpu communication. This is useful when doing distributed training.

fastreid.utils.comm.**get_world_size**() → int

fastreid.utils.comm.**get_rank**() → int

fastreid.utils.comm.**get_local_rank**() → int

**Returns** The rank of the current process within the local (per-machine) process group.

fastreid.utils.comm.**get_local_size**() → int

**Returns** The size of the per-machine process group, i.e. the number of processes per machine.

fastreid.utils.comm.**is_main_process**() → bool

fastreid.utils.comm.**synchronize**()
Helper function to synchronize (barrier) among all processes when using distributed training

fastreid.utils.comm.**all_gather**(*data*, *group=None*)
Run all_gather on arbitrary picklable data (not necessarily tensors). :param data: any picklable object :param group: a torch process group. By default, will use a group which

contains all ranks on gloo backend.

**Returns** *list[data]* – list of data gathered from each rank

fastreid.utils.comm.**gather**(*data*, *dst=0*, *group=None*)
Run gather on arbitrary picklable data (not necessarily tensors). :param data: any picklable object :param dst: destination rank :type dst: int :param group: a torch process group. By default, will use a group which

contains all ranks on gloo backend.

**Returns**

*list[data]* –

**on dst, a list of data gathered from each rank. Otherwise,** an empty list.

fastreid.utils.comm.**shared_random_seed**()

**Returns**

*int* –

**a random number that is the same across all workers.** If workers need a shared RNG, they can use this shared seed to create one.

All workers must call this function, otherwise it will deadlock.

fastreid.utils.comm.**reduce_dict**(*input_dict*, *average=True*)
Reduce the values in the dictionary from all processes so that process with rank 0 has the reduced results. :param input_dict: inputs to be reduced. All the values must be scalar CUDA Tensor. :type input_dict: dict :param average: whether to do average or sum :type average: bool

**Returns** a dict with the same keys as input_dict, after reduction.

### 1.10.3 fastreid.utils.events module

fastreid.utils.events.**get_event_storage**()

**Returns** The *EventStorage* object that's currently being used. Throws an error if no *EventStorage* is currently enabled.

**class** fastreid.utils.events.**JSONWriter**(*json_file*, *window_size=20*)
Bases: fastreid.utils.events.EventWriter

Write scalars to a json file. It saves scalars as one json per line (instead of a big json) for easy parsing. Examples parsing such a json file:

```
$ cat metrics.json | jq -s '.[0:2]'
[
  {
    "data_time": 0.008433341979980469,
    "iteration": 19,
    "loss": 1.9228371381759644,
    "loss_box_reg": 0.050025828182697296,
    "loss_classifier": 0.5316952466964722,
    "loss_mask": 0.7236229181289673,
    "loss_rpn_box": 0.0856662318110466,
    "loss_rpn_cls": 0.48198649287223816,
    "lr": 0.007173333333333333,
    "time": 0.25401854515075684
  },
  {
    "data_time": 0.007216215133666992,
    "iteration": 39,
    "loss": 1.282649278640747,
    "loss_box_reg": 0.06222952902317047,
    "loss_classifier": 0.30682939291000366,
    "loss_mask": 0.6970193982124329,
    "loss_rpn_box": 0.038663312792778015,
    "loss_rpn_cls": 0.1471673548221588,
    "lr": 0.007706666666666667,
    "time": 0.2490077018737793
  }
]
$ cat metrics.json | jq '.loss_mask'
0.7126231789588928
0.689423680305481
0.6776131987571716
...
```

**__init__**(*json_file*, *window_size=20*)

**Parameters**

- **json_file** (*str*) – path to the json file. New data will be appended if the file exists.
- **window_size** (*int*) – the window size of median smoothing for the scalars whose *smoothing_hint* are True.

**write**()

**close**()

**class** fastreid.utils.events.**TensorboardXWriter**(*log_dir: str*, *window_size: int = 20*, ***kwargs*)

Bases: fastreid.utils.events.EventWriter

Write all scalars to a tensorboard file.

**__init__**(*log_dir: str*, *window_size: int = 20*, ***kwargs*)

**Parameters**

- **log_dir** (*str*) – the directory to save the output events
- **window_size** (*int*) – the scalars will be median-smoothed by this window size
- **kwargs** – other arguments passed to *torch.utils.tensorboard.SummaryWriter(...)*

**write**()

**close**()

**class** fastreid.utils.events.**CommonMetricPrinter**(*max_iter*)
Bases: fastreid.utils.events.EventWriter

Print **common** metrics to the terminal, including iteration time, ETA, memory, all losses, and the learning rate. It also applies smoothing using a window of 20 elements. It's meant to print common metrics in common ways. To print something in more customized ways, please implement a similar printer by yourself.

**__init__**(*max_iter*)

**Parameters** **max_iter** (*int*) – the maximum number of iterations to train. Used to compute ETA.

**write**()

**class** fastreid.utils.events.**EventStorage**(*start_iter=0*)
Bases: object

The user-facing class that provides metric storage functionalities. In the future we may add support for storing / logging other types of data if needed.

**__init__**(*start_iter=0*)

**Parameters** **start_iter** (*int*) – the iteration number to start with

**put_image**(*img_name*, *img_tensor*)
Add an *img_tensor* associated with *img_name*, to be shown on tensorboard. :param img_name: The name of the image to put into tensorboard. :type img_name: str :param img_tensor: An *uint8* or *float*

Tensor of shape *[channel, height, width]* where *channel* is 3. The image format should be RGB. The elements in img_tensor can either have values in [0, 1] (float32) or [0, 255] (uint8). The *img_tensor* will be visualized in tensorboard.

**put_scalar**(*name*, *value*, *smoothing_hint=True*)
Add a scalar *value* to the *HistoryBuffer* associated with *name*. :param smoothing_hint: a 'hint' on whether this scalar is noisy and should be

smoothed when logged. The hint will be accessible through *EventStorage.smoothing_hints()*. A writer may ignore the hint and apply custom smoothing rule. It defaults to True because most scalars we save need to be smoothed to provide any useful signal.

**put_scalars**(*, *smoothing_hint=True*, ***kwargs*)
Put multiple scalars from keyword arguments. .. rubric:: Examples

storage.put_scalars(loss=my_loss, accuracy=my_accuracy, smoothing_hint=True)

**put_histogram**(*hist_name*, *hist_tensor*, *bins=1000*)
Create a histogram from a tensor. :param hist_name: The name of the histogram to put into tensorboard. :type hist_name: str :param hist_tensor: A Tensor of arbitrary shape to be converted

into a histogram.

**Parameters** **bins** (*int*) – Number of histogram bins.

**history**(*name*)

**Returns** *HistoryBuffer* – the scalar history for name

**histories**()

**Returns** *dict[name -> HistoryBuffer]* – the HistoryBuffer for all scalars

**latest**()

**Returns**

*dict[str -> (float, int)]* –

**mapping from the name of each scalar to the most** recent value and the iteration number its added.

**latest_with_smoothing_hint**(*window_size=20*)

Similar to `latest()`, but the returned values are either the un-smoothed original latest value, or a median of the given window_size, depend on whether the smoothing_hint is True. This provides a default behavior that other writers can use.

**smoothing_hints**()

**Returns**

*dict[name -> bool]* –

**the user-provided hint on whether the scalar** is noisy and needs smoothing.

**step**()

User should either: (1) Call this function to increment storage.iter when needed. Or (2) Set *storage.iter* to the correct iteration number before each iteration. The storage will then be able to associate the new data with an iteration number.

**property iter**

Returns: int: The current iteration number. When used together with a trainer,

this is ensured to be the same as trainer.iter.

**property iteration**

**name_scope**(*name*)

**Yields** A context within which all the events added to this storage will be prefixed by the name scope.

**clear_images**()

Delete all the stored images for visualization. This should be called after images are written to tensorboard.

**clear_histograms**()

Delete all the stored histograms for visualization. This should be called after histograms are written to tensorboard.

### 1.10.4 fastreid.utils.logger module

fastreid.utils.logger.**setup_logger**(*output=None*, *distributed_rank=0*, ***, *color=True*, *name='fastreid'*, *abbrev_name=None*)

**Parameters**

- **output** (*str*) – a file name or a directory to save log. If None, will not save log file. If ends with ".txt" or ".log", assumed to be a file name. Otherwise, logs will be saved to *output/log.txt*.
- **name** (*str*) – the root module name of this logger

- **abbrev_name** (*str*) – an abbreviation of the module, to avoid long names in logs. Set to "" to not log the root module in logs. By default, will abbreviate "detectron2" to "d2" and leave other modules unchanged.

fastreid.utils.logger.**log_first_n**(*lvl*, *msg*, *n=1*, ***, *name=None*, *key='caller'*)
Log only for the first n times. :param lvl: the logging level :type lvl: int :param msg: :type msg: str :param n: :type n: int :param name: name of the logger to use. Will use the caller's module by default. :type name: str :param key: the string(s) can be one of "caller" or

> "message", which defines how to identify duplicated logs. For example, if called with *n=1, key="caller"*, this function will only log the first call from the same caller, regardless of the message content. If called with *n=1, key="message"*, this function will log the same content only once, even if they are called from different places. If called with *n=1, key=("caller", "message")*, this function will not log only if the same caller has logged the same message before.

fastreid.utils.logger.**log_every_n**(*lvl*, *msg*, *n=1*, ***, *name=None*)
Log once per n times. :param lvl: the logging level :type lvl: int :param msg: :type msg: str :param n: :type n: int :param name: name of the logger to use. Will use the caller's module by default. :type name: str

fastreid.utils.logger.**log_every_n_seconds**(*lvl*, *msg*, *n=1*, ***, *name=None*)
Log no more than once per n seconds. :param lvl: the logging level :type lvl: int :param msg: :type msg: str :param n: :type n: int :param name: name of the logger to use. Will use the caller's module by default. :type name: str

### 1.10.5 fastreid.utils.registry module

**class** fastreid.utils.registry.**Registry**(*name: str*)
Bases: object

The registry that provides name -> object mapping, to support third-party users' custom modules. To create a registry (e.g. a backbone registry): .. code-block:: python

> BACKBONE_REGISTRY = Registry('BACKBONE')

To register an object: .. code-block:: python

> @BACKBONE_REGISTRY.register() class MyBackbone():
>
> > ...

Or: .. code-block:: python

> BACKBONE_REGISTRY.register(MyBackbone)

**__init__**(*name: str*) → None

> **Parameters** **name** (*str*) – the name of this registry

**register**(*obj: Optional[object] = None*) → Optional[object]
Register the given object under the the name *obj.__name__*. Can be used as either a decorator or not. See docstring of this class for usage.

**get**(*name: str*) → object

### 1.10.6 fastreid.utils.memory module

### 1.10.7 fastreid.utils.analysis module

### 1.10.8 fastreid.utils.visualizer module

### 1.10.9 fastreid.utils.video_visualizer module

# PYTHON MODULE INDEX

## f

# INDEX

## Symbols



## A



## C



## E



## F



## G



## H



## I



## J



## L



## M

## N



## P



## R



## S



## T



## W