MMEval

发布 *0.2.1*

MMEval Contributors

2023 年 04 月 03 日

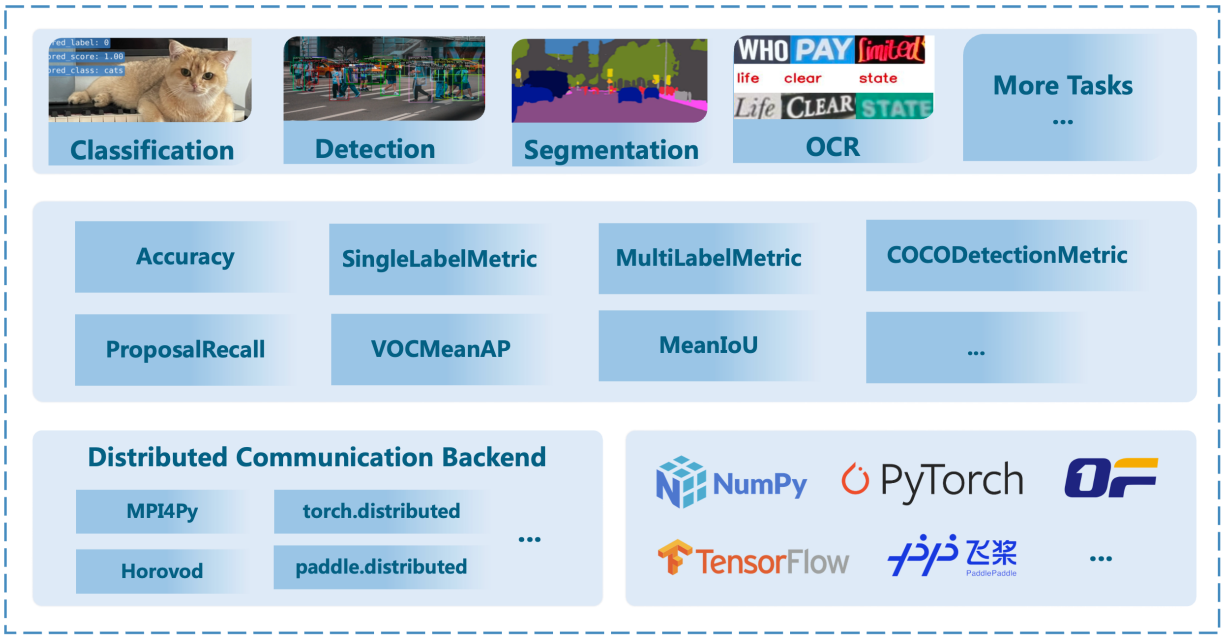
开始你的第一步

CHAPTER 1

介绍

MMEval 是一个机器学习算法评测库，提供高效准确的分布式评测以及多种机器学习框架后端支持，具有以下特点：

- 提供丰富的计算机视觉各细分方向评测指标（自然语言处理方向的评测指标正在支持中）
- 支持多种分布式通信库，实现高效准确的分布式评测。
- 支持多种机器学习框架，根据输入自动分发对应实现。



CHAPTER 2

安装与使用

2.1 安装

MMEval 依赖 Python 3.6+，可以通过 pip 来安装 MMEval。安装 MMEval 的过程中会安装一些 MMEval 运行时的依赖库：

```
pip install mmeval
```

如果要安装 MMEval 中所有评测指标都需要的依赖，可以通过以下命令安装：

```
pip install 'mmeval[all]'
```

2.2 使用

MMEval 中的评测指标提供两种使用方式，以*Accuracy* 为例：

```
from mmeval import Accuracy
import numpy as np

accuracy = Accuracy()
```

第一种是直接调用实例化的 Accuracy 对象，计算评测指标：

```
labels = np.asarray([0, 1, 2, 3])
preds = np.asarray([0, 2, 1, 3])
accuracy(preds, labels)
# {'top1': 0.5}
```

第二种是累积多个批次的数据后，计算评测指标：

```
for i in range(10):
    labels = np.random.randint(0, 4, size=(100, ))
    predicts = np.random.randint(0, 4, size=(100, ))
    accuracy.add(predicts, labels)

accuracy.compute()
# {'top1': ...}
```

CHAPTER 3

支持矩阵

3.1 支持的分布式通信后端

3.2 支持的评测指标及机器学习框架

注解： 下表列出 MMEval 已实现的评测指标与对应的机器学习框架支持情况，打勾表示能够直接接收对应框架的数据类型（如 Tensor）进行计算。

注解： MMEval 在 PyTorch 1.6+，TensorFlow 2.4+, Paddle 2.2+ 和 OneFlow 0.8+ 测试通过。

CHAPTER 4

自定义评测指标

在 MMEval 中实现一个自定义评测指标，需要继承*BaseMetric* 并且实现 `add` 和 `compute_metric` 方法。

在评测过程中，评测指标需要在调用 `add` 后更新 `_results` 以存储中间结果。在最后进行指标计算的时候，将会对 `_results` 进行进程同步后调用 `compute_metric` 进行指标的计算。

以实现 `Accuracy` 指标为例：

```python
import numpy as np
from mmeval.core import BaseMetric

class Accuracy(BaseMetric):

    def add(self, predictions, labels):
        self._results.append((predictions, labels))

    def compute_metric(self, results):
        predictions = np.concatenate(
            [res[0] for res in results])
        labels = np.concatenate(
            [res[1] for res in results])
        correct = (predictions == labels)
        accuracy = sum(correct) / len(predictions)
        return {'accuracy': accuracy}
```

使用 `Accuracy`：

```
# stateless call
accuracy = Accuracy()
metric_results = accuracy(predictions=[1, 2, 3, 4], labels=[1, 2, 3, 1])
print(metric_results)
# {'accuracy': 0.75}

# Accumulate batch
for i in range(10):
    predicts = np.random.randint(0, 4, size=(10,))
    labels = predicts = np.random.randint(0, 4, size=(10,))
    accuracy.add(predicts, labels)

metric_results = accuracy.compute()
accuracy.reset()  # clear the intermediate results
```

CHAPTER 5

使用分布式评测

分布式评测一般采用数据并行的策略，每个进程执行相同的程序来处理不同的数据。

MMEval 中已支持的分布式通信后端可以通过*list_all_backends* 查看：

```
import mmeval

print(mmeval.core.dist.list_all_backends())
# ['non_dist', 'mpi4py', 'tf_horovod', 'torch_cpu', 'torch_cuda', ...]
```

本节将以 CIFAR-10 数据集的评测为例，分别介绍如何使用 MMEval 结合 `torch.distributed` 和 `MPI4Py` 进行分布式评测，相关代码可以在 mmeval/examples/cifar10_dist_eval 中找到。

5.1 评测数据与模型准备

首先我们需要加载 CIFAR-10 测试数据，我们可以使用 `Torchvison` 提供的数据集类。

另外，为了能够在分布式评测中将数据集根据进程数量进行切分，我们需要引入 `DistributedSampler`。

```
import torchvision as tv
from torch.utils.data import DataLoader, DistributedSampler

def get_eval_dataloader(rank=0, num_replicas=1):
    dataset = tv.datasets.CIFAR10(
        root='./', train=False, download=True,
```

(下页继续)

(续上页)

```
        transform=tv.transforms.ToTensor())
    dist_sampler = DistributedSampler(
        dataset, num_replicas=num_replicas, rank=rank)
    data_loader = DataLoader(dataset, batch_size=1, sampler=dist_sampler)
    return data_loader, len(dataset)
```

其次，我们需要准备待评测的模型，这里我们使用 Torchvision 中的 resnet18。

```
import torch
import torchvision as tv

def get_model(pretrained_model_fpath=None):
    model = tv.models.resnet18(num_classes=10)
    if pretrained_model_fpath is not None:
        model.load_state_dict(torch.load(pretrained_model_fpath))
    return model.eval()
```

5.2 单进程评测

有了待评测的数据集与模型，就可以使用*mmeval.Accuracy* 指标对模型预测结果进行评测，下面是一个单进程评测的示例：

```
import tqdm
import torch
from mmeval import Accuracy

eval_dataloader, total_num_samples = get_eval_dataloader()
model = get_model()
# 实例化 `Accuracy`，计算 top1 与 top3 准确率
accuracy = Accuracy(topk=(1, 3))

with torch.no_grad():
    for images, labels in tqdm.tqdm(eval_dataloader):
        predicted_score = model(images)
        # 累计批次数据，中间结果将保存在 `accuracy._results` 中
        accuracy.add(predictions=predicted_score, labels=labels)

# 调用 `accuracy.compute` 进行指标计算
print(accuracy.compute())
# 调用 `accuracy.reset` 清除保存在 `accuracy._results` 中的中间结果
accuracy.reset()
```

5.3 使用 torch.distributed 进行分布式评测

在 MMEval 中为 torch.distributed 实现了两个分布式通信后端，分别是*TorchCPUDist* 和*TorchCUDADist*。

为 MMEval 设置分布式通信后端的方式有两种：

```
from mmeval.core import set_default_dist_backend
from mmeval import Accuracy

# 1. 设置全局默认分布式通信后端
set_default_dist_backend('torch_cpu')

# 2. 初始化评测指标时候通过 `dist_backend` 传参
accuracy = Accuracy(dist_backend='torch_cpu')
```

结合上述单进程评测的代码，再加入分布式环境启动以及初始化即可实现分布式评测。

```
import tqdm
import torch
from mmeval import Accuracy


def eval_fn(rank, process_num):
    # 分布式环境初始化
    torch.distributed.init_process_group(
        backend='gloo',
        init_method=f'tcp://127.0.0.1:2345',
        world_size=process_num,
        rank=rank)

    eval_dataloader, total_num_samples = get_eval_dataloader(rank, process_num)
    model = get_model()
    # 实例化 Accuracy 并设置分布式通信后端
    accuracy = Accuracy(topk=(1, 3), dist_backend='torch_cpu')

    with torch.no_grad():
        for images, labels in tqdm.tqdm(eval_dataloader, disable=(rank!=0)):
            predicted_score = model(images)
            accuracy.add(predictions=predicted_score, labels=labels)

    # 通过 size 指定数据集样本数量，以便去除 DistributedSampler 补齐的重复样本。
    print(accuracy.compute(size=total_num_samples))
    accuracy.reset()
```

(下页继续)

(续上页)

```
if __name__ == "__main__":
    # 分布式进程数量
    process_num = 3
    # 采用 spawn 的方式启动分布式
    torch.multiprocessing.spawn(
        eval_fn, nprocs=process_num, args=(process_num, ))
```

5.4 使用 MPI4Py 进行分布式评测

MMEval 将分布式通信功能抽象解耦了，因此虽然上述例子使用的是 PyTorch 模型和数据加载，我们仍然可以使用除 torch.distributed 以外的分布式通信后端来实现分布式评测，下面将展示如何使用 MPI4Py 作为分布式通信后端来进行分布式评测。

首先需要安装 MPI4Py 以及 openmpi，建议使用 conda 进行安装：

```
conda install openmpi
conda install mpi4py
```

然后将上述代码修改为使用 MPI4Py 做为分布式通信后端：

```
# cifar10_eval_mpi4py.py

import tqdm
from mpi4py import MPI
import torch
from mmeval import Accuracy


def eval_fn(rank, process_num):
    eval_dataloader, total_num_samples = get_eval_dataloader(rank, process_num)
    model = get_model()
    accuracy = Accuracy(topk=(1, 3), dist_backend='mpi4py')

    with torch.no_grad():
        for images, labels in tqdm.tqdm(eval_dataloader, disable=(rank!=0)):
            predicted_score = model(images)
            accuracy.add(predictions=predicted_score, labels=labels)

    print(accuracy.compute(size=total_num_samples))
    accuracy.reset()
```

(下页继续)

(续上页)

```
if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    eval_fn(comm.Get_rank(), comm.Get_size())
```

使用 mpirun 作为分布式评测启动方式：

```
# 使用 mpirun 启动 3 个进程
mpirun -np 3 python3 cifar10_eval_mpi4py.py
```

CHAPTER 6

MMCls

MMEval 中的*BaseMetric* 参照了 mmengine.evaluator 模块的设计，在此基础上引入了分布式通信后端的组件，以满足多样的分布式通信库需求。

因此 MMEval 天然的支持基于 OpenMMLab 2.0 算法库的评测，在 OpenMMLab 2.0 算法库中使用 MMEval 的评测指标无需多做修改。

以在 MMCls 中使用*mmeval.Accuracy* 为例，只需要在 config 中配置好使用的 Metric 为 Accuracy 即可：

```
val_evaluator = dict(type='Accuracy', topk=(1, ))

test_evaluator = val_evaluator
```

MMEval 对 OpenMMLab 2.0 算法库评测的支持正在逐步完善中，已支持的评测指标可以在支持矩阵 中查看。

CHAPTER 7

TensorPack

TensorPack 是一个基于 TensorFlow 的深度学习训练库，具有高效与灵活的特点。

在 TensorPack 代码仓库中，提供了许多经典模型与任务的示例，本小节展示如何在 TensorPack-FasterRCNN 中使用*mmeval.COCODetection* 进行评测，相关代码可以在 mmeval/examples/tensorpack 中找到。

首先需要安装 TensorFlow 与 TensorPack，然后按照 TensorPack-FasterRCNN 示例中的准备步骤，安装依赖和准备 COCO 数据集，以及下载需要评测的预训练模型权重。

TensorPack-FasterRCNN 自带了评测功能，可以通过以下命令执行评测：

```
./predict.py --evaluate output.json --load /path/to Trained-Model-Checkpoint --config␣
↪SAME-AS-TRAINING
```

MMEval 为 TensorPack-FasterRCNN 提供了适配*mmeval.COCODetection* 的评测脚本，需要将该脚本放至 TensorPack-FasterRCNN 示例目录下，然后通过以下命令执行评测：

```
# 单卡评测
python tensorpack_mmeval.py --load <model_path> --config SAME-AS-TRAINING

# 支持基于 MPI4Py 的分布式评测，通过 mpirun 启动多卡评测
mpirun -np 8 python tensorpack_mmeval.py --load <model_path> --config SAME-AS-TRAINING
```

我们在 COCO-MaskRCNN-R50C41x 配置上测试了该评测脚本，与 TensorPack-FasterRCNN 报告的评测结果一致。

CHAPTER 8

PaddleSeg

PaddleSeg 是一个基于 Paddle 的语义分割算法库，支持许多和语义分割相关的下游任务。

本小节展示如何在 PaddleSeg 中使用*mmeval.MeanIoU* 进行评测，相关代码可以在 mmeval/examples/paddleseg 中找到。

首先需要安装 Paddle 与 PaddleSeg，可以参照 PaddleSeg 中的安装文档进行。另外需要下载待评测的预训练模型，以及根据配置文件准备数据集。

PaddleSeg 算法库中提供了进行模型评测的脚本，可以通过以下命令对模型进行评测：

```
python val.py --config <config_path> --model_path <model_path>
```

需要注意，PaddleSeg 算法仓库中的 `val.py` 评测脚本只支持单卡评测，尚未支持多卡评测。

MMEval 为 PaddleSeg 提供了适配*mmeval.MeanIoU* 的评测脚本，可以通过以下命令执行评测：

```
# 单卡评测
python ppseg_mmeval.py --config <config_path> --model_path <model_path>

# 单机多卡评测
python ppseg_mmeval.py --config <config_path> --model_path <model_path> --launcher␣
↪paddle --num_process <num_gpus>
```

我们在 fastfcn_resnet50_os8_ade20k_480x480_120k 配置上测试了该评测脚本，与 PaddleSeg 中的 `val.py` 得到的评测结果一致。

CHAPTER 9

BaseMetric 设计

在评测过程中，通常会以数据并行的形式，在每张卡上推理部分数据集的结果，以加快评测速度。

而在每个数据子集上计算得到的评测结果，通常不能通过简单的求平均来与整个数据集的评测结果进行等价。因此通常的做法是在分布式评测过程中，将每张卡得到的推理结果或者指标计算中间结果保存下来，在所有进程中进行 all-gather 操作，最后再计算整个评测数据集的指标结果。

上述操作在 MMEval 中由*BaseMetric* 来完成，其接口设计如下图所示：

其中 `add` 与 `compute_metric` 方法为需要用户继承实现的接口，具体可以参考自定义评测指标。

通过*BaseMetric* 接口可以看出，*BaseMetric* 主要功能是提供分布式评测，其基本流程为：

1. 用户调用 `add` 方法，将推理结果或者指标计算中间结果保存在 `BaseMetric._results` 列表中。
2. 用户调用 `compute` 方法，`BaseMetric` 将 `_results` 列表中的数据进行进程间同步并调用用户定义的 `compute_metric` 方法进行指标的计算。

除此之外，*BaseMetric* 还考虑到数据并行过程中，为了保证所有进程中的数据样本数量一致，部分进程会有补齐重复数据样本的情况，比如 PyTorch 中的 `DistributedSampler`，这会影响到指标计算的正确性。

为了应对这个问题，*BaseMetric.compute* 可以接收一个 `size` 参数，表示整个评测数据集的真实样本数量，在 `_results` 进程同步之后，调用 `compute_metric` 方法之前，根据 `dist_collect_mode` 去除用来补齐的重复样本，以实现正确的指标计算。

注解： 为了能够在分布式评测时候将补齐的重复样本删除掉，存储在 `_results` 列表的中间值需要和评测数据集样本是一一对应的关系。

CHAPTER 10

分布式通信后端

MMEval 在分布式评测过程中所需的分布式通信需求，主要有以下两个：

- 将各个进程中保存的评测指标计算中间结果 all-gather
- 将 rank 0 进程计算得到的指标结果 broadcast 给所有进程

为了能够灵活的支持多种分布式通信库，MMEval 将上述分布式通信需求抽象定义了一个分布式通信接口*BaseDistBackend*：

实现一个分布式通信后端，需要继承*BaseDistBackend* 并且实现上述接口，其中：

- is_initialized，标识当前是否已经完成分布式通信环境的初始化。
- rank，当前进程所在进程组的序号。
- world_size，进程数量。
- all_gather_object，对任意可以被 Pickle 序列化的 Python 对象进行 all_tather 操作。
- broadcast_object，对任意可以被 Pickle 序列化的 Python 对象进行广播操作。

以实现*MPI4PyDist* 为例：

```
from mpi4py import MPI


class MPI4PyDist(BaseDistBackend):
    """A distributed communication backend for mpi4py."""
```

(下页继续)

(续上页)

```
    @property
    def is_initialized(self) -> bool:
        """Returns True if the distributed environment has been initialized."""
        return 'OMPI_COMM_WORLD_SIZE' in os.environ

    @property
    def rank(self) -> int:
        """Returns the rank index of the current process group."""
        comm = MPI.COMM_WORLD
        return comm.Get_rank()

    @property
    def world_size(self) -> int:
        """Returns the world size of the current process group."""
        comm = MPI.COMM_WORLD
        return comm.Get_size()

    def all_gather_object(self, obj: Any) -> List[Any]:
        """All gather the given object from the current process group and
        returns a list consisting gathered object of each process."""
        comm = MPI.COMM_WORLD
        return comm.allgather(obj)

    def broadcast_object(self, obj: Any, src: int = 0) -> Any:
        """Broadcast the given object from source process to the current
        process group."""
        comm = MPI.COMM_WORLD
        return comm.bcast(obj, root=src)
```

MMEval 中已经预置实现了一些分布式通信后端，具体可以在支持矩阵中查看。

CHAPTER 11

基于参数类型注释的多分派

MMEval 希望能够支持多种机器学习框架，一个最为简单的方案是让所有评测指标的计算都支持 NumPy 即可。

这样做可以实现大部分评测需求，因为所有机器学习框架的 Tensor 数据类型都可以转为 numpy.ndarray。

但是在某些情况下可能会存在一些问题：

- NumPy 有一些常用算子尚未实现，如 topk，会影响评测指标的计算速度。
- 大量的 Tensor 从 CUDA 设备搬运到 CPU 内存会比较耗时。

另外，如果希望评测指标的计算过程是可导的，那么就需要用各自机器学习框架的 Tensor 数据类型进行计算。

为了应对上述问题，MMEval 的评测指标提供了一些特定机器学习框架的指标计算实现，具体可以在支持矩阵 中查看。

同时，为了应对不同指标计算方式的分发问题，MMEval 采用了基于类型注释的动态多分派机制，可以根据输入的数据类型，动态的选择不同的计算方式。

一个基于类型注释的多分派简单示例如下：

```
from mmeval.core import dispatch


@dispatch
def compute(x: int, y: int):
    print('this is int')
```

(下页继续)

(续上页)

```
@dispatch
def compute(x: str, y: str):
    print('this is str')

compute(1, 1)
# this is int

compute('1', '1')
# this is str
```

目前，我们使用 plum-dispatch 来实现 MMEval 中的分发机制，在 plum-dispatch 基础上，做了一些速度上的优化，并且扩展支持了 `typing.ForwardRef`。

警告： 受限于 Python 动态类型的特性，在运行时确定一个变量的具体类型可能会比较耗时，尤其是碰到一些大的嵌套结构数据。因此基于类型注释的动态多分派机制可能会存在一些性能问题，更多信息可以参考：wesselb/plum/issues/53

CHAPTER 12

mmeval.core

mmeval.core

12.1 base_metric

BaseMetric	Base class for metric.

12.1.1 BaseMetric

class mmeval.core.**BaseMetric**(*dataset_meta: Optional[Dict] = None*, *dist_collect_mode: str = 'unzip'*, *dist_backend: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Base class for metric.

To implement a metric, you should implement a subclass of `BaseMetric` that overrides the `add` and `compute_metric` methods. `BaseMetric` will automatically complete the distributed synchronization be-

tween processes.

In the evaluation process, each metric will update `self._results` to store intermediate results after each call of `add`. When computing the final metric result, the `self._results` will be synchronized between processes.

参数

- **dataset_meta** (*dict, optional*) –Meta information of the dataset, this is required for some metrics that require dataset information. Defaults to None.
- **dist_collect_mode** (*str, optional*) –The method of concatenating the collected synchronization results. This depends on how the distributed data is split. Currently only 'unzip' and 'cat' are supported. For PyTorch's `DistributedSampler`, 'unzip' should be used. Defaults to 'unzip'.
- **dist_backend** (*str, optional*) –The name of the distributed communication backend, you can get all the backend names through `mmeval.core.list_all_backends()`. If `None`, use the default backend. Defaults to None.
- **logger** (*Logger, optional*) –The logger used to log messages. If `None`, use the default logger of mmeval. Defaults to None.

Example to implement an accuracy metric:

```
>>> import numpy as np
>>> from mmeval.core import BaseMetric
>>>
>>> class Accuracy(BaseMetric):
...     def add(self, predictions, labels):
...         self._results.append((predictions, labels))
...     def compute_metric(self, results):
...         predictions = np.concatenate([res[0] for res in results])
...         labels = np.concatenate([res[1] for res in results])
...         correct = (predictions == labels)
...         accuracy = sum(correct) / len(predictions)
...         return {'accuracy': accuracy}
```

Stateless call of metric:

```
>>> accuracy = Accuracy()
>>> accuracy(predictions=[1, 2, 3, 4], labels=[1, 2, 3, 1])
{'accuracy': 0.75}
```

Accumulate batch:

```
>>> for i in range(10):
>>>     predicts = np.random.randint(0, 4, size=(10,))
>>>     labels = predicts = np.random.randint(0, 4, size=(10,))
```

(下页继续)

(续上页)

```
>>>     accuracy.add(predicts, labels)
>>> accuracy.compute()
```

abstract add(**args*, ***kwargs*)

Override this method to add the intermediate results to `self._results`.

注解： For performance issues, what you add to the `self._results` should be as simple as possible. But be aware that the intermediate results stored in `self._results` should correspond one-to-one with the samples, in that we need to remove the padded samples for the most accurate result.

---

compute(*size: Optional[int] = None*) → Dict

Synchronize intermediate results and then call `self.compute_metric`.

参数 size(*int, optional*) –The length of the entire dataset, it is only used when distributed evaluation. When batch size > 1, the dataloader may pad some data samples to make sure all ranks have the same length of dataset slice. The `compute` will drop the padded data based on this size. If None, do nothing. Defaults to None.

返回 The computed metric results.

返回类型 dict

abstract compute_metric(*results: List[Any]*) → Dict

Override this method to compute the metric result from collectd intermediate results.

The returned result of the metric compute should be a dictionary.

property dataset_meta: Optional[Dict]

Meta information of the dataset.

property name: str

The metric name, defaults to the name of the class.

reset() → None

Clear the metric stored results.

12.2 dist

list_all_backends	Returns a list of all distributed backend names.
set_default_dist_backend	Set the given distributed backend as the default distributed backend.

下页继续

表 2 – 续上页

get_dist_backend	Returns distributed backend by the given distributed backend name.

12.2.1 mmeval.core.list_all_backends

mmeval.core.**list_all_backends**() → List[str]

Returns a list of all distributed backend names.

返回 A list of all distributed backend names.

返回类型 List[str]

12.2.2 mmeval.core.set_default_dist_backend

mmeval.core.**set_default_dist_backend**(*dist_backend: str*) → None

Set the given distributed backend as the default distributed backend.

参数 **dist_backend** (*str*) –The distribute backend name to set.

12.2.3 mmeval.core.get_dist_backend

mmeval.core.**get_dist_backend**(*dist_backend: Optional[str] = None*) → *mmeval.core.dist_backends.base_backend.BaseDistBackend*

Returns distributed backend by the given distributed backend name.

参数 **dist_backend** (*str, optional*) –The distributed backend name want to get. if None, return the default distributed backend.

返回 The distributed backend instance.

返回类型 BaseDistBackend

12.3 dispatch

dispatch	A Dispatcher inherited from plum.Dispatcher that resolve typing.ForwardRef.

12.3.1 mmeval.core.dispatch

mmeval.core.**dispatch = <mmeval.core.dispatcher._MMEvalDispatcher object>**

A Dispatcher inherited from `plum.Dispatcher` that resolve `typing.ForwardRef`.

This dispatcher tries to use `importlib.import_moudle` to import ForwardRerf type and convert unimportable type as a placeholder.

With the `_MMEvalDispatcher`, we can run the following code example without PyTorch installed, which is `plum.dispatch` can't do.

示例

```
>>> from mmeval.core import dispatch
```

```
>>> @dispatch
>>> def compute(x: 'torch.Tensor'):
...     print('The input is a `torch.Tensor`')
```

```
>>> @dispatch
>>> def compute(x: 'numpy.ndarray'):
...     print('The input is a `numpy.ndarray`')
```

CHAPTER 13

mmeval.core.dist_backends

mmeval.core.dist_backends

13.1 dist_backends

BaseDistBackend	The base backend of distributed communication used by mmeval Metric.
TensorBaseDistBackend	A base backend of Tensor base distributed communication like PyTorch.
NonDist	A dummy distributed communication for non-distributed environment.
MPI4PyDist	A distributed communication backend for mpi4py.
TorchCPUDist	A cpu distributed communication backend for torch.distributed.
TorchCUDADist	A cuda distributed communication backend for torch.distributed.
TFHorovodDist	A distributed communication backend for horovod.tensorflow.

下页继续

表 1 – 续上页

PaddleDist	A distributed communication backend for paddle.distributed.
OneFlowDist	A distributed communication backend for oneflow.

13.1.1 BaseDistBackend

class mmeval.core.dist_backends.**BaseDistBackend**

The base backend of distributed communication used by mmeval Metric.

abstract all_gather_object(*obj: Any*) → List[Any]

All gather the given object from the current process group and returns a list consisting gathered object of each process..

参数 **obj** (*any*) –Any pickle-able python object for all gather.

返回 A list of the all gathered object.

返回类型 list

abstract broadcast_object(*obj: Any*, *src: int*) → Any

Broadcast the given object from source process to the current process group.

参数

- **obj** (*any*) –Any pickle-able python object for broadcast.
- **src** (*int*) –The source rank index.

返回 The broadcast object.

返回类型 any

abstract property is_initialized: bool

Returns True if the distributed environment has been initialized.

返回 Returns True if the distributed environment has been initialized, otherwise returns False.

返回类型 bool

abstract property rank: int

Returns the rank index of the current process group.

返回 The rank index of the current process group.

返回类型 int

abstract property world_size: int

Returns the world size of the current process group.

The *world size* is the size of the communication process group.

返回 The size of the current process group.

返回类型 int

13.1.2 TensorBaseDistBackend

class mmeval.core.dist_backends.**TensorBaseDistBackend**

A base backend of Tensor base distributed communication like PyTorch.

all_gather_object(*obj: Any*) → List[Any]

All gather the given object from the current process group and returns a list consisting gathered object of each process..

There are 3 steps to all gather a python object using Tensor distributed communication:

1. Serialize picklable python object to tensor.
2. All gather the tensor size and padding the tensor with the same size.
3. All gather the padded tensor and deserialize tensor to picklable python object.

参数 **obj** (*any*) –Any pickle-able python object for all gather.

返回 A list of the all gathered object.

返回类型 list

broadcast_object(*obj: Any*, *src: int = 0*) → Any

Broadcast the given object from source process to the current process group.

There are 3 steps to broadcast a python object use Tensor distributed communication:

1. Serialize picklable python object to tensor.
2. Broadcast the tensor size and padding the tensor with the same size.
3. Broadcast the padded tensor and deserialize tensor to picklable python object.

参数

- **obj** (*any*) –Any pickle-able python object for broadcast.
- **src** (*int*) –The source rank index.

返回 The broadcast object.

返回类型 any

13.1.3 NonDist

class mmeval.core.dist_backends.**NonDist**

A dummy distributed communication for non-distributed environment.

all_gather_object(*obj: Any*) → List[Any]

Returns the list with given obj in a non-distributed environment.

broadcast_object(*obj: Any*, *src: int = 0*) → Any

Returns the given obj directly in a non-distributed environment.

property is_initialized: bool

Returns False directly in a non-distributed environment.

property rank: int

Returns 0 as the rank index in a non-distributed environment.

property world_size: int

Returns 1 as the world_size in a non-distributed environment.

13.1.4 MPI4PyDist

class mmeval.core.dist_backends.**MPI4PyDist**

A distributed communication backend for mpi4py.

all_gather_object(*obj: Any*) → List[Any]

All gather the given object from the current process group and returns a list consisting gathered object of each process.

参数 **obj** (*any*) –Any pickle-able python object for all gather.

返回 A list of the all gathered object.

返回类型 list

broadcast_object(*obj: Any*, *src: int = 0*) → Any

Broadcast the given object from source process to the current process group.

参数

- **obj** (*any*) –Any pickle-able python object for broadcast.
- **src** (*int*) –The source rank index.

返回 The broadcast object.

返回类型 any

property is_initialized: bool

Returns True if the distributed environment has been initialized.

返回 Returns True if the distributed environment has been initialized, otherwise returns False.

返回类型 bool

property rank: int

Returns the rank index of the current process group.

property world_size: int

Returns the world size of the current process group.

13.1.5 TorchCPUDist

class mmeval.core.dist_backends.**TorchCPUDist**

A cpu distributed communication backend for torch.distributed.

property is_initialized: bool

Returns True if the distributed environment has been initialized.

返回 Returns True if the distributed environment has been initialized, otherwise returns False.

返回类型 bool

property rank: int

Returns the rank index of the current process group.

property world_size: int

Returns the world size of the current process group.

13.1.6 TorchCUDADist

class mmeval.core.dist_backends.**TorchCUDADist**

A cuda distributed communication backend for torch.distributed.

13.1.7 TFHorovodDist

class mmeval.core.dist_backends.**TFHorovodDist**

A distributed communication backend for horovod.tensorflow.

all_gather_object(*obj: Any*) → List[Any]

All gather the given object from the current process group and returns a list consisting gathered object of each process..

参数 **obj** (*any*) –Any pickle-able python object for all gather.

返回 A list of the all gathered object.

返回类型 list

broadcast_object(*obj: Any*, *src: int = 0*) → Any

Broadcast the given object from source process to the current process group.

参数

- **obj** (*any*) –Any pickle-able python object for broadcast.
- **src** (*int*) –The source rank index.

返回 The broadcast object.

返回类型 any

property is_initialized: bool

Returns True if the distributed environment has been initialized.

返回 Returns True if the distributed environment has been initialized, otherwise returns False.

返回类型 bool

property rank: int

Returns the rank index of the current process group.

property world_size: int

Returns the world size of the current process group.

13.1.8 PaddleDist

class mmeval.core.dist_backends.**PaddleDist**

A distributed communication backend for paddle.distributed.

property is_initialized: bool

Returns True if the distributed environment has been initialized.

返回 Returns True if the distributed environment has been initialized, otherwise returns False.

返回类型 bool

property rank: int

Returns the rank index of the current process group.

property world_size: int

Returns the world size of the current process group.

13.1.9 OneFlowDist

class mmeval.core.dist_backends.**OneFlowDist**

A distributed communication backend for oneflow.

property is_initialized: bool

Returns True if the distributed environment has been initialized.

返回 Returns True if the distributed environment has been initialized, otherwise returns False.

返回类型 bool

property rank: int

Returns the rank index of the current process group.

property world_size: int

Returns the world size of the current process group.

CHAPTER 14

mmeval.fileio

mmeval.fileio

14.1 File Backend

`BaseStorageBackend`	Abstract class of storage backends.
`LocalBackend`	Raw local storage backend.
`HTTPBackend`	HTTP and HTTPS storage bachend.
`LmdbBackend`	Lmdb storage backend.
`MemcachedBackend`	Memcached storage backend.
`PetrelBackend`	Petrel storage backend (for internal usage).

14.1.1 BaseStorageBackend

class mmeval.fileio.**BaseStorageBackend**

Abstract class of storage backends.

All backends need to implement two apis: *get()* and *get_text()*.

- *get()* reads the file as a byte stream.
- *get_text()* reads the file as texts.

14.1.2 LocalBackend

class mmeval.fileio.**LocalBackend**

Raw local storage backend.

exists(*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path exists.

参数 **filepath** (*str or Path*) –Path to be checked whether exists.

返回 Return `True` if `filepath` exists, `False` otherwise.

返回类型 bool

实际案例

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.exists(filepath)
True
```

get(*filepath: Union[str, pathlib.Path]*) → bytes

Read bytes from a given `filepath` with 'rb' mode.

参数 **filepath** (*str or Path*) –Path to read data.

返回 Expected bytes object.

返回类型 bytes

实际案例

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.get(filepath)
b'hello world'
```

get_local_path(*filepath: Union[str, pathlib.Path]*) → Generator[Union[str, pathlib.Path], None, None]

Only for unified API and do nothing.

参数

- **filepath** (*str or Path*) –Path to be read data.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

实际案例

```
>>> backend = LocalBackend()
>>> with backend.get_local_path('s3://bucket/abc.jpg') as path:
...     # do something here
```

get_text(*filepath: Union[str, pathlib.Path]*, *encoding: str = 'utf-8'*) → str

Read text from a given `filepath` with ‘r’ mode.

参数

- **filepath** (*str or Path*) –Path to read data.
- **encoding** (*str*) –The encoding format used to open the `filepath`. Defaults to ‘utf-8’.

返回 Expected text reading from `filepath`.

返回类型 str

实际案例

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.get_text(filepath)
'hello world'
```

isdir(*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path is a directory.

参数 **filepath** (*str or Path*) –Path to be checked whether it is a directory.

返回 Return `True` if `filepath` points to a directory, `False` otherwise.

返回类型 bool

实际案例

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/dir'
>>> backend.isdir(filepath)
True
```

isfile (*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path is a file.

参数 **filepath** (*str or Path*) –Path to be checked whether it is a file.

返回 Return `True` if `filepath` points to a file, `False` otherwise.

返回类型 bool

实际案例

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.isfile(filepath)
True
```

join_path (*filepath: Union[str, pathlib.Path]*, **filepaths: Union[str, pathlib.Path]*) → str

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

参数 **filepath** (*str or Path*) –Path to be concatenated.

返回 The result of concatenation.

返回类型 str

实际案例

```
>>> backend = LocalBackend()
>>> filepath1 = '/path/of/dir1'
>>> filepath2 = 'dir2'
>>> filepath3 = 'path/of/file'
>>> backend.join_path(filepath1, filepath2, filepath3)
'/path/of/dir/dir2/path/of/file'
```

list_dir_or_file(*dir_path: Union[str, pathlib.Path]*, *list_dir: bool = True*, *list_file: bool = True*, *suffix: Optional[Union[str, Tuple[str]]] = None*, *recursive: bool = False*) → Iterator[str]

Scan a directory to find the interested directories or files in arbitrary order.

注解： *list_dir_or_file()* returns the path relative to `dir_path`.

参数

- **dir_path** (*str or Path*) –Path of the directory.
- **list_dir** (*bool*) –List the directories. Defaults to True.
- **list_file** (*bool*) –List the path of files. Defaults to True.
- **suffix** (*str or tuple[str], optional*) –File suffix that we are interested in. Defaults to None.
- **recursive** (*bool*) –If set to True, recursively scan the directory. Defaults to False.

生成器 *Iterable[str]* –A relative path to `dir_path`.

实际案例

```
>>> backend = LocalBackend()
>>> dir_path = '/path/of/dir'
>>> # list those files and directories in current directory
>>> for file_path in backend.list_dir_or_file(dir_path):
...     print(file_path)
>>> # only list files
>>> for file_path in backend.list_dir_or_file(dir_path, list_dir=False):
...     print(file_path)
>>> # only list directories
>>> for file_path in backend.list_dir_or_file(dir_path, list_file=False):
...     print(file_path)
>>> # only list files ending with specified suffixes
```

(下页继续)

(续上页)

```
>>> for file_path in backend.list_dir_or_file(dir_path, suffix='.txt'):
...     print(file_path)
>>> # list all files and directory recursively
>>> for file_path in backend.list_dir_or_file(dir_path, recursive=True):
...     print(file_path)
```

14.1.3 HTTPBackend

class mmeval.fileio.**HTTPBackend**

HTTP and HTTPS storage bachend.

get(*filepath: str*) → bytes

Read bytes from a given `filepath`.

参数 **filepath** (*str*) –Path to read data.

返回 Expected bytes object.

返回类型 bytes

实际案例

```
>>> backend = HTTPBackend()
>>> backend.get('http://path/of/file')
b'hello world'
```

get_local_path(*filepath: str*) → Generator[Union[str, pathlib.Path], None, None]

Download a file from `filepath` to a local temporary directory, and return the temporary path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

参数 **filepath** (*str*) –Download a file from `filepath`.

生成器 *Iterable[str]* –Only yield one temporary path.

实际案例

```
>>> backend = HTTPBackend()
>>> # After existing from the ``with`` clause,
>>> # the path will be removed
>>> with backend.get_local_path('http://path/of/file') as path:
...     # do something here
```

get_text (*filepath*, *encoding='utf-8'*) → str

Read text from a given `filepath`.

参数

- **filepath** (*str*) –Path to read data.
- **encoding** (*str*) –The encoding format used to open the `filepath`. Defaults to 'utf-8'.

返回 Expected text reading from `filepath`.

返回类型 str

实际案例

```
>>> backend = HTTPBackend()
>>> backend.get_text('http://path/of/file')
'hello world'
```

14.1.4 LmdbBackend

class mmeval.fileio.**LmdbBackend**(*db_path*, *readonly=True*, *lock=False*, *readahead=False*, ***kwargs*)

Lmdb storage backend.

参数

- **db_path** (*str*) –Lmdb database path.
- **readonly** (*bool*) –Lmdb environment parameter. If True, disallow any write operations. Defaults to True.
- **lock** (*bool*) –Lmdb environment parameter. If False, when concurrent access occurs, do not lock the database. Defaults to False.
- **readahead** (*bool*) –Lmdb environment parameter. If False, disable the OS filesystem readahead mechanism, which may improve random read performance when a database is larger than RAM. Defaults to False.
- ****kwargs** –Keyword arguments passed to *lmdb.open*.

db_path

Lmdb database path.

Type str

get (*filepath: Union[str, pathlib.Path]*) → bytes

Get values according to the filepath.

参数 **filepath** (*str or Path*) –Here, filepath is the lmdb key.

返回 Expected bytes object.

返回类型 bytes

实际案例

```
>>> backend = LmdbBackend('path/to/lmdb')
>>> backend.get('key')
b'hello world'
```

14.1.5 MemcachedBackend

class mmeval.fileio.**MemcachedBackend**(*server_list_cfg*, *client_cfg*, *sys_path=None*)

Memcached storage backend.

server_list_cfg

Config file for memcached server list.

Type str

client_cfg

Config file for memcached client.

Type str

sys_path

Additional path to be appended to *sys.path*. Defaults to None.

Type str, optional

get(*filepath: Union[str, pathlib.Path]*)

Get values according to the filepath.

参数 **filepath** (*str or Path*) –Path to read data.

返回 Expected bytes object.

返回类型 bytes

实际案例

```
>>> server_list_cfg = '/path/of/server_list.conf'
>>> client_cfg = '/path/of/mc.conf'
>>> backend = MemcachedBackend(server_list_cfg, client_cfg)
>>> backend.get('/path/of/file')
b'hello world'
```

14.1.6 PetrelBackend

class mmeval.fileio.**PetrelBackend**(*path_mapping: Optional[dict] = None*, *enable_mc: bool = True*)

Petrel storage backend (for internal usage).

PetrelBackend supports reading and writing data to multiple clusters. If the file path contains the cluster name, PetrelBackend will read data from specified cluster or write data to it. Otherwise, PetrelBackend will access the default cluster.

参数

- **path_mapping** (*dict, optional*) –Path mapping dict from local path to Petrel path. When `path_mapping={'src': 'dst'}`, `src` in `filepath` will be replaced by `dst`. Defaults to None.
- **enable_mc** (*bool, optional*) –Whether to enable memcached support. Defaults to True.

实际案例

```
>>> backend = PetrelBackend()
>>> filepath1 = 'petrel://path/of/file'
>>> filepath2 = 'cluster-name:petrel://path/of/file'
>>> backend.get(filepath1)  # get data from default cluster
>>> client.get(filepath2)  # get data from 'cluster-name' cluster
```

exists(*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path exists.

参数 **filepath** (*str or Path*) –Path to be checked whether exists.

返回 Return `True` if `filepath` exists, `False` otherwise.

返回类型 bool

实际案例

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.exists(filepath)
True
```

get(*filepath: Union[str, pathlib.Path]*) → bytes

Read bytes from a given `filepath` with 'rb' mode.

参数 **filepath** (*str or Path*) –Path to read data.

返回 Return bytes read from filepath.

返回类型 bytes

实际案例

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.get(filepath)
b'hello world'
```

get_local_path(*filepath: Union[str, pathlib.Path]*) → Generator[Union[str, pathlib.Path], None, None]

Download a file from `filepath` to a local temporary directory, and return the temporary path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

参数 **filepath** (*str or Path*) –Download a file from `filepath`.

生成器 *Iterable[str]* –Only yield one temporary path.

实际案例

```
>>> backend = PetrelBackend()
>>> # After existing from the ``with`` clause,
>>> # the path will be removed
>>> filepath = 'petrel://path/of/file'
>>> with backend.get_local_path(filepath) as path:
...     # do something here
```

get_text(*filepath: Union[str, pathlib.Path]*, *encoding: str = 'utf-8'*) → str

Read text from a given `filepath` with ‘r’ mode.

参数

- **filepath** (*str or Path*) –Path to read data.
- **encoding** (*str*) –The encoding format used to open the `filepath`. Defaults to ‘utf-8’ .

返回 Expected text reading from `filepath`.

返回类型 str

实际案例

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.get_text(filepath)
'hello world'
```

isdir (*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path is a directory.

参数 **filepath** (*str or Path*) –Path to be checked whether it is a directory.

返回 Return `True` if `filepath` points to a directory, `False` otherwise.

返回类型 bool

实际案例

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/dir'
>>> backend.isdir(filepath)
True
```

isfile (*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path is a file.

参数 **filepath** (*str or Path*) –Path to be checked whether it is a file.

返回 Return `True` if `filepath` points to a file, `False` otherwise.

返回类型 bool

实际案例

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.isfile(filepath)
True
```

join_path (*filepath: Union[str, pathlib.Path]*, **filepaths: Union[str, pathlib.Path]*) → str

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

参数 **filepath** (*str or Path*) –Path to be concatenated.

返回 The result after concatenation.

返回类型 str

实际案例

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.join_path(filepath, 'another/path')
'petrel://path/of/file/another/path'
>>> backend.join_path(filepath, '/another/path')
'petrel://path/of/file/another/path'
```

list_dir_or_file(*dir_path: Union[str, pathlib.Path]*, *list_dir: bool = True*, *list_file: bool = True*, *suffix: Optional[Union[str, Tuple[str]]] = None*, *recursive: bool = False*) → Iterator[str]

Scan a directory to find the interested directories or files in arbitrary order.

注解: Petrel has no concept of directories but it simulates the directory hierarchy in the filesystem through public prefixes. In addition, if the returned path ends with '/', it means the path is a public prefix which is a logical directory.

注解: *list_dir_or_file()* returns the path relative to `dir_path`. In addition, the returned path of directory will not contains the suffix '/' which is consistent with other backends.

参数

- **dir_path** (*str | Path*) –Path of the directory.
- **list_dir** (*bool*) –List the directories. Defaults to True.
- **list_file** (*bool*) –List the path of files. Defaults to True.
- **suffix** (*str or tuple[str], optional*) –File suffix that we are interested in. Defaults to None.
- **recursive** (*bool*) –If set to True, recursively scan the directory. Defaults to False.

生成器 *Iterable[str]* –A relative path to `dir_path`.

实际案例

```
>>> backend = PetrelBackend()
>>> dir_path = 'petrel://path/of/dir'
>>> # list those files and directories in current directory
>>> for file_path in backend.list_dir_or_file(dir_path):
...     print(file_path)
>>> # only list files
>>> for file_path in backend.list_dir_or_file(dir_path, list_dir=False):
...     print(file_path)
>>> # only list directories
>>> for file_path in backend.list_dir_or_file(dir_path, list_file=False):
...     print(file_path)
>>> # only list files ending with specified suffixes
>>> for file_path in backend.list_dir_or_file(dir_path, suffix='.txt'):
...     print(file_path)
>>> # list all files and directory recursively
>>> for file_path in backend.list_dir_or_file(dir_path, recursive=True):
...     print(file_path)
```

register_backend	Register a backend.

14.1.7 mmeval.fileio.register_backend

mmeval.fileio.**register_backend**(*name: str*, *backend: Optional[Type[*mmeval.fileio.backends.base.BaseStorageBackend*]] = None*, *force: bool = False*, *prefixes: Optional[Union[str, list, tuple]] = None*)

Register a backend.

参数

- **name** (*str*) –The name of the registered backend.
- **backend** (*class, optional*) –The backend class to be registered, which must be a subclass of *BaseStorageBackend*. When this method is used as a decorator, backend is None. Defaults to None.
- **force** (*bool*) –Whether to override the backend if the name has already been registered. Defaults to False.
- **prefixes** (*str or list[str] or tuple[str], optional*) –The prefix of the registered storage backend. Defaults to None.

This method can be used as a normal method or a decorator.

实际案例

```
>>> class NewBackend(BaseStorageBackend):
...     def get(self, filepath):
...         return filepath
...
...     def get_text(self, filepath):
...         return filepath
>>> register_backend('new', NewBackend)
```

```
>>> @register_backend('new')
... class NewBackend(BaseStorageBackend):
...     def get(self, filepath):
...         return filepath
...
...     def get_text(self, filepath):
...         return filepath
```

14.2 File Handler

BaseFileHandler	A base class for file handler.
JsonHandler	A Json handler that parse json data from file object.
PickleHandler	A Pickle handler that parse pickle data from file object.
YamlHandler	A Yaml handler that parse yaml data from file object.

14.2.1 BaseFileHandler

class mmeval.fileio.**BaseFileHandler**

A base class for file handler.

14.2.2 JsonHandler

class mmeval.fileio.**JsonHandler**

A Json handler that parse json data from file object.

14.2.3 PickleHandler

class mmeval.fileio.**PickleHandler**

A Pickle handler that parse pickle data from file object.

14.2.4 YamlHandler

class mmeval.fileio.**YamlHandler**

A Yaml handler that parse yaml data from file object.

register_handler	A decorator that register a handler for some file extensions.

14.2.5 mmeval.fileio.register_handler

mmeval.fileio.**register_handler**(*file_formats*, ***kwargs*)

A decorator that register a handler for some file extensions.

14.3 File IO

load	Load data from json/yaml/pickle files.
exists	Check whether a file path exists.
get	Read bytes from a given `filepath` with 'rb' mode.
get_file_backend	Return a file backend based on the prefix of uri or backend_args.
get_local_path	Download data from `filepath` and write the data to local path.
get_text	Read text from a given `filepath` with 'r' mode.
isdir	Check whether a file path is a directory.
isfile	Check whether a file path is a file.
join_path	Concatenate all file paths.
list_dir_or_file	Scan a directory to find the interested directories or files in arbitrary order.

14.3.1 mmeval.fileio.load

mmeval.fileio.**load**(*file*, *file_format=None*, *backend_args=None*, ***kwargs*)

Load data from json/yaml/pickle files.

This method provides a unified api for loading data from serialized files.

`load` supports loading data from serialized files those can be storaged in different backends.

参数

- **file** (str or `Path` or file-like object) –Filename or a file-like object.
- **file_format** (*str, optional*) –If not specified, the file format will be inferred from the file extension, otherwise use the specified one. Currently supported formats include "json", "yaml/yml" and "pickle/pkl".
- **backend_args** (*dict, optional*) –Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

实际案例

```
>>> load('/path/of/your/file')  # file is storaged in disk
>>> load('https://path/of/your/file')  # file is storaged in Internet
>>> load('s3://path/of/your/file')  # file is storaged in petrel
```

返回 The content from the file.

14.3.2 mmeval.fileio.exists

mmeval.fileio.**exists**(*filepath: Union[str, pathlib.Path]*, *backend_args: Optional[dict] = None*) → bool

Check whether a file path exists.

参数

- **filepath** (*str or Path*) –Path to be checked whether exists.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

返回 Return `True` if `filepath` exists, `False` otherwise.

返回类型 bool

实际案例

```
>>> filepath = '/path/of/file'
>>> exists(filepath)
True
```

14.3.3 mmeval.fileio.get

mmeval.fileio.**get**(*filepath: Union[str, pathlib.Path]*, *backend_args: Optional[dict] = None*) → bytes

Read bytes from a given `filepath` with 'rb' mode.

参数

- **filepath** (*str or Path*) –Path to read data.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

返回 Expected bytes object.

返回类型 bytes

实际案例

```
>>> filepath = '/path/of/file'
>>> get(filepath)
b'hello world'
```

14.3.4 mmeval.fileio.get_file_backend

mmeval.fileio.**get_file_backend**(*uri: Optional[Union[str, pathlib.Path]] = None*, ***, *backend_args: Optional[dict] = None*, *enable_singleton: bool = False*)

Return a file backend based on the prefix of uri or backend_args.

参数

- **uri** (*str or Path*) –Uri to be parsed that contains the file prefix.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.
- **enable_singleton** (*bool*) –Whether to enable the singleton pattern. If it is True, the backend created will be reused if the signature is same with the previous one. Defaults to False.

返回 Instantiated Backend object.

返回类型 *BaseStorageBackend*

实际案例

```
>>> # get file backend based on the prefix of uri
>>> uri = 's3://path/of/your/file'
>>> backend = get_file_backend(uri)
>>> # get file backend based on the backend_args
>>> backend = get_file_backend(backend_args={'backend': 'petrel'})
>>> # backend name has a higher priority if 'backend' in backend_args
>>> backend = get_file_backend(uri, backend_args={'backend': 'petrel'})
```

14.3.5 mmeval.fileio.get_local_path

mmeval.fileio.**get_local_path**(*filepath: Union[str, pathlib.Path]*, *backend_args: Optional[dict] = None*) → Generator[Union[str, pathlib.Path], None, None]

Download data from `filepath` and write the data to local path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

注解： If the `filepath` is a local path, just return itself and it will not be released (removed).

参数

- **filepath** (*str or Path*) –Path to be read data.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

生成器 *Iterable[str]* –Only yield one path.

实际案例

```
>>> with get_local_path('s3://bucket/abc.jpg') as path:
...     # do something here
```

14.3.6 mmeval.fileio.get_text

mmeval.fileio.**get_text**(*filepath: Union[str, pathlib.Path]*, *encoding='utf-8'*, *backend_args: Optional[dict] = None*) → str

Read text from a given `filepath` with 'r' mode.

参数

- **filepath** (*str or Path*) –Path to read data.
- **encoding** (*str*) –The encoding format used to open the `filepath`. Defaults to 'utf-8'.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

返回 Expected text reading from `filepath`.

返回类型 str

实际案例

```
>>> filepath = '/path/of/file'
>>> get_text(filepath)
'hello world'
```

14.3.7 mmeval.fileio.isdir

mmeval.fileio.**isdir**(*filepath: Union[str, pathlib.Path]*, *backend_args: Optional[dict] = None*) → bool

Check whether a file path is a directory.

参数

- **filepath** (*str or Path*) –Path to be checked whether it is a directory.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

返回 Return `True` if `filepath` points to a directory, `False` otherwise.

返回类型 bool

实际案例

```
>>> filepath = '/path/of/dir'
>>> isdir(filepath)
True
```

14.3.8 mmeval.fileio.isfile

mmeval.fileio.**isfile**(*filepath: Union[str, pathlib.Path]*, *backend_args: Optional[dict] = None*) → bool

Check whether a file path is a file.

参数

- **filepath** (*str or Path*) –Path to be checked whether it is a file.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

返回 Return `True` if `filepath` points to a file, `False` otherwise.

返回类型 bool

实际案例

```
>>> filepath = '/path/of/file'
>>> isfile(filepath)
True
```

14.3.9 mmeval.fileio.join_path

mmeval.fileio.**join_path**(*filepath: Union[str, pathlib.Path]*, **filepaths: Union[str, pathlib.Path]*, *backend_args: Optional[dict] = None*) → Union[str, pathlib.Path]

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

参数

- **filepath** (*str or Path*) –Path to be concatenated.
- ***filepaths** (*str or Path*) –Other paths to be concatenated.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

返回 The result of concatenation.

返回类型 str

实际案例

```
>>> filepath1 = '/path/of/dir1'
>>> filepath2 = 'dir2'
>>> filepath3 = 'path/of/file'
>>> join_path(filepath1, filepath2, filepath3)
'/path/of/dir/dir2/path/of/file'
```

14.3.10 mmeval.fileio.list_dir_or_file

mmeval.fileio.**list_dir_or_file**(*dir_path: Union[str, pathlib.Path]*, *list_dir: bool = True*, *list_file: bool = True*, *suffix: Optional[Union[str, Tuple[str]]] = None*, *recursive: bool = False*, *backend_args: Optional[dict] = None*) → Iterator[str]

Scan a directory to find the interested directories or files in arbitrary order.

注解: *list_dir_or_file()* returns the path relative to dir_path.

参数

- **dir_path** (*str or Path*) –Path of the directory.
- **list_dir** (*bool*) –List the directories. Defaults to True.
- **list_file** (*bool*) –List the path of files. Defaults to True.
- **suffix** (*str or tuple[str], optional*) –File suffix that we are interested in. Defaults to None.
- **recursive** (*bool*) –If set to True, recursively scan the directory. Defaults to False.
- **backend_args** (*dict, optional*) –Arguments to instantiate the corresponding backend. Defaults to None.

生成器 *Iterable[str]* –A relative path to dir_path.

实际案例

```
>>> dir_path = '/path/of/dir'
>>> for file_path in list_dir_or_file(dir_path):
...     print(file_path)
>>> # list those files and directories in current directory
>>> for file_path in list_dir_or_file(dir_path):
...     print(file_path)
>>> # only list files
>>> for file_path in list_dir_or_file(dir_path, list_dir=False):
...     print(file_path)
>>> # only list directories
>>> for file_path in list_dir_or_file(dir_path, list_file=False):
...     print(file_path)
>>> # only list files ending with specified suffixes
>>> for file_path in list_dir_or_file(dir_path, suffix='.txt'):
...     print(file_path)
>>> # list all files and directory recursively
>>> for file_path in list_dir_or_file(dir_path, recursive=True):
...     print(file_path)
```

14.4 Parse File

dict_from_file	Load a text file and parse the content as a dict.
list_from_file	Load a text file and parse the content as a list of strings.

14.4.1 mmeval.fileio.dict_from_file

mmeval.fileio.**dict_from_file**(*filename*, *key_type=<class 'str'>*, *encoding='utf-8'*, *backend_args=None*)

Load a text file and parse the content as a dict.

Each line of the text file will be two or more columns split by whitespaces or tabs. The first column will be parsed as dict keys, and the following columns will be parsed as dict values.

`dict_from_file` supports loading a text file which can be storaged in different backends and parsing the content as a dict.

参数

- **filename** (*str*) –Filename.
- **key_type** (*type*) –Type of the dict keys. str is user by default and type conversion will be performed if specified.

- **encoding** (*str*) –Encoding used to open the file. Defaults to utf-8.
- **backend_args** (*dict, optional*) –Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

实际案例

```
>>> dict_from_file('/path/of/your/file')  # disk
{'key1': 'value1', 'key2': 'value2'}
>>> dict_from_file('s3://path/of/your/file')  # ceph or petrel
{'key1': 'value1', 'key2': 'value2'}
```

返回 The parsed contents.

返回类型 dict

14.4.2 mmeval.fileio.list_from_file

mmeval.fileio.**list_from_file**(*filename*, *prefix=''*, *offset=0*, *max_num=0*, *encoding='utf-8'*, *backend_args=None*)

Load a text file and parse the content as a list of strings.

list_from_file supports loading a text file which can be storaged in different backends and parsing the content as a list for strings.

参数

- **filename** (*str*) –Filename.
- **prefix** (*str*) –The prefix to be inserted to the beginning of each item.
- **offset** (*int*) –The offset of lines.
- **max_num** (*int*) –The maximum number of lines to be read, zeros and negatives mean no limitation.
- **encoding** (*str*) –Encoding used to open the file. Defaults to utf-8.
- **backend_args** (*dict, optional*) –Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

实际案例

```
>>> list_from_file('/path/of/your/file')  # disk
['hello', 'world']
>>> list_from_file('s3://path/of/your/file')  # ceph or petrel
['hello', 'world']
```

返回 A list of strings.

返回类型 list[str]

CHAPTER 15

mmeval.metrics

mmeval.metrics

15.1 Metrics

Accuracy	Top-k accuracy evaluation metric.
SingleLabelMetric	alias of `mmeval.metrics.precision_recall_f1score.SingleLabelPrecisionRecallF1score`
MultiLabelMetric	alias of `mmeval.metrics.precision_recall_f1score.MultiLabelPrecisionRecallF1score`
AveragePrecision	Calculate the average precision with respect of classes.
MeanIoU	MeanIoU evaluation metric.
COCODetection	COCO object detection task evaluation metric.
ProposalRecall	Proposals recall evaluation metric.
VOCMeanAP	Pascal VOC evaluation metric.
OIDMeanAP	Open Images Dataset detection evaluation metric.

下页继续

表 1 – 续上页

F1Score	Compute F1 scores.
HmeanIoU	HmeanIoU metric.
EndPointError	EndPointError evaluation metric.
PCKAccuracy	PCK accuracy evaluation metric, which is widely used in pose estimation.
MpiiPCKAccuracy	PCKh accuracy evaluation metric for MPII dataset.
JhmdbPCKAccuracy	PCK accuracy evaluation metric for Jhmdb dataset.
AVAMeanAP	AVA evaluation metric.
StructuralSimilarity	Calculate StructuralSimilarity (structural similarity).
SignalNoiseRatio	Signal-to-Noise Ratio.
PeakSignalNoiseRatio	Peak Signal-to-Noise Ratio.
MeanAbsoluteError	Mean Absolute Error metric for image.
MeanSquaredError	Mean Squared Error metric for image.
BLEU	Bilingual Evaluation Understudy metric.
SumAbsoluteDifferences	Sum of Absolute Differences metric for image.
GradientError	Gradient error for evaluating alpha matte prediction.
MattingMeanSquaredError	Mean Squared Error metric for image matting.
ConnectivityError	Connectivity error for evaluating alpha matte prediction.
DOTAMeanAP	DOTA evaluation metric.
ROUGE	Calculate Rouge Score used for automatic summarization.
NaturalImageQualityEvaluator	Calculate Natural Image Quality Evaluator(NIQE) metric.
Perplexity	Perplexity measures how well a language model predicts a text sample.
CharRecallPrecision	Calculate the char level recall & precision.
KeypointEndPointError	EPE evaluation metric.
KeypointAUC	AUC evaluation metric.
KeypointNME	NME evaluation metric.
WordAccuracy	Calculate the word level accuracy.

15.1.1 Accuracy

class mmeval.metrics.**Accuracy**(*topk: Union[int, Sequence[int]] = (1)*, *thrs: Optional[Union[float, Sequence[Optional[float]]]] = 0.0*, ***kwargs*)

Top-k accuracy evaluation metric.

This metric computes the accuracy based on the given topk and thresholds.

Currently, this metric supports 5 kinds of inputs, i.e. `numpy.ndarray`, `torch.Tensor`, `oneflow.`

`Tensor`, `tensorflow.Tensor` and `paddle.Tensor`, and the implementation for the calculation depends on the inputs type.

参数

- **topk** (*int | Sequence[int]*) –If the predictions in `topk` matches the target, the predictions will be regarded as correct ones. Defaults to 1.
- **thrs** (*Sequence[float | None] | float | None*) –Predictions with scores under the thresholds are considered negative. None means no thresholds. Defaults to 0.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import Accuracy
>>> accuracy = Accuracy()
```

Use NumPy implementation:

```
>>> import numpy as np
>>> labels = np.asarray([0, 1, 2, 3])
>>> preds = np.asarray([0, 2, 1, 3])
>>> accuracy(preds, labels)
{'top1': 0.5}
```

Use PyTorch implementation:

```
>>> import torch
>>> labels = torch.Tensor([0, 1, 2, 3])
>>> preds = torch.Tensor([0, 2, 1, 3])
>>> accuracy(preds, labels)
{'top1': 0.5}
```

Computing top-k accuracy with specified threold:

```
>>> labels = np.asarray([0, 1, 2, 3])
>>> preds = np.asarray([
    [0.7, 0.1, 0.1, 0.1],
    [0.1, 0.3, 0.4, 0.2],
    [0.3, 0.4, 0.2, 0.1],
    [0.0, 0.0, 0.1, 0.9]])
>>> accuracy = Accuracy(topk=(1, 2, 3))
>>> accuracy(preds, labels)
{'top1': 0.5, 'top2': 0.75, 'top3': 1.0}
>>> accuracy = Accuracy(topk=2, thrs=(0.1, 0.5))
```

(下页继续)

(续上页)

```
>>> accuracy(preds, labels)
{'top2_thr-0.10': 0.75, 'top2_thr-0.50': 0.5}
```

Accumulate batch:

```
>>> for i in range(10):
...     labels = torch.randint(0, 4, size=(100, ))
...     predicts = torch.randint(0, 4, size=(100, ))
...     accuracy.add(predicts, labels)
>>> accuracy.compute()
```

add(*predictions: Sequence*, *labels: Sequence*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence*) –Predictions from the model. It can be labels (N,), or scores of every class (N, C).
- **labels** (*Sequence*) –The ground truth labels. It should be (N,).

compute_metric(*results: List[Union[Iterable, numpy.number, torch.Tensor, tensorflow.Tensor, paddle.Tensor, jax.Array, flow.Tensor]]*) → Dict[str, float]

Compute the accuracy metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*list*) –A list that consisting the correct numbers. This list has already been synced across all ranks.

返回 The computed accuracy metric.

返回类型 Dict[str, float]

15.1.2 SingleLabelMetric

mmeval.metrics.**SingleLabelMetric**

alias of `mmeval.metrics.precision_recall_f1score.SingleLabelPrecisionRecallF1score`

15.1.3 MultiLabelMetric

mmeval.metrics.**MultiLabelMetric**

alias of mmeval.metrics.precision_recall_f1score.MultiLabelPrecisionRecallF1score

15.1.4 AveragePrecision

class mmeval.metrics.**AveragePrecision**(*average: Optional[str] = 'macro'*, ***kwargs*)

Calculate the average precision with respect of classes.

参数

- **average** (*str, optional*) –The average method. It supports two modes:
  - ”*macro*”：**Calculate metrics for each category, and calculate** the mean value over all categories.
  - *None*: Return scores of all categories.
- **to "macro".** (*Defaults*) –

引用

实际案例

```
>>> from mmeval import AveragePrecision
>>> average_precision = AveragePrecision()
```

Use Builtin implementation with label-format labels:

```
>>> preds = [[0.9, 0.8, 0.3, 0.2],
             [0.1, 0.2, 0.2, 0.1],
             [0.7, 0.5, 0.9, 0.3],
             [0.8, 0.1, 0.1, 0.2]]
>>> labels = [[0, 1], [1], [2], [0]]
>>> average_precision(preds, labels)
{'mAP': 70.833..}
```

Use Builtin implementation with one-hot encoding labels:

```
>>> preds = [[0.9, 0.8, 0.3, 0.2],
             [0.1, 0.2, 0.2, 0.1],
             [0.7, 0.5, 0.9, 0.3],
             [0.8, 0.1, 0.1, 0.2]]
>>> labels = [[1, 1, 0, 0],
```

(下页继续)

(续上页)

```
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [1, 0, 0, 0]]
>>> average_precision(preds, labels)
{'mAP': 70.833..}
```

Use NumPy implementation with label-format labels:

```
>>> import numpy as np
>>> preds = np.array([[0.9, 0.8, 0.3, 0.2],
                      [0.1, 0.2, 0.2, 0.1],
                      [0.7, 0.5, 0.9, 0.3],
                      [0.8, 0.1, 0.1, 0.2]])
>>> labels = [np.array([0, 1]), np.array([1]), np.array([2]), np.array([0])] #␣
↪noqa
>>> average_precision(preds, labels)
{'mAP': 70.833..}
```

Use PyTorch implementation with one-hot encoding labels:

```
>>> import torch
>>> preds = torch.Tensor([[0.9, 0.8, 0.3, 0.2],
                          [0.1, 0.2, 0.2, 0.1],
                          [0.7, 0.5, 0.9, 0.3],
                          [0.8, 0.1, 0.1, 0.2]])
>>> labels = torch.Tensor([[1, 1, 0, 0],
                           [0, 1, 0, 0],
                           [0, 0, 1, 0],
                           [1, 0, 0, 0]])
>>> average_precision(preds, labels)
{'mAP': 70.833..}
```

Computing with *None* average mode:

```
>>> preds = np.array([[0.9, 0.8, 0.3, 0.2],
                      [0.1, 0.2, 0.2, 0.1],
                      [0.7, 0.5, 0.9, 0.3],
                      [0.8, 0.1, 0.1, 0.2]])
>>> labels = [np.array([0, 1]), np.array([1]), np.array([2]), np.array([0])] #␣
↪noqa
>>> average_precision = AveragePrecision(average=None)
>>> average_precision(preds, labels)
{'AP_classwise': [100.0, 83.33, 100.00, 0.0]}  # rounded results
```

Accumulate batch:

```
>>> for i in range(10):
...     preds = torch.randint(0, 4, size=(100, 10))
...     labels = torch.randint(0, 4, size=(100, ))
...     average_precision.add(preds, labels)
>>> average_precision.compute()
```

add(*preds: Sequence*, *labels: Sequence*) → None

Add the intermediate results to *self._results*.

参数

- **preds** (*Sequence*) –Predictions from the model. It should be scores of every class (N, C).
- **labels** (*Sequence*) –The ground truth labels. It should be (N,) for label-format, or (N, C) for one-hot encoding.

compute_metric(*results: List[Union[Tuple[Union[numpy.ndarray, numpy.number], Union[numpy.ndarray, numpy.number]], Tuple[torch.Tensor, torch.Tensor], Tuple[oneflow.Tensor, oneflow.Tensor], Tuple[Union[int, Sequence[Union[int, float]]], Union[int, Sequence[int]]]]]*) → Dict[str, float]

Compute the metric.

Currently, there are 3 implementations of this method: NumPy and PyTorch and OneFlow. Which implementation to use is determined by the type of the calling parameters. e.g. *numpy.ndarray* or *torch.Tensor*, *oneflow.Tensor*.

This method would be invoked in *BaseMetric.compute* after distributed synchronization.

参数

- **(List[Union[NUMPY_IMPL_HINTS**(*results*) –
- **TORCH_IMPL_HINTS** –

:param : :param ONEFLOW_IMPL_HINTS]]): A list of tuples that consisting the :param prediction and label. This list has already been synced across: :param all ranks.:

返回 The computed metric.

返回类型 Dict[str, float]

15.1.5 MeanIoU

class mmeval.metrics.**MeanIoU**(*num_classes: Optional[int] = None*, *ignore_index: int = 255*, *nan_to_num: Optional[int] = None*, *beta: int = 1*, *classwise_results: bool = False*, ***kwargs*)

MeanIoU evaluation metric.

MeanIoU is a widely used evaluation metric for image semantic segmentation.

In addition to mean iou, it will also compute and return accuracy, mean accuracy, mean dice, mean precision, mean recall and mean f-score.

This metric supports 6 kinds of inputs, i.e. `numpy.ndarray`, `torch.Tensor`, `oneflow.Tensor`, `tensorflow.Tensor`, `paddle.Tensor``and``jax.Array`, and the implementation for the calculation depends on the inputs type.

参数

- **num_classes** (*int, optional*) –The number of classes. If None, it will be obtained from the 'num_classes' or 'classes' field in *self.dataset_meta*. Defaults to None.
- **ignore_index** (*int, optional*) –Index that will be ignored in evaluation. Defaults to 255.
- **nan_to_num** (*int, optional*) –If specified, NaN values will be replaced by the numbers defined by the user. Defaults to None.
- **beta** (*int, optional*) –Determines the weight of recall in the F-score. Defaults to 1.
- **classwise_results** (*bool, optional*) –Whether to return the computed results of each class. Defaults to False.
- ****kwargs** –Keyword arguments passed to `BaseMetric`.

实际案例

```
>>> from mmeval import MeanIoU
>>> miou = MeanIoU(num_classes=4)
```

Use NumPy implementation:

```
>>> import numpy as np
>>> labels = np.asarray([[[0, 1, 1], [2, 3, 2]]])
>>> preds = np.asarray([[[0, 2, 1], [1, 3, 2]]])
>>> miou(preds, labels)
{'aAcc': 0.6666666666666666,
 'mIoU': 0.6666666666666666,
 'mAcc': 0.75,
```

(下页继续)

(续上页)

```
 'mDice': 0.75,
 'mPrecision': 0.75,
 'mRecall': 0.75,
 'mFscore': 0.75,
 'kappa': 0.5384615384615384}
```

Use PyTorch implementation:

```
>>> import torch
>>> labels = torch.Tensor([[[0, 1, 1], [2, 3, 2]]])
>>> preds = torch.Tensor([[[0, 2, 1], [1, 3, 2]]])
>>> miou(preds, labels)
{'aAcc': 0.6666666666666666,
 'mIoU': 0.6666666666666666,
 'mAcc': 0.75,
 'mDice': 0.75,
 'mPrecision': 0.75,
 'mRecall': 0.75,
 'mFscore': 0.75,
 'kappa': 0.5384615384615384}
```

Accumulate batch:

```
>>> for i in range(10):
...     labels = torch.randint(0, 4, size=(100, 10, 10))
...     predicts = torch.randint(0, 4, size=(100, 10, 10))
...     miou.add(predicts, labels)
>>> miou.compute()
```

add(*predictions: Sequence*, *labels: Sequence*) → None

Process one batch of data and predictions.

Calculate the following 3 stuff from the inputs and store them in `self._results`:

- num_tp_per_class: the number of true positive per-class.
- num_gts_per_class: the number of ground truth per-class.
- num_preds_per_class: the number of predicition per-class.

参数

- **predictions** (*Sequence*) –A sequence of the predicted segmentation mask.
- **labels** (*Sequence*) –A sequence of the segmentation mask labels.

compute_confusion_matrix

Compute confusion matrix with NumPy.

参数

- **prediction** (*numpy.ndarray*) –The predicition.
- **label** (*numpy.ndarray*) –The ground truth.
- **num_classes** (*int*) –The number of classes.

返回 The computed confusion matrix.

返回类型 numpy.ndarray

compute_metric(*results: List[Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]]*) → dict

Compute the MeanIoU metric.

This method would be invoked in *BaseMetric.compute* after distributed synchronization.

参数 **results** (*List[tuple]*) –This list has already been synced across all ranks. This is a list of tuple, and each tuple has the following elements:

- (List[numpy.ndarray]): Each element in the list is the number of true positive per-class on a sample.
- (List[numpy.ndarray]): Each element in the list is the number of ground truth per-class on a sample.
- (List[numpy.ndarray]): Each element in the list is the number of predicition per-class on a sample.

返回

The computed metric, with following keys:

- aAcc, the overall accuracy, namely pixel accuracy.
- mIoU, the mean Intersection-Over-Union (IoU) for all classes.
- mAcc, the mean accuracy for all classes, namely mean pixel

accuracy. - mDice, the mean dice coefficient for all claases. - mPrecision, the mean precision for all classes. - mRecall, the mean recall for all classes. - mFscore, the mean f-score for all classes. - kappa, the Cohen’s kappa coefficient. - classwise_result, the evaluate results of each classes. This would be returned if `self.classwise_result` is True.

返回类型 Dict

property num_classes: int

Returns the number of classes.

The number of classes should be set during initialization, otherwise it will be obtained from the ‘classes’ or ‘num_classes’ field in `self.dataset_meta`.

引发 **RuntimeError** –If the num_classes is not set.

返回 The number of classes.

返回类型 int

15.1.6 COCODetection

class mmeval.metrics.**COCODetection**(*ann_file: Optional[str] = None*, *metric: Union[str, List[str]] = 'bbox'*, *iou_thrs: Optional[Union[float, Sequence[float]]] = None*, *classwise: bool = False*, *proposal_nums: Sequence[int] = (1, 10, 100)*, *metric_items: Optional[Sequence[str]] = None*, *format_only: bool = False*, *outfile_prefix: Optional[str] = None*, *gt_mask_area: bool = True*, *backend_args: Optional[dict] = None*, *print_results: bool = True*, ***kwargs*)

COCO object detection task evaluation metric.

Evaluate AR, AP, and mAP for detection tasks including proposal/box detection and instance segmentation. Please refer to https://cocodataset.org/#detection-eval for more details.

参数

- **ann_file** (*str, optional*) –Path to the coco format annotation file. If not specified, ground truth annotations from the dataset will be converted to coco format. Defaults to None.
- **metric** (*str | List[str]*) –Metrics to be evaluated. Valid metrics include 'bbox', 'segm', and 'proposal'. Defaults to 'bbox'.
- **iou_thrs** (*float | List[float], optional*) –IoU threshold to compute AP and AR. If not specified, IoUs from 0.5 to 0.95 will be used. Defaults to None.
- **classwise** (*bool*) –Whether to return the computed results of each class. Defaults to False.
- **proposal_nums** (*Sequence[int]*) –Numbers of proposals to be evaluated. Defaults to (1, 10, 100).
- **metric_items** (*List[str], optional*) –Metric result names to be recorded in the evaluation result. Defaults to None.
- **format_only** (*bool*) –Format the output results without perform evaluation. It is useful when you want to format the result to a specific format and submit it to the test server. Defaults to False.
- **outfile_prefix** (*str, optional*) –The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Defaults to None.

- **gt_mask_area** (*bool*) –Whether to calculate GT mask area when not loading ann_file. If True, the GT instance area will be the mask area, else the bounding box area. It will not be used when loading ann_file. Defaults to True.
- **backend_args** (*dict, optional*) –Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.
- **print_results** (*bool*) –Whether to print the results. Defaults to True.
- **logger** (*Logger, optional*) –logger used to record messages. When set to `None`, the default logger will be used. Defaults to None.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> import numpy as np
>>> from mmeval import COCODetection
>>> try:
>>>     from mmeval.metrics.utils.coco_wrapper import mask_util
>>> except ImportError as e:
>>>     mask_util = None
>>>
>>> num_classes = 4
>>> fake_dataset_metas = {
...     'classes': tuple([str(i) for i in range(num_classes)])
... }
>>>
>>> coco_det_metric = COCODetection(
...     dataset_meta=fake_dataset_metas,
...     metric=['bbox', 'segm']
... )
>>> def _gen_bboxes(num_bboxes, img_w=256, img_h=256):
...     # random generate bounding boxes in 'xyxy' formart.
...     x = np.random.rand(num_bboxes, ) * img_w
...     y = np.random.rand(num_bboxes, ) * img_h
...     w = np.random.rand(num_bboxes, ) * (img_w - x)
...     h = np.random.rand(num_bboxes, ) * (img_h - y)
...     return np.stack([x, y, x + w, y + h], axis=1)
>>>
>>> def _gen_masks(bboxes, img_w=256, img_h=256):
...     if mask_util is None:
...         raise ImportError(
...             'Please try to install official pycocotools by '
...             '"pip install pycocotools"')
```

(下页继续)

(续上页)

```
...     masks = []
...     for i, bbox in enumerate(bboxes):
...         mask = np.zeros((img_h, img_w))
...         bbox = bbox.astype(np.int32)
...         box_mask = (np.random.rand(
...             bbox[3] - bbox[1],
...             bbox[2] - bbox[0]) > 0.3).astype(np.int32)
...         mask[bbox[1]:bbox[3], bbox[0]:bbox[2]] = box_mask
...         masks.append(
...             mask_util.encode(
...                 np.array(mask[:, :, np.newaxis], order='F',
...                          dtype='uint8'))[0])  # encoded with RLE
...     return masks
>>>
>>> img_id = 1
>>> img_w, img_h = 256, 256
>>> num_bboxes = 10
>>> pred_boxes = _gen_bboxes(
...     num_bboxes=num_bboxes,
...     img_w=img_w,
...     img_h=img_h)
>>> pred_masks = _gen_masks(
...     bboxes=pred_boxes,
...     img_w=img_w,
...     img_h=img_h)
>>> prediction = {
...     'img_id': img_id,
...     'bboxes': pred_boxes,
...     'scores': np.random.rand(num_bboxes, ),
...     'labels': np.random.randint(0, num_classes, size=(num_bboxes, )),
...     'masks': pred_masks
... }
>>> gt_boxes = _gen_bboxes(
...     num_bboxes=num_bboxes,
...     img_w=img_w,
...     img_h=img_h)
>>> gt_masks = _gen_masks(
...     bboxes=pred_boxes,
...     img_w=img_w,
...     img_h=img_h)
>>> groundtruth = {
...     'img_id': img_id,
...     'width': img_w,
```

(下页继续)

(续上页)

```
...     'height': img_h,
...     'bboxes': gt_boxes,
...     'labels': np.random.randint(0, num_classes, size=(num_bboxes, )),
...     'masks': gt_masks,
...     'ignore_flags': np.zeros(num_bboxes)
... }
>>> coco_det_metric(predictions=[prediction, ], groundtruths=[groundtruth, ])
{'bbox_mAP': ..., 'bbox_mAP_50': ..., ...,
 'segm_mAP': ..., 'segm_mAP_50': ..., ...,
 'bbox_result': ..., 'segm_result': ..., ...}
```

add(*predictions: Sequence[Dict]*, *groundtruths: Sequence[Dict]*) → None

Add the intermediate results to *self._results*.

参数

- **predictions** (*Sequence[dict]*) –A sequence of dict. Each dict representing a detection result for an image, with the following keys:
  - img_id (int): Image id.
  - bboxes (numpy.ndarray): Shape (N, 4), the predicted bounding bboxes of this image, in 'xyxy' foramrt.
  - scores (numpy.ndarray): Shape (N,), the predicted scores of bounding boxes.
  - labels (numpy.ndarray): Shape (N,), the predicted labels of bounding boxes.
  - masks (list[RLE], optional): The predicted masks.
  - mask_scores (np.array, optional): Shape (N,), the predicted scores of masks.
- **groundtruths** (*Sequence[dict]*) –A sequence of dict. If load from *ann_file*, the dict inside can be empty. Else, each dict represents a groundtruths for an image, with the following keys:
  - img_id (int): Image id.
  - width (int): The width of the image.
  - height (int): The height of the image.
  - bboxes (numpy.ndarray): Shape (K, 4), the ground truth bounding bboxes of this image, in 'xyxy' foramrt.
  - labels (numpy.ndarray): Shape (K,), the ground truth labels of bounding boxes.
  - masks (list[RLE], optional): The predicted masks.
  - ignore_flags (numpy.ndarray, optional): Shape (K,), the ignore flags.

add_predictions(*predictions: Sequence[Dict]*) → None

Add predictions only.

If the *ann_file* has been passed, we can add predictions only.

参数 **predictions**(*Sequence[dict]*) –Refer to *COCODetection.add*.

property classes: list

Get classes from self.dataset_meta.

compute_metric(*results: list*) → dict

Compute the COCO metrics.

参数 **results**(*List[tuple]*) –A list of tuple. Each tuple is the prediction and ground truth of an image. This list has already been synced across all ranks.

返回 The computed metric. The keys are the names of the metrics, and the values are corresponding results.

返回类型 dict

gt_to_coco_json(*gt_dicts: Sequence[dict]*, *outfile_prefix: str*) → str

Convert ground truth to coco format json file.

参数

- **gt_dicts**(*Sequence[dict]*) –Ground truth of the dataset.
- **outfile_prefix**(*str*) –The filename prefix of the json files. If the prefix is "somepath/xxx", the json file will be named "somepath/xxx.gt.json".

返回 The filename of the json file.

返回类型 str

results2json(*results: Sequence[dict]*, *outfile_prefix: str*) → dict

Dump the detection results to a COCO style json file.

There are 3 types of results: proposals, bbox predictions, mask predictions, and they have different data types. This method will automatically recognize the type, and dump them to json files.

参数

- **results**(*Sequence[dict]*) –Testing results of the dataset.
- **outfile_prefix**(*str*) –The filename prefix of the json files. If the prefix is "somepath/xxx", the json files will be named "somepath/xxx.bbox.json", "somepath/xxx.segm.json", "somepath/xxx.proposal.json".

返回 Possible keys are "bbox", "segm", "proposal", and values are corresponding filenames.

返回类型 dict

xyxy2xywh(*bbox: numpy.ndarray*) → list

Convert `xyxy` style bounding boxes to `xywh` style for COCO evaluation.

参数 **bbox** (*np.ndarray*) –The bounding boxes, shape (4,), in `xyxy` order.

返回 The converted bounding boxes, in `xywh` order.

返回类型 list[float]

15.1.7 ProposalRecall

class mmeval.metrics.**ProposalRecall**(*iou_thrs: Optional[Union[float, Sequence[float]]] = None*, *proposal_nums: Union[int, Sequence[int]] = (1, 10, 100, 1000)*, *use_legacy_coordinate: bool = False*, *nproc: int = 4*, *print_results: bool = True*, ***kwargs*)

Proposals recall evaluation metric.

The speed of calculating recall is faster than COCO Detection metric.

参数

- **iou_thrs** (*float | List[float], optional*) –IoU thresholds. If not specified, IoUs from 0.5 to 0.95 will be used. Defaults to None.
- **proposal_nums** (*Sequence[int]*) –Numbers of proposals to be evaluated. Defaults to (1, 10, 100, 1000).
- **use_legacy_coordinate** (*bool*) –Whether to use coordinate system in mmdet v1.x. which means width, height should be calculated as 'x2 - x1 + 1' and 'y2 - y1 + 1' respectively. Defaults to False.
- **nproc** (*int*) –Processes used for computing TP and FP. If nproc is less than or equal to 1, multiprocessing will not be used. Defaults to 4.
- **print_results** (*bool*) –Whether to print the results. Defaults to True.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> import numpy as np
>>> from mmeval import ProposalRecall
>>>
>>> proposal_recall = ProposalRecall()
>>> def _gen_bboxes(num_bboxes, img_w=256, img_h=256):
...     # random generate bounding boxes in 'xyxy' formart.
...     x = np.random.rand(num_bboxes, ) * img_w
```

(下页继续)

(续上页)

```
...     y = np.random.rand(num_bboxes, ) * img_h
...     w = np.random.rand(num_bboxes, ) * (img_w - x)
...     h = np.random.rand(num_bboxes, ) * (img_h - y)
...     return np.stack([x, y, x + w, y + h], axis=1)
>>>
>>> prediction = {
...     'bboxes': _gen_bboxes(10),
...     'scores': np.random.rand(10, ),
... }
>>> groundtruth = {
...     'bboxes': _gen_bboxes(10),
... }
>>> proposal_recall(predictions=[prediction, ], groundtruths=[groundtruth, ])
{'AR@1': ..., 'AR@10': ..., 'AR@100': ..., 'AR@1000': ...}
```

add(*predictions: Sequence[Dict]*, *groundtruths: Sequence[Dict]*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence[dict]*) –A sequence of dict. Each dict representing a detection result for an image, with the following keys:
  - bboxes (numpy.ndarray): Shape (N, 4), the predicted bounding bboxes of this image, in 'xyxy' foramrt.
  - scores (numpy.ndarray): Shape (N, 1), the predicted scores of bounding boxes.
- **groundtruths** (*Sequence[dict]*) –A sequence of dict. Each dict represents a groundtruths for an image, with the following keys:
  - bboxes (numpy.ndarray): Shape (M, 4), the ground truth bounding bboxes of this image, in 'xyxy' foramrt.

calculate_recall(*predictions: List[dict]*, *groundtruths: List[dict]*, *pool: Optional[multiprocessing.pool.Pool]*)

Calculate recall.

参数

- **predictions** (*List[dict]*) –A list of dict. Each dict is the detection result of an image. Same as *ProposalRecall.add*.
- **groundtruths** (*List[dict]*) –A list of dict. Each dict is the ground truth of an image. Same as *ProposalRecall.add*.
- **pool** (*Optional[Pool]*) –A instance of `multiprocessing.Pool`. If None, do not use multiprocessing.

返回 Shape (len(proposal_nums), len(iou_thrs)), the recall results.

返回类型 numpy.ndarry

compute_metric(*results: list*) → dict

Compute the ProposalRecall metric.

参数 **results** (*List[tuple]*) –A list of tuple. Each tuple is the prediction and ground truth of an image. This list has already been synced across all ranks.

返回 The computed metric. The keys are the names of the proposal numbers, and the values are corresponding results.

返回类型 dict

process_proposals(*predictions: List[dict]*, *groundtruths: List[dict]*) → Tuple[List, List, int]

Process the proposals(bboxes) in predictions and groundtruths.

参数

- **predictions** (*List[dict]*) –A list of dict. Each dict is the detection result of an image. Same as *ProposalRecall.add*.
- **groundtruths** (*list[dict]*) –Same as *ProposalRecall.add*.

返回

- proposal_preds (List[numpy.ndarray]): The sorted proposals of the prediction.
- proposal_gts (List[numpy.ndarray]): The proposals of the groundtruths.
- num_gts (int): The total groundtruth numbers.

返回类型 tuple (proposal_preds, proposal_gts, num_gts)

15.1.8 VOCMeanAP

class mmeval.metrics.**VOCMeanAP**(*iou_thrs: Union[float, List[float]] = 0.5*, *scale_ranges: Optional[List[Tuple]] = None*, *num_classes: Optional[int] = None*, *eval_mode: str = 'area'*, *use_legacy_coordinate: bool = False*, *nproc: int = 4*, *drop_class_ap: bool = True*, *classwise: bool = False*, ***kwargs*)

Pascal VOC evaluation metric.

This metric computes the VOC mAP (mean Average Precision) with the given IoU thresholds and scale ranges.

参数

- **iou_thrs** (*float | List[float]*) –IoU thresholds. Defaults to 0.5.
- **scale_ranges** (*List[tuple], optional*) –Scale ranges for evaluating mAP. If not specified, all bounding boxes would be included in evaluation. Defaults to None.

- **num_classes** (*int, optional*) –The number of classes. If None, it will be obtained from the ‘classes’ field in `self.dataset_meta`. Defaults to None.
- **eval_mode** (*str*) – ‘area’ or ‘11points’, ‘area’ means calculating the area under precision-recall curve, ‘11points’ means calculating the average precision of recalls at [0, 0.1, ···, 1]. The PASCAL VOC2007 defaults to use ‘11points’, while PASCAL VOC2012 defaults to use ‘area’. Defaults to ‘area’.
- **use_legacy_coordinate** (*bool*) –Whether to use coordinate system in mmdet v1.x. which means width, height should be calculated as ‘x2 - x1 + 1‘ and ‘y2 - y1 + 1’ respectively. Defaults to False.
- **nproc** (*int*) –Processes used for computing TP and FP. If nproc is less than or equal to 1, multiprocessing will not be used. Defaults to 4.
- **drop_class_ap** (*bool*) –Whether to drop the class without ground truth when calculating the average precision for each class.
- **classwise** (*bool*) –Whether to return the computed results of each class. Defaults to False.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> import numpy as np
>>> from mmeval import VOCMeanAP
>>> num_classes = 4
>>> voc_map = VOCMeanAP(num_classes=4)
>>>
>>> def _gen_bboxes(num_bboxes, img_w=256, img_h=256):
...     # random generate bounding boxes in 'xyxy' formart.
...     x = np.random.rand(num_bboxes, ) * img_w
...     y = np.random.rand(num_bboxes, ) * img_h
...     w = np.random.rand(num_bboxes, ) * (img_w - x)
...     h = np.random.rand(num_bboxes, ) * (img_h - y)
...     return np.stack([x, y, x + w, y + h], axis=1)
>>>
>>> prediction = {
...     'bboxes': _gen_bboxes(10),
...     'scores': np.random.rand(10, ),
...     'labels': np.random.randint(0, num_classes, size=(10, ))
... }
>>> groundtruth = {
...     'bboxes': _gen_bboxes(10),
...     'labels': np.random.randint(0, num_classes, size=(10, )),
```

(下页继续)

(续上页)

```
...     'bboxes_ignore': _gen_bboxes(5),
...     'labels_ignore': np.random.randint(0, num_classes, size=(5, ))
... }
>>> voc_map(predictions=[prediction, ], groundtruths=[groundtruth, ])
{'AP50': ..., 'mAP': ...}
```

add(*predictions: Sequence[Dict]*, *groundtruths: Sequence[Dict]*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence[dict]*) –A sequence of dict. Each dict representing a detection result for an image, with the following keys:
  - bboxes (numpy.ndarray): Shape (N, 4), the predicted bounding bboxes of this image, in 'xyxy' foramrt.
  - scores (numpy.ndarray): Shape (N, 1), the predicted scores of bounding boxes.
  - labels (numpy.ndarray): Shape (N, 1), the predicted labels of bounding boxes.
- **groundtruths** (*Sequence[dict]*) –A sequence of dict. Each dict represents a groundtruths for an image, with the following keys:
  - bboxes (numpy.ndarray): Shape (M, 4), the ground truth bounding bboxes of this image, in 'xyxy' foramrt.
  - labels (numpy.ndarray): Shape (M, 1), the ground truth labels of bounding boxes.
  - bboxes_ignore (numpy.ndarray): Shape (K, 4), the ground truth ignored bounding bboxes of this image, in 'xyxy' foramrt.
  - labels_ignore (numpy.ndarray): Shape (K, 1), the ground truth ignored labels of bounding boxes.

calculate_class_tpfp(*predictions: List[dict]*, *groundtruths: List[dict]*, *class_index: int*, *pool: Optional[multiprocessing.pool.Pool]*) → Tuple

Calculate the tp and fp of the given class index.

参数

- **predictions** (*List[dict]*) –A list of dict. Each dict is the detection result of an image. Same as *VOCMeanAP.add*.
- **groundtruths** (*List[dict]*) –A list of dict. Each dict is the ground truth of an image. Same as *VOCMeanAP.add*.
- **class_index** (*int*) –The class index.
- **pool** (*Optional[Pool]*) –A instance of `multiprocessing.Pool`. If None, do not use multiprocessing.

返回

- tp (numpy.ndarray): Shape (num_ious, num_scales, num_pred), the true positive flag of each predicted bbox for this class.
- fp (numpy.ndarray): Shape (num_ious, num_scales, num_pred), the false positive flag of each predicted bbox for this class.
- num_gts (numpy.ndarray): Shape (num_ious, num_scales), the number of ground truths.

返回类型 tuple (tp, fp, num_gts)

compute_metric(*results: list*) → dict

Compute the VOCMeanAP metric.

参数 **results** (*List[tuple]*) –A list of tuple. Each tuple is the prediction and ground truth of an image. This list has already been synced across all ranks.

返回

The computed metric, with the following keys:

- mAP, the averaged across all IoU thresholds and all class.
- AP{IoU}, the mAP of the specified IoU threshold.
- mAP@{scale_range}, the mAP of the specified scale range.
- classwise, the evaluation results of each class. This would be returned if `self.classwise` is True.

返回类型 dict

get_class_gts(*groundtruths: List[dict]*, *class_index: int*) → Tuple

Get prediciton gt information of a certain class index.

参数

- **groundtruths** (*list[dict]*) –Same as *VOCMeanAP.add*.
- **class_index** (*int*) –Index of a specific class.

返回

- class_gts (List[numpy.ndarray]): The gt bboxes of this class.
- class_ignore_gts (List[numpy.ndarray]): The ignored gt bboxes of this class. This is necessary when counting tp and fp.

返回类型 tuple (class_gts, class_ignore_gts)

get_class_predictions(*predictions: List[dict]*, *class_index: int*) → List

Get prediciton results of a certain class index.

参数

- **predictions** (*list[dict]*) –Same as *VOCMeanAP.add*.
- **class_index** (*int*) –Index of a specific class.

返回 A list of predicted bboxes of this class. Each predicted score of the bbox is concatenated behind the predicted bbox.

返回类型 list[np.ndarray]

property num_classes: int

Returns the number of classes.

The number of classes should be set during initialization, otherwise it will be obtained from the 'classes' field in `self.dataset_meta`.

返回 The number of classes.

返回类型 int

引发 **RuntimeError** –If the num_classes is not set.

15.1.9 OIDMeanAP

class mmeval.metrics.**OIDMeanAP**(*iof_thrs: Union[float, List[float]] = 0.5*, *use_group_of: bool = True*, *get_supercategory: bool = True*, *filter_labels: bool = True*, *class_relation_matrix: Optional[numpy.ndarray] = None*, ***kwargs*)

Open Images Dataset detection evaluation metric.

The Open Images Dataset detection evaluation uses a variant of the standard PASCAL VOC 2010 mean Average Precision (mAP) at IoU > 0.5. There are some key features of Open Images annotations, which are addressed by the new metric.

For more see: https://storage.googleapis.com/openimages/web/evaluation.html

参数

- **iof_thrs** (*float | List[float]*) –IoF thresholds. Defaults to 0.5.
- **use_group_of** (*bool*) –Whether consider group of groud truth bboxes during evaluating. Defaults to True.
- **get_supercategory** (*bool, optional*) –Whether to get parent class of the current class. Defaults to True.
- **filter_labels** (*bool, optional*) –Whether filter unannotated classes. Defaults to True.
- **class_relation_matrix** (*numpy.ndarray, optional*) –The matrix of the corresponding relationship between the parent class and the child class. If None, it will be obtained from the 'relation_matrix' field in `self.dataset_meta`. Defaults to None.
- ****kwargs** –Keyword parameters passed to *VOCMeanAP*.

实际案例

```
>>> import numpy as np
>>> from mmeval import OIDMeanAP
>>> num_classes = 4
>>> # use a fake relation_matrix
>>> relation_matrix = np.eye(num_classes, num_classes)
>>> oid_map = OIDMeanAP(
...     num_classes=4, class_relation_matrix=relation_matrix)
>>>
>>> def _gen_bboxes(num_bboxes, img_w=256, img_h=256):
...     # random generate bounding boxes in 'xyxy' formart.
...     x = np.random.rand(num_bboxes, ) * img_w
...     y = np.random.rand(num_bboxes, ) * img_h
...     w = np.random.rand(num_bboxes, ) * (img_w - x)
...     h = np.random.rand(num_bboxes, ) * (img_h - y)
...     return np.stack([x, y, x + w, y + h], axis=1)
>>>
>>> prediction = {
...     'bboxes': _gen_bboxes(10),
...     'scores': np.random.rand(10, ),
...     'labels': np.random.randint(0, num_classes, size=(10, ))
... }
>>> instances = []
>>> for bbox in _gen_bboxes(20):
...     instances.append({
...         'bbox': bbox.tolist(),
...         'bbox_label': random.randint(0, num_classes - 1),
...         'is_group_of': random.randint(0, 1) == 0,
...     })
>>> groundtruth = {
...     'instances': instances,
...     'image_level_labels': np.random.randint(0, num_classes, size=(10, )),  #␣
↪noqa: E501
... }
>>> oid_map(predictions=[prediction, ], groundtruths=[groundtruth, ])
{'AP@50': ..., 'mAP': ...}
```

add(*predictions: Sequence[dict]*, *groundtruths: Sequence[dict]*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence[dict]*) –A sequence of dict. Each dict representing a detection result for an image, with the following keys:

  - bboxes (numpy.ndarray): Shape (N, 4), the predicted bounding bboxes of this image, in 'xyxy' foramrt.
  - scores (numpy.ndarray): Shape (N, 1), the predicted scores of bounding boxes.
  - labels (numpy.ndarray): Shape (N, 1), the predicted labels of bounding boxes.

- **groundtruths** (*Sequence[dict]*) –A sequence of dict. Each dict representing a groundtruths for an image, with the following keys:
  - instances (List[dict]): Each dict representing a bounding box with the following keys:
    * bbox (list): Containing box location in 'xyxy' foramrt.
    * bbox_label (int): Box label index.
    * is_group_of (bool): Whether the box is group or not.
  - image_level_labels (numpy.ndarray): The image level labels.

calculate_class_tpfp(*predictions: List[dict]*, *groundtruths: List[dict]*, *class_index: int*, *pool: Optional[multiprocessing.pool.Pool]*) → Tuple

Calculate the tp and fp of the given class index.

This is an overridden method of *VOCMeanAP*.

参数

- **predictions** (*List[dict]*) –A list of dict. Each dict is the detection result of an image. Same as *VOCMeanAP.add*.
- **groundtruths** (*List[dict]*) –A list of dict. Each dict is the ground truth of an image. Same as *VOCMeanAP.add*.
- **class_index** (*int*) –The class index.
- **pool** (*Pool, optional*) –An instance of class:*multiprocessing.Pool*. If None, do not use multiprocessing.

返回

- tp (numpy.ndarray): Shape (num_ious, num_scales, num_pred), the true positive flag of each predict bbox.
- fp (numpy.ndarray): Shape (num_ious, num_scales, num_pred), the false positive flag of each predict bbox.
- num_gts (numpy.ndarray): Shape (num_ious, num_scales), the number of ground truths.

返回类型 tuple (tp, fp, num_gts)

property class_relation_matrix: numpy.ndarray

Returns the class relation matrix.

The class relation matrix should be set during initialization, otherwise it will be obtained from the 'relation_matrix' field in `self.dataset_meta`.

返回 The class relation matrix.

返回类型 numpy.ndarray

引发 **RuntimeError** –If the class relation matrix is not set.

get_class_gts(*groundtruths: List[dict]*, *class_index: int*) → Tuple

Get prediciton gt information of a certain class index.

This is an overridden method of *VOCMeanAP*.

参数

- **groundtruths** (*list[dict]*) –Same as `self.add`.
- **class_index** (*int*) –Index of a specific class.

返回 gt bboxes.

返回类型 List[np.ndarray]

15.1.10 F1Score

class mmeval.metrics.**F1Score**(*num_classes: int*, *mode: Union[str, Sequence[str]] = 'micro'*, *cared_classes: Sequence[int] = []*, *ignored_classes: Sequence[int] = []*, ***kwargs*)

Compute F1 scores.

参数

- **num_classes** (*int*) –Number of labels.
- **mode** (*str or list[str]*) –There are 2 options:
  - 'micro': Calculate metrics globally by counting the total true positives, false negatives and false positives.
  - 'macro': Calculate metrics for each label, and find their unweighted mean.

  If mode is a list, then metrics in mode will be calculated separately. Defaults to 'micro'.
- **cared_classes** (*list[int]*) –The indices of the labels participated in the metric computing. If both `cared_classes` and `ignored_classes` are empty, all classes will be taken into account. Defaults to []. Note: `cared_classes` and `ignored_classes` cannot be specified together.
- **ignored_classes** (*list[int]*) –The index set of labels that are ignored when computing metrics. If both `cared_classes` and `ignored_classes` are empty, all classes will be taken into account. Defaults to []. Note: `cared_classes` and `ignored_classes` cannot be specified together.

- ****kwargs** –Keyword arguments passed to `BaseMetric`.

警告： Only non-negative integer labels are involved in computing. All negative ground truth labels will be ignored.

实际案例

```
>>> from mmeval import F1Score
>>> f1 = F1Score(num_classes=5, mode=['macro', 'micro'])
```

Use NumPy implementation:

```
>>> import numpy as np
>>> labels = np.asarray([0, 1, 4])
>>> preds = np.asarray([0, 1, 2])
>>> f1(preds, labels)
{'macro_f1': 0.4,
 'micro_f1': 0.6666666666666666}
```

Use PyTorch implementation:

```
>>> import torch
>>> labels = torch.Tensor([0, 1, 4])
>>> preds = torch.Tensor([0, 1, 2])
>>> f1(preds, labels)
{'macro_f1': 0.4,
 'micro_f1': 0.6666666666666666}
```

Accumulate batch:

```
>>> for i in range(10):
...     labels = torch.randint(0, 4, size=(20, ))
...     predicts = torch.randint(0, 4, size=(20, ))
...     f1.add(predicts, labels)
>>> f1.compute()
```

add(*predictions: Sequence[Union[Sequence[int], numpy.ndarray]]*, *labels: Sequence[Union[Sequence[int], numpy.ndarray]]*) → None

Process one batch of data and predictions.

Calculate the following 2 stuff from the inputs and store them in `self._results`:

- prediction: prediction labels.
- label: ground truth labels.

参数

- **predictions** (*Sequence[Sequence[int] or np.ndarray]*) –A batch of sequences of non-negative integer labels.
- **labels** (*Sequence[Sequence[int] or np.ndarray]*) –A batch of sequences of non-negative integer labels.

compute_metric(*results: Sequence[Tuple[numpy.ndarray, numpy.ndarray]]*) → Dict

Compute the metrics from processed results.

参数 **results** (*list[(ndarray, ndarray)]*) –The processed results of each batch.

返回 The f1 scores. The keys are the names of the metrics, and the values are corresponding results. Possible keys are ‘micro_f1’ and ‘macro_f1’.

返回类型 dict[str, float]

15.1.11 HmeanIoU

class mmeval.metrics.**HmeanIoU**(*match_iou_thr: float = 0.5*, *ignore_precision_thr: float = 0.5*, *pred_score_thrs: Dict = {'start': 0.3, 'step': 0.1, 'stop': 0.9}*, *strategy: str = 'vanilla'*, ***kwargs*)

HmeanIoU metric.

This method computes the hmean iou metric of polygons, and accepts parameters orgnaized as follows:

- batch_pred_polygons (Sequence[Sequence[np.ndarray]]): A batch of prediction polygons, where each element is a sequence of polygons. Each polygon is represented in the form of [x1, y1, x2, y2, ···].
- batch_pred_scores (Sequence): A batch of prediction scores, where each element is a sequence of scores.
- batch_pred_polygons (Sequence[Sequence[np.ndarray]]): A batch of ground truth polygons, where each element is a sequence of polygons. Each polygon is represented in the form of [x1, y1, x2, y2, ···].
- batch_gt_ignore_flags (Sequence): A batch of boolean flags indicating whether to ignore the corresponding ground truth polygon. Each element is a sequence of flags.

The evaluation is done in the following steps:

- Filter the prediction polygon:
  - Score is smaller than minimum prediction score threshold.
  - The proportion of the area that intersects with gt ignored polygon is greater than ignore_precision_thr.
- Computing an M x N IoU matrix, where each element indexing E_mn represents the IoU between the m-th valid GT and n-th valid prediction.
- Based on different prediction score threshold:

  - Obtain the ignored predictions according to prediction score. The filtered predictions will not be involved in the later metric computations.
  - Based on the IoU matrix, get the match metric according to `match_iou_thr`.
  - Based on different *strategy*, accumulate the match number.
- calculate H-mean under different prediction score threshold.

参数

- **match_iou_thr** (*float*) –IoU threshold for a match. Defaults to 0.5.
- **ignore_precision_thr** (*float*) –Precision threshold when prediction and gt ignored polygons are matched. Defaults to 0.5.
- **pred_score_thrs** (*dict*) –Best prediction score threshold searching space. Defaults to dict(start=0.3, stop=0.9, step=0.1).
- **strategy** (*str*) –Polygon matching strategy. Options are 'max_matching' and 'vanilla'. 'max_matching' refers to the optimum strategy that maximizes the number of matches. Vanilla strategy matches gt and pred polygons if both of them are never matched before. It was used in academia. Defaults to 'vanilla'.
- ****kwargs** –Keyword arguments passed to `BaseMetric`.

实际案例

```
>>> from mmeval import HmeanIoU
>>> import numpy as np
>>> hmeaniou = HmeanIoU(pred_score_thrs=dict(start=0.5, stop=0.7, step=0.1))  # noqa
>>> gt_polygons = [[np.array([0, 0, 1, 0, 1, 1, 0, 1])]]
>>> pred_polygons = [[
...     np.array([0, 0, 1, 0, 1, 1, 0, 1]),
...     np.array([0, 0, 1, 0, 1, 1, 0, 1]),
... ]]
>>> pred_scores = [np.array([1, 0.5])]
>>> gt_ignore_flags = [[False]]
>>> hmeaniou(pred_polygons, pred_scores, gt_polygons, gt_ignore_flags)
{
    0.5: {'precision': 0.5, 'recall': 1.0, 'hmean': 0.6666666666666666},  # noqa
    0.6: {'precision': 1.0, 'recall': 1.0, 'hmean': 1.0},
    'best': {'precision': 1.0, 'recall': 1.0, 'hmean': 1.0}
}
```

add(*batch_pred_polygons: Sequence[Sequence[numpy.ndarray]]*, *batch_pred_scores: Sequence*, *batch_gt_polygons: Sequence[Sequence[numpy.ndarray]]*, *batch_gt_ignore_flags: Sequence*) → None

Process one batch of data and predictions.

参数

- **batch_pred_polygons** (*Sequence[Sequence[np.ndarray]]*) –A batch of prediction polygons, where each element is a sequence of polygons. Each polygon is represented in the form of [x1, y1, x2, y2, ···].
- **batch_pred_scores** (*Sequence*) –A batch of prediction scores, where each element is a sequence of scores.
- **batch_pred_polygons** –A batch of ground truth polygons, where each element is a sequence of polygons. Each polygon is represented in the form of [x1, y1, x2, y2, ···].
- **batch_gt_ignore_flags** (*Sequence*) –A batch of boolean flags indicating whether to ignore the corresponding ground truth polygon. Each element is a sequence of flags.

compute_metric(*results: Sequence[Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]]*) → Dict

Compute the metrics from processed results.

参数 **results** (*list[(np.ndarray, np.ndarray, np.ndnarray, np.ndarray]*) –The processed results of each batch.

返回 Nested dicts as results. The inner dict contains the "precision", "recall", and "hmean" scores under different prediction score thresholds, which can be indexed by the corresponding threshold value from the outer dict. The outer dict also contains the "best" key, which is the result of the best hmean score.

返回类型 dict[float or "best", dict[str, float]]

15.1.12 EndPointError

class mmeval.metrics.**EndPointError**(***kwargs*)

EndPointError evaluation metric.

EndPointError is a widely used evaluation metric for optical flow estimation.

This metric supports 3 kinds of inputs, i.e. *numpy.ndarray* and *torch.Tensor*, *oneflow.Tensor*, and the implementation for the calculation depends on the inputs type.

参数 ****kwargs** –Keyword arguments passed to `BaseMetric`.

实际案例

```
>>> from mmeval import EndPointError
>>> epe = EndPointError()
```

Use NumPy implementation:

```
>>> import numpy as np
>>> predictions = np.array(
...     [[[10., 5.], [0.1, 3.]],
...     [[3., 15.2], [2.4, 4.5]]])
>>> labels = np.array(
...     [[[10.1, 4.8], [10, 3.]],
...     [[6., 10.2], [2.0, 4.1]]])
>>> valid_masks = np.array([[1., 1.], [1., 0.]])
>>> epe(predictions, labels, valid_masks)
{'EPE': 5.318186230865093}
```

Use PyTorch implementation:

```
>>> import torch
>>> predictions = torch.Tensor(
...     [[[10., 5.], [0.1, 3.]],
...     [[3., 15.2], [2.4, 4.5]]])
>>> labels = torch.Tensor(
...     [[[10.1, 4.8], [10, 3.]],
...     [[6., 10.2], [2.0, 4.1]]])
>>> valid_masks = torch.Tensor([[1., 1.], [1., 0.]])
>>> epe(predictions, labels, valid_masks)
{'EPE': 5.3181863}
```

Accumulate batch:

```
>>> for i in range(10):
...     predictions = torch.randn(10, 10, 2)
...     labels = torch.randn(10, 10, 2)
...     epe.add(predictions, labels)
>>> epe.compute()
```

add(*predictions: Sequence*, *labels: Sequence*, *valid_masks: Optional[Sequence] = None*) → None

Process one batch of predictions and labels.

参数

- **predictions** (*Sequence*) –Predicted sequence of flow map with shape (H, W, 2).
- **labels** (*Sequence*) –The ground truth sequence of flow map with shape (H, W, 2).

- **valid_masks** (*Sequence*) –The Sequence of valid mask for labels with shape (H, W). If it is None, this function will automatically generate a map filled with 1. Defaults to None.

compute_metric(*results: List[Tuple[numpy.ndarray, int]]*) → dict

Compute the EndPointError metric.

This method would be invoked in *BaseMetric.compute* after distributed synchronization.

参数 **results** (*List[np.ndarray]*) –This list has already been synced across all ranks. This is a list of *np.ndarray*, which is the end point error map between the prediction and the label.

返回

The computed metric, with following key:

- EPE, the mean end point error of all pairs.

返回类型 Dict

end_point_error_map

Calculate end point error map.

参数

- **prediction** (*np.ndarray*) –Prediction with shape (H, W, 2).
- **label** (*np.ndarray*) –Ground truth with shape (H, W, 2).
- **valid_mask** (*np.ndarray, optional*) –Valid mask with shape (H, W).

返回 The mean of end point error and the numbers of valid labels.

返回类型 Tuple

15.1.13 PCKAccuracy

class mmeval.metrics.**PCKAccuracy**(*thr: float = 0.2*, *norm_item: Union[str, Sequence[str]] = 'bbox'*, ***kwargs*)

PCK accuracy evaluation metric, which is widely used in pose estimation.

Calculate the pose accuracy of Percentage of Correct Keypoints (PCK) for each individual keypoint and the averaged accuracy across all keypoints. PCK metric measures the accuracy of the localization of the body joints. The distances between predicted positions and the ground-truth ones are typically normalized by the person bounding box size. The threshold (thr) of the normalized distance is commonly set as 0.05, 0.1 or 0.2 etc.

注解:

- length of dataset: N

- num_keypoints: K
- number of keypoint dimensions: D (typically D = 2)

参数

- **thr** (*float*) –Threshold of PCK calculation. Defaults to 0.2.
- **norm_item** (*str | Sequence[str]*) –The item used for normalization. Valid items include 'bbox', 'head', 'torso', which correspond to 'PCK', 'PCKh' and 'tPCK' respectively. Defaults to 'bbox'.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval import PCKAccuracy
>>> import numpy as np
>>> num_keypoints = 15
>>> keypoints = np.random.random((1, num_keypoints, 2)) * 10
>>> predictions = [{'coords': keypoints}]
>>> keypoints_visible = np.ones((1, num_keypoints)).astype(bool)
>>> bbox_size = np.random.random((1, 2)) * 10
>>> groundtruths = [{
...     'coords': keypoints,
...     'mask': keypoints_visible,
...     'bbox_size': bbox_size,
... }]
>>> pck_metric = PCKAccuracy(thr=0.5, norm_item='bbox')
>>> pck_metric(predictions, groundtruths)
OrderedDict([('PCK@0.5', 1.0)])
```

add(*predictions: List[Dict]*, *groundtruths: List[Dict]*) → None

Process one batch of predictions and groundtruths and add the intermediate results to *self._results*.

参数

- **predictions** (*Sequence[dict]*) –Predictions from the model. Each prediction dict has the following keys:
  - coords (np.ndarray, [1, K, D]): predicted keypoints coordinates
- **groundtruths** (*Sequence[dict]*) –The ground truth labels. Each groundtruth dict has the following keys:
  - coords (np.ndarray, [1, K, D]): ground truth keypoints coordinates
  - mask (np.ndarray, [1, K]): ground truth keypoints_visible

- bbox_size (np.ndarray, optional, [1, 2]): ground truth bbox size
- head_size (np.ndarray, optional, [1, 2]): ground truth head size
- torso_size (np.ndarray, optional, [1, 2]): ground truth torso size

compute_metric(*results: list*) → Dict[str, float]

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are the corresponding results.

返回类型 Dict[str, float]

15.1.14 MpiiPCKAccuracy

class mmeval.metrics.**MpiiPCKAccuracy**(*thr: float = 0.5*, *norm_item: Union[str, Sequence[str]] = 'head'*, ***kwargs*)

PCKh accuracy evaluation metric for MPII dataset.

Calculate the pose accuracy of Percentage of Correct Keypoints (PCK) for each individual keypoint and the averaged accuracy across all keypoints. PCK metric measures accuracy of the localization of the body joints. The distances between predicted positions and the ground-truth ones are typically normalized by the person bounding box size. The threshold (thr) of the normalized distance is commonly set as 0.05, 0.1 or 0.2 etc.

注解:

- length of dataset: N
- num_keypoints: K
- number of keypoint dimensions: D (typically D = 2)

参数

- **thr** (*float*) –Threshold of PCK calculation. Defaults to 0.5.
- **norm_item** (*str | Sequence[str]*) –The item used for normalization. Valid items include 'bbox', 'head', 'torso', which correspond to 'PCK', 'PCKh' and 'tPCK' respectively. Defaults to `'head'`.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import MpiiPCKAccuracy
>>> import numpy as np
>>> num_keypoints = 16
>>> keypoints = np.random.random((1, num_keypoints, 2)) * 10
>>> predictions = [{'coords': keypoints}]
>>> keypoints_visible = np.ones((1, num_keypoints)).astype(bool)
>>> head_size = np.random.random((1, 2)) * 10
>>> groundtruths = [{
...     'coords': keypoints + 1.0,
...     'mask': keypoints_visible,
...     'head_size': head_size,
... }]
>>> mpii_pckh_metric = MpiiPCKAccuracy(thr=0.3, norm_item='head')
>>> mpii_pckh_metric(predictions, groundtruths)
OrderedDict([('Head', 100.0), ('Shoulder', 100.0), ('Elbow', 100.0),
('Wrist', 100.0), ('Hip', 100.0), ('Knee', 100.0), ('Ankle', 100.0),
('PCKh', 100.0), ('PCKh@0.1', 100.0)])
```

compute_metric (*results: list*) → Dict[str, float]

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.15 JhmdbPCKAccuracy

class mmeval.metrics.**JhmdbPCKAccuracy** (*thr: float = 0.5*, *norm_item: Union[str, Sequence[str]] = 'bbox'*, ***kwargs*)

PCK accuracy evaluation metric for Jhmdb dataset.

Calculate the pose accuracy of Percentage of Correct Keypoints (PCK) for each individual keypoint and the averaged accuracy across all keypoints. PCK metric measures accuracy of the localization of the body joints. The distances between predicted positions and the ground-truth ones are typically normalized by the person bounding box size. The threshold (thr) of the normalized distance is commonly set as 0.05, 0.1 or 0.2 etc.

注解:

- length of dataset: N
- num_keypoints: K

- number of keypoint dimensions: D (typically D = 2)

参数

- **thr** (*float*) –Threshold of PCK calculation. Defaults to 0.5.
- **norm_item** (*str | Sequence[str]*) –The item used for normalization. Valid items include 'bbox', 'head', 'torso', which correspond to 'PCK', 'PCKh' and 'tPCK' respectively. Defaults to 'bbox'.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval import JhmdbPCKAccuracy
>>> import numpy as np
>>> num_keypoints = 15
>>> keypoints = np.random.random((1, num_keypoints, 2)) * 10
>>> predictions = [{'coords': keypoints}]
>>> keypoints_visible = np.ones((1, num_keypoints)).astype(bool)
>>> torso_size = np.random.random((1, 2)) * 10
>>> groundtruths = [{
...     'coords': keypoints,
...     'mask': keypoints_visible,
...     'torso_size': torso_size,
... }]
>>> jhmdb_pckh_metric = JhmdbPCKAccuracy(thr=0.2, norm_item='torso')
>>> jhmdb_pckh_metric(predictions, groundtruths)
OrderedDict([('Head tPCK', 1.0), ('Sho tPCK', 1.0), ('Elb tPCK', 1.0),
('Wri tPCK', 1.0), ('Hip tPCK', 1.0), ('Knee tPCK', 1.0),
('Ank tPCK', 1.0), ('Mean tPCK', 1.0)])
```

compute_metric(*results: list*) → Dict[str, float]

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.16 AVAMeanAP

class mmeval.metrics.**AVAMeanAP**(*ann_file: str*, *label_file: str*, *exclude_file: Optional[str] = None*, *num_classes: int = 81*, *custom_classes: Optional[List[int]] = None*, *verbose: bool = True*, ***kwargs*)

AVA evaluation metric.

AVA(Atomic Visual Action): https://research.google.com/ava.

This metric computes mAP using the ava evaluation toolkit provided by the author.

参数

- **ann_file** (*str*) –The annotation file path.
- **label_file** (*str*) –The label file path.
- **exclude_file** (*str, optional*) –The excluded timestamp file path. Defaults to None.
- **num_classes** (*int*) –Number of classes. Defaults to 81.
- **custom_classes** (*list(int), optional*) –A subset of class ids from origin dataset. Defaults to None.
- **verbose** (*bool*) –Whether to print messages in the evaluation process. Defaults to True.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import AVAMeanAP
>>> import numpy as np
>>>
>>> ann_file = 'tests/test_metrics/ava_detection_gt.csv'
>>> label_file = 'tests/test_metrics/ava_action_list.txt'
>>> num_classes = 4
>>> ava_metric = AVAMeanAP(ann_file=ann_file, label_file=label_file,
>>>                        num_classes=4)
>>>
>>> predictions = [
>>> {
>>>     'video_id': '3reY9zJKhqN',
>>>     'timestamp': 1774,
>>>     'outputs': [
>>>         np.array([[0.362, 0.156, 0.969, 0.666, 0.106],
>>>                   [0.442, 0.083, 0.721, 0.947, 0.162]]),
>>>         np.array([[0.288, 0.365, 0.766, 0.551, 0.706],
```

(下页继续)

(续上页)

```
>>>                     [0.178, 0.296, 0.707, 0.995, 0.223]]),
>>>         np.array([[0.417, 0.167, 0.843, 0.939, 0.015],
>>>                     [0.35, 0.421, 0.57, 0.689, 0.427]]) ]
>>> },
>>> {
>>>     'video_id': 'HmR8SmNIoxu',
>>>     'timestamp': 1384,
>>>     'outputs': [
>>>         np.array([[0.256, 0.338, 0.726, 0.799, 0.563],
>>>                     [0.071, 0.256, 0.64, 0.75, 0.297]]),
>>>         np.array([[0.326, 0.036, 0.513, 0.991, 0.405],
>>>                     [0.351, 0.035, 0.729, 0.936, 0.945]]),
>>>         np.array([[0.051, 0.005, 0.975, 0.942, 0.424],
>>>                     [0.347, 0.05, 0.97, 0.944, 0.396]])]
>>> },
>>> {
>>>     'video_id': '5HNXoce1raG',
>>>     'timestamp': 1097,
>>>     'outputs': [
>>>         np.array([[0.39, 0.087, 0.833, 0.616, 0.447],
>>>                     [0.461, 0.212, 0.627, 0.527, 0.036]]),
>>>         np.array([[0.022, 0.394, 0.93, 0.527, 0.109],
>>>                     [0.208, 0.462, 0.874, 0.948, 0.954]]),
>>>         np.array([[0.206, 0.456, 0.564, 0.725, 0.685],
>>>                     [0.106, 0.445, 0.782, 0.673, 0.367]])]}
>>> ]
>>> ava_metric(predictions)
{'mAP@0.5IOU': 0.027777778}
```

add (*predictions: Sequence[dict]*) → None

Add detection results to the results list.

参数 **predictions** (*Sequence[dict]*) –A list of prediction dict which contains the following keys:

- *video_id*: The id of the video, e.g., *3reY9zJKhqN*.
- *timestamp*: The timestamp of the video e.g., *1774*.
- *outputs*: A list bbox results of each class with the format of [x1, y1, x2, y2, score].

ava_eval (*result_file: str*) → dict

Perform ava evaluation.

参数 **result_file** (*str*) –The dumped results file path.

返回 The evaluation results.

返回类型 dict

compute_metric(*results: list*) → dict

Compute the AVA MeanAP.

参数 **results** (*list*) –A list of detection results.

返回 The computed ava metric.

返回类型 dict

read_csv(*csv_file: str*, *class_whitelist: Optional[set] = None*) → tuple

Loads boxes and class labels from a CSV file in the AVA format.

CSV file format described at https://research.google.com/ava/download.html.

参数

- **csv_file** (*str*) –A csv file path.
- **class_whitelist** (*set, optional*) –If provided, boxes corresponding to (integer) class labels not in this set are skipped.

返回

- boxes (dict): A dictionary mapping each unique image key (string) to a list of boxes, given as coordinates [y1, x1, y2, x2].
- labels (dict): A dictionary mapping each unique image key (string) to a list of integer class labels, matching the corresponding box in *boxes*.
- scores (dict): A dictionary mapping each unique image key (string) to a list of score values labels, matching the corresponding label in *labels*. If scores are not provided in the csv, then they will default to 1.0.

返回类型 tuple (boxes, labels, scores)

read_exclusions(*exclude_file: str*) → set

Reads a CSV file of excluded timestamps.

参数 **exclude_file** (*str*) –The path of exclude file.

返回 A set of strings containing excluded image keys, e.g. "aaaaaaaaaaa,0904" or an empty set if exclusions file is None.

返回类型 excluded (set)

read_label(*label_file: str*) → tuple

Reads a label mapping file.

参数 **label_file** (*str*) –The path of label file.

返回

- labelmap (list): The label map in the form used by the object_detection_evaluation module - a list of { “id” : integer, “name” : classname } dicts.
- class_ids (set): A set containing all of the valid class id integers.

返回类型 tuple (labelmap, class_ids)

results2csv(*results: List[dict]*, *out_file: str*) → None

Dump the results to a csv file.

参数

- **results** (*list[dict]*) –A list of detection results.
- **out_file** (*str*) –The output csv file path.

15.1.17 StructuralSimilarity

class mmeval.metrics.**StructuralSimilarity**(*crop_border: int = 0*, *input_order: str = 'CHW'*, *convert_to: Optional[str] = None*, *channel_order: str = 'rgb'*, ***kwargs*)

Calculate StructuralSimilarity (structural similarity).

Ref: Image quality assessment: From error visibility to structural similarity

The results are the same as that of the official released MATLAB code in https://ece.uwaterloo.ca/~z70wang/research/ssim/.

For three-channel images, StructuralSimilarity is calculated for each channel and then averaged.

参数

- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the PeakSignalNoiseRatio calculation. Defaults to 0.
- **input_order** (*str*) –Whether the input order is ‘HWC’ or ‘CHW’. Defaults to ‘HWC’.
- **convert_to** (*str, optional*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for ‘Y’, the images are assumed to be in BGR order. Options are ‘Y’ and None. Defaults to None.
- **channel_order** (*str*) –The channel order of image. Choices are ‘rgb’ and ‘bgr’. Defaults to ‘rgb’.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import StructuralSimilarity as SSIM
>>> import numpy as np
>>>
>>> ssim = SSIM(input_order='CHW', convert_to='Y', channel_order='rgb')
>>> gts = np.random.randint(0, 255, size=(3, 32, 32))
>>> preds = np.random.randint(0, 255, size=(3, 32, 32))
>>> ssim(preds, gts)
{'ssim': ...}
```

Calculate StructuralSimilarity between 2 single channel images:

```
>>> img1 = np.ones((32, 32)) * 2
>>> img2 = np.ones((32, 32))
>>> SSIM.compute_ssim(img1, img2)
0.913062377743969
```

add(*predictions: Sequence[numpy.ndarray]*, *groundtruths: Sequence[numpy.ndarray]*, *channel_order: Optional[str] = None*) → None

Add the StructuralSimilarity score of the batch to `self._results`.

For three-channel images, StructuralSimilarity is calculated for each channel and then averaged.

参数

- **predictions** (*Sequence[np.ndarray]*) –Predictions of the model.
- **groundtruths** (*Sequence[np.ndarray]*) –The ground truth images.
- **channel_order** (*Optional[str]*) –The channel order of the input samples. If not passed, will set as `self.channel_order`. Defaults to None.

compute_metric(*results: List[numpy.float64]*) → Dict[str, float]

Compute the StructuralSimilarity metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[np.float64]*) –A list that consisting the StructuralSimilarity scores. This list has already been synced across all ranks.

返回 The computed StructuralSimilarity metric.

返回类型 Dict[str, float]

static compute_ssim(*img1: numpy.ndarray*, *img2: numpy.ndarray*) → numpy.float64

Calculate StructuralSimilarity (structural similarity) between two single channel image.

Ref: Image quality assessment: From error visibility to structural similarity

The results are the same as that of the official released MATLAB code in https://ece.uwaterloo.ca/~z70wang/research/ssim/.

参数

- **img1** (*np.ndarray*) –Single channels Images with range [0, 255].
- **img2** (*np.ndarray*) –Single channels Images with range [0, 255].

返回 StructuralSimilarity result.

返回类型 np.float64

15.1.18 SignalNoiseRatio

class mmeval.metrics.**SignalNoiseRatio**(*crop_border: int = 0*, *input_order: str = 'CHW'*, *convert_to: Optional[str] = None*, *channel_order: str = 'rgb'*, ***kwargs*)

Signal-to-Noise Ratio.

Ref: https://en.wikipedia.org/wiki/Signal-to-noise_ratio

参数

- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the PeakSignalNoiseRatio calculation. Defaults to 0.
- **input_order** (*str*) –Whether the input order is ‘HWC’ or ‘CHW’. Defaults to ‘HWC’.
- **convert_to** (*str, optional*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for ‘Y’, the images are assumed to be in BGR order. Options are ‘Y’ and None. Defaults to None.
- **channel_order** (*str*) –The channel order of image. Choices are ‘rgb’ and ‘bgr’. Defaults to ‘rgb’.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval import SignalNoiseRatio
>>> import numpy as np
>>>
>>> snr = SignalNoiseRatio(crop_border=1, input_order='CHW',
...                        convert_to='Y', channel_order='rgb')
>>> gts = np.random.randint(0, 255, size=(3, 32, 32))
>>> preds = np.random.randint(0, 255, size=(3, 32, 32))
>>> snr(preds, gts)
{'snr': ...}
```

Calculate SignalNoiseRatio between 2 images:

```
>>> gts = np.ones((3, 32, 32)) * 2
>>> preds = np.ones((3, 32, 32))
>>> SignalNoiseRatio.compute_snr(preds, gts)
6.020599913279624
```

add(*predictions: Sequence[numpy.ndarray]*, *groundtruths: Sequence[numpy.ndarray]*, *channel_order: Optional[str] = None*) → None

Add SignalNoiseRatio score of batch to `self._results`

参数

- **predictions** (*Sequence[np.ndarray]*) –Predictions of the model.
- **groundtruths** (*Sequence[np.ndarray]*) –The ground truth images.
- **channel_order** (*Optional[str]*) –The channel order of the input samples. If not passed, will set as `self.channel_order`. Defaults to None.

compute_metric(*results: List[numpy.float64]*) → Dict[str, float]

Compute the SignalNoiseRatio metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[np.float64]*) –A list that consisting the SignalNoiseRatio score. This list has already been synced across all ranks.

返回 The computed SignalNoiseRatio metric.

返回类型 Dict[str, float]

static compute_snr(*prediction: numpy.ndarray*, *groundtruth: numpy.ndarray*) → numpy.float64

Calculate SignalNoiseRatio (Signal-to-Noise Ratio).

Ref: https://en.wikipedia.org/wiki/Signal-to-noise_ratio

参数

- **prediction** (*np.ndarray*) –Images with range [0, 255].
- **groundtruth** (*np.ndarray*) –Images with range [0, 255].

返回 SignalNoiseRatio result.

返回类型 np.float64

15.1.19 PeakSignalNoiseRatio

class mmeval.metrics.**PeakSignalNoiseRatio**(*crop_border: int = 0*, *input_order: str = 'CHW'*, *convert_to: Optional[str] = None*, *channel_order: str = 'rgb'*, ***kwargs*)

Peak Signal-to-Noise Ratio.

Ref: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

参数

- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the PeakSignalNoiseRatio calculation. Defaults to 0.
- **input_order** (*str*) –Whether the input order is 'HWC' or 'CHW'. Defaults to 'CHW'.
- **convert_to** (*str*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for 'Y', the images are assumed to be in BGR order. Options are 'Y' and None. Defaults to None.
- **channel_order** (*str*) –The channel order of image. Defaults to 'rgb'.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import PeakSignalNoiseRatio as PSNR
>>> import numpy as np
>>>
>>> psnr = PSNR(input_order='CHW', convert_to='Y', channel_order='rgb')
>>> gts = np.random.randint(0, 255, size=(3, 32, 32))
>>> preds = np.random.randint(0, 255, size=(3, 32, 32))
>>> psnr(preds, gts)
{'psnr': ...}
```

Calculate PeakSignalNoiseRatio between 2 single channel images:

```
>>> img1 = np.ones((32, 32))
>>> img2 = np.ones((32, 32))
>>> PSNR.compute_psnr(img1, img2)
49.45272242415597
```

add(*predictions: Sequence[numpy.ndarray]*, *groundtruths: Sequence[numpy.ndarray]*, *channel_order: Optional[str] = None*) → None

Add PeakSignalNoiseRatio score of batch to `self._results`

参数

- **predictions** (*Sequence[np.ndarray]*) –Predictions of the model.
- **groundtruths** (*Sequence[np.ndarray]*) –The ground truth images.
- **channel_order** (*Optional[str]*) –The channel order of the input samples. If not passed, will set as `self.channel_order`. Defaults to None.

compute_metric (*results: List[numpy.float64]*) → Dict[str, float]

Compute the PeakSignalNoiseRatio metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[np.float64]*) –A list that consisting the PeakSignalNoiseRatio score. This list has already been synced across all ranks.

返回 The computed PeakSignalNoiseRatio metric.

返回类型 Dict[str, float]

static compute_psnr (*prediction: numpy.ndarray*, *groundtruth: numpy.ndarray*) → numpy.float64

Calculate PeakSignalNoiseRatio (Peak Signal-to-Noise Ratio).

Ref: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

参数

- **prediction** (*np.ndarray*) –Images with range [0, 255].
- **groundtruth** (*np.ndarray*) –Images with range [0, 255].

返回 PeakSignalNoiseRatio result.

返回类型 np.float64

15.1.20 MeanAbsoluteError

class mmeval.metrics.**MeanAbsoluteError** (***kwargs*)

Mean Absolute Error metric for image.

Formula: mean(abs(a-b)).

参数 ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import MeanAbsoluteError as MAE
>>> import numpy as np
>>>
>>> mae = MAE()
>>> gts = np.random.randint(0, 255, size=(3, 32, 32))
```

(下页继续)

(续上页)

```
>>> preds = np.random.randint(0, 255, size=(3, 32, 32))
>>> mae(preds, gts)
{'mae': ...}
```

Calculate MeanAbsoluteError between 2 images with mask:

```
>>> img1 = np.ones((32, 32, 3))
>>> img2 = np.ones((32, 32, 3)) * 2
>>> mask = np.ones((32, 32, 3)) * 2
>>> mask[:16] *= 0
>>> MAE.compute_mae(img1, img2, mask)
0.003921568627
```

add (*predictions: Sequence[numpy.ndarray]*, *groundtruths: Sequence[numpy.ndarray]*, *masks: Optional[Sequence[numpy.ndarray]] = None*) → None

Add MeanAbsoluteError score of batch to `self._results`

参数

- **predictions** (*Sequence[np.ndarray]*) –Predictions of the model.
- **groundtruths** (*Sequence[np.ndarray]*) –The ground truth images.
- **masks** (*Sequence[np.ndarray], optional*) –Mask images. Defaults to None.

static compute_mae (*prediction: numpy.ndarray*, *groundtruth: numpy.ndarray*, *mask: Optional[numpy.ndarray] = None*) → numpy.float32

Calculate MeanAbsoluteError (Mean Absolute Error).

参数

- **prediction** (*np.ndarray*) –Images with range [0, 255].
- **groundtruth** (*np.ndarray*) –Images with range [0, 255].
- **mask** (*np.ndarray, optional*) –Mask of evaluation.

返回 MeanAbsoluteError result.

返回类型 np.float32

compute_metric (*results: List[numpy.float32]*) → Dict[str, float]

Compute the MeanAbsoluteError metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[np.float32]*) –A list that consisting the MeanAbsoluteError score. This list has already been synced across all ranks.

返回 The computed MeanAbsoluteError metric.

返回类型 Dict[str, float]

15.1.21 MeanSquaredError

class mmeval.metrics.**MeanSquaredError**(***kwargs*)

Mean Squared Error metric for image.

Formula: mean((a-b)^2).

参数 ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import MeanSquaredError as MSE
>>> import numpy as np
>>>
>>> mse = MSE()
>>> gts = np.random.randint(0, 255, size=(3, 32, 32))
>>> preds = np.random.randint(0, 255, size=(3, 32, 32))
>>> mse(preds, gts)
{'mse': ...}
```

Calculate MeanSquaredError between 2 images with mask:

```
>>> img1 = np.ones((32, 32, 3))
>>> img2 = np.ones((32, 32, 3)) * 2
>>> mask = np.ones((32, 32, 3)) * 2
>>> mask[:16] *= 0
>>> MSE.compute_mse(img1, img2, mask)
0.000015378700496
```

add(*predictions: Sequence[numpy.ndarray]*, *groundtruths: Sequence[numpy.ndarray]*, *masks: Optional[Sequence[numpy.ndarray]] = None*) → None

Add MeanSquaredError score of batch to `self._results`

参数

- **predictions** (*Sequence[np.ndarray]*) –Predictions of the model.
- **groundtruths** (*Sequence[np.ndarray]*) –The ground truth images.
- **masks** (*Sequence[np.ndarray], optional*) –Mask images.

compute_metric(*results: List[numpy.float32]*) → Dict[str, float]

Compute the MeanSquaredError metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[np.float32]*) –A list that consisting the MeanSquaredError score. This list has already been synced across all ranks.

返回 The computed MeanSquaredError metric.

返回类型 Dict[str, float]

static compute_mse (*prediction: numpy.ndarray*, *groundtruth: numpy.ndarray*, *mask: Optional[numpy.ndarray] = None*) → numpy.float32

Calculate MeanSquaredError (Mean Squared Error).

参数

- **prediction** (*np.ndarray*) –Images with range [0, 255].
- **groundtruth** (*np.ndarray*) –Images with range [0, 255].
- **mask** (*np.ndarray, optional*) –Mask of evaluation.

返回 MeanSquaredError result.

返回类型 np.float32

15.1.22 BLEU

class mmeval.metrics.**BLEU** (*n_gram: int = 4*, *smooth: bool = False*, *ngram_weights: Optional[Sequence[float]] = None*, *tokenizer_fn: Optional[Union[Callable, str]] = None*, ***kwargs*)

Bilingual Evaluation Understudy metric.

This metric proposed in BLEU: a Method for Automatic Evaluation of Machine Translation is a tool for evaluating the quality of machine translation. The closer the translation is to human translation, the higher the score will be.

参数

- **n_gram** (*int*) –The maximum number of words contained in a phrase when calculating word fragments. Defaults to 4.
- **smooth** (*bool*) –Whether or not to apply to smooth. Defaults to False.
- **ngram_weights** (*Sequence[float], optional*) –Weights used for unigrams, bigrams, etc. to calculate BLEU score. If not provided, uniform weights are used. Defaults to None.
- **tokenizer_fn** (*Union[Callable, str, None]*) –A user's own tokenizer function. Defaults to None. New in version 0.3.0.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval import BLEU
>>> predictions = ['the cat is on the mat', 'There is a big tree near the park␣
↪here']  # noqa: E501
>>> references = [['a cat is on the mat'], ['A big tree is growing near the park␣
↪here']]  # noqa: E501
>>> bleu = BLEU()
>>> bleu_results = bleu(predictions, references)
{'bleu': 0.5226045319355426}
```

```
>>> # Calculate BLEU with smooth:
>>> from mmeval import BLEU
>>> predictions = ['the cat is on the mat', 'There is a big tree near the park␣
↪here']  # noqa: E501
>>> references = [['a cat is on the mat'], ['A big tree is growing near the park␣
↪here']]  # noqa: E501
>>> bleu = BLEU(smooth = True)
>>> bleu_results = bleu(predictions, references)
{'bleu': 0.566315716093867}
```

add (*predictions: Sequence[str]*, *references: Sequence[Sequence[str]]*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence[str]*) –An iterable of predicted sentences.
- **references** (*Sequence[Sequence[str]]*) –An iterable of referenced sentences.

compute_metric (*results: List[Tuple[int, int, numpy.ndarray, numpy.ndarray]]*) → dict

Compute the bleu metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[Tuple[int, int, np.ndarray, np.ndarray]]*) –A list that consisting the tuple of correct numbers. Tuple contains pred_len, references_len, precision_matches, precision_total. This list has already been synced across all ranks.

返回 The computed bleu score.

返回类型 Dict[str, float]

15.1.23 SumAbsoluteDifferences

class mmeval.metrics.**SumAbsoluteDifferences**(*norm_const: int = 1000*, ***kwargs*)

Sum of Absolute Differences metric for image.

This metric computes per-pixel absolute difference and sum across all pixels. i.e. sum(abs(a-b)) / norm_const

参数

- **norm_const** (*int*) –Divide the result to reduce its magnitude. Default to 1000.
- ****kwargs** –Keyword parameters passed to BaseMetric.

注解： The current implementation assumes the image a numpy array with pixel values ranging from 0 to 255.

实际案例

```
>>> from mmeval import SumAbsoluteDifferences as SAD
>>> import numpy as np
>>>
>>> sad = SAD()
>>> prediction = np.zeros((32, 32), dtype=np.uint8)
>>> groundtruth = np.ones((32, 32), dtype=np.uint8) * 255
>>> sad(prediction, groundtruth)
{'sad': ...}
```

add(*predictions: Sequence[numpy.ndarray]*, *groundtruths: Sequence[numpy.ndarray]*) → None

Add SumAbsoluteDifferences score of batch to self._results

参数

- **predictions** (*Sequence[np.ndarray]*) –Sequence of predicted image.
- **groundtruths** (*Sequence[np.ndarray]*) –Sequence of groundtruth image.

compute_metric(*results: List*) → Dict[str, float]

Compute the SumAbsoluteDifferences metric.

参数 **results** (*List*) –A list that consisting the SumAbsoluteDifferences score. This list has already been synced across all ranks.

返回 The computed SumAbsoluteDifferences metric. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.24 GradientError

class mmeval.metrics.**GradientError**(*sigma: float = 1.4*, *norm_const: int = 1000*, ***kwargs*)

Gradient error for evaluating alpha matte prediction.

参数

- **sigma** (*float*) –Standard deviation of the gaussian kernel. Defaults to 1.4 .
- **norm_const** (*int*) –Divide the result to reduce its magnitude. Defaults to 1000.
- ****kwargs** –Keyword parameters passed to BaseMetric.

注解： The current implementation assumes the image / alpha / trimap a numpy array with pixel values ranging from 0 to 255.

The pred_alpha should be masked by trimap before passing into this metric.

The trimap is the most commonly used prior knowledge. As the name implies, trimap is a ternary graph and each pixel takes one of {0, 128, 255}, representing the foreground, the unknown and the background respectively.

实际案例

```
>>> from mmeval import GradientError
>>> import numpy as np
>>>
>>> gradient_error = GradientError()
>>> np.random.seed(0)
>>> pred_alpha = np.random.randn(32, 32).astype('uint8')
>>> gt_alpha = np.ones((32, 32), dtype=np.uint8) * 255
>>> trimap = np.zeros((32, 32), dtype=np.uint8)
>>> trimap[:16, :16] = 128
>>> trimap[16:, 16:] = 255
>>> gradient_error(pred_alpha, gt_alpha, trimap)
{'gradient_error': ...}
```

add(*pred_alphas: Sequence[numpy.ndarray]*, *gt_alphas: Sequence[numpy.ndarray]*, *trimaps: Sequence[numpy.ndarray]*) → None

Add GradientError score of batch to self._results

参数

- **pred_alphas** (*Sequence[np.ndarray]*) –Predict the probability that pixels belong to the foreground.

- **gt_alphas** (*Sequence[np.ndarray]*) –Probability that the actual pixel belongs to the foreground.
- **trimaps** (*Sequence[np.ndarray]*) –Broadly speaking, the trimap consists of foreground and unknown region.

compute_metric(*results: List*) → Dict[str, float]

Compute the GradientError metric.

参数 **results** (*List*) –A list that consisting the GradientError score. This list has already been synced across all ranks.

返回 The computed GradientError metric. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.25 MattingMeanSquaredError

class mmeval.metrics.**MattingMeanSquaredError**(***kwargs*)

Mean Squared Error metric for image matting.

This metric computes the per-pixel squared error average across all pixels. i.e. mean((a-b)^2)

参数 ****kwargs** –Keyword parameters passed to `BaseMetric`.

注解: The current implementation assumes the image / alpha / trimap a numpy array with pixel values ranging from 0 to 255.

The pred_alpha should be masked by trimap before passing into this metric.

The trimap is the most commonly used prior knowledge. As the name implies, trimap is a ternary graph and each pixel takes one of {0, 128, 255}, representing the foreground, the unknown and the background respectively.

实际案例

```
>>> from mmeval import MattingMeanSquaredError as MattingMSE
>>> import numpy as np
>>>
>>> matting_mse = MattingMSE()
>>> pred_alpha = np.zeros((32, 32), dtype=np.uint8)
>>> gt_alpha = np.ones((32, 32), dtype=np.uint8) * 255
>>> trimap = np.zeros((32, 32), dtype=np.uint8)
>>> trimap[:16, :16] = 128
>>> trimap[16:, 16:] = 255
```

(下页继续)

(续上页)

```
>>> matting_mse(pred_alpha, gt_alpha, trimap)
{'matting_mse': ...}
```

add(*pred_alphas: Sequence[numpy.ndarray]*, *gt_alphas: Sequence[numpy.ndarray]*, *trimaps: Sequence[numpy.ndarray]*) → None

Add MattingMeanSquaredError score of batch to `self._results`

参数

- **pred_alphas** (*Sequence[np.ndarray]*) –Predict the probability that pixels belong to the foreground.
- **gt_alphas** (*Sequence[np.ndarray]*) –Probability that the actual pixel belongs to the foreground.
- **trimaps** (*Sequence[np.ndarray]*) –Broadly speaking, the trimap consists of foreground and unknown region.

compute_metric(*results: List*) → Dict[str, float]

Compute the MattingMeanSquaredError metric.

参数 **results** (*List*) –A list that consisting the MattingMeanSquaredError score. This list has already been synced across all ranks.

返回 The computed MattingMeanSquaredError metric. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.26 ConnectivityError

class mmeval.metrics.**ConnectivityError**(*step: float = 0.1*, *norm_const: int = 1000*, ***kwargs*)

Connectivity error for evaluating alpha matte prediction.

参数

- **step** (*float*) –Step of threshold when computing intersection between *alpha* and *pred_alpha*. Default to 0.1 .
- **norm_const** (*int*) –Divide the result to reduce its magnitude. Defaults to 1000 .
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

注解: The current implementation assumes the image / alpha / trimap a numpy array with pixel values ranging from 0 to 255.

The pred_alpha should be masked by trimap before passing into this metric.

The trimap is the most commonly used prior knowledge. As the name implies, trimap is a ternary graph and each pixel takes one of {0, 128, 255}, representing the foreground, the unknown and the background respectively.

实际案例

```
>>> from mmeval import ConnectivityError
>>> import numpy as np
>>>
>>> connectivity_error = ConnectivityError()
>>> pred_alpha = np.zeros((32, 32), dtype=np.uint8)
>>> gt_alpha = np.ones((32, 32), dtype=np.uint8) * 255
>>> trimap = np.zeros((32, 32), dtype=np.uint8)
>>> trimap[:16, :16] = 128
>>> trimap[16:, 16:] = 255
>>> connectivity_error(pred_alpha, gt_alpha, trimap)
{'connectivity_error': ...}
```

add(*pred_alphas: Sequence[numpy.ndarray]*, *gt_alphas: Sequence[numpy.ndarray]*, *trimaps: Sequence[numpy.ndarray]*) → None

Add ConnectivityError score of batch to `self._results`

参数

- **pred_alphas** (*Sequence[np.ndarray]*) –Predict the probability that pixels belong to the foreground.
- **gt_alphas** (*Sequence[np.ndarray]*) –Probability that the actual pixel belongs to the foreground.
- **trimaps** (*Sequence[np.ndarray]*) –Broadly speaking, the trimap consists of foreground and unknown region.

compute_metric(*results: List*) → Dict[str, float]

Compute the ConnectivityError metric.

参数 **results** (*List*) –A list that consisting the ConnectivityError score. This list has already been synced across all ranks.

返回 The computed ConnectivityError metric. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.27 DOTAMeanAP

class mmeval.metrics.**DOTAMeanAP**(*iou_thrs: Union[float, List[float]] = 0.5*, *scale_ranges: Optional[List[Tuple]] = None*, *num_classes: Optional[int] = None*, *eval_mode: str = '11points'*, *nproc: int = 4*, *drop_class_ap: bool = True*, *classwise: bool = False*, ***kwargs*)

DOTA evaluation metric.

DOTA is a large-scale dataset for object detection in aerial images which is introduced in https://arxiv.org/abs/1711.10398. This metric computes the DOTA mAP (mean Average Precision) with the given IoU thresholds and scale ranges.

参数

- **iou_thrs** (*float | List[float]*) –IoU thresholds. Defaults to 0.5.
- **scale_ranges** (*List[tuple], optional*) –Scale ranges for evaluating mAP. If not specified, all bounding boxes would be included in evaluation. Defaults to None.
- **num_classes** (*int, optional*) –The number of classes. If None, it will be obtained from the 'CLASSES' field in `self.dataset_meta`. Defaults to None.
- **eval_mode** (*str*) – 'area' or '11points', 'area' means calculating the area under precision-recall curve, '11points' means calculating the average precision of recalls at [0, 0.1, ···, 1]. The PASCAL VOC2007 defaults to use '11points', while PASCAL VOC2012 defaults to use 'area'. Defaults to '11points'.
- **nproc** (*int*) –Processes used for computing TP and FP. If nproc is less than or equal to 1, multiprocessing will not be used. Defaults to 4.
- **drop_class_ap** (*bool*) –Whether to drop the class without ground truth when calculating the average precision for each class.
- **classwise** (*bool*) –Whether to return the computed results of each class. Defaults to False.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> import numpy as np
>>> from mmeval import DOTAMetric
>>> num_classes = 15
>>> dota_metric = DOTAMetric(num_classes=15)
>>>
>>> def _gen_bboxes(num_bboxes, img_w=256, img_h=256):
...     # random generate bounding boxes in 'xywha' formart.
```

(下页继续)

(续上页)

```
...     x = np.random.rand(num_bboxes, ) * img_w
...     y = np.random.rand(num_bboxes, ) * img_h
...     w = np.random.rand(num_bboxes, ) * (img_w - x)
...     h = np.random.rand(num_bboxes, ) * (img_h - y)
...     a = np.random.rand(num_bboxes, ) * np.pi / 2
...     return np.stack([x, y, w, h, a], axis=1)
>>> prediction = {
...     'bboxes': _gen_bboxes(10),
...     'scores': np.random.rand(10, ),
...     'labels': np.random.randint(0, num_classes, size=(10, ))
... }
>>> groundtruth = {
...     'bboxes': _gen_bboxes(10),
...     'labels': np.random.randint(0, num_classes, size=(10, )),
...     'bboxes_ignore': _gen_bboxes(5),
...     'labels_ignore': np.random.randint(0, num_classes, size=(5, ))
... }
>>> dota_metric(predictions=[prediction, ], groundtruths=[groundtruth, ])
{'mAP@0.5': ..., 'mAP': ...}
```

add (*predictions: Sequence[Dict]*, *groundtruths: Sequence[Dict]*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence[Dict]*) –A sequence of dict. Each dict representing a detection result for an image, with the following keys: - bboxes (numpy.ndarray): Shape (N, 5) or shape (N, 8).

  bounding bboxes of this image. The box format is depend on predict_box_type. Details in Note.

  - scores (numpy.ndarray): Shape (N,), the predicted scores of bounding boxes.
  - labels (numpy.ndarray): Shape (N,), the predicted labels of bounding boxes.

- **groundtruths** (*Sequence[Dict]*) –A sequence of dict. Each dict represents a groundtruths for an image, with the following keys:
  - bboxes (numpy.ndarray): Shape (M, 5) or shape (M, 8), the groundtruth bounding bboxes of this image, The box format is depend on predict_box_type. Details in Note.
  - labels (numpy.ndarray): Shape (M,), the ground truth labels of bounding boxes.
  - bboxes_ignore (numpy.ndarray): Shape (K, 5) or shape(K, 8), the groundtruth ignored bounding bboxes of this image. The box format is depend on `self.`

predict_box_type.Details in upper note.

- labels_ignore (numpy.ndarray): Shape (K,), the ground truth ignored labels of bounding boxes.

注解: The box shape of predictions and groundtruths is depends on the predict_box_type. If predict_box_type is ‘rbox’, the box shape should be (N, 5) which represents the (x, y,w, h, angle), otherwise the box shape should be (N, 8) which represents the (x1, y1, x2, y2, x3, y3, x4, y4).

15.1.28 ROUGE

class mmeval.metrics.**ROUGE**(*rouge_keys: Union[List, Tuple, int, str] = (1, 2, 'L')*, *use_stemmer: bool = False*, *normalizer: Optional[Callable] = None*, *tokenizer: Optional[Union[Callable, str]] = None*, *accumulate: str = 'best'*, *lowercase: bool = True*, ***kwargs: Any*)

Calculate Rouge Score used for automatic summarization.

This metric proposed in ROUGE: A Package for Automatic Evaluation of Summaries are common evaluation indicators in the fields of machine translation, automatic summarization, question and answer generation, etc.

参数

- **rouge_keys** (*List or Tuple or int or str*) –A list of rouge types to calculate. Keys that are allowed are L, and 1 through 9. Defaults to (1, 2, 'L').
- **use_stemmer** (*bool*) –Use Porter stemmer to strip word suffixes to improve matching. Defaults to False.
- **normalizer** (*Callable, optional*) –A user’s own normalizer function. If this is None, replacing any non-alpha-numeric characters with spaces is default. Defaults to None.
- **tokenizer** (*Callable or str, optional*) –A user’s own tokenizer function. Defaults to None.
- **accumulate** (*str*) –Useful in case of multi-reference rouge score. avg takes the average of all references with respect to predictions. best takes the best fmeasure score obtained between prediction and multiple corresponding references. Defaults to best.
- **lowercase** (*bool*) –If it is True, all characters will be lowercase. Defaults to True.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval import ROUGE
>>> predictions = ['the cat is on the mat']
>>> references = [['a cat is on the mat']]
>>> metric = ROUGE(rouge_keys='L')
>>> metric.add(predictions, references)
>>> results = metric.compute_metric()
{'rougeL_fmeasure': 0.8333333,
 'rougeL_precision': 0.8333333,
 'rougeL_recall': 0.8333333}
```

add(*predictions: Sequence[str]*, *references: Sequence[Sequence[str]]*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence[str]*) –An iterable of predicted sentences.
- **references** (*Sequence[Sequence[str]]*) –An iterable of referenced sentences. Each predicted sentence may correspond to multiple referenced sentences.

compute_metric(*results: List[Any]*) → dict

Compute the rouge metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List*) –A list that consists correct numbers. This list has already been synced across all ranks.

返回 The computed rouge score.

返回类型 Dict[str, float]

15.1.29 NaturalImageQualityEvaluator

class mmeval.metrics.**NaturalImageQualityEvaluator**(*crop_border: int = 0*, *input_order: str = 'CHW'*, *convert_to: str = 'gray'*, *channel_order: str = 'rgb'*, ***kwargs*)

Calculate Natural Image Quality Evaluator(NIQE) metric.

Ref: Making a "Completely Blind" Image Quality Analyzer. This implementation could produce almost the same results as the official MATLAB codes: http://live.ece.utexas.edu/research/quality/niqe_release.zip

参数

- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the NIQE calculation. Defaults to 0.

- **input_order** (*str*) –Whether the input order is ‘HWC’ or ‘CHW’. Defaults to ‘CHW’.
- **convert_to** (*str*) –Convert the images to other color models. Options are ‘y’ and ‘gray’. Defaults to ‘gray’.
- **channel_order** (*str*) –The channel order of image. Defaults to ‘rgb’.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import NaturalImageQualityEvaluator
>>> import numpy as np
>>>
>>> niqe = NaturalImageQualityEvaluator()
>>> preds = np.random.randint(0, 255, size=(3, 32, 32))
>>> niqe(preds)
{'niqe': ...}
```

add(*predictions: Sequence[numpy.ndarray]*, *channel_order: Optional[str] = None*) → None

Add NIQE score of batch to `self._results`

参数

- **predictions** (*Sequence[np.ndarray]*) –Predictions of the model. Each prediction should be a 4D (as a video, not batches of images) or 3D (as an image) array.
- **channel_order** (*Optional[str]*) –The channel order of the input samples. If not passed, will set as `self.channel_order`. Defaults to None.

compute_feature(*block: numpy.ndarray*) → List

Compute features.

参数 **block** (*np.ndarray*) –2D Image block.

返回 Features with length of 18.

返回类型 List

compute_metric(*results: List[numpy.float64]*) → Dict[str, float]

Compute the NaturalImageQualityEvaluator metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*List[np.float64]*) –A list that consisting the NIQE score. This list has already been synced across all ranks.

返回 The computed NIQE metric.

返回类型 Dict[str, float]

compute_niqe(*prediction: numpy.ndarray*, *channel_order: str*, *mu_pris_param: numpy.ndarray*, *cov_pris_param: numpy.ndarray*, *gaussian_window: numpy.ndarray*, *block_size_h: int = 96*, *block_size_w: int = 96*) → numpy.float64

Calculate NIQE (Natural Image Quality Evaluator) metric. We use the official params estimated from the pristine dataset. We use the recommended block size (96, 96) without overlaps.

Note that we do not include block overlap height and width, since they are always 0 in the official implementation.

For good performance, it is advisable by the official implementation to divide the distorted image into the same size patched as used for the construction of multivariate Gaussian model.

参数

- **prediction** (*np.ndarray*) –Input image whose quality to be computed. Range [0, 255] with float type.
- **channel_order** (*str*) –The channel order of image.
- **mu_pris_param** (*np.ndarray*) –Mean of a pre-defined multivariate Gaussian model calculated on the pristine dataset.
- **cov_pris_param** (*np.ndarray*) –Covariance of a pre-defined multivariate Gaussian model calculated on the pristine dataset.
- **gaussian_window** (*ndarray*) –A 7x7 Gaussian window used for smoothing the image.
- **block_size_h** (*int*) –Height of the blocks in to which image is divided. Defaults to 96.
- **block_size_w** (*int*) –Width of the blocks in to which image is divided. Defaults to 96.

返回 NIQE result.

返回类型 np.float64

estimate_aggd_param(*block: numpy.ndarray*) → Tuple

Estimate AGGD (Asymmetric Generalized Gaussian Distribution) parameters.

参数 **block** (*np.ndarray*) –2D Image block.

返回 alpha(float), beta_l(float) and beta_r(float) for the AGGD distribution (Estimating parameters in Equation 7 in the paper).

返回类型 Tuple

get_size_from_scale(*input_size: Tuple*, *scale_factor: List[float]*) → List[int]

Get the output size given input size and scale factor.

参数

- **input_size** (*Tuple*) –The size of the input image.
- **scale_factor** (*List[float]*) –The resize factor.

返回 The size of the output image.

返回类型 List[int]

get_weights_indices(*input_length: int*, *output_length: int*, *scale: float*, *kernel: Callable*, *kernel_width: float*) → Tuple[List[numpy.ndarray], List[numpy.ndarray]]

Get weights and indices for interpolation.

参数

- **input_length** (*int*) –Length of the input sequence.
- **output_length** (*int*) –Length of the output sequence.
- **scale** (*float*) –Scale factor.
- **kernel** (*Callable*) –The kernel used for resizing.
- **kernel_width** (*float*) –The width of the kernel.

返回 The weights and the indices for interpolation.

返回类型 Tuple[List[np.ndarray], List[np.ndarray]]

matlab_resize(*img: numpy.ndarray*, *scale: float*) → numpy.ndarray

Resize an image to the required size.

参数

- **img** (*np.ndarray*) –The original image.
- **scale** (*float*) –The scale factor of the resize operation.

返回 The resized image.

返回类型 np.ndarray

resize_along_dim(*img_in: numpy.ndarray*, *weights: numpy.ndarray*, *indices: numpy.ndarray*, *dim: int*) → numpy.ndarray

Resize along a specific dimension.

参数

- **img_in** (*np.ndarray*) –The input image.
- **weights** (*np.ndarray*) –The weights used for interpolation, computed from [get_weights_indices].
- **indices** (*np.ndarray*) –The indices used for interpolation, computed from [get_weights_indices].
- **dim** (*int*) –Which dimension to undergo interpolation.

返回 Interpolated (along one dimension) image.

返回类型 np.ndarray

15.1.30 Perplexity

class mmeval.metrics.**Perplexity**(*ignore_labels: Optional[Union[int, List[int]]] = None*, ***kwargs*)

Perplexity measures how well a language model predicts a text sample.

It is commonly used as a metric for evaluating the quality of a language model. It is defined as 2 to the power of the cross-entropy loss of the model (or the negative log-likelihood of the sample).

参数

- **ignore_labels** (*int or list[int], optional*) –Integer specifying a target class to ignore. If given, this class index does not contribute to the returned score. Defaults to None.
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import Perplexity
>>> import numpy as np
>>>
>>> preds = np.random.rand(2, 4, 2)
>>> targets = np.random.randint(low=0, high=2, size=(2, 4))
>>> metric = Perplexity()
>>> result = metric(preds, targets)
{'perplexity': ...}
```

add(*predictions: Sequence*, *targets: Sequence*) → None

Add the intermediate results to `self._results`.

参数

- **predictions** (*Sequence*) –Probabilities assigned to each token in a sequence with shape [batch_size, seq_len, vocab_size].
- **targets** (*Sequence*) –Ground truth values with a shape [batch_size, seq_len].

compute_metric(*results: List[Tuple[float, int]]*) → Dict[str, float]

Compute the perplexity metric.

This method would be invoked in `BaseMetric.compute` after distributed synchronization.

参数 **results** (*list*) –A list that consisting the total and count. This list has already been synced across all ranks.

返回 The computed perplexity metric.

返回类型 Dict[str, float]

15.1.31 CharRecallPrecision

class mmeval.metrics.**CharRecallPrecision**(*letter_case: str = 'unchanged'*, *invalid_symbol: str = '[^A-Za-z0-9 —-🄵]'*, ***kwargs*)

Calculate the char level recall & precision.

参数

- **letter_case** (*str*) –There are three options to alter the letter cases
  - unchanged: Do not change prediction texts and labels.
  - upper: Convert prediction texts and labels into uppercase characters.
  - lower: Convert prediction texts and labels into lowercase characters.

  Usually, it only works for English characters. Defaults to 'unchanged'.
- **invalid_symbol** (*str*) –A regular expression to filter out invalid or not cared characters. Defaults to '[^A-Za-z0-9u4e00-u9fa5]'.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval import CharRecallPrecision
>>> metric = CharRecallPrecision()
>>> metric(['helL', 'HEL'], ['hello', 'HELLO'])
{'char_recall': 0.6, 'char_precision': 0.8571428571428571}
>>> metric = CharRecallPrecision(letter_case='upper')
>>> metric(['helL', 'HEL'], ['hello', 'HELLO'])
{'char_recall': 0.7, 'char_precision': 1.0}
```

add(*predictions: Sequence[str]*, *groundtruths: Sequence[str]*) → None

Process one batch of data and predictions.

参数

- **predictions** (*list[str]*) –The prediction texts.
- **groundtruths** (*list[str]*) –The ground truth texts.

compute_metric(*results: Sequence[Tuple[int, int, int]]*) → Dict

Compute the metrics from processed results.

参数 **results** (*list[tuple]*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict

15.1.32 KeypointEndPointError

class mmeval.metrics.**KeypointEndPointError**(*dataset_meta: Optional[Dict] = None*, *dist_collect_mode: str = 'unzip'*, *dist_backend: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

EPE evaluation metric.

Calculate the end-point error (EPE) of keypoints.

注解:

- length of dataset: N
- num_keypoints: K
- number of keypoint dimensions: D (typically D = 2)

实际案例

```
>>> from mmeval.metrics import KeypointEndPointError
>>> import numpy as np
>>> output = np.array([[[10.,  4.],
...     [10., 18.],
...     [ 0.,  0.],
...     [40., 40.],
...     [20., 10.]]])
>>> target = np.array([[[10.,  0.],
...     [10., 10.],
...     [ 0., -1.],
...     [30., 30.],
...     [ 0., 10.]]])
>>> keypoints_visible = np.array([[True, True, False, True, True]])
>>> predictions = [{'coords': output}]
>>> groundtruths = [{'coords': target, 'mask': keypoints_visible}]
>>> epe_metric = KeypointEndPointError()
>>> epe_metric(predictions, groundtruths)
{'EPE': 11.535533905029297}
```

add(*predictions: Sequence[Dict]*, *groundtruths: Sequence[Dict]*) → None

Process one batch of predictions and groundtruths and add the intermediate results to *self._results*.

参数

- **predictions** (*Sequence[dict]*) –Predictions from the model. Each prediction dict has the following keys:
  - coords (np.ndarray, [1, K, D]): predicted keypoints coordinates
- **groundtruths** (*Sequence[dict]*) –The ground truth labels. Each groundtruth dict has the following keys:
  - coords (np.ndarray, [1, K, D]): ground truth keypoints coordinates
  - mask (np.ndarray, [1, K]): ground truth keypoints_visible

compute_metric(*results: list*) → Dict[str, float]

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.33 KeypointAUC

class mmeval.metrics.**KeypointAUC**(*norm_factor: float = 30*, *num_thrs: int = 20*, ***kwargs*)

AUC evaluation metric.

Calculate the Area Under Curve (AUC) of keypoint PCK accuracy.

By altering the threshold percentage in the calculation of PCK accuracy, AUC can be generated to further evaluate the pose estimation algorithms.

注解:

- length of dataset: N
- num_keypoints: K
- number of keypoint dimensions: D (typically D = 2)

参数

- **norm_factor** (*float*) –AUC normalization factor, Defaults to 30 (pixels).
- **num_thrs** (*int*) –Number of thresholds to calculate AUC. Defaults to 20.

- ****kwargs** –Keyword parameters passed to `mmeval.BaseMetric`. Must include `dataset_meta` in order to compute the metric.

实际案例

```
>>> from mmeval import KeypointAUC
>>> import numpy as np
>>> auc_metric = KeypointAUC(norm_factor=20, num_thrs=4)
>>> output = np.array([[[10.,  4.],
...     [10., 18.],
...     [ 0.,  0.],
...     [40., 40.],
...     [20., 10.]]])
>>> target = np.array([[[10.,  0.],
...     [10., 10.],
...     [ 0., -1.],
...     [30., 30.],
...     [ 0., 10.]]])
>>> keypoints_visible = np.array([[True, True, False, True, True]])
>>> num_keypoints = 15
>>> prediction = {'coords': output}
>>> groundtruth = {'coords': target, 'mask': keypoints_visible}
>>> predictions = [prediction]
>>> groundtruths = [groundtruth]
>>> auc_metric(predictions, groundtruths)
OrderedDict([('AUC@4', 0.375)])
```

add(*predictions: List[Dict]*, *groundtruths: List[Dict]*) → None

Process one batch of predictions and groundtruths and add the intermediate results to *self._results*.

参数

- **predictions** (*Sequence[dict]*) –Predictions from the model. Each prediction dict has the following keys:
  - coords (np.ndarray, [1, K, D]): predicted keypoints coordinates
- **groundtruths** (*Sequence[dict]*) –The ground truth labels. Each groundtruth dict has the following keys:
  - coords (np.ndarray, [1, K, D]): ground truth keypoints coordinates
  - mask (np.ndarray, [1, K]): ground truth keypoints_visible

compute_metric(*results: list*) → Dict[str, float]

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict[str, float]

15.1.34 KeypointNME

class mmeval.metrics.**KeypointNME**(*norm_mode: str*, *norm_item: Optional[str] = None*, *keypoint_indices: Optional[Sequence[int]] = None*, ***kwargs*)

NME evaluation metric.

Calculate the normalized mean error (NME) of keypoints.

注解:

- length of dataset: N
- num_keypoints: K
- number of keypoint dimensions: D (typically D = 2)

参数

- **norm_mode** (*str*) –The normalization mode, which should be one of the following options:
  - 'use_norm_item': Should specify the argument *norm_item*, which represents the item in the datainfo that will be used as the normalization factor.
  - 'keypoint_distance': Should specify the argument *keypoint_indices* that are used to calculate the keypoint distance as the normalization factor.
- **norm_item** (*str, optional*) –The item used as the normalization factor. For example, '*box_size*' in '*AFLWDataset*'. Only valid when norm_mode is use_norm_item. Defaults to None.
- **keypoint_indices** (*Sequence[int], optional*) –The keypoint indices used to calculate the keypoint distance as the normalization factor. Only valid when norm_mode is keypoint_distance. If set as None, will use the default keypoint_indices in *DEFAULT_KEYPOINT_INDICES* for specific datasets, else use the given keypoint_indices of the dataset. Defaults to None.
- ****kwargs** –Keyword parameters passed to BaseMetric.

实际案例

```
>>> from mmeval.metrics import KeypointNME
>>> import numpy as np
>>> aflw_dataset_meta = {
...     'dataset_name': 'aflw',
...     'num_keypoints': 19,
...     'sigmas': np.array([]),
... }
>>> nme_metric = KeypointNME(
...     norm_mode='use_norm_item',
...     norm_item=norm_item,
...     dataset_meta=aflw_dataset_meta)
>>> batch_size = 2
>>> predictions = [{
...     'coords': np.zeros((1, 19, 2))
... } for _ in range(batch_size)]
>>> groundtruths = [{
...     'coords': np.zeros((1, 19, 2)) + 0.5,
...     'mask': np.ones((1, 19)).astype(bool),
...     'box_size': np.ones((1, 1)) * i * 20
... } for i in range(batch_size)]
>>> norm_item = 'box_size'
>>> nme_metric(predictions, groundtruths)
OrderedDict([('NME', 0.03535533892480951)])
```

add (*predictions: Sequence[Dict]*, *groundtruths: Sequence[Dict]*) → None

Add the intermediate results to *self._results*.

参数

- **predictions** (*Sequence[dict]*) –A sequence of dict. Each prediction dict has the following keys:
  - coords (np.ndarray, [1, K, D]): predicted keypoints coordinates
- **groundtruths** (*Sequence[dict]*) –The ground truth labels. Each groundtruth dict has the following keys:
  - coords (np.ndarray, [1, K, D]): ground truth keypoints coordinates
  - mask (np.ndarray, [1, K]): ground truth keypoints_visible

  There are some optional keys as well:
  - bboxes: it is necessary when `self.norm_item` is *'bbox_size'*
  - `self.norm_item`: it is necessary when `self.norm_item` is neither `None` nor *'bbox_size'*

compute_metric(*results: list*) → Dict[str, float]

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are the corresponding results.

返回类型 Dict[str, float]

15.1.35 WordAccuracy

class mmeval.metrics.**WordAccuracy**(*mode: Union[str, Sequence[str]] = 'ignore_case_symbol'*, *invalid_symbol: str = '[^A-Za-z0-9 —-🄵]'*, ***kwargs*)

Calculate the word level accuracy.

参数

- **mode** (*str or list[str]*) –Options are:
  - ' exact' : Accuracy at word level.
  - ' ignore_case' : Accuracy at word level, ignoring letter case.
  - ' ignore_case_symbol' : Accuracy at word level, ignoring letter case and symbol. (Default metric for academic evaluation)

  If mode is a list, then metrics in mode will be calculated separately. Defaults to 'ignore_case_symbol' .
- **invalid_symbol** (*str*) –A regular expression to filter out invalid or not cared characters. Defaults to '[^A-Za-z0-9u4e00-u9fa5]'
- ****kwargs** –Keyword parameters passed to `BaseMetric`.

实际案例

```
>>> from mmeval import WordAccuracy
>>> metric = WordAccuracy()
>>> metric(['hello', 'hello', 'hello'], ['hello', 'HELLO', '$HELLO$'])
{'ignore_case_symbol_accuracy': 1.0}
>>> metric = WordAccuracy(mode=['exact', 'ignore_case',
>>>                             'ignore_case_symbol'])
>>> metric(['hello', 'hello', 'hello'], ['hello', 'HELLO', '$HELLO$'])
{'accuracy': 0.333333333,
 'ignore_case_accuracy': 0.666666667,
 'ignore_case_symbol_accuracy': 1.0}
```

add(*predictions: Sequence[str]*, *groundtruths: Sequence[str]*) → None

Process one batch of data and predictions.

参数

- **predictions** (*list[str]*) –The prediction texts.
- **groundtruths** (*list[str]*) –The ground truth texts.

compute_metric(*results: List[Tuple[int, int, int]]*) → Dict

Compute the metrics from processed results.

参数 **results** (*list[float]*) –The processed results of each batch.

返回

Nested dicts as results. Provided keys are:

- accuracy (float): Accuracy at word level.
- ignore_case_accuracy (float): Accuracy at word level, ignoring letter case.
- ignore_case_symbol_accuracy (float): Accuracy at word level, ignoring letter case and symbol.

返回类型 dict[str, float]

CHAPTER 16

mmeval.utils

mmeval.utils

- *misc*

16.1 misc

try_import	Try to import a module.
has_method	Check whether the object has a method.
is_seq_of	Check whether it is a sequence of some type.
is_list_of	Check whether it is a list of some type.
is_tuple_of	Check whether it is a tuple of some type.
is_filepath	Check if the given object is Path-like.

16.1.1 mmeval.utils.try_import

mmeval.utils.**try_import**(*name: str*) → Optional[module]

Try to import a module.

参数 **name** (*str*) –Specifies what module to import in absolute or relative terms (e.g. either pkg.mod or ..mod).

返回 If importing successfully, returns the imported module, otherwise returns None.

返回类型 ModuleType or None

16.1.2 mmeval.utils.has_method

mmeval.utils.**has_method**(*obj: object*, *method: str*) → bool

Check whether the object has a method.

参数

- **method** (*str*) –The method name to check.
- **obj** (*object*) –The object to check.

返回 True if the object has the method else False.

返回类型 bool

16.1.3 mmeval.utils.is_seq_of

mmeval.utils.**is_seq_of**(*seq: Any*, *expected_type: Union[Type, tuple]*, *seq_type: Optional[Type] = None*) → bool

Check whether it is a sequence of some type.

参数

- **seq** (*Sequence*) –The sequence to be checked.
- **expected_type** (*type or tuple*) –Expected type of sequence items.
- **seq_type** (*type, optional*) –Expected sequence type. Defaults to None.

返回 Return True if `seq` is valid else False.

返回类型 bool

实际案例

```
>>> from mmeval.utils import is_seq_of
>>> seq = ['a', 'b', 'c']
>>> is_seq_of(seq, str)
True
>>> is_seq_of(seq, int)
False
```

16.1.4 mmeval.utils.is_list_of

mmeval.utils.**is_list_of**(*seq*, *expected_type*)

Check whether it is a list of some type.

A partial method of *is_seq_of()*.

16.1.5 mmeval.utils.is_tuple_of

mmeval.utils.**is_tuple_of**(*seq*, *expected_type*)

Check whether it is a tuple of some type.

A partial method of *is_seq_of()*.

16.1.6 mmeval.utils.is_filepath

mmeval.utils.**is_filepath**(*x*)

Check if the given object is Path-like.

参数 **x** (*object*) – Any object.

返回 Returns True if the given is a str or `pathlib.Path`. Otherwise, returns False.

返回类型 bool

CHAPTER 17

Changelog of v0.x

CHAPTER 18

English

CHAPTER 19

简体中文

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

索引

A

B

C

D

E

F

G

H

I

J

K

L

M

N

X

Y