Python机器学习
实验管理与可视化工具介绍

合肥工業大學

杨宇轩

2022.06.16

提　纲

Sacred

Sacred是一个Python库，可以帮助研究人员配置、组织、记录和复制实验。它旨在完成研究人员需要围绕实际实验进行的所有繁琐的日常工作，以便：

- 跟踪实验的所有参数：Experiment, Run
- 轻松进行不同设置的实验：@ex.config
- 将单个运行的配置保存在数据库中：Observer
- 重现结果：Seed

- Sacred文档：https://sacred.readthedocs.io/en/stable/index.html
- 中文博客：https://www.jarvis73.com/2020/11/15/Sacred/
- Meet post

Sacred

例子

```python
from numpy.random import permutation
from sklearn import svm, datasets




C = 1.0
gamma = 0.7



iris = datasets.load_iris()
per = permutation(iris.target.size)
iris.data = iris.data[per]
iris.target = iris.target[per]

clf = svm.SVC(C=C, kernel='rbf', gamma=gamma)
clf.fit(iris.data[:90], iris.target[:90])

print(clf.score(iris.data[90:], iris.target[90:]))
```

```python
from numpy.random import permutation
from sklearn import svm, datasets
from sacred import Experiment                # Sacred 相关
ex = Experiment('iris_rbf_svm')              # Sacred 相关

@ex.config                   # Sacred 相关
def cfg():
    C = 1.0
    gamma = 0.7

@ex.automain                 # Sacred 相关
def run(C, gamma):
    iris = datasets.load_iris()
    per = permutation(iris.target.size)
    iris.data = iris.data[per]
    iris.target = iris.target[per]

    clf = svm.SVC(C=C, kernel='rbf', gamma=gamma)
    clf.fit(iris.data[:90], iris.target[:90])
    # Sacred 相关
    return clf.score(iris.data[90:], iris.target[90:])
```

```
PS C:\Users\17356\Codebase\temp> python iris_rbf_svm.py
WARNING - iris_rbf_svm - No observers have been added to this run
INFO - iris_rbf_svm - Running command 'run'
INFO - iris_rbf_svm - Started
INFO - iris_rbf_svm - Result: 0.9666666666666667
INFO - iris_rbf_svm - Completed after 0:00:00
```

Sacred

程序入口：@ex.main, @ex.automain

- @ex.automain

```python
from sacred import Experiment

# 实例化Experiment类
ex = Experiment()

# 程序执行入口
@ex.automain
def main():
    print("Hello world!")
```

等同

- @ex.main搭配ex.run_commandline()

```python
from sacred import Experiment

ex = Experiment()

@ex.main
def main():
    print("Hello world!")

ex.run_commandline()
```

- @ex.automain相当于@ex.main+ex.run_commandline()
- 但要注意：带有 @ex.automain装饰器的函数必须放到脚本文件的末尾

```
PS C:\Users\17356\Codebase\temp> python hello_world.py
WARNING - hello_world - No observers have been added to this run
INFO - hello_world - Running command 'main'
INFO - hello_world - Started
Hello world!
INFO - hello_world - Completed after 0:00:00
```

Sacred

轻松进行不同设置的实验：@ex.config

单个文件

```python
from sacred import Experiment
import numpy as np
ex = Experiment('cfg demo')

@ex.config
def cfg():
    a = 1
    b = 2.3
    c = np.array([4,5,6])

@ex.automain
def main(a, b, c):
    result = (c + a) * b
    return result
```

等同

分配置文件和运行文件

```python
from sacred import Experiment
import numpy as np
ex = Experiment('cfg demo')

@ex.config
def cfg():
    a = 1
    b = 2.3
    c = np.array([4,5,6])
```

```python
from cfg_demo_cfg import ex

@ex.automain
def main(a, b, c):
    result = (c + a) * b
    return result
```

- 配置函数中支持任何Python语法，可以使用分支结构等来动态控制参数
- 参数类型支持整型，浮点型，字符串，元组，列表，字典等可以Json序列化的类型

```
PS C:\Users\17356\Codebase\temp> python cfg_demo_run.py
WARNING - cfg demo - No observers have been added to this run
INFO - cfg demo - Running command 'main'
INFO - cfg demo - Started
INFO - cfg demo - Result: [11.5 13.8 16.1]
INFO - cfg demo - Completed after 0:00:00
```

Sacred

支持配置注入的捕获函数：@ex.capture

```python
from sacred import Experiment
ex = Experiment('capture demo')

@ex.config
def cfg():
    a = 1       # int
    b = 'abc'   # str
    c = 1.2     # float

@ex.capture
def print_param(a, b, c=2.3):
    print(f'a={a}, b={b}, c={c}')

@ex.automain
def main():
    print_param()
    print_param(2, 'def')
    print_param(b=2, c=10.5)
```

配置域中的参数可以直接注入到捕获函数的参数列表中。捕获函数包括：

- @ex.main
- @ex.automain
- @ex.capture
- @ex.command

注意：c=2.3这个默认参数永远不会被使用，参数优先级顺序：调用时传参 > Sacred 参数 > 默认参数。

```
PS C:\Users\17356\Codebase\temp> python capture_demo.py
WARNING - capture demo - No observers have been added to this run
INFO - capture demo - Running command 'main'
INFO - capture demo - Started
a=1, b=abc, c=1.2
a=2, b=def, c=1.2
a=1, b=2, c=10.5
INFO - capture demo - Completed after 0:00:00
```

Sacred

支持配置注入的捕获函数：@ex.capture

```python
from sacred import Experiment
ex = Experiment('capture demo')

@ex.config
def cfg():
    a = 1       # int
    b = 'abc'   # str
    c = 1.2     # float

@ex.capture
def some_function(_config, _log):
    _log.warning(f"Config a={_config['a']}")

@ex.capture
def do_random_stuff(_rnd, _seed):
    print(_seed)
    print(_rnd.randint(1, 100))

@ex.automain
def main():
    some_function()
    do_random_stuff()
```

捕获函数可以获取一些 Sacred 内置的变量，要使用时写在捕获函数的参数列表中就可以直接调用：

- _config：所有的参数作为一个字典（只读的）
- _log：Python标准Logger
- _seed：每次调用函数都不同的随机数
- _rnd：伪随机数生成器
- _run：当前实验运行时的 run 对象

```
PS C:\Users\17356\Codebase\temp> python capture_demo.py
WARNING - capture demo - No observers have been added to this run
INFO - capture demo - Running command 'main'
INFO - capture demo - Started
WARNING - some_function - Config a=1
165208810
39
INFO - capture demo - Completed after 0:00:00
```

Sacred

命令行接口

```python
from sacred import Experiment
import numpy as np
ex = Experiment('cmd demo')

@ex.config
def cfg():
    a = 1       # int
    b = 'abc'   # str

@ex.automain
def main(a, b, c):
    print(f'a: {type(a)}, {a}')
    print(f'b: {type(b)}, {b}')
    print(f'c: {type(c)}, {c}')
```

命令行接口最大的作用就是更新实验参数（with）。

```
PS C:\Users\17356\Codebase\temp> python cmd_demo.py with c=[1,2,3]
WARNING - root - Added new config entry: "c"
WARNING - cmd demo - No observers have been added to this run
INFO - cmd demo - Running command 'main'
INFO - cmd demo - Started
a: <class 'int'>, 1
b: <class 'str'>, abc
c: <class 'sacred.config.custom_containers.ReadOnlyList'>, [1, 2, 3]
INFO - cmd demo - Completed after 0:00:00
```

使用配置文件更新实验参数，
支持json、yaml、pickle格式

```
{
    "a": 2,
    "b": 1.5,
    "c": "hello world"
}
```

```
PS C:\Users\17356\Codebase\temp> python cmd_demo.py with config.json
WARNING - root - Added new config entry: "c"
WARNING - root - Changed type of config entry "b" from str to float
WARNING - cmd demo - No observers have been added to this run
INFO - cmd demo - Running command 'main'
INFO - cmd demo - Started
a: <class 'int'>, 2
b: <class 'float'>, 1.5
c: <class 'str'>, hello world
INFO - cmd demo - Completed after 0:00:00
```

Sacred

命令行接口

Sacred内置了一系列命令，同时用户可以自定义命令。

➢ 内置命令

命令	参数
`print_config`	仅打印参数. 对于同时更新了的参数, 会使用三种颜色来标记: 更改的(蓝色), 增加的(绿色), 类型改变的(红色)
`print_dependencies`	打印程序依赖, 源文件, git 版本控制
`save_config`	保存当前参数到文件, 默认保存到 `config.json`
`print_named_configs`	打印 `@ex.named_config` 修饰的参数组

```
PS C:\Users\17356\Codebase\temp> python cmd_demo.py print_config
INFO - cmd demo - Running command 'print_config'
INFO - cmd demo - Started
Configuration (modified, added, typechanged, doc):
  a = 1                                # int
  b = 'abc'                            # str
  seed = 103631398                     # the random seed for this experiment
INFO - cmd demo - Completed after 0:00:00
```

➢ 自定义命令：使用@ex.command

```python
from sacred import Experiment
ex = Experiment('cmd demo')

@ex.command
def train(_run, _config):
    print("Training a network.")

@ex.automain
def main():
    pass
```

```
PS C:\Users\17356\Codebase\temp> python cmd_def_demo.py train
WARNING - cmd demo - No observers have been added to this run
INFO - cmd demo - Running command 'train'
INFO - cmd demo - Started
Training a network.
INFO - cmd demo - Completed after 0:00:00
```

Sacred

将单个运行的配置保存在数据库中：Observer

Sacred 会在每次实验中搜集大量的信息，为了获取这些信息，我们需要显式的添加观察器。

- Mongo Observer
- File Storage Observer
- TinyDB Observer
- SQL Observer
- S3 Observer
- gcs_observer
- Queue Observer
- Custom Observer（自定义Observer）

https://sacred.readthedocs.io/en/stable/observers.html

Sacred

将单个运行的配置保存在数据库中：Observer

➢ File Storage Observer

- **通过代码设置**

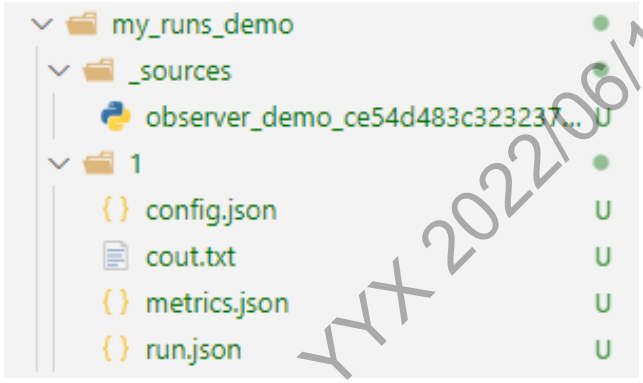
```
from sacred import Experiment
from sacred.observers import FileStorageObserver

ex = Experiment('observer demo')
ex.observers.append(FileStorageObserver('my_runs_demo'))
```

- **通过命令行设置**

```
>> python observer_demo.py -F 'my_runs_demo'
>> python observer_demo.py --file_storage='my_runs_demo'
```

在my_runs_demo文件夹下保存了以下内容：

- **源代码**
- **配置参数**
- **输出**
- **追踪的metric**
- **运行详细配置，包括环境、依赖库等**



Sacred

跟踪实验的所有参数：Experiment, Run

ex=Experiment()为Experiment对象。

Run对象可以通过捕获函数接收的_run参数获得，run=ex.run()完成之后返回的run变量也是Run对象。

- **Metrics**

使用_run.log_scalar()或者ex.log_scalar()，记录如损失、精度等信息。

`log_scalar`(*metric_name*, *value*, *step=None*)

Add a new measurement.

The measurement will be processed by the MongoDB observer during a heartbeat event. Other observers are not yet supported.

Parameters:

- **metric_name** – The name of the metric, e.g. training.loss
- **value** – The measured value
- **step** – The step number (integer), e.g. the iteration number If not specified, an internal counter for each metric is used, incremented by one.

Sacred

跟踪实验的所有参数：Experiment, Run

- **Metrics**

使用_run.log_scalar()或者ex.log_scalar()，记录如损失、精度等信息。

```python
from sacred import Experiment
from sacred.observers import FileStorageObserver

ex = Experiment('metrics demo')
ex.observers.append(FileStorageObserver('my_runs_demo'))

@ex.config
def cfg():
    a = list(range(10))

@ex.capture
def linear(a, _run):
    for i in a:
        _run.log_scalar('data.x', i)      # step从0开始自动递增
        _run.log_scalar('data.y', i*2)
        _run.log_scalar('data,z', i, 2)  # step=2

@ex.automain
def main():
    linear()
```


```
},
"data.y": {
  "steps": [
    0,
    1,
    2,
    3,
    4,
    5,
    6,
    7,
    8,
    9
  ],
  "timestamps": [
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525",
    "2022-06-08T03:10:03.493525"
  ],
  "values": [
    0,
    2,
    4,
    6,
    8,
    10,
    12,
    14,
    16,
    18
  ]
}
```

Sacred

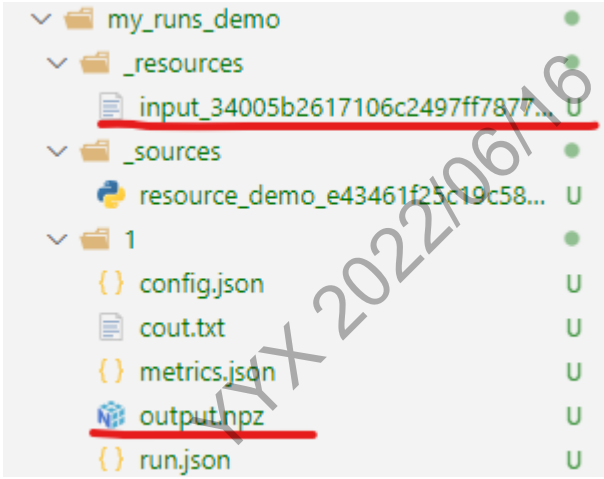
跟踪实验的所有参数：Experiment, Run

- 资源（resource）和工件（artifact）

```
from sacred import Experiment
from sacred.observers import FileStorageObserver
import numpy as np

ex = Experiment('resource demo')
ex.observers.append(FileStorageObserver('my_runs_demo'))

@ex.config
def cfg():
    input_name = 'input.txt'
    output_name = 'output.npz'

@ex.capture
def to_npz(input_name, output_name, _run):
    array = []
    with _run.open_resource(input_name) as f:
        for line in f.readlines():
            array.append(list(map(float,
line.strip().split(' '))))
    array = np.array(array)
    np.savez(output_name, array)
    _run.add_artifact(output_name)

@ex.automain
def main():
    to_npz()
```

资源指实验中需要用到的文件。使用open_resource(filename, mode='r')或add_resource(filename)，观察器会记录资源信息。

工件指实验中产生的文件。使用add_artifact(filename)以提供给观察器记录。



Sacred

重现结果：Seed

➢ 全局随机数种子

Sacred会为每次实验自动生成一个种子，作为配置的一部分。也可以自行设置：

```
>> python experiment.py with seed=123
```

在开始实验时，Sacred会自动设置以下库的seed：

- random
- numpy.random
- tensorflow.set_random_seed
- pytorch.manual_seed

➢ 捕获函数的特殊参数

- _seed：每次调用函数都不同的随机数
- _rnd：伪随机数生成器

https://sacred.readthedocs.io/en/stable/randomness.html

Tensorboard

在机器学习中，我们一般需要记录模型训练的评估指标、参数等等。TensorBoard就是一个功能极为强大的机器学习实时检测并可视化的工具。它可以实时跟踪并可视化loss、acc等标量，可以使用直方图、分布图来展示权重和梯度分布的变化，还可以将将嵌入向量投影到较低维度的空间从而可视化等等。

TensorBoard最早是TensorFlow的可视化工具包。在PyTorch 1.1.0版本，官方与TensoBoard合作加入了对Tensorboard的支持接口torch.utils.tensorboard。

- Tensorboard官网：https://www.tensorflow.org/tensorboard
- Pytorch教程：https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html
- Pytorch文档：https://pytorch.org/docs/stable/tensorboard.html

Tensorboard

安装

```
>> pip install tensorboard
```

使用

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter('./path/to/log')
```

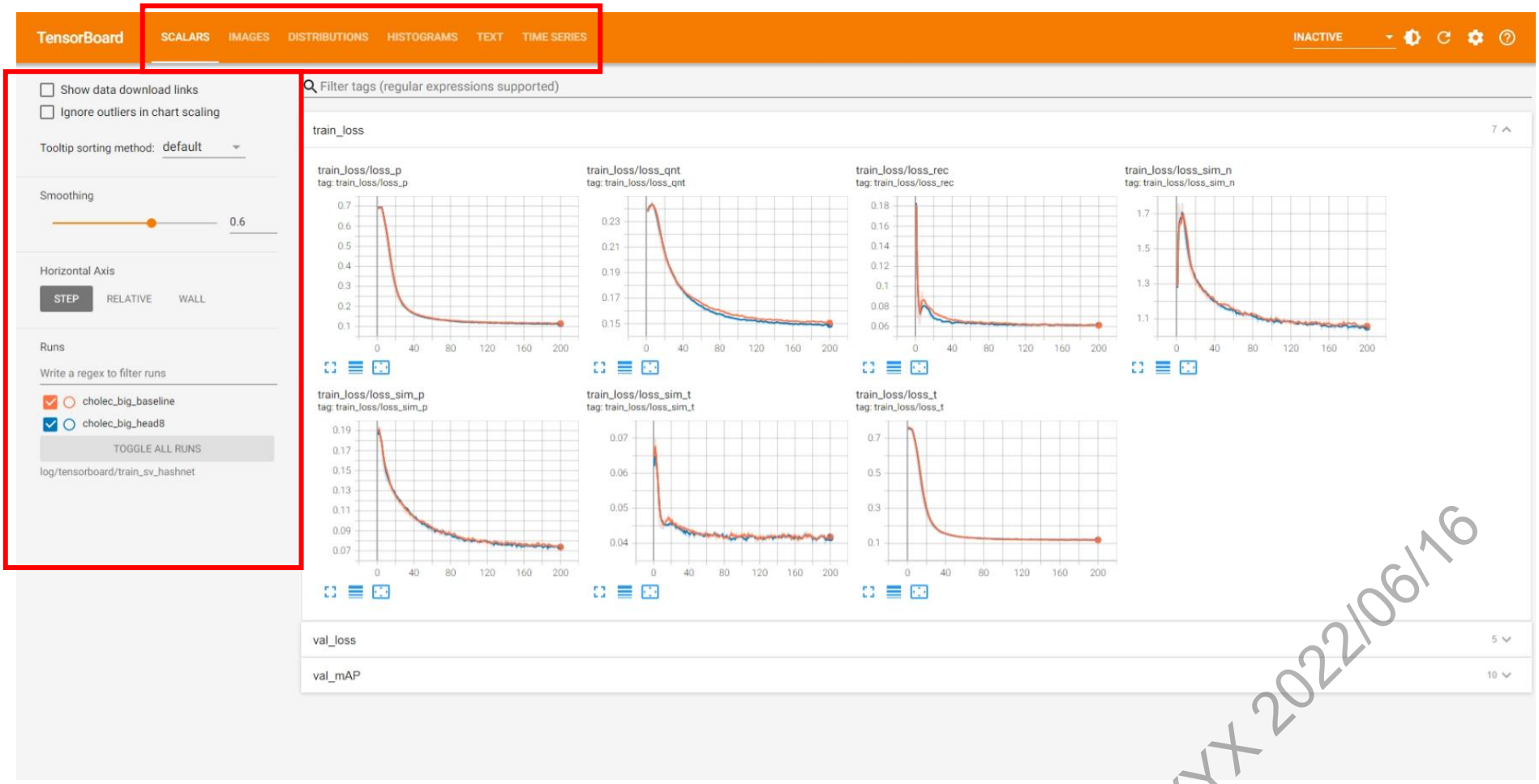
定义SummaryWriter对象，供TensorBoard记录。

```
tensorboard --logdir='./path/to/log' --port 6006 --bind_all
```

从终端启动Tensorboard。TensorBoard将递归遍历以logdir为根的目录结构，查找.*tfevents.* 文件。如果需要访问不在本地启动的Tensorboard，需要添加—bind_all参数。启动后打开http://主机地址:端口/（如http://localhost:6006/），端口默认为6006。

Tensorboard

➢ 界面



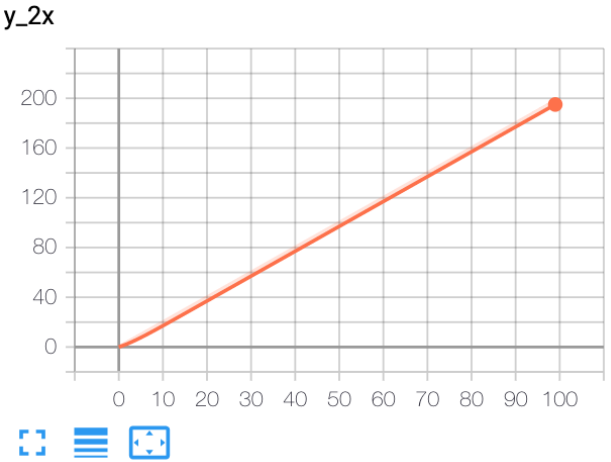
YYX 2022/06/16

Tensorboard

标量

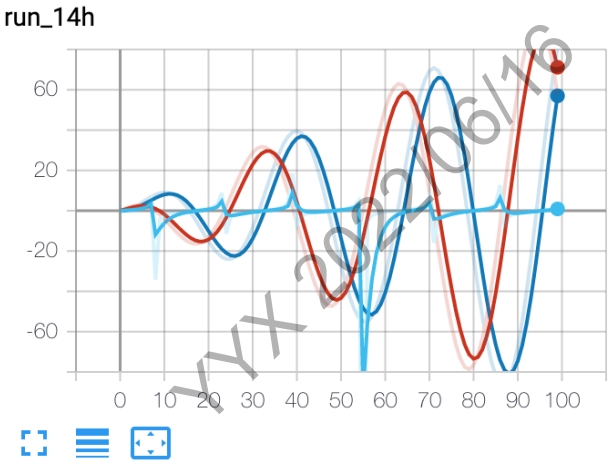
```
add_scalar(tag, scalar_value, global_step=None, walltime=None, new_style=False,
double_precision=False) [SOURCE]
```

```python
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
x = range(100)
for i in x:
    writer.add_scalar('y=2x', i * 2, i)
writer.close()
```




```
add_scalars(main_tag, tag_scalar_dict, global_step=None, walltime=None)
```

```python
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
r = 5
for i in range(100):
    writer.add_scalars('run_14h', {'xsinx':i*np.sin(i/r),
                                   'xcosx':i*np.cos(i/r),
                                   'tanx': np.tan(i/r)}, i)
writer.close()
```



Tensorboard

图像的分组

当记录了很多信息后，会出现图像大量堆叠导致UI混乱不美观。可以使用分组功能，只需要将参数tag分层命名（‘/’）即可。不局限于标量图像，所有带tag参数的图像都可以分组。

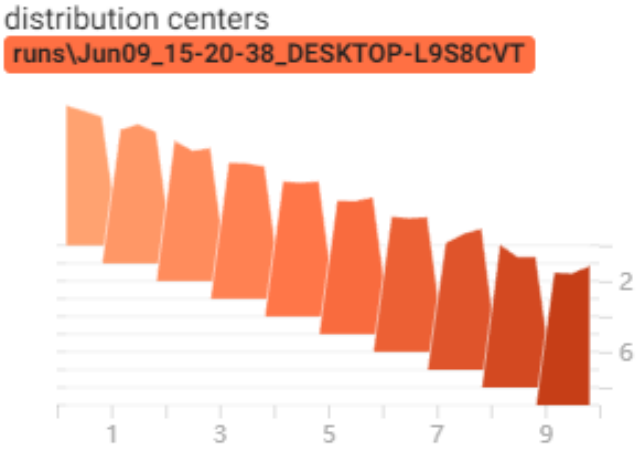
```
for n_iter in range(100):
    writer.add_scalar('Loss/train', np.random.random(), n_iter)
    writer.add_scalar('Loss/test', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/train', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/test', np.random.random(), n_iter)
```

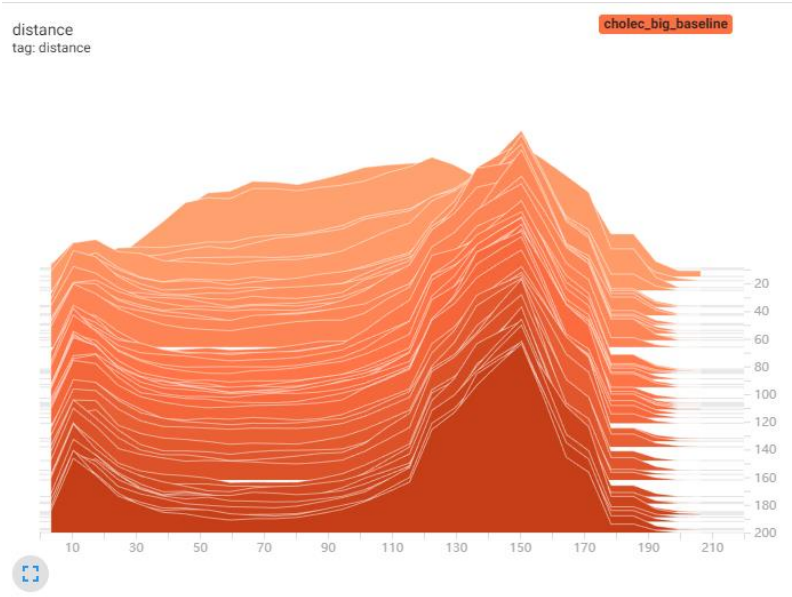
Tensorboard

直方图：可视化数据分布

```
add_histogram(tag, values, global_step=None, bins='tensorflow', walltime=None,
max_bins=None) [SOURCE]
```

```python
from torch.utils.tensorboard import SummaryWriter
import numpy as np
writer = SummaryWriter()
for i in range(10):
    x = np.random.random(1000)
    writer.add_histogram('distribution centers', x + i, i)
writer.close()
```





YYX 2022/06/16

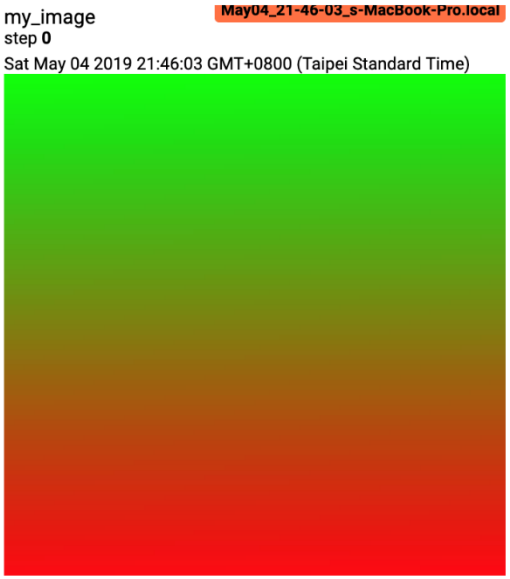
Tensorboard

图片

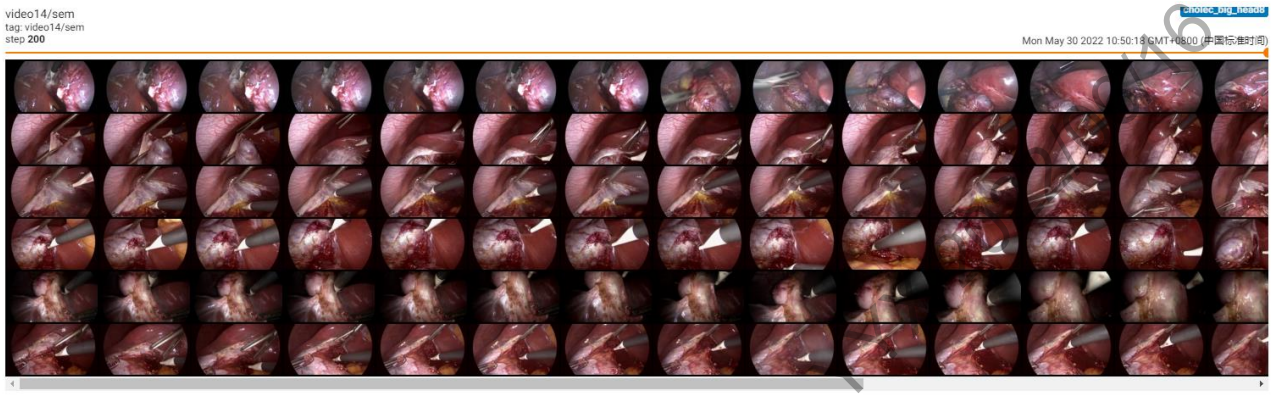
```
add_image(tag, img_tensor, global_step=None, walltime=None, dataformats='CHW')
```

Shape:

img_tensor: Default is $(3, H, W)$. You can use `torchvision.utils.make_grid()` to convert a batch of tensor into 3xHxW format or call `add_images` and let us do the job. Tensor with $(1, H, W)$, (H, W), $(H, W, 3)$ is also suitable as long as corresponding `dataformats` argument is passed, e.g. `CHW`, `HWC`, `HW`.



如果不是在本机使用Tensorboard，建议适当降低图片分辨率，以降低网络延迟和存储消耗。



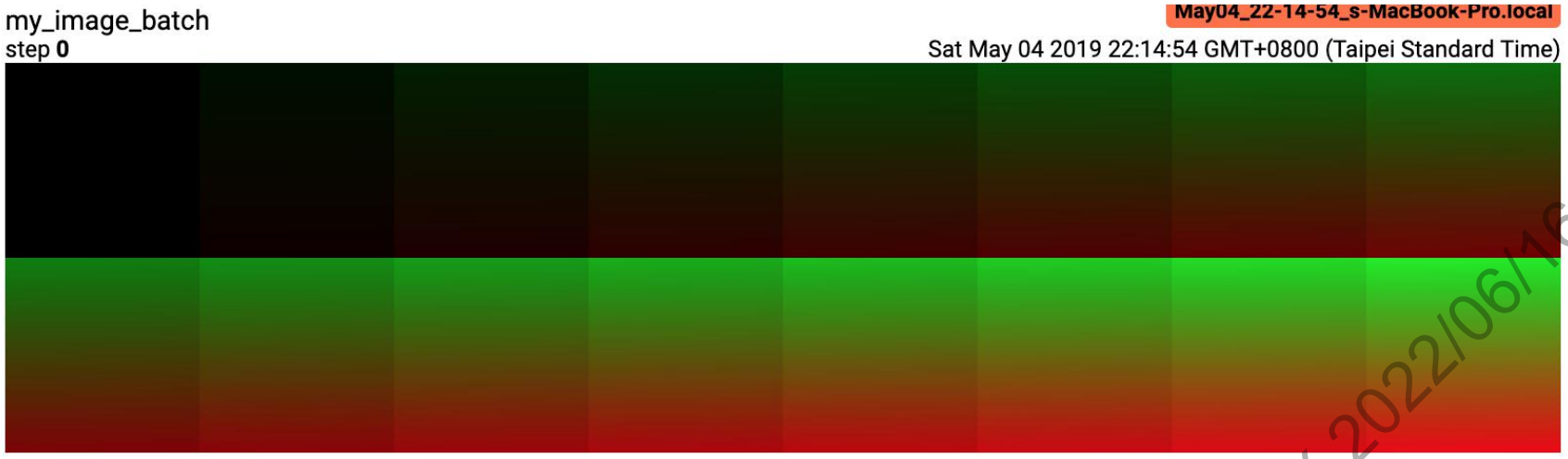
Tensorboard

图片

```
add_images(tag, img_tensor, global_step=None, walltime=None, dataformats='NCHW')
```

Shape:

img_tensor: Default is $(N, 3, H, W)$. If `dataformats` is specified, other shape will be accepted. e.g. NCHW or NHWC.



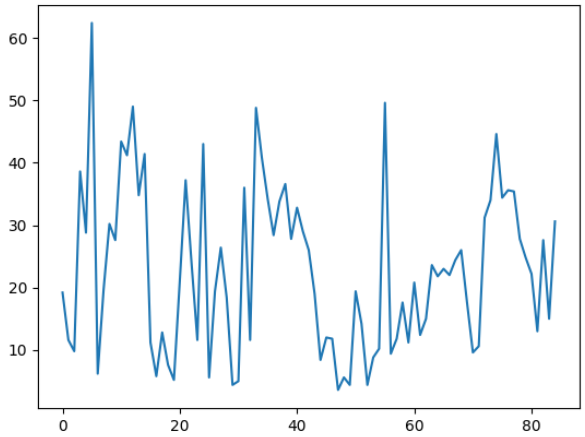
Tensorboard

图像

`add_figure(tag, figure, global_step=None, close=True, walltime=None)` [SOURCE]

Render matplotlib figure into an image and add it to summary.

Note that this requires the `matplotlib` package.



Parameters

- **tag** (*string*) – Data identifier
- **figure** (*matplotlib.pyplot.figure*) – Figure or a list of figures
- **global_step** (*int*) – Global step value to record
- **close** (*bool*) – Flag to automatically close the figure
- **walltime** (*float*) – Optional override default walltime (time.time()) seconds after epoch of event

Tensorboard

视频

```
add_video(tag, vid_tensor, global_step=None, fps=4, walltime=None)
```

Shape:

vid_tensor: (N, T, C, H, W). The values should lie in [0, 255] for type *uint8* or [0, 1] for type *float*.

音频

```
add_audio(tag, snd_tensor, global_step=None, sample_rate=44100, walltime=None)
```

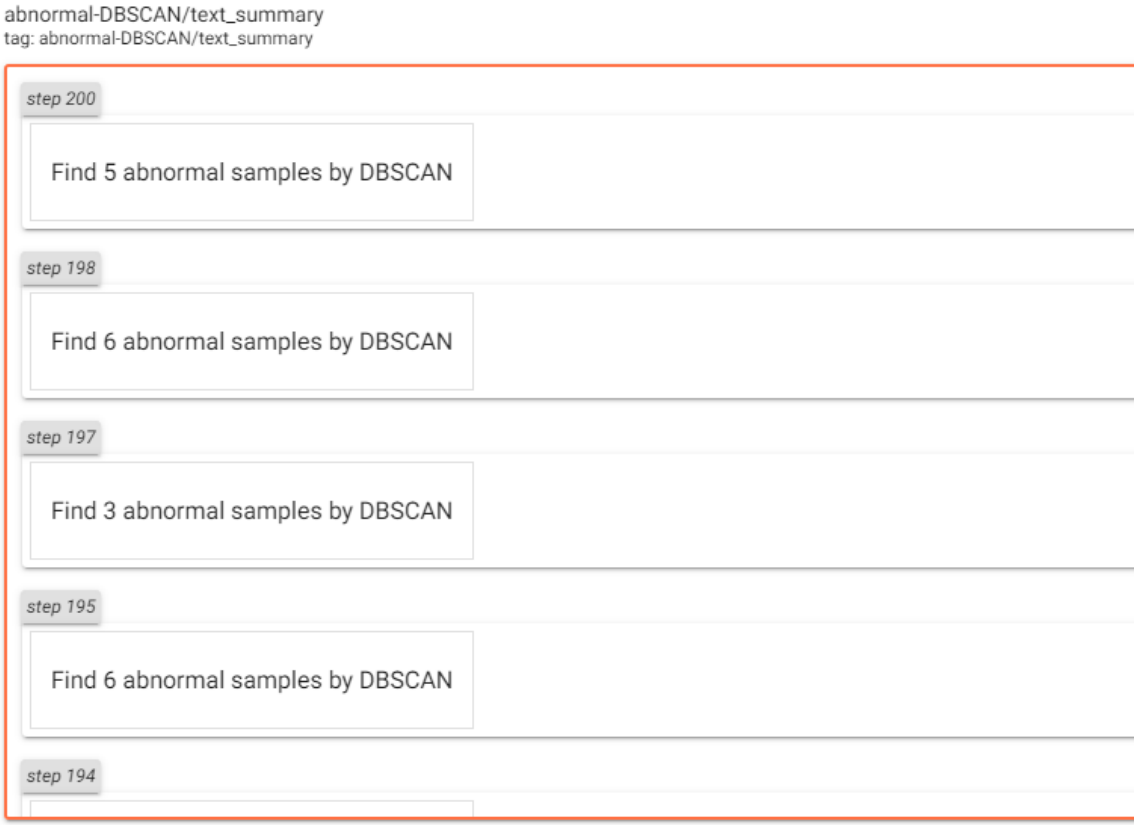
Shape:

snd_tensor: $(1, L)$. The values should lie between [-1, 1].

Tensorboard

文本

```
add_text(tag, text_string, global_step=None, walltime=None)
```



YYX 2022/06/16

Tensorboard

向量

```
add_embedding(mat, metadata=None, label_img=None, global_step=None, tag='default',
metadata_header=None) [SOURCE]
```

Add embedding projector data to summary.

Parameters

- **mat** (*torch.Tensor or numpy.array*) – A matrix which each row is the feature vector of the data point
- **metadata** (*list*) – A list of labels, each element will be convert to string
- **label_img** (*torch.Tensor*) – Images correspond to each data point
- **global_step** (*int*) – Global step value to record
- **tag** (*string*) – Name for the embedding

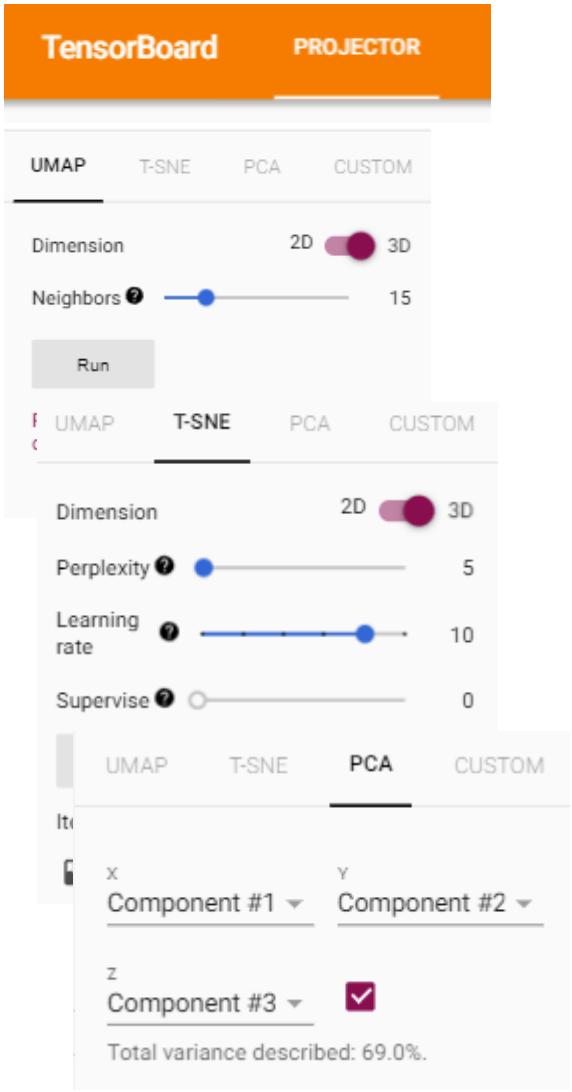
Shape:

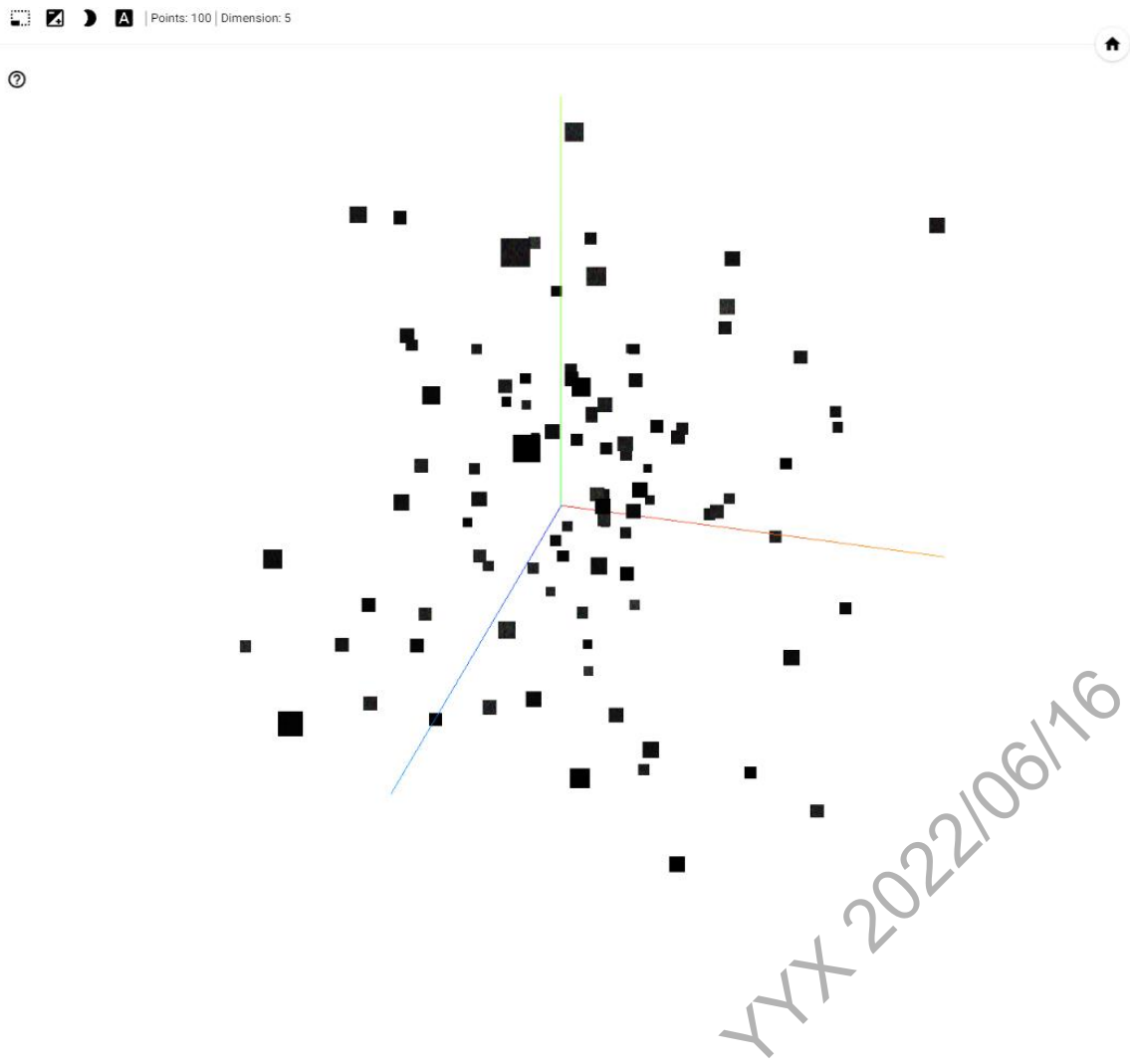
mat: (N, D), where N is number of data and D is feature dimension

label_img: (N, C, H, W)

Tensorboard

向量





Tensorboard

模型图

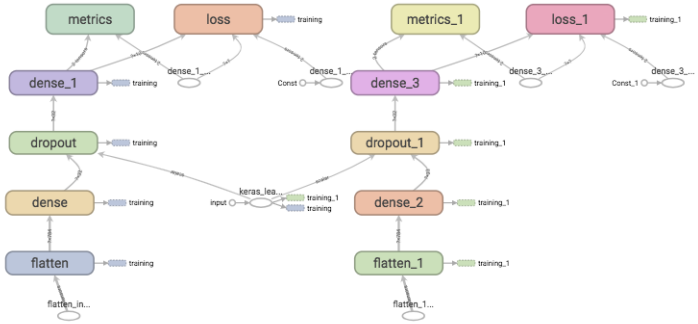
`add_graph(model, input_to_model=None, verbose=False, use_strict_trace=True)` [SOURCE]

Add graph data to summary.

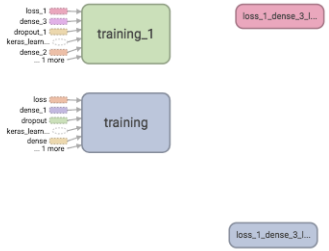
Parameters

- **model** (*torch.nn.Module*) – Model to draw.
- **input_to_model** (*torch.Tensor or list of torch.Tensor*) – A variable or a tuple of variables to be fed.
- **verbose** (*bool*) – Whether to print graph structure in console.
- **use_strict_trace** (*bool*) – Whether to pass keyword argument *strict* to *torch.jit.trace*. Pass False when you want the tracer to record your mutable container types (list, dict)

Main Graph

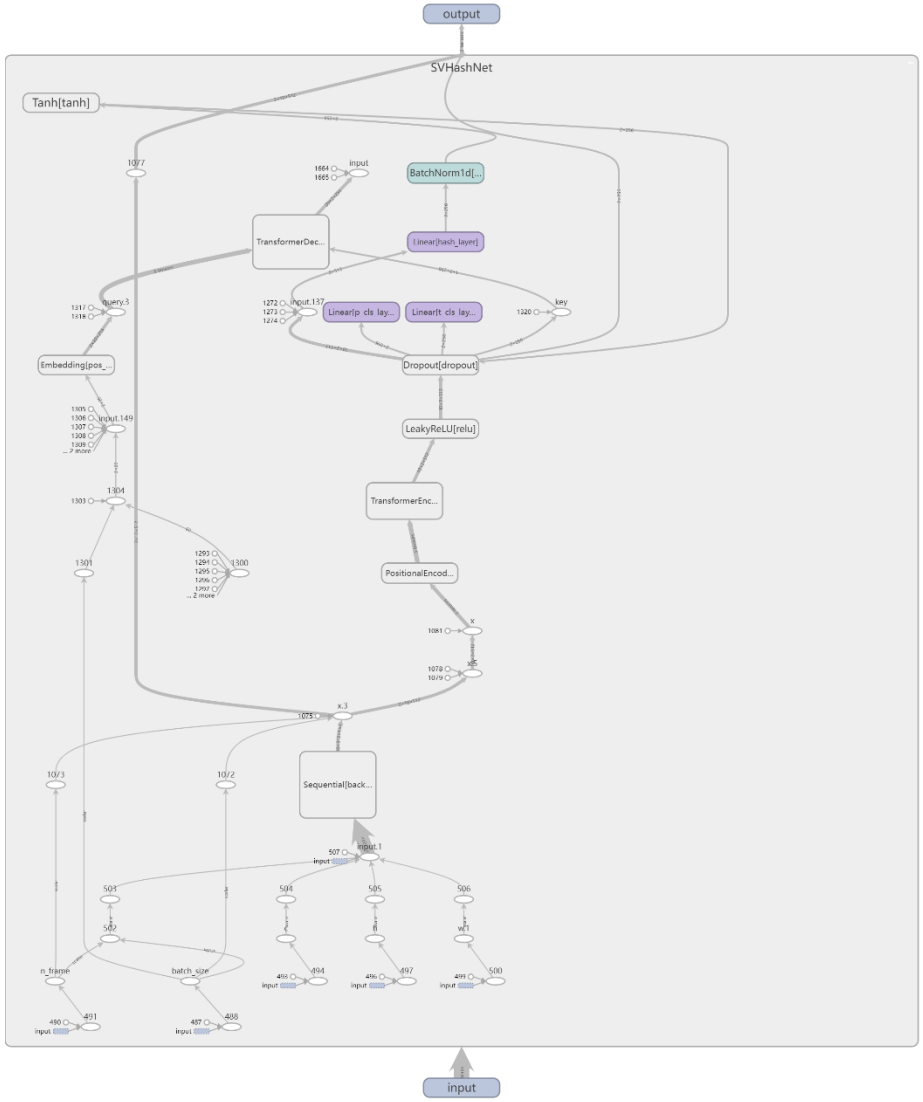


Auxiliary Nodes



Tensorboard

模型图



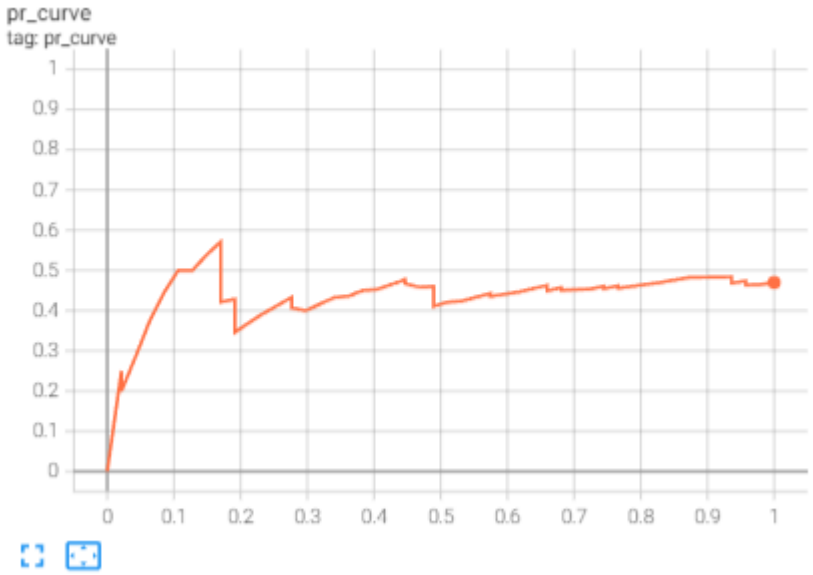
YYX 2022/06/16

Tensorboard

PR曲线

```
add_pr_curve(tag, labels, predictions, global_step=None, num_thresholds=127,
weights=None, walltime=None) [SOURCE]
```

```python
from torch.utils.tensorboard import SummaryWriter
import numpy as np
labels = np.random.randint(2, size=100)  # binary label
predictions = np.random.rand(100)
writer = SummaryWriter()
writer.add_pr_curve('pr_curve', labels, predictions, 0)
writer.close()
```

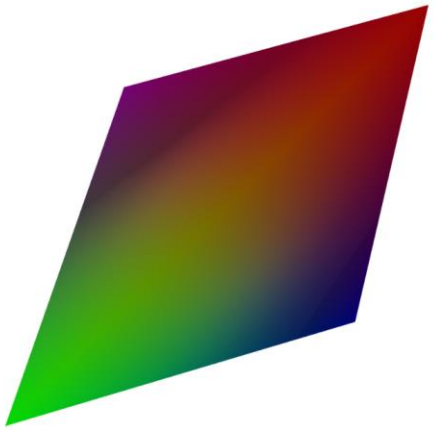


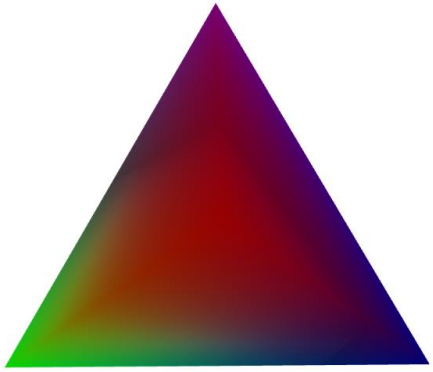
YYX 2022/06/16

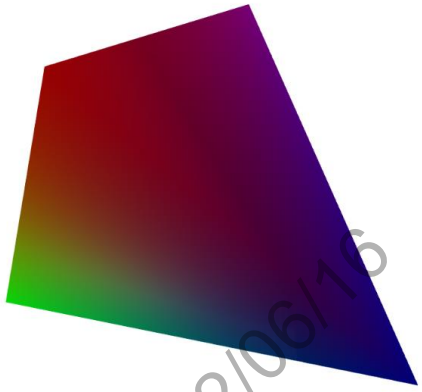
Tensorboard

3D网格、点云

```
add_mesh(tag, vertices, colors=None, faces=None, config_dict=None, global_step=None,
walltime=None) [SOURCE]
```







Tensorboard

超参数

`add_hparams(hparam_dict, metric_dict, hparam_domain_discrete=None, run_name=None)` [SOURCE]

Add a set of hyperparameters to be compared in TensorBoard.

Parameters

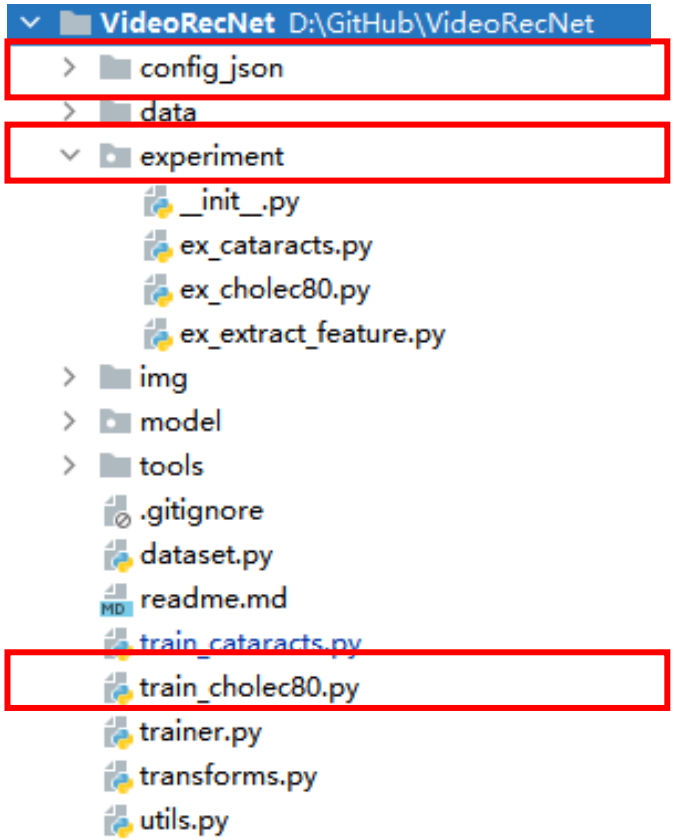
- **hparam_dict** (*dict*) – Each key-value pair in the dictionary is the name of the hyper parameter and it's corresponding value. The type of the value can be one of *bool*, *string*, *float*, *int*, or *None*.
- **metric_dict** (*dict*) – Each key-value pair in the dictionary is the name of the metric and it's corresponding value. Note that the key used here should be unique in the tensorboard record. Otherwise the value you added by `add_scalar` will be displayed in hparam plugin. In most cases, this is unwanted.
- **hparam_domain_discrete** – (Optional[Dict[str, List[Any]]]) A dictionary that contains names of the hyperparameters and all discrete values they can hold
- **run_name** (*str*) – Name of the run, to be included as part of the logdir. If unspecified, will use current timestamp.

Session Group Name.	Show Metrics	lr	bsize	accuracy	loss
Jul16_21-44-19_s-...	☐	0.0000	0.0000	0.0000	0.0000
Jul16_21-44-19_s-...	☐	0.10000	1.0000	10.000	10.000
Jul16_21-44-19_s-...	☐	0.20000	2.0000	20.000	20.000
Jul16_21-44-19_s-...	☐	0.30000	3.0000	30.000	30.000
Jul16_21-44-19_s-...	☐	0.40000	4.0000	40.000	40.000

实际使用

Sacred参数设置

➢ 文件结构



配置文件

实验参数文件

主程序

YYX 2022/06/16

实际使用

Sacred参数

➢ 实验参数文件

```
@ingredient.config
def base_cfg():
    # ==================== Path Config ====================
    label_dir = ""
    data_dir = ""

    # ==================== Learning Config ====================
    batch_size = 256
    epochs = 200

def parse_config(_config):
    """ Parse base configurations and add new configurations. """
    config = Map({k: v for k, v in _config.items()})
    ......
    return config


class Map(dict):
    """ Support getting dict item by dot(.) operation """
    ......
```

基础配置，用@ex.config注解，不作产生中间变量的计算，能够保证配置文件简洁、不冗余

进一步的配置，基于基础参数计算，并加入新的参数

使得配置支持点操作符(.)

实际使用

Sacred参数

- 实验参数文件

```
class Map(dict):
    """ Support getting dict item by dot(.) operation """
    __getattr__ = dict.__getitem__
    __setattr__ = dict.__setitem__
    __delattr__ = dict.__delitem__

    def __init__(self, obj):
        new_dict = {}
        if isinstance(obj, dict):
            for k, v in obj.items():
                if isinstance(v, dict):
                    new_dict[k] = Map(v)
                else:
                    new_dict[k] = v
        else:
            raise TypeError(f"`obj` must be a dict, got {type(obj)}")
        super(Map, self).__init__(new_dict)
```

实际使用

Sacred参数

➢ 主函数

```
@ex.main
def main(_config):
    config = parse_config(_config)
```

主函数用@ex.main注解，通过捕获函数的功能，将_config作为参数注入。接下来使用parse_config函数进一步处理，得到的config包含所有配置参数，可以使用config.xxx调用。

➢ 其他函数

其他函数需要调用实验参数时（如train函数），可以在参数列表里加上config参数，传入config。尽量少使用捕获函数的功能，代码可读性降低，难以debug。

```
def train(model, dataloader, criterions, optimizer, epoch, config, ex, writer):
    ......
```

实际使用

将Sacred和Tensorboard一同使用

- **主程序内初始化**

```
# init logger
ex_name = 'experiment'

from sacred import Experiment
from sacred.observers import FileStorageObserver

ex = Experiment(ex_name, save_git_info=False, ingredients=[ingredient])
ex.observers.append(FileStorageObserver(f'log/sacred/{ex_name}'))

from torch.utils.tensorboard import SummaryWriter
import datetime

log_dir = f'log/tensorboard/{ex_name}/{datetime.datetime.now().strftime("%Y%m%d-%H%M%S")}'
writer = SummaryWriter(log_dir)
```

实际使用

将Sacred和Tensorboard一同使用

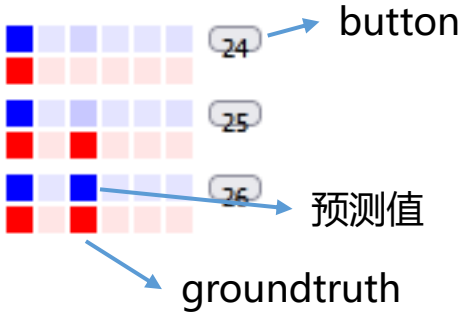
➢ 记录标量

```python
def log_metrics(metrics: dict, epoch, sacred_ex=None, tb_writer=None):
    parsed = {}

    def r_parse(metrics: dict, key_pref=''):
        for k, v in metrics.items():
            full_key = '/'.join([key_pref, k]) if key_pref != '' else k
            if isinstance(v, dict):
                r_parse(v, full_key)  # `v` is a dict
            else:
                parsed[full_key] = v  # `v` is a value

    r_parse(metrics)
    for k, v in parsed.items():
        if sacred_ex is not None:
            sacred_ex.log_scalar(k, v, epoch)
        if tb_writer is not None:
            tb_writer.add_scalar(k, v, epoch)
```

```python
metrics = {
    'val_loss': {
        'loss_p': loss_p_meter.avg,
        'loss_t': loss_t_meter.avg,
        'loss_qnt': loss_qnt_meter.avg,
        'loss_sim_p': loss_sim_p_meter.avg,
        'loss_sim_t': loss_sim_t_meter.avg
    },
    'val_mAP': {
        f'top-{k}': {
            'mAP_p': mAP_p[k],
            'mAP_t': mAP_t[k]
        } for k in topk
    }
}
log_metrics(metrics, epoch, ex, writer)
```

基于Web的可视化

Tensorboard能够满足机器学习最常见的可视化需求，但对于特定任务而言，可能需要定制化的可视化功能。这里给出一个基于本地html+服务器flask服务的方案。



流程：

(1) 保存实验测试集上的详细结果（.npy, .npz）

(2) 基于结果生成html，实现结果的可视化

(3) 通过html的按钮、js脚本等发送Web请求

(4) 服务器端基于flask接收请求并响应，包含读取或生成的数据

```
from flask import Flask, Response
# ......

app = Flask(__name__)

@app.route("/show/<int:video_i>/<int:f_id>")
def send_img(video_i, f_id):
    filename = os.path.join('../', data_dir, test_video_names[video_i], '{0:06d}.png'.format(f_id))
    f = open(filename, 'rb')
    resp = Response(f, mimetype="image/jpeg")
    return resp
```

Q&A