视觉神经网络模型优秀开源工作：timm库使用方法和最新代码解读

原创 CV开发者都爱看的 极市平台 2021-09-01 22:00:00 手机阅读

收录于话题
#神经网络结构设计

作者丨科技猛兽
编辑丨极市平台

极市导读

timm是一个非常优秀的神经网络开源模型库，并且一直处在更新维护当中，本文是对timm库代码的最新解读，不只局限于视觉transformer模型。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

1 什么是timm库?

PyTorchImageModels，简称timm，是一个巨大的**PyTorch**代码集合，包括了一系列：

- **image models**
- **layers**
- **utilities**
- **optimizers**
- **schedulers**
- **data-loaders / augmentations**
- **training / validation scripts**

旨在将各种SOTA模型整合在一起，并具有复现ImageNet训练结果的能力。

作者：Ross Wightman， 来自加拿大温哥华。首先致敬大佬！

> 作者github链接：https://github.com/rwightman

> timm库链接：https://github.com/rwightman/pytorch-image-models

> 作者官方指南：https://rwightman.github.io/pytorch-image-models/

timm库实现了**最新的**几乎**所有的具有影响力**的**视觉**模型，它不仅提供了模型的权重，还提供了一个很棒的**分布式训练**和**评估**的**代码框架**，方便后人开发。更难能可贵的是它还在**不断地更新**迭代**新的训练方法，新的视觉模型**和**优化代码**。

但是毫无疑问，训练、测试和维护这些代码库和模型权重需要大量的 GPU (或 TPU) 资源和大量的电力/冷却费用。Ross Wightman 也确实需要额外的资源来提供和训练更多具有更好技术的模型，所以作者打了广告邀请各界人士赞助 orz。

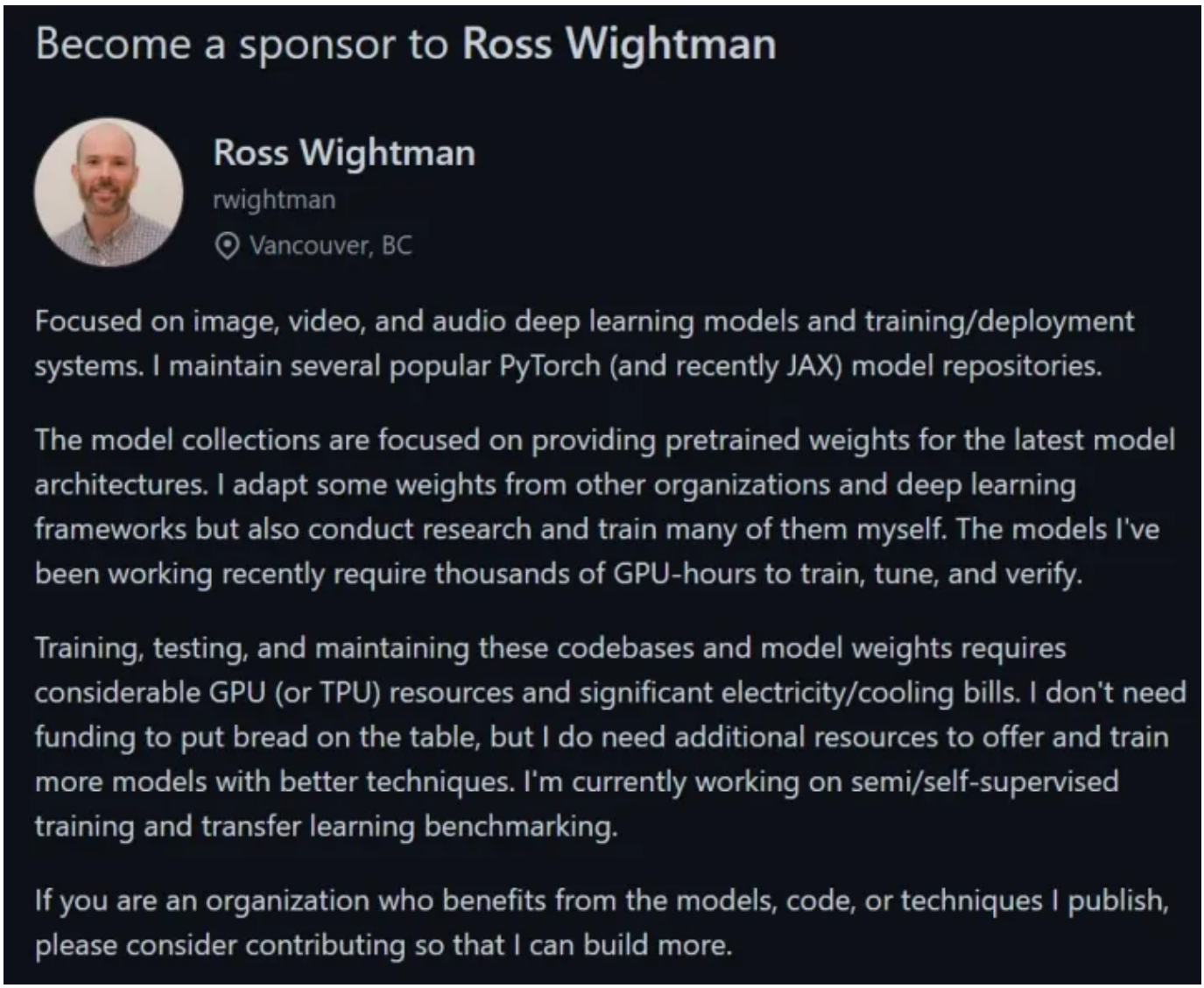

图1：作者的邀请赞助广告

我在2021年3月写了 timm 库的 vision transformer 的代码解读，链接如下。

用Pytorch轻松实现28个视觉Transformer，开源库 timm 了解一下！（附代码解读）

本文写于2021年9月，这半年期间 timm 库又做了**诸多模型的更新和代码的优化**。比如：

- **CNN模型：**

添加了经典的 NFNet，RegNet，TResNet，Lambda Networks，GhostNet，ByoaNet 等以及 TResNet, MobileNet-V3, ViT 的 ImageNet-21k 训练的权重，EfficientNet-V2 ImageNet-1k，ImageNet-21k 训练的权重。

- **Transformer模型：**

添加了经典的 TNT，Swin Transformer，PiT，Bottleneck Transformers，Halo Nets，CoaT，CaiT，LeViT, Visformer, Con ViT，Twins，BiT 等。

- **MLP模型：**

添加了经典的 MLP-Mixer，ResMLP，gMLP等。

- **优化器层面：**

更新了Adabelief optimizer等。

所以本文是对 timm 库代码的**最新解读**，不只限于视觉 transformer 模型。

所有的PyTorch模型及其对应arxiv链接如下：

- Aggregating Nested Transformers - https://arxiv.org/abs/2105.12723
- Big Transfer ResNetV2 (BiT) - https://arxiv.org/abs/1912.11370
- Bottleneck Transformers - https://arxiv.org/abs/2101.11605
- CaiT (Class-Attention in Image Transformers) - https://arxiv.org/abs/2103.17239
- CoaT (Co-Scale Conv-Attentional Image Transformers) - https://arxiv.org/abs/2104.06399
- ConViT (Soft Convolutional Inductive Biases Vision Transformers)- https://arxiv.org/abs/2103.10697
- CspNet (Cross-Stage Partial Networks) - https://arxiv.org/abs/1911.11929
- DeiT (Vision Transformer) - https://arxiv.org/abs/2012.12877
- DenseNet - https://arxiv.org/abs/1608.06993
- DLA - https://arxiv.org/abs/1707.06484
- DPN (Dual-Path Network) - https://arxiv.org/abs/1707.01629
- EfficientNet (MBConvNet Family)
  - EfficientNet NoisyStudent (B0-B7, L2) - https://arxiv.org/abs/1911.04252
  - EfficientNet AdvProp (B0-B8) - https://arxiv.org/abs/1911.09665
  - EfficientNet (B0-B7) - https://arxiv.org/abs/1905.11946
  - EfficientNet-EdgeTPU (S, M, L) - https://ai.googleblog.com/2019/08/efficientnet-edgetpu-creating.html
  - EfficientNet V2 - https://arxiv.org/abs/2104.00298
  - FBNet-C - https://arxiv.org/abs/1812.03443

    - MixNet - https://arxiv.org/abs/1907.09595
    - MNASNet B1, A1 (Squeeze-Excite), and Small - https://arxiv.org/abs/1807.11626
    - MobileNet-V2 - https://arxiv.org/abs/1801.04381
    - Single-Path NAS - https://arxiv.org/abs/1904.02877
- GhostNet - https://arxiv.org/abs/1911.11907
- gMLP - https://arxiv.org/abs/2105.08050
- GPU-Efficient Networks - https://arxiv.org/abs/2006.14090
- Halo Nets - https://arxiv.org/abs/2103.12731
- HardCoRe-NAS - https://arxiv.org/abs/2102.11646
- HRNet - https://arxiv.org/abs/1908.07919
- Inception-V3 - https://arxiv.org/abs/1512.00567
- Inception-ResNet-V2 and Inception-V4 - https://arxiv.org/abs/1602.07261
- Lambda Networks - https://arxiv.org/abs/2102.08602
- LeViT (Vision Transformer in ConvNet's Clothing) - https://arxiv.org/abs/2104.01136
- MLP-Mixer - https://arxiv.org/abs/2105.01601
- MobileNet-V3 (MBConvNet w/ Efficient Head) - https://arxiv.org/abs/1905.02244
- NASNet-A - https://arxiv.org/abs/1707.07012
- NFNet-F - https://arxiv.org/abs/2102.06171
- NF-RegNet / NF-ResNet - https://arxiv.org/abs/2101.08692
- PNasNet - https://arxiv.org/abs/1712.00559
- Pooling-based Vision Transformer (PiT) - https://arxiv.org/abs/2103.16302
- RegNet - https://arxiv.org/abs/2003.13678

- RepVGG - https://arxiv.org/abs/2101.03697
- ResMLP - https://arxiv.org/abs/2105.03404
- ResNet/ResNeXt
  - ResNet (v1b/v1.5) - https://arxiv.org/abs/1512.03385
  - ResNeXt - https://arxiv.org/abs/1611.05431
  - 'Bag of Tricks' / Gluon C, D, E, S variations - https://arxiv.org/abs/1812.01187
  - Weakly-supervised (WSL) Instagram pretrained / ImageNet tuned ResNeXt101 - https://arxiv.org/abs/1805.00932
  - Semi-supervised (SSL) / Semi-weakly Supervised (SWSL) ResNet/ResNeXts - https://arxiv.org/abs/1905.00546
  - ECA-Net (ECAResNet) - https://arxiv.org/abs/1910.03151v4
  - Squeeze-and-Excitation Networks (SEResNet) - https://arxiv.org/abs/1709.01507
  - ResNet-RS - https://arxiv.org/abs/2103.07579
- Res2Net - https://arxiv.org/abs/1904.01169
- ResNeSt - https://arxiv.org/abs/2004.08955
- ReXNet - https://arxiv.org/abs/2007.00992
- SelecSLS - https://arxiv.org/abs/1907.00837
- Selective Kernel Networks - https://arxiv.org/abs/1903.06586
- Swin Transformer - https://arxiv.org/abs/2103.14030
- Transformer-iN-Transformer (TNT) - https://arxiv.org/abs/2103.00112
- TResNet - https://arxiv.org/abs/2003.13630
- Twins (Spatial Attention in Vision Transformers) - https://arxiv.org/pdf/2104.13840.pdf
- Vision Transformer - https://arxiv.org/abs/2010.11929
- VovNet V2 and V1 - https://arxiv.org/abs/1911.06667
- Xception - https://arxiv.org/abs/1610.02357

- Xception (Modified Aligned, Gluon) - https://arxiv.org/abs/1802.02611
- Xception (Modified Aligned, TF) -https://arxiv.org/abs/1802.02611
- XCiT (Cross-Covariance Image Transformers) - https://arxiv.org/abs/2106.09681

2 使用教程

2.1 开始使用 timm

安装库 (Python3, PyTorch version 1.4+)：

```
pip install timm
```

加载你需要的预训练模型权重：

```
import timm

m = timm.create_model('mobilenetv3_large_100', pretrained=True)
m.eval()
```

加载所有的预训练模型列表 (pprint 是美化打印的标准库)：

```
import timm
from pprint import pprint
model_names = timm.list_models(pretrained=True)
pprint(model_names)
>>> ['adv_inception_v3',
 'cspdarknet53',
 'cspresnext50',
 'densenet121',
 'densenet161',
 'densenet169',
 'densenet201',
 'densenetblur121d',
 'dla34',
 'dla46_c',
...
]
```

利用通配符加载所有的预训练模型列表：

```
import timm
from pprint import pprint
model_names = timm.list_models('*resne*t*')
pprint(model_names)
>>> ['cspresnet50',
 'cspresnet50d',
 'cspresnet50w',
 'cspresnext50',
...
]
```

2.2 支持的全部模型列表和相关论文链接以及官方代码实现

https://rwightman.github.io/pytorch-image-models/models/

2.3 如何使用某个模型

这里以著名的 **MobileNet v3** 为例。MobileNetV3 是一种卷积神经网络，专为手机 CPU 设计。网络设计包括在 MBConv 块中使用 hard swish activation 激活函数和 squeeze-and-excitation 模块。

hard swish activation：https://paperswithcode.com/method/hard-swish

squeeze-and-excitation：https://paperswithcode.com/method/squeeze-and-excitation-block

- **加载 MobileNet v3 预训练模型：**

```
import timm
model = timm.create_model('mobilenetv3_large_100', pretrained=True)
model.eval()
```

- **加载图片和预处理：**

```
import urllib
from PIL import Image
from timm.data import resolve_data_config
from timm.data.transforms_factory import create_transform

config = resolve_data_config({}, model=model)
transform = create_transform(**config)

url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
urllib.request.urlretrieve(url, filename)
img = Image.open(filename).convert('RGB')
tensor = transform(img).unsqueeze(0) # transform and add batch dimension
```

- **获取模型预测结果：**

```
import torch
with torch.no_grad():
    out = model(tensor)
probabilities = torch.nn.functional.softmax(out[0], dim=0)
print(probabilities.shape)
# prints: torch.Size([1000])
```

- **获取预测前5名的类名称：**

```
# Get imagenet class mappings
url, filename = ("https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt", "imagenet_classes.txt")
urllib.request.urlretrieve(url, filename)
with open("imagenet_classes.txt", "r") as f:
    categories = [s.strip() for s in f.readlines()]

# Print top categories per image
top5_prob, top5_catid = torch.topk(probabilities, 5)
for i in range(top5_prob.size(0)):
    print(categories[top5_catid[i]], top5_prob[i].item())
# prints class names and probabilities like:
# [('Samoyed', 0.6425196528434753), ('Pomeranian', 0.04062102362513542), ('keeshond', 0.03186424449086189), ('white wolf', (
```

2.4 timm库所有模型在 ImageNet 数据集结果汇总

https://rwightman.github.io/pytorch-image-models/results/

2.5 开始训练你的模型

对于训练数据集文件夹，指定包含 train 和 validation 的基础文件夹。

- **想训练一个 SE-ResNet34 在 ImageNet 数据集，4 GPUs，分布式训练，使用 cosine 的 learning rate schedule，命令为：**

```
./distributed_train.sh 4 /data/imagenet \--model seresnet34 \--sched cosine \--epochs 150 \--warmup-epochs 5 \--lr 0.4 \--reprob 0.5 \--remode pixel \--batch-size 256 \--amp \-j 4
```

> 注： `--amp` 默认使用 native AMP。--apex-amp 将强制使用 Apex 组件。

2.6 一些训练示例

- **想训练 EfficientNet-B2 with RandAugment - 80.4 top-1, 95.1 top-5：**

These params are for dual Titan RTX cards with NVIDIA Apex installed:

```
./distributed_train.sh 2 /imagenet/ \--model efficientnet_b2 \-b 128 \--sched step \--epochs 450 \--decay-epochs 2.4 \--decay-rate .97 \--opt rmsproptf \--opt-eps .001 \-j 8 \--warmup-lr 1e-6 \--weight-decay 1e-5 \--drop 0.3 \--drop-connect 0.2 \--model-ema \--model-ema-decay 0.9999 \--aa rand-m9-mstd0.5 \--remode pixel \--reprob 0.2 \--amp \--lr .016
```

- **想训练 MixNet-XL with RandAugment - 80.5 top-1, 94.9 top-5：**

This params are for dual Titan RTX cards with NVIDIA Apex installed:

```
./distributed_train.sh 2 /imagenet/ \--model mixnet_xl \-b 128 \--sched step \--epochs 450 \--decay-epochs 2.4 \--decay-rate .969 \--opt rmsproptf \--opt-eps .001 \-j 8 \--warmup-lr 1e-6 \--weight-decay 1e-5 \--drop 0.3 \--drop-connect 0.2 \--model-ema \--model-ema-decay 0.9999 \--aa rand-m9-mstd0.5 \--remode pixel \--reprob 0.3 \--amp \--lr .016 \--dist-bn reduce
```

- **想训练 SE-ResNeXt-26-D and SE-ResNeXt-26-T：**

These hparams (or similar) work well for a wide range of ResNet architecture, generally a good idea to increase the epoch # as the model size increases... ie approx 180-200 for ResNe(X)t50, and 220+ for larger. Increase batch size and LR proportionally for better GPUs or with AMP enabled. These params were for 2 1080Ti cards:

```
./distributed_train.sh 2 /imagenet/ \--model seresnext26t_32x4d \--lr 0.1 \--warmup-epochs 5 \--epochs 160 \--weight-decay 1e-4 \--sched cosine \--reprob 0.4 \--remode pixel \-b 112
```

- **想训练 EfficientNet-B3 with RandAugment - 81.5 top-1, 95.7 top-5：**

The training of this model started with the same command line as EfficientNet-B2 w/ RA above. After almost three weeks of training the process crashed. The results weren't looking amazing so I resumed the training several times with tweaks to a few params (increase RE prob, decrease rand-aug, increase ema-decay). Nothing looked great. I ended up averaging the best checkpoints from all restarts. The result is mediocre at default res/crop but oddly performs much better with a full image test crop of 1.0.

- **想训练 EfficientNet-B0 with RandAugment - 77.7 top-1, 95.3 top-5：**

Michael Klachko achieved these results with the command line for B2 adapted for larger batch size, with the recommended B0 dropout rate of 0.2.

```
./distributed_train.sh 2 /imagenet/ \--model efficientnet_b0 \-b 384 \--sched step \--epochs 450 \--decay-epochs 2.4 \--decay-rate .97 \--opt rmsproptf \--opt-eps .001 \-j 8 \--warmup-lr 1e-6 \--weight-decay 1e-5 \--drop 0.2 \--drop-connect 0.2 \--model-ema \--model-ema-decay 0.9999 \--aa rand-m9-mstd0.5 \--remode pixel \--reprob 0.2 \--amp \--lr .048
```

- **想训练 ResNet50 with JSD loss and RandAugment (clean + 2x RA augs) - 79.04 top-1, 94.39 top-5：**

```
./distributed_train.sh 2 /imagenet \-b 64 \--model resnet50 \--sched cosine \--epochs 200 \--lr 0.05 \--amp \--remode pixel \--reprob 0.6 \--aug-splits 3 \--aa rand-m9-mstd0.5-inc1 \--resplit \--split-bn \--jsd \--dist-bn reduce
```

- **想训练 EfficientNet-ES (EdgeTPU-Small) with RandAugment - 78.066 top-1, 93.926 top-5**

```
./distributed_train.sh 8 /imagenet \--model efficientnet_es \-b 128 \--sched step \--epochs 450 \--decay-epochs 2.4 \--decay-rate .97 \--opt rmsproptf \--opt-eps .001 \-j 8 \--warmup-lr 1e-6 \--weight-decay 1e-5 \--drop 0.2 \--drop-connect 0.2 \--aa rand-m9-mstd0.5 \--remode pixel \--reprob 0.2 \--amp \--lr .064
```

- **想训练 MobileNetV3-Large-100 - 75.766 top-1, 92,542 top-5：**

```
./distributed_train.sh 2 /imagenet/ \--model mobilenetv3_large_100 \-b 512 \--sched step \--epochs 600 \--decay-epochs 2.4 \--decay-rate .973 \--opt rmsproptf \--opt-eps .001 \-j 7 \--warmup-lr 1e-6 \--weight-decay 1e-5 \--drop 0.2 \--drop-connect 0.2 \--model-ema \--model-ema-decay 0.9999 \--aa rand-m9-mstd0.5 \--remode pixel \--reprob 0.2 \--amp \--lr .064 \--lr-noise 0.42 0.9
```

- **想训练 ResNeXt-50 32x4d w/ RandAugment - 79.762 top-1, 94.60 top-5：**.

```
./distributed_train.sh 8 /imagenet \--model resnext50_32x4d \--lr 0.6 \--warmup-epochs 5 \--epochs 240 \--weight-decay 1e-4 \--sched cosine \--reprob 0.4 \--recount 3 \--remode pixel \--aa rand-m7-mstd0.5-inc1 \-b 192 \-j 6 \--amp \--dist-bn reduce
```

2.7 验证/推理你的模型

对于验证集文件夹，指定在 validation 的文件夹位置。

- **验证带有预训练权重的模型：**

```
python validate.py /imagenet/validation/ \--model seresnext26_32x4d \--pretrained
```

- **根据给定的 checkpoint 作前向推理：**

```
python inference.py /imagenet/validation/ \--model mobilenetv3_large_100 \--checkpoint ./output/train/model_best.pth.tar
```

2.8 特征提取

timm 中的所有模型都可以从模型中获取各种类型的特征，用于除分类之外的任务。

- **获取 Penultimate Layer Features：**

Penultimate Layer Features的中文含义是 "倒数第2层的特征"，即 classifier 之前的特征。timm 库可以通过多种方式获得倒数第二个模型层的特征，而无需进行模型的手术。

```
import torch
import timm
m = timm.create_model('resnet50', pretrained=True, num_classes=0)
o = m(torch.randn(2, 3, 224, 224))
print(f'Pooled shape: {o.shape}')
```

输出：

```
Pooled shape: torch.Size([2, 2048])
```

- **获取分类器之后的特征：**

```
import torch
import timm
m = timm.create_model('ese_vovnet19b_dw', pretrained=True)
o = m(torch.randn(2, 3, 224, 224))
print(f'Original shape: {o.shape}')
m.reset_classifier(0)
o = m(torch.randn(2, 3, 224, 224))
print(f'Pooled shape: {o.shape}')
```

输出：

```
Pooled shape: torch.Size([2, 1024])
```

- **输出多尺度特征：**

默认情况下，大多数模型将输出 5 个stride (并非所有模型都有那么多)，第一个从 stride = 2 开始 (有些从 1 或 4 开始)。

```
import torch
import timm
m = timm.create_model('resnest26d', features_only=True, pretrained=True)
o = m(torch.randn(2, 3, 224, 224))
for x in o:
  print(x.shape)
```

输出：

```
torch.Size([2, 64, 112, 112])
torch.Size([2, 256, 56, 56])
torch.Size([2, 512, 28, 28])
torch.Size([2, 1024, 14, 14])
torch.Size([2, 2048, 7, 7])
```

- **.feature_info 属性是一个封装了特征提取信息的类：**

> 比如这个例子输出各个特征的通道数：

```
import torch
import timm
m = timm.create_model('regnety_032', features_only=True, pretrained=True)
print(f'Feature channels: {m.feature_info.channels()}')
```

```
o = m(torch.randn(2, 3, 224, 224))
for x in o:
  print(x.shape)
```

输出：

```
Feature channels: [32, 72, 216, 576, 1512]
torch.Size([2, 32, 112, 112])
torch.Size([2, 72, 56, 56])
torch.Size([2, 216, 28, 28])
torch.Size([2, 576, 14, 14])
torch.Size([2, 1512, 7, 7])
```

- **选择特定的 feature level 或限制 stride：**

> out_indices：指定输出特征的索引 (实际是指定通道数)。
> output_stride：指定输出特征的 stride 值，通过将特征进行 dilated convolution 得到。

```
import torch
import timm
m = timm.create_model('ecaresnet101d', features_only=True, output_stride=8, out_indices=(2, 4), pretrained=True)
print(f'Feature channels: {m.feature_info.channels()}')
print(f'Feature reduction: {m.feature_info.reduction()}')
o = m(torch.randn(2, 3, 320, 320))
for x in o:
  print(x.shape)
```

输出：

```
Feature channels: [512, 2048]
Feature reduction: [8, 8]
torch.Size([2, 512, 40, 40])
torch.Size([2, 2048, 40, 40])
```

> 这个例子里面 out_indices=8，代表输出 stride=8 的特征。out_indices=(2,4) 代表输出特征的索引是2和4，即channel数分别是512和2048。

3 代码解读

以上就是读者在使用 timm 库时的基本方法，其实到这里你应该已经能够使用它训练自己的分类模型了。但是如果还想进一步搞清楚它的框架原理，并在它的基础上做修改，本节可能会帮到你。

3.1 创建dataset

timm 库通过 create_dataset 函数来得到 dataset_train 和 dataset_eval 这两个 dataset 类。

```
dataset_train = create_dataset(
    args.dataset,
    root=args.data_dir, split=args.train_split, is_training=True,
    batch_size=args.batch_size, repeats=args.epoch_repeats)
dataset_eval = create_dataset(
    args.dataset, root=args.data_dir, split=args.val_split, is_training=False, batch_size=args.batch_size)
```

create_dataset 函数如下面所示，实际返回的是：ImageDataset(root, parser=name, **kwargs)

```
def create_dataset(name, root, split='validation', search_split=True, is_training=False, batch_size=None, **kwargs):
    name = name.lower()
    if name.startswith('tfds'):
        ds = IterableImageDataset(
            root, parser=name, split=split, is_training=is_training, batch_size=batch_size, **kwargs)

    else:
        # FIXME support more advance split cfg for ImageFolder/Tar datasets in the future
        kwargs.pop('repeats', 0)  # FIXME currently only Iterable dataset support the repeat multiplier
        if search_split and os.path.isdir(root):
            root = _search_split(root, split)
        ds = ImageDataset(root, parser=name, **kwargs)
    return ds
```

ImageDataset 类如下所示，它的内部定义了最关键的 __getitem__ 函数。

```
class ImageDataset(data.Dataset):

    def __init__(
            self,
            root,
            parser=None,
            class_map='',
            load_bytes=False,
            transform=None,
    ):
        if parser is None or isinstance(parser, str):
            parser = create_parser(parser or '', root=root, class_map=class_map)
        self.parser = parser
        self.load_bytes = load_bytes
        self.transform = transform
        self._consecutive_errors = 0

    def __getitem__(self, index):
        img, target = self.parser[index]
        try:
            img = img.read() if self.load_bytes else Image.open(img).convert('RGB')
        except Exception as e:
            _logger.warning(f'Skipped sample (index {index}, file {self.parser.filename(index)}). {str(e)}')
            self._consecutive_errors += 1
            if self._consecutive_errors < _ERROR_RETRY:
                return self.__getitem__((index + 1) % len(self.parser))
            else:
                raise e
        self._consecutive_errors = 0
        if self.transform is not None:
            img = self.transform(img)
        if target is None:
            target = torch.tensor(-1, dtype=torch.long)
        return img, target

    def __len__(self):
        return len(self.parser)

    def filename(self, index, basename=False, absolute=False):
        return self.parser.filename(index, basename, absolute)

    def filenames(self, basename=False, absolute=False):
        return self.parser.filenames(basename, absolute)
```

那么这个函数里面最关键的一句是：

img, target = self.parser[index]

而这里的 parser 来自于parser = create_parser(parser or '', root=root, class_map=class_map)，所以有必要看看这个 create_parser 函数。

create_parser 函数的定义如下所示，最后返回的 parser 来自：parser = ParserImageFolder(root, **kwargs)

```
def create_parser(name, root, split='train', **kwargs):
    name = name.lower()
    name = name.split('/', 2)
    prefix = ''
    if len(name) > 1:
        prefix = name[0]
    name = name[-1]

    # FIXME improve the selection right now just tfds prefix or fallback path, will need options to
    # explicitly select other options shortly
    if prefix == 'tfds':
        from .parser_tfds import ParserTfds  # defer tensorflow import
        parser = ParserTfds(root, name, split=split, shuffle=kwargs.pop('shuffle', False), **kwargs)
    else:
        assert os.path.exists(root)
        # default fallback path (backwards compat), use image tar if root is a .tar file, otherwise image folder
        # FIXME support split here, in parser?
        if os.path.isfile(root) and os.path.splitext(root)[1] == '.tar':
            parser = ParserImageInTar(root, **kwargs)
        else:
            parser = ParserImageFolder(root, **kwargs)
    return parser
```

所以有必要看看这个 ParserImageFolder 函数。

```
class ParserImageFolder(Parser):

    def __init__(
            self,
            root,
            class_map=''):
        super().__init__()

        self.root = root
        class_to_idx = None
        if class_map:
            class_to_idx = load_class_map(class_map, root)
        self.samples, self.class_to_idx = find_images_and_targets(root, class_to_idx=class_to_idx)
        if len(self.samples) == 0:
            raise RuntimeError(
                f'Found 0 images in subfolders of {root}. Supported image extensions are {", ".join(IMG_EXTENSIONS)}')

    def __getitem__(self, index):
        path, target = self.samples[index]
        return open(path, 'rb'), target

    def __len__(self):
        return len(self.samples)

    def _filename(self, index, basename=False, absolute=False):
        filename = self.samples[index][0]
        if basename:
            filename = os.path.basename(filename)
        elif not absolute:
            filename = os.path.relpath(filename, self.root)
        return filename
```

ParserImageFolder 函数通过 self.samples, self.class_to_idx = find_images_and_targets(root, class_to_idx=class_to_idx) 来找到所有的 samples 的类别(0-1000) 和一个类名映射索引的 class_to_idx 表。

然后直接通过 path, target = self.samples[index] 找到某个索引的图片路径和类索引 (0-1000)。

所以说 img, target = self.parser[index] 的返回值其实就是 ParserImageFolder 类的 __getitem__ 函数的返回值，即：**某个索引的图片路径和类索引 (0-1000)**。也就是 dataset 的功能。

3.2 构建dataloader

timm 库通过 create_loader 函数来创建dataloader，需要传入上一步构建的 dataset_train。

```
loader_train = create_loader(
    dataset_train,
    input_size=data_config['input_size'],
    batch_size=args.batch_size,
    is_training=True,
    use_prefetcher=args.prefetcher,
    no_aug=args.no_aug,
    re_prob=args.reprob,
    re_mode=args.remode,
    re_count=args.recount,
    re_split=args.resplit,
    scale=args.scale,
    ratio=args.ratio,
    hflip=args.hflip,
    vflip=args.vflip,
    color_jitter=args.color_jitter,
    auto_augment=args.aa,
    num_aug_splits=num_aug_splits,
    interpolation=train_interpolation,
    mean=data_config['mean'],
    std=data_config['std'],
    num_workers=args.workers,
    distributed=args.distributed,
    collate_fn=collate_fn,
    pin_memory=args.pin_mem,
    use_multi_epochs_loader=args.use_multi_epochs_loader
)
```

create_loader 函数内部通过：

```
loader_class = torch.utils.data.DataLoader
```

得到 loader_class，再通过下面的语句建立 DataLoader (需要的参数 batch_size, shuffle, num_workers, sampler, collate_fn, drop_last 等等都以字典的形式保存在 loader_args 中)：

```
loader_args = dict(
    batch_size=batch_size,
    shuffle=not isinstance(dataset, torch.utils.data.IterableDataset) and sampler is None and is_training,
    num_workers=num_workers,
    sampler=sampler,
    collate_fn=collate_fn,
    pin_memory=pin_memory,
    drop_last=is_training,
    persistent_workers=persistent_workers)
try:
    loader = loader_class(dataset, **loader_args)
```

最后返回 loader。

3.3 创建模型

timm 库通过 create_model 函数来创建模型。

```
    model = create_model(
        args.model,
        pretrained=args.pretrained,
        num_classes=args.num_classes,
        drop_rate=args.drop,
        drop_connect_rate=args.drop_connect,  # DEPRECATED, use drop_path
        drop_path_rate=args.drop_path,
        drop_block_rate=args.drop_block,
        global_pool=args.gp,
        bn_tf=args.bn_tf,
        bn_momentum=args.bn_momentum,
        bn_eps=args.bn_eps,
        scriptable=args.torchscript,
        checkpoint_path=args.initial_checkpoint)
```

函数 create_model 的具体实现是：

```
def create_model(
        model_name,
        pretrained=False,
        checkpoint_path='',
        scriptable=None,
        exportable=None,
        no_jit=None,
        **kwargs):
    """Create a model

    Args:
        model_name (str): name of model to instantiate
        pretrained (bool): load pretrained ImageNet-1k weights if true
        checkpoint_path (str): path of checkpoint to load after model is initialized
        scriptable (bool): set layer config so that model is jit scriptable (not working for all models yet)
        exportable (bool): set layer config so that model is traceable / ONNX exportable (not fully impl/obeyed yet)
        no_jit (bool): set layer config so that model doesn't utilize jit scripted layers (so far activations only)

    Keyword Args:
        drop_rate (float): dropout rate for training (default: 0.0)
        global_pool (str): global pool type (default: 'avg')
        **: other kwargs are model specific
    """
    source_name, model_name = split_model_name(model_name)

    # Only EfficientNet and MobileNetV3 models have support for batchnorm params or drop_connect_rate passed as args
    is_efficientnet = is_model_in_modules(model_name, ['efficientnet', 'mobilenetv3'])
    if not is_efficientnet:
        kwargs.pop('bn_tf', None)
        kwargs.pop('bn_momentum', None)
        kwargs.pop('bn_eps', None)

    # handle backwards compat with drop_connect -> drop_path change
    drop_connect_rate = kwargs.pop('drop_connect_rate', None)
    if drop_connect_rate is not None and kwargs.get('drop_path_rate', None) is None:
        print("WARNING: 'drop_connect' as an argument is deprecated, please use 'drop_path'."
              " Setting drop_path to %f." % drop_connect_rate)
        kwargs['drop_path_rate'] = drop_connect_rate

    # Parameters that aren't supported by all models or are intended to only override model defaults if set
    # should default to None in command line args/cfg. Remove them if they are present and not set so that
    # non-supporting models don't break and default args remain in effect.
    kwargs = {k: v for k, v in kwargs.items() if v is not None}

    if source_name == 'hf_hub':
        # For model names specified in the form `hf_hub:path/architecture_name#revision`,
        # load model weights + default_cfg from Hugging Face hub.
        hf_default_cfg, model_name = load_model_config_from_hf(model_name)
        kwargs['external_default_cfg'] = hf_default_cfg  # FIXME revamp default_cfg interface someday

    if is_model(model_name):
        create_fn = model_entrypoint(model_name)
```

```
    else:
        raise RuntimeError('Unknown model (%s)' % model_name)

    with set_layer_config(scriptable=scriptable, exportable=exportable, no_jit=no_jit):
        model = create_fn(pretrained=pretrained, **kwargs)

    if checkpoint_path:
        load_checkpoint(model, checkpoint_path)

    return model
```

timm 库每次新定义一个模型，都类似于这样的形式 (这里以 vit_base_patch32_384 为例)：

```
@register_model
def vit_base_patch32_384(pretrained=False, **kwargs):
    """ ViT-Base model (ViT-B/32) from original paper (https://arxiv.org/abs/2010.11929).
    ImageNet-1k weights fine-tuned from in21k @ 384x384, source https://github.com/google-research/vision_transformer.
    """
    model_kwargs = dict(patch_size=32, embed_dim=768, depth=12, num_heads=12, **kwargs)
    model = _create_vision_transformer('vit_base_patch32_384', pretrained=pretrained, **model_kwargs)
    return model
```

这里的 register_model 来自 register.py 文件的 register_model 函数，如下。

register_model 函数的输入是 fn，也就是例子里面的 **vit_base_patch32_384**。register_model 函数的功能是把这个模型的函数的信息存到 _model_to_module 和 _model_entrypoints 等等的字典里面，相当于把 **vit_base_patch32_384** 给注册一下。

```
def register_model(fn):
    # lookup containing module
    mod = sys.modules[fn.__module__]
    module_name_split = fn.__module__.split('.')
    module_name = module_name_split[-1] if len(module_name_split) else ''

    # add model to __all__ in module
    model_name = fn.__name__
    if hasattr(mod, '__all__'):
        mod.__all__.append(model_name)
    else:
        mod.__all__ = [model_name]

    # add entries to registry dict/sets
    _model_entrypoints[model_name] = fn
    _model_to_module[model_name] = module_name
    _module_to_models[module_name].add(model_name)
    has_pretrained = False  # check if model has a pretrained url to allow filtering on this
    if hasattr(mod, 'default_cfgs') and model_name in mod.default_cfgs:
        # this will catch all models that have entrypoint matching cfg key, but miss any aliasing
        # entrypoints or non-matching combos
        has_pretrained = 'url' in mod.default_cfgs[model_name] and 'http' in mod.default_cfgs[model_name]['url']
        _model_default_cfgs[model_name] = deepcopy(mod.default_cfgs[model_name])
    if has_pretrained:
        _model_has_pretrained.add(model_name)
    return fn
```

注册完以后，通过 create_model 函数中的 create_fn = model_entrypoint(model_name) 语句得到 **vit_base_patch32_384** 函数。所以 **create_fn() 就相当于是 vit_base_patch32_384 ()**。

最后就是使用 create_fn 函数得到模型并返回 model。

```
    with set_layer_config(scriptable=scriptable, exportable=exportable, no_jit=no_jit):
        model = create_fn(pretrained=pretrained, **kwargs)
```

```
    if checkpoint_path:
        load_checkpoint(model, checkpoint_path)

    return model
```

3.4 构建优化器

timm 库通过 create_optimizer_v2 函数来构建优化器。

```
optimizer = create_optimizer_v2(model, **optimizer_kwargs(cfg=args))
```

create_optimizer_v2 函数的具体实现如下，需要传入的参数是：模型参数，优化器类型，出示学习率，weight_decay 等等。之后通过 opt_lower 的选择来构建不同类型的优化器。

```
def create_optimizer_v2(
        model_or_params,
        opt: str = 'sgd',
        lr: Optional[float] = None,
        weight_decay: float = 0.,
        momentum: float = 0.9,
        filter_bias_and_bn: bool = True,
        **kwargs):
    """ Create an optimizer.

    TODO currently the model is passed in and all parameters are selected for optimization.
    For more general use an interface that allows selection of parameters to optimize and lr groups, one of:
      * a filter fn interface that further breaks params into groups in a weight_decay compatible fashion
      * expose the parameters interface and leave it up to caller

    Args:
        model_or_params (nn.Module): model containing parameters to optimize
        opt: name of optimizer to create
        lr: initial learning rate
        weight_decay: weight decay to apply in optimizer
        momentum:  momentum for momentum based optimizers (others may use betas via kwargs)
        filter_bias_and_bn:  filter out bias, bn and other 1d params from weight decay
        **kwargs: extra optimizer specific kwargs to pass through

    Returns:
        Optimizer
    """
    if isinstance(model_or_params, nn.Module):
        # a model was passed in, extract parameters and add weight decays to appropriate layers
        if weight_decay and filter_bias_and_bn:
            skip = {}
            if hasattr(model_or_params, 'no_weight_decay'):
                skip = model_or_params.no_weight_decay()
            parameters = add_weight_decay(model_or_params, weight_decay, skip)
            weight_decay = 0.
        else:
            parameters = model_or_params.parameters()
    else:
        # iterable of parameters or param groups passed in
        parameters = model_or_params

    opt_lower = opt.lower()
    opt_split = opt_lower.split('_')
    opt_lower = opt_split[-1]
    if 'fused' in opt_lower:
        assert has_apex and torch.cuda.is_available(), 'APEX and CUDA required for fused optimizers'

    opt_args = dict(weight_decay=weight_decay, **kwargs)
    if lr is not None:
        opt_args.setdefault('lr', lr)

    # basic SGD & related
```

```
    if opt_lower == 'sgd' or opt_lower == 'nesterov':
        # NOTE 'sgd' refers to SGD + nesterov momentum for legacy / backwards compat reasons
        opt_args.pop('eps', None)
        optimizer = optim.SGD(parameters, momentum=momentum, nesterov=True, **opt_args)
    elif opt_lower == 'momentum':
        opt_args.pop('eps', None)
        optimizer = optim.SGD(parameters, momentum=momentum, nesterov=False, **opt_args)
    elif opt_lower == 'sgdp':
        optimizer = SGDP(parameters, momentum=momentum, nesterov=True, **opt_args)

    # adaptive
    elif opt_lower == 'adam':
        optimizer = optim.Adam(parameters, **opt_args)
    elif opt_lower == 'adamw':
        optimizer = optim.AdamW(parameters, **opt_args)
    elif opt_lower == 'adamp':
        optimizer = AdamP(parameters, wd_ratio=0.01, nesterov=True, **opt_args)
    elif opt_lower == 'nadam':
        try:
            # NOTE PyTorch >= 1.10 should have native NAdam
            optimizer = optim.Nadam(parameters, **opt_args)
        except AttributeError:
            optimizer = Nadam(parameters, **opt_args)
    elif opt_lower == 'radam':
        optimizer = RAdam(parameters, **opt_args)
    elif opt_lower == 'adamax':
        optimizer = optim.Adamax(parameters, **opt_args)
    elif opt_lower == 'adabelief':
        optimizer = AdaBelief(parameters, rectify=False, **opt_args)
    elif opt_lower == 'radabelief':
        optimizer = AdaBelief(parameters, rectify=True, **opt_args)
    elif opt_lower == 'adadelta':
        optimizer = optim.Adadelta(parameters, **opt_args)
    elif opt_lower == 'adagrad':
        opt_args.setdefault('eps', 1e-8)
        optimizer = optim.Adagrad(parameters, **opt_args)
    elif opt_lower == 'adafactor':
        optimizer = Adafactor(parameters, **opt_args)
    elif opt_lower == 'lamb':
        optimizer = Lamb(parameters, **opt_args)
    elif opt_lower == 'lambc':
        optimizer = Lamb(parameters, trust_clip=True, **opt_args)
    elif opt_lower == 'larc':
        optimizer = Lars(parameters, momentum=momentum, trust_clip=True, **opt_args)
    elif opt_lower == 'lars':
        optimizer = Lars(parameters, momentum=momentum, **opt_args)
    elif opt_lower == 'nlarc':
        optimizer = Lars(parameters, momentum=momentum, trust_clip=True, nesterov=True, **opt_args)
    elif opt_lower == 'nlars':
        optimizer = Lars(parameters, momentum=momentum, nesterov=True, **opt_args)
    elif opt_lower == 'madgrad':
        optimizer = MADGRAD(parameters, momentum=momentum, **opt_args)
    elif opt_lower == 'madgradw':
        optimizer = MADGRAD(parameters, momentum=momentum, decoupled_decay=True, **opt_args)
    elif opt_lower == 'novograd' or opt_lower == 'nvnovograd':
        optimizer = NvNovoGrad(parameters, **opt_args)
    elif opt_lower == 'rmsprop':
        optimizer = optim.RMSprop(parameters, alpha=0.9, momentum=momentum, **opt_args)
    elif opt_lower == 'rmsproptf':
        optimizer = RMSpropTF(parameters, alpha=0.9, momentum=momentum, **opt_args)

    # second order
    elif opt_lower == 'adahessian':
        optimizer = Adahessian(parameters, **opt_args)

    # NVIDIA fused optimizers, require APEX to be installed
    elif opt_lower == 'fusedsgd':
        opt_args.pop('eps', None)
        optimizer = FusedSGD(parameters, momentum=momentum, nesterov=True, **opt_args)
    elif opt_lower == 'fusedmomentum':
        opt_args.pop('eps', None)
        optimizer = FusedSGD(parameters, momentum=momentum, nesterov=False, **opt_args)
```

```
    elif opt_lower == 'fusedadam':
        optimizer = FusedAdam(parameters, adam_w_mode=False, **opt_args)
    elif opt_lower == 'fusedadamw':
        optimizer = FusedAdam(parameters, adam_w_mode=True, **opt_args)
    elif opt_lower == 'fusedlamb':
        optimizer = FusedLAMB(parameters, **opt_args)
    elif opt_lower == 'fusednovograd':
        opt_args.setdefault('betas', (0.95, 0.98))
        optimizer = FusedNovoGrad(parameters, **opt_args)

    else:
        assert False and "Invalid optimizer"
        raise ValueError

    if len(opt_split) > 1:
        if opt_split[0] == 'lookahead':
            optimizer = Lookahead(optimizer)

    return optimizer
```

3.5 构建scheduler

timm 库通过 create_scheduler 函数来构建 scheduler。

```
lr_scheduler, num_epochs = create_scheduler(args, optimizer)
```

内部通过 args.sched 参数控制具体创建什么类型的 scheduler。

3.6 构建训练 engine

timm 库通过 train_one_epoch 函数来构建训练 engine。

```
def train_one_epoch(
        epoch, model, loader, optimizer, loss_fn, args,
        lr_scheduler=None, saver=None, output_dir=None, amp_autocast=suppress,
        loss_scaler=None, model_ema=None, mixup_fn=None):

    if args.mixup_off_epoch and epoch >= args.mixup_off_epoch:
        if args.prefetcher and loader.mixup_enabled:
            loader.mixup_enabled = False
        elif mixup_fn is not None:
            mixup_fn.mixup_enabled = False

    second_order = hasattr(optimizer, 'is_second_order') and optimizer.is_second_order
    batch_time_m = AverageMeter()
    data_time_m = AverageMeter()
    losses_m = AverageMeter()

    model.train()

    end = time.time()
    last_idx = len(loader) - 1
    num_updates = epoch * len(loader)
    for batch_idx, (input, target) in enumerate(loader):
        last_batch = batch_idx == last_idx
        data_time_m.update(time.time() - end)
        if not args.prefetcher:
            input, target = input.cuda(), target.cuda()
            if mixup_fn is not None:
                input, target = mixup_fn(input, target)
        if args.channels_last:
            input = input.contiguous(memory_format=torch.channels_last)

        with amp_autocast():
```

```
                output = model(input)
                loss = loss_fn(output, target)

            if not args.distributed:
                losses_m.update(loss.item(), input.size(0))

            optimizer.zero_grad()
            if loss_scaler is not None:
                loss_scaler(
                    loss, optimizer,
                    clip_grad=args.clip_grad, clip_mode=args.clip_mode,
                    parameters=model_parameters(model, exclude_head='agc' in args.clip_mode),
                    create_graph=second_order)
            else:
                loss.backward(create_graph=second_order)
                if args.clip_grad is not None:
                    dispatch_clip_grad(
                        model_parameters(model, exclude_head='agc' in args.clip_mode),
                        value=args.clip_grad, mode=args.clip_mode)
                optimizer.step()

            if model_ema is not None:
                model_ema.update(model)

            torch.cuda.synchronize()
            num_updates += 1
            batch_time_m.update(time.time() - end)
            if last_batch or batch_idx % args.log_interval == 0:
                lrl = [param_group['lr'] for param_group in optimizer.param_groups]
                lr = sum(lrl) / len(lrl)

                if args.distributed:
                    reduced_loss = reduce_tensor(loss.data, args.world_size)
                    losses_m.update(reduced_loss.item(), input.size(0))

                if args.local_rank == 0:
                    _logger.info(
                        'Train: {} [{:>4d}/{} ({:>3.0f}%)]  '
                        'Loss: {loss.val:>9.6f} ({loss.avg:>6.4f})  '
                        'Time: {batch_time.val:.3f}s, {rate:>7.2f}/s  '
                        '({batch_time.avg:.3f}s, {rate_avg:>7.2f}/s)  '
                        'LR: {lr:.3e}  '
                        'Data: {data_time.val:.3f} ({data_time.avg:.3f})'.format(
                            epoch,
                            batch_idx, len(loader),
                            100. * batch_idx / last_idx,
                            loss=losses_m,
                            batch_time=batch_time_m,
                            rate=input.size(0) * args.world_size / batch_time_m.val,
                            rate_avg=input.size(0) * args.world_size / batch_time_m.avg,
                            lr=lr,
                            data_time=data_time_m))

                    if args.save_images and output_dir:
                        torchvision.utils.save_image(
                            input,
                            os.path.join(output_dir, 'train-batch-%d.jpg' % batch_idx),
                            padding=0,
                            normalize=True)

            if saver is not None and args.recovery_interval and (
                    last_batch or (batch_idx + 1) % args.recovery_interval == 0):
                saver.save_recovery(epoch, batch_idx=batch_idx)

            if lr_scheduler is not None:
                lr_scheduler.step_update(num_updates=num_updates, metric=losses_m.avg)

            end = time.time()
            # end for

        if hasattr(optimizer, 'sync_lookahead'):
            optimizer.sync_lookahead()
```

```
    return OrderedDict([('loss', losses_m.avg)])
```

这个函数里面值得注意的是 loss_scaler 函数，它的作用本质上是 **loss.backward(create_graph=create_graph) 和 optimizer.step()**。

loss_scaler 继承 NativeScaler 这个类。这个类的实例在调用时需要传入 loss, optimizer, clip_grad 等参数，在 **__call__ () 函数**的内部实现了 **loss.backward(create_graph=create_graph) 功能和 optimizer.step() 功能**。

```
class NativeScaler:
    state_dict_key = "amp_scaler"

    def __init__(self):
        self._scaler = torch.cuda.amp.GradScaler()

    def __call__(self, loss, optimizer, clip_grad=None, clip_mode='norm', parameters=None, create_graph=False):
        self._scaler.scale(loss).backward(create_graph=create_graph)
        if clip_grad is not None:
            assert parameters is not None
            self._scaler.unscale_(optimizer)  # unscale the gradients of optimizer's assigned params in-place
            dispatch_clip_grad(parameters, clip_grad, mode=clip_mode)
        self._scaler.step(optimizer)
        self._scaler.update()

    def state_dict(self):
        return self._scaler.state_dict()

    def load_state_dict(self, state_dict):
        self._scaler.load_state_dict(state_dict)
```

总结

本文简要介绍了优秀的PyTorch Image Model 库：timm库的使用方法以及框架实现。图像分类，顾名思义，是一个输入图像，输出对该图像内容分类的描述的问题。它是计算机视觉的核心，实际应用广泛。图像分类的传统方法是特征描述及检测，这类传统方法可能对于一些简单的图像分类是有效的，但由于实际情况非常复杂，传统的分类方法不堪重负。本文的目的是为学者介绍一系列的优秀的视觉分类深度学习模型的PyTorch实现，以便更快地开展相关实验。

极市平台签约作者#

科技猛兽

知乎：科技猛兽

清华大学自动化系19级硕士

研究领域：AI边缘计算 (Efficient AI with Tiny Resource)：专注模型压缩，搜索，量化，加速，加法网络，以及它们与其他任务的结合，更好地服务于端侧设备。

作品精选

搞懂 Vision Transformer 原理和代码，看这篇技术综述就够了

用Pytorch轻松实现28个视觉Transformer，开源库 timm 了解一下！（附代码解读）

轻量高效！清华智能计算实验室开源基于PyTorch的视频（图片）去模糊框架SimDeblur

投稿方式：

添加小编微信Fengcall（微信号：fengcall19），备注：**姓名-投稿**

△长按添加极市平台小编

觉得有用麻烦给个在看啦~

阅读原文

喜欢此内容的人还喜欢

基于OpenCV的焊件缺陷检测

小白学视觉

TorchVision重磅升级：支持多权重的API

机器学习算法工程师

收藏 | 各种 Optimizer 梯度下降优化算法回顾和总结

深度学习算法与计算机视觉