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

原创 CV开发者都爱看的 极市平台 2021-07-13 22:00:00 手机阅读

作者丨科技猛兽
编辑丨极市平台

极市导读

清华大学自动化系智能计算实验室团队开源基于 PyTorch 的视频（图片）去模糊框架 SimDeblur，涵盖经典的视频（图像）去模糊算法且轻量高效。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

清华大学自动化系**智能计算实验室团队**开源基于 PyTorch 的**视频 (图片) 去模糊**框架 **SimDeblur**。

基于 PyTorch 的视频 (图片) 去模糊框架 SimDeblur

它的特点是：

- **全面：** 涵盖经典的视频 (图像) 去模糊算法，如 MSCNN, SRN, DeblurGAN, EDVR, 等等。
- **高效：** 支持 DDP 多机多卡训练。
- **轻量：** 便于拓展，易上手，让更多的人能更快地上手使用。
- **专注：** 使我们在实现自己的新模型时只需要关注一个文件或很少的几个文件。

Github link：

ljzycmd/SimDeblurgithub.com

目录

1 为什么要做这个开源框架?

在深度学习领域，有几个问题我觉得很有必要提一下：

1.1 怎么总是这几个baseline?

> **比如说**
> 在检测领域，baseline一般都有：Faster R-CNN,FCOS等等。
> 在Vision Transformer领域，baseline一般都有：DeiT,T2T-ViT,TNT等等。
> 在超分领域，baseline一般都有：CARN，EDSR，EDVR等等。

大家都不比较那些“最好”的baseline，而是去比较很 Popular 的baseline。

这就像买显卡时，

> 1060说：我比960好。
> 1080说：我比960好。
> 2080Ti说：我比960好。

有很多自称达到了 SOTA 的模型，涨到了比较高的性能，但是很难考证。所以后续研究者在选择比较对象的时候就会选择一些**性能相对较低的，但是代码高质量开源的论文**去比较。原因有2点：

1. 这些论文因为代码高质量开源，所以引用量高，大家都知道且信服，比较 Popular。
2. 这些论文性能相对低一点，和他们比较显得自己提出的方法厉害一点，也就更容易发论文。

这样做的好处是有百花齐放百家争鸣的感觉。但坏处是有的真正好的 baseline 模型被忽略掉了，导致了劣币驱逐良币。

如果今天你问一个你所在领域的专家，随便挑一个人，你问他：

> **" 我们这个任务目前最好的模型是哪个？ "**

他一定也很难回答。

你可能会问了：

> **" 这有啥难的？我直接把最新的论文都找出来，看看这个任务里面，谁超过baseline最多，谁提升的幅度最大，谁不就是最好的吗？ "**

这就引出了第2个问题：

1.2 同一个baseline，在不同论文中的质量差别很大

这句话的意思是说：**同一个baseline模型，相同的任务，不同论文中给出的结果性能是不同的。** 为什么呢？

这是因为：很多研究者对baseline的复现，其实并没有做到“全心全意”。换句话说，对baseline参数的调整其实带有相当大的随意性，对baseline的调整不会下过多的功夫，导致得到的baseline的性能没有达到其可以达到的最佳状态。

在这种情况下，如果你想比较2个**自称达到了SOTA的模型**的性能，因为它们对比的baseline的性能有差距，所以假设它们都相对baseline涨了3个点，但其实它们的性能是有差别的，所以就不具备很好的可比性。可能甲把baseline调得非常好，另一个乙把baseline没有调得很好，那么乙的提升就不具备很高的可信度。

你可能又会问了：

> **" 那我就直接找出baseline论文中给出的它在某个数据集上的性能，直接使用它的结果不就好了吗？ "**

这就引出了第3个问题：

- **1.3 同一个baseline，同一个数据集实验结果可比吗？**

即使baseline在用一个数据集上，其实验结果也是不可比的。这是因为实验中的很多其他变量无法得到相同的控制。比如在数据预处理环节，每篇论文所列的baseline方法是否做到了完全一致？再比如在超参数的设置上，每篇论文所列的baseline方法是否做到了完全相同？

我们看下面的2张图，图1是DeiT模型的超参数设置 (DeiT是一种用于分类任务的视觉Transformer模型)，图2是不同超参数设置下的模型性能对比。我们可以看到，相同的模型在相同的数据集下面，性能还是有差别的。所以这些看似不起眼的设定，其实是对模型的性能有着相对重大的影响，而这些却不会出现在引用DeiT的论文里面。所以你可能会看到：相同的模型在相同的数据集下面，结果又是会出现很大的差异。假设我们有8个超参数，每个超参数只有2种选择，那么不同的组合就多达2^8种。

Epochs	300
Batch size	1024
Base learning rate	0.0005
Learning rate decay	cosine
Weight decay	0.05
Label smoothing ε	0.1
Dropout	0.1
Warmup epochs	5
Rand Augment	9/0.5
Mixup prob.	0.8
Cutmix prob.	1.0
Erasing prob.	0.25
Distillation soft weight λ	0.1
Distillation soft temperature τ	3.0

图1：DeiT模型的超参数设置

											top-1 accuracy	
Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	pre-trained 224^2	fine-tuned 384^2
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3

optimizer	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

图2：DeiT模型不同超参数设置下的模型性能对比

总之这里想说的就是：很难保证 A 和 B 两篇论文的一切实验设置都是相同的。这就导致即使我们找到了 A 和 B 两篇在相同的模型在相同的数据集下面进行的实验，它们的结果也不是那么的可比。

你可能又会问了：

> **" 那很多论文都提供了开源代码，我直接下载下来在自己的任务上跑跑不就行了吗？ "**

这就引出了第4个问题：

1.4 低质量的代码开源

目前一篇顶会论文开源代码的最低要求是：**能复现论文中所列的实验结果**。但遗憾的是，许多开源代码根本无法达到这个要求。对于有些达到了这个要求的代码，它们的可重用性也非常差，想把它移植到你自己的实验环境下也十分地困难。我之前遇到过很多种奇葩的开源代码，这里随便举一个例子 (具体的论文就不说了。。)。比如它做 NAS 的论文，开源的代码里面没有 NAS 搜索的代码，只有模型的 model.py，那这样的开源代码就缺乏了最核心的 NAS 算法的开源，就是无意义的。那遇到这样的情况可能一周过去了，你还是无法复现出原论文的结果，这时候开组会时：

> 导师：你这周干了啥?
> 你：复现某某某论文失败了。
> 导师：这代码不是开源了吗，怎么还是复现不出来，你有没有认真做实验?
> 你：。。。。。。(委屈脸)

这种情况其实是很普遍且很不合理的情况，真的不是你的能力不行，而是目前领域中广泛存在的问题，Are we really making progress?所以在目前领域文章看似百花齐放的前提下，其实隐藏着一个潜在的，使领域停滞不前的问题。

这里我在举一个良性的例子。

比如2020年是视觉Transformer爆火的一年，从20年下半年开始一直持续到21年，Transformer模型被应用在了视觉的各个领域，想详细了解的童鞋们可以参考：

科技猛兽：Vision Transformer 超详细解读 (原理分析+代码解读)

https://zhuanlan.zhihu.com/p/348593638

但是，在2020年爆火的Vision Transformer背后，其实是**有一个重要的依托**，就是**Ross Wightman大佬创建的timm库**（https://github.com/rwightman/pytorch-image-models/tree/master/timm）。Py**T**orch**Im**age**M**odels，简称timm，包含很多种PyTorch的视觉模型，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，详细的介绍如下：

科技猛兽：视觉Transformer优秀开源工作：timm库vision transformer代码解读

https://zhuanlan.zhihu.com/p/350837279

许多Vision Transformer，包含高引的DeiT，CaiT等，其实都是基于timm库来实现的。所以这给了我们启发：**我们需要一个benchmark平台，包含多种模型，使得它们在同一条件下得到公平的评测**，这也是我们开发这一框架的初衷。

在设计这个框架时，我们的思想是：

- 首先它应该**轻量**，易上手，让更多的人能更快地上手使用。
- 其次它应该**高效**，使使用者专注于模型的实现，对于训练和评估的过程尽量少关心。
- 其次它应该**灵活**，适配不同的数据输入格式和实验设定。
- 最后就是**专注**，使我们在实现新模型时只需要关注一个文件。

2 SimDeblur: 基于PyTorch的视频 (图片) 去模糊框架

2.1 已实现模型

(粗体表示已经实现的模型，其他是待实现的模型)

- Single Image Deblurring
  - MSCNN

    Paper：https://arxiv.org/abs/1612.02177

    Project：https://github.com/SeungjunNah/DeepDeblur-PyTorch
  - SRN

    Paper：https://arxiv.org/abs/1802.01770

    Project：https://github.com/jiangsutx/SRN-Deblur
- Video Deblurring
  - DBN

    Paper：https://arxiv.org/abs/1611.08387

    Project：http://www.cs.ubc.ca/labs/imager/tr/2017/DeepVideoDeblurring/

- STRCNN [paper：https://arxiv.org/abs/1704.03285]
- DBLRNet [Paper：https://arxiv.org/abs/1804.00533]
- EDVR

  Paper：https://arxiv.org/abs/1905.02716

  Project：https://github.com/xinntao/EDVR
- STFAN

  Paper：https://arxiv.org/abs/1904.12257

  Project：https://shangchenzhou.com/projects/stfan/
- IFIRNN [Paper：https://openaccess.thecvf.com/content_CVPR_2019/html/Nah_Recurrent_Neural_Networks_With_Intra-Frame_Iterations_for_Video_Deblurring_CVPR_2019_paper.html]
- CDVD-TSP

  Paper:https://arxiv.org/abs/2004.02501

  Project:https://github.com/csbhr/CDVD-TSP
- ESTRNN

  Paper:https://www.ecva.net/papers/eccv_2020/papers_ECCV/html/5116_ECCV_2020_paper.php

  Project:https://github.com/zzh-tech/ESTRNN

- Benchmarks
  - GoPro

    Paper:https://arxiv.org/abs/1612.02177

    Data:https://seungjunnah.github.io/Datasets/gopro
  - DVD

    Paper:https://arxiv.org/abs/1611.08387

    Data:http://www.cs.ubc.ca/labs/imager/tr/2017/DeepVideoDeblurring/
  - REDS

    Paper:https://openaccess.thecvf.com/content_CVPRW_2019/html/NTIRE/Nah_NTIRE_2019_Challenge_on_Video_Deblurring_and_Super-Resolution_Dataset_and_CVPRW_2019_paper.html

    Data:https://seungjunnah.github.io/Datasets/reds

2.2 使用方法

1) 安装依赖

```
Python 3 (Conda is recommended)
Pytorch 1.5.1 (with GPU)
CUDA 10.2+
```

Clone the repositry or download the zip file:

```
git clone https://github.com/ljzycmd/SimDeblur.git
```

Install SimDeblur:

```
# create a pytorch env
conda create -n simdeblur python=3.7
conda activate simdeblur
# install the packages
cd SimDeblur
bash Install.sh
```

2) 使用默认的 trainer 来搭建一个训练进程，如下所示：

```
from simdeblur.config import build_config, merge_args
from simdeblur.engine.parse_arguments import parse_arguments
from simdeblur.engine.trainer import Trainer


args = parse_arguments()

cfg = build_config(args.config_file)
cfg = merge_args(cfg, args)
cfg.args = args

trainer = Trainer(cfg)
trainer.train()
```

3) 单卡训练：

```
CUDA_VISIBLE_DEVICES=0 bash ./tools/train.sh ./config/dbn/dbn_dvd.yaml 1
```

4) 多卡训练：

```
CUDA_VISIBLE_DEVICES=0,1,2,3 bash ./tools/train.sh ./config/dbn/dbn_dvd.yaml 4
```

train.sh：

```
CONFIG=$1
GPUS=$2
PORT=${PORT:=10086}
# PORT=10086
# single gpu training
if [ GPUS == 1 ]
then
echo start single GPU training
python train.py $CONFIG --gpus=$GPUS
else
echo start distributed training
# distributed training
PYTHONPATH="$(dirname $0)/..":$PYTHONPATH \
python -m torch.distributed.launch --nproc_per_node=$GPUS --master_port=$PORT \
```

```
        train.py $CONFIG --gpus=$GPUS
fi
```

5) 也可以直接通过 SimDeblur 中的函数构建各种模块：

build the a dataset:

```
from easydict import EasyDict as edict
from simdeblur.dataset import build_dataset

dataset = build_dataset(edict({
    "name": "DVD",
    "mode": "train",
    "sampling": "n_c",
    "overlapping": True,
    "interval": 1,
    "root_gt": "./dataset/DVD/quantitative_datasets",
    "num_frames": 5,
    "augmentation": {
        "RandomCrop": {
            "size": [256, 256] },
        "RandomHorizontalFlip": {
            "p": 0.5 },
        "RandomVerticalFlip": {
            "p": 0.5 },
        "RandomRotation90": {
            "p": 0.5 },
    }
}))

print(dataset[0])
```

build the model:

```
from simdeblur.model import build_backbone

model = build_backbone({
    "name": "DBN",
    "num_frames": 5,
    "in_channels": 3,
    "inner_channels": 64
})

x = torch.randn(1, 5, 3, 256, 256)
out = model(x)
```

build the loss:

```
from simdeblur.model import build_loss

```

```
criterion = build_loss({
    "name": "MSELoss",
})
x = torch.randn(2, 3, 256, 256)
y = torch.randn(2, 3, 256, 256)
print(criterion(x, y))
```

2.3 代码解读：

1 框架架构：

> **/configs**
> → /dblrnet: dblrnet配置文件
> → /dbn: dbn配置文件
> → /edvr: edvr配置文件
> → /...
> **/datasets:** 数据集位置
> **/docs**
> **/simdeblur**
> → __init__.py
> **→ /config**
> → → __init__.py
> → → build.py：读取配置信息的一些函数
> → → default_config.py：默认配置信息
> **→ /dataset**
> → → __init__.py
> → → build.py：创建数据集的接口
> → → augment.py：数据增强的函数
> → → dvd.py
> → → gopro.py
> → → red.py
> **→ /engine**
> → → __init__.py
> → → parse_arguments.py
> → → trainer.py：主要的训练代码
> → → hook.py
> **→ /model**
> **→ → __init__.py**
> **→ → build.py：创建模型的接口**
> **→ → /backbone：各种 backbone 具体实现**
> → → →/dblrnet：dblrnet 具体实现
> → → →/dbn：dbn 具体实现
> → → →/edvr：edvr 具体实现
> → → →/ifirnn：ifirnn 具体实现
> → → →/stfan：stfan 具体实现
> → → →/strcnn：strcnn 具体实现
> **→ → /layer：各种 layer 具体实现**
> → → →__init__.py
> → → →non_local.py：non_local block 具体实现
> → → →res_block.py：残差块具体实现
> → → →vgg.py：VGG 块具体实现
> **→ → /loss：各种损失函数具体实现**

> → → →__init__.py
> → → →loss.py
> → → →perceptual_loss.py
> **→ → /meta_arch**
> **→ /scheduler**： 优化器和学习率 scheduler 函数
> **→ /utils**： 打印日志的相关函数
> **/tools:** 生成demo的一些工具函数，以及启动文件 train.sh
> **/utils:** 其它涉及到的一些工具函数
> **/requirements.txt:** 运行需要的依赖库
> **setup.py:** 上传 PYPI 需要的文件
> **test.py**： 模型测试的接口文件，需要传入.yaml格式的配置文件
> **train.py**： 模型训练的接口文件，需要传入.yaml格式的配置文件

2 train.py：

```
import torch

from simdeblur.config import build_config, merge_args
from simdeblur.engine.parse_arguments import parse_arguments
from simdeblur.engine.trainer import Trainer


def main():
    args = parse_arguments()

    cfg = build_config(args.config_file)
    cfg = merge_args(cfg, args)
    cfg.args = args

    trainer = Trainer(cfg)
    trainer.train()


if __name__ == "__main__":
    main()
```

> build_config：根据配置文件 (.yaml) 得到配置信息cfg (字典)。
> merge_args：融合命令行参数。
> 得到包含了所有配置信息的变量 cfg，传入Trainer类。

3 Trainer 类介绍：

(a) 定义 Trainer 类属性：

```
from simdeblur.dataset import build_dataset
from simdeblur.scheduler import build_optimizer, build_lr_scheduler
from simdeblur.model import build_backbone, build_meta_arch, build_loss
from simdeblur.utils.logger import LogBuffer, SimpleMetricPrinter, TensorboardWriter
from simdeblur.utils.metrics import calculate_psnr, calculate_ssim
from simdeblur.utils import dist_utils
```

```
from simdeblur.engine import hooks


logging.basicConfig(format='%(asctime)s - %(levelname)s - SimDeblur: %(message)s',level=logging.INFO)
logging.info("******* A simple deblurring framework ********")

class Trainer:
    def __init__(self, cfg):
        """
        Args
            cfg(edict): the config file, which contains arguments form comand line
        """
        self.cfg = copy.deepcopy(cfg)
        # initialize the distributed training
        if cfg.args.gpus > 1:
            dist_utils.init_distributed(cfg)

        # create the working dirs
        self.current_work_dir = os.path.join(cfg.work_dir, cfg.name)
        if not os.path.exists(self.current_work_dir):
            os.makedirs(self.current_work_dir, exist_ok=True)

        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        # self.device = torch.device("cpu")

        # default logger
        logger = logging.getLogger("simdeblur")
        logger.setLevel(logging.INFO)
        logger.addHandler(
            logging.FileHandler(
                os.path.join(
                    self.current_work_dir, self.cfg.name.split("_")[0] + ".json"))
        )

        # construct the modules
        self.model = self.build_model(cfg).to(self.device)
        self.criterion = build_loss(cfg.loss).to(self.device)
        self.train_dataloader, self.train_sampler = self.build_dataloder(cfg, mode="train")
        self.val_datalocaer, _ = self.build_dataloder(cfg, mode="val")
        self.optimizer = self.build_optimizer(cfg, self.model)
        self.lr_scheduler = self.build_lr_scheduler(cfg, self.optimizer)

        # trainer hooks
        self._hooks = self.build_hooks()

        # some induces when training
        self.epochs = 0
        self.iters = 0
        self.batch_idx = 0

        self.start_epoch = 0
        self.start_iter = 0
```

```
        self.total_train_epochs = self.cfg.schedule.epochs
        self.total_train_iters = self.total_train_epochs * len(self.train_dataloader)

        # resume or load the ckpt as init-weights
        if self.cfg.resume_from != "None":
            self.resume_or_load_ckpt(ckpt_path=self.cfg.resume_from)

        # log bufffer(dict to save)
        self.log_buffer = LogBuffer()
```

(b) 每个 epoch 开始前 shuffle the dataloader when dist training：

```
    def before_epoch(self):
        for h in self._hooks:
            h.before_epoch(self)
        # shuffle the data when dist training ...
        if self.train_sampler:
            self.train_sampler.set_epoch(self.epochs)
```

(c) 每个 iteration 开始前 shuffle the dataloader when dist training：

```
    def before_epoch(self):
        for h in self._hooks:
            h.before_epoch(self)
        # shuffle the data when dist training ...
        if self.train_sampler:
            self.train_sampler.set_epoch(self.epochs)
```

(d) 准备输入信息：

```
    def preprocess(self, batch_data):
        """
        prepare for input
        """
        return batch_data["input_frames"].to(self.device)
```

(e) 模型输出的后处理：

```
def postprocess(self):
        """
        post process for model outputs
        """
        # When the outputs is a img tensor
        if isinstance(self.outputs, torch.Tensor) and self.outputs.dim() == 5:
            self.outputs = self.outputs.flatten(0, 1)
```

(f) 计算损失：

```
    def calculate_loss(self, batch_data, model_outputs):
```

```
        """
        calculate the loss
        """
        gt_frames = batch_data["gt_frames"].to(self.device).flatten(0, 1)
        if model_outputs.dim() == 5:
                model_outputs = model_outputs.flatten(0, 1) # (b*n, c, h, w)
        return self.criterion(gt_frames, model_outputs)
```

(g) 优化器更新参数：

```
    def update_params(self):
        """
        update params
        pipline: zero_grad, backward and update grad
        """
        self.optimizer.zero_grad()
        self.loss.backward()
        self.optimizer.step()
```

(h) 每个 iteration 或者 epoch 结束以后，使用 hook 干一些事情，比如：lr_scheduler 更新，calculate metrics，保存日志等等，具体可以查看 /simdeblur/engine.hook.py 文件。

```
    def after_iter(self):
        for h in self._hooks:
            h.after_iter(self)
    def after_epoch(self):
        for h in self._hooks:
            h.after_epoch(self)
```

(i) 根据以上工具函数写训练函数 train()：

```
     def train(self, **kwargs):
         self.model.train()
         self.before_train()
         logger = logging.getLogger("simdeblur")
         logger.info("Starting training...")
         for self.epochs in range(self.start_epoch, self.cfg.schedule.epochs):
             # shuffle the dataloader when dist training: dist_data_loader.set_epoch(epoch)
             self.before_epoch()
             for self.batch_idx, self.batch_data in enumerate(self.train_dataloader):
                 self.before_iter()

                 input_frames = self.preprocess(self.batch_data)

                 self.outputs = self.model(input_frames)
                 self.postprocess()

                 self.loss = self.calculate_loss(self.batch_data, self.outputs)

                 self.update_params()

```

```
                self.iters += 1
                self.after_iter()

            if self.epochs % self.cfg.schedule.val_epochs == 0:
                self.val()

            self.after_epoch()
```

> before_epoch(), after_epoch(), before_iter(), after_iter() 这四个函数都是通过 hook 来定义每个 epoch 之前或之后，每个 iteration 之前或之后要做的事情，具体可以查看 /simdeblur/engine.hook.py 文件。

3 作者团队信息

曹铭登：

> 清华大学自动化系19级硕士，目前实习于腾讯 AI Lab。
> **邮箱：mingdengcao@gmail.com**

王家豪：

> 清华大学自动化系19级硕士，目前实习于北京华为诺亚方舟实验室。
> **邮箱：wang-jh19@mails.tsinghua.edu.cn**

智能计算实验室信息：

https://sites.google.com/view/iigroup-thusites.google.com

学术合作 or 沟通交流欢迎私信联系~

cite as:

```
@Article{wang2021simdeblur,
  author  = {Mingdeng Cao, Jiahao Wang},
  title   = {清华智能计算实验室团队开源基于PyTorch的视频（图片）去模糊框架SimDeblur},
  journal = {https://zhuanlan.zhihu.com/},
  howpublished = {\url{https://github.com/ljzycmd/SimDeblur}},
  year    = {2021},
  url= {https://zhuanlan.zhihu.com/p/368312516/},
}
```

极市干货

YOLO教程：一文读懂YOLO V5 与 YOLO V4｜大盘点｜YOLO 系目标检测算法总览｜全面解析YOLO V4网络结构

实操教程：PyTorch vs LibTorch：网络推理速度谁更快？｜只用两行代码，我让Transformer推理加速了50倍｜PyTorch AutoGrad C++层实现

算法技巧（trick）：深度学习训练tricks总结（有实验支撑）｜深度强化学习调参Tricks合集｜长尾识别中的Tricks汇总（AAAI2021）

最新CV竞赛：2021 高通人工智能应用创新大赛｜CVPR 2021 | Short-video Face Parsing Challenge｜3D人体目标检测与行为分析竞赛开赛，奖池7万+，数据集达16671张！

极市平台签约作者#

科技猛兽

清华大学自动化系19级硕士，目前实习于北京华为诺亚方舟实验室。

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。

知乎：科技猛兽

相关作品：

MLP三大工作超详细解读：why do we need?

搞懂 Vision Transformer 原理和代码，看这篇技术综述就够了（十三）

进可暴力提性能，退可无损做压缩：结构重参数化技术综述

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

15个目标检测开源数据集汇总

极市平台