

gtcv算法库介绍

一、gtcv

1、积累高质量的工业AI算法库

1-1、通用AI模型标准化

- 标准化模型输入、输出
- 训练、推理、验证、配置接口保持一致

1-2、可视化分析

- 支持数据集可视化分析、清洗
- 支持模型评估
- 支持错图分析

1-3、支持快速修改定制

- 模型的所有部分都可以通过重载接口直接修改（model, loss、step、optimizer、lr schedule...

1-4、积累传统数字图像处理算法

- 尺寸测量
- 模板匹配
- blob分析等

2、实现机制【pytorch-lightning + timm】

2-1、pytorch-lightning

Github: <https://github.com/Lightning-AI/lightning>

官网: <https://lightning.ai/pytorch-lightning>

减少pytorch模型训练、部署过程中的大量重复代码。



Advantages over unstructured PyTorch

- Models become hardware agnostic

- Code is clear to read because engineering code is abstracted away
- Easier to reproduce
- Make fewer mistakes because lightning handles the tricky engineering
- Keeps all the flexibility (LightningModules are still PyTorch modules), but removes a ton of boilerplate
- Lightning has dozens of integrations with popular machine learning tools.
- Tested rigorously with every new PR. We test every combination of PyTorch and Python supported versions, every OS, multi GPUs and even TPUs.
- Minimal running speed overhead (about 300 ms per epoch compared with pure PyTorch).

```

PYTORCH
# models
encoder = nn.Sequential(nn.Linear(28 * 28, 64), nn.ReLU(), nn.Linear(64, 3))
decoder = nn.Sequential(nn.Linear(3, 64), nn.ReLU(), nn.Linear(64, 28 * 28))

encoder.cuda(0)
decoder.cuda(0)

# download on rank 0 only
if global_rank == 0:
    mnist_train = MNIST(os.getcwd(), train=True, download=True)

# split dataset
transform=transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize(0.5, 0.5)])
mnist_train = MNIST(os.getcwd(), train=True, download=True, transform=transform)

# train (55,000 images), val split (5,000 images)
mnist_train, mnist_val = random_split(mnist_train, [55000, 5000])

# The dataloaders handle shuffling, batching, etc...
mnist_train = DataLoader(mnist_train, batch_size=64)
mnist_val = DataLoader(mnist_val, batch_size=64)

# optimizer
params = [encoder.parameters(), decoder.parameters()]
optimizer = torch.optim.Adam(params, lr=1e-3)

# TRAIN LOOP
model.train()
num_epochs = 1
for epoch in range(num_epochs):
    for train_batch in mnist_train:
        x, y = train_batch
        x = x.cuda(0)
        x = x.view(x.size(0), -1)
        z = encoder(x)
        x_hat = decoder(z)
        loss = F.mse_loss(x_hat, x)
        print('train loss: ', loss.item())

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

# EVAL LOOP
model.eval()
with torch.no_grad():
    val_loss = []
    for val_batch in mnist_val:
        x, y = val_batch
        x = x.cuda(0)
        x = x.view(x.size(0), -1)
        z = encoder(x)
        x_hat = decoder(z)
        loss = F.mse_loss(x_hat, x)
        val_loss.append(loss)
    val_loss = torch.mean(torch.tensor(val_loss))
    model.train()

```

PYTORCH LIGHTNING

Turn PyTorch into Lightning

Lightning is just plain PyTorch.

2-2、timm

提供大量SOTA的预训练图像模型。官网：<https://github.com/huggingface/pytorch-image-models>
可直接输出不同层级的特征金字塔，方便各种模型定制

```

1 import torch
2 import timm
3 m = timm.create_model('mobilenetv3_large_100', features_only=True,
    pretrained=True)

```

```

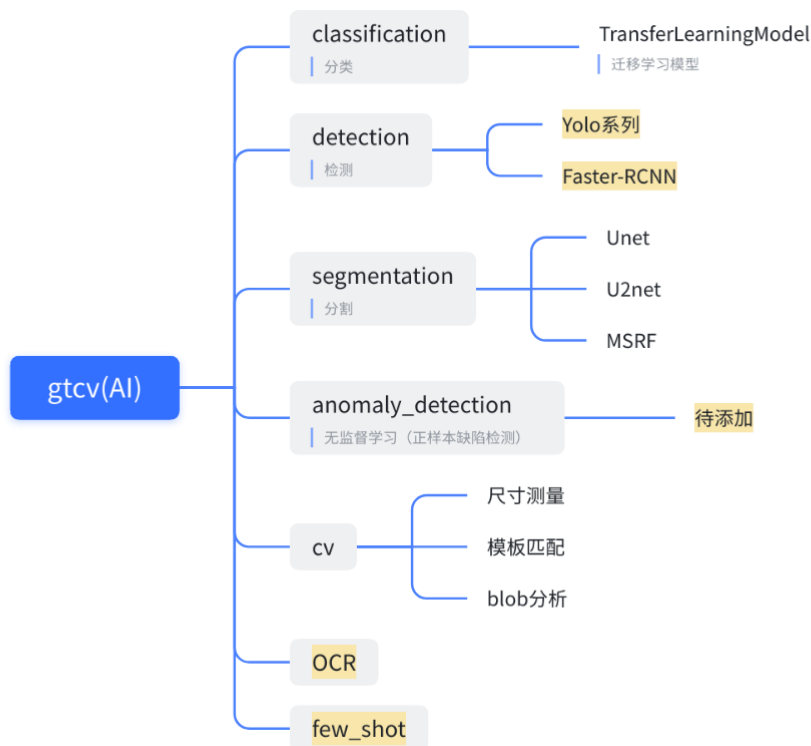
4 print(f'Feature channels: {m.feature_info.channels()}')
5 o = m(torch.randn(2, 3, 224, 224))
6 for x in o:
7     print(x.shape)
8
9 """ Out
10 Feature channels: [16, 24, 40, 112, 960]
11 torch.Size([2, 16, 112, 112])
12 torch.Size([2, 24, 56, 56])
13 torch.Size([2, 40, 28, 28])
14 torch.Size([2, 112, 14, 14])
15 torch.Size([2, 960, 7, 7])
16 """

```

3、模型说明



！黄色会在近期添加



二、算法开发服务器（镜像/容器）管理

📖 算法开发服务器（镜像/容器）管理

三、demo演示

1、容器

参考[国算法开发服务器（镜像/容器）管理](#)

```
1 docker load -i Ind_Vision_Base.tar
2 # docker images
3 nvidia/cuda:11.7-cudnn8-devel-ubuntu18.04
4
5 # docker run
6 nvidia-docker run -dit --name cv_algo_demo -p 7322:22 -p 7330-7399:7330-7399 -
  v /etc/localtime:/etc/localtime -v /var/wdcvml:/var/wdcvml -v
  /var/gtcvml:/var/gtcvml -v /etc/machine-id:/etc/machine-id -v
  /data/algorithm/cv_algo/dataset/public:/root/dataset/public -v
  /data/algorithm/cv_algo/dataset/cv_algo:/root/dataset/cv_algo -v
  /data/algorithm/cv_algo/project/cv_algo:/root/project/cv_algo -v
  /data/algorithm/cv_algo/shared:/root/shared -v
  /data/algorithm/cv_algo/common/pretrained/_.torch:/root/.torch -v
  /data/algorithm/cv_algo/common/pretrained/_.cache:/root/.cache -v
  /dev/shm:/dev/shm --privileged nvidia/cuda:11.7-cudnn8-devel-ubuntu18.04
7
8 # 进入容器
9 docker exec -it cv_algo_demo bash
10
11 # 开启ssh服务
12 service ssh start
13
14 # ssh连接
15 root@192.168.1.6:7322
16
17 # 虚拟环境 同一项目不同开发人员可以克隆虚拟环境或者create新的虚拟环境，独立环境开发。
18 cv_env
19
20 # jupyter启动配置jupyter密码（getech）：
21 jupyter notebook password
22
23 # 在~/目录下，启动jupyter服务： nohub jupyter-notebook --no-browser --ip 0.0.0.0 -
  -port xx40 --allow-root > jupyter.nohub.out &
24 nohub jupyter-notebook --no-browser --ip 0.0.0.0 --port 7340 --allow-root >
  jupyter.nohub.out &
25
26 # 浏览器打开
27 http://192.168.1.6:7340/tree
```

2、算法包

闭源算法包： [📄 gtcv-0.0.1-cp37-cp37m-linux_x86_64.whl](#)

开源算法包： [📄 gtcv-0.0.1-py3-none-any.whl](#)

```
1 # gtcv-0.0.1-py3-none-any.whl
2 pip install gtcv-0.0.1-py3-none-any.whl
```

gitlab: <https://gitlab.poros.getech.cn/operation-getech-algo/gtcv>

2-1、项目中算法包代码释放：

修改源码 - 生成whl包 - 随项目算法包发布到部署机器

- 闭源方式【部署时闭源】

生成 whl 的python包: dist 目录下 gtcv-0.0.1-cp37-cp37m-linux_x86_64.whl

```
python setup.py bdist_wheel BUILD_ONLY_SO
```

- 开源方式

生成 whl 的python包: dist目录下 gtcv-0.0.1-py3-none-any.whl

```
python setup.py bdist_wheel
```

- 源码开发者模式【方便算法人员服务器上修改更新源码，并立即看到更改的效果】

```
python setup.py develop
```

2-2、算法库核心算法贡献：

修改源码 - 测试OK - 提交代码 - review - 合并代码

2-3、examples:

```
gtcv/examples/
```

四、期望

- 覆盖工业日常使用所用模型，保证各项目可以正常使用gtcv的各种模型
- 各类模型指标达到业界领先
- 算法成员都参与到核心算法的贡献