**RGB-T目标跟踪**任务执行记录

**执行目录均为：(apf) user@user-Super-Server:~/chenzimeng/APFNet/AAAICode$**

**另记为 /home/user/chenzimeng/APFNet/AAAICode**

1 任务背景

参考资料：

[1] [RGB-T追踪——【数据集基准】](https://blog.csdn.net/qq_42312574/article/details/125944796?ops_request_misc=&request_id=&biz_id=102&utm_term=RGB-T%E7%9B%AE%E6%A0%87%E8%B7%9F%E8%B8%AA&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-3-125944796.142%5ev100%5epc_search_result_base2&spm=1018.2226.3001.4187)

1.1 RGB-T目标跟踪任务概述

RGB-T目标跟踪（RGB-Thermal Target Tracking）是多模态目标跟踪中的一种任务，它利用RGB（可见光）和热红外（Thermal）两种模态的数据，进行对目标在视频中的实时跟踪。RGB-T目标跟踪任务的主要目标是将这两种数据源的信息进行融合，以实现对目标的更稳定、准确的跟踪。

**RGB-T目标跟踪的优势**

1. **模态互补性**：RGB和热红外图像具有不同的特性，**RGB图像能够提供丰富的颜色和纹理信息**，适合在白天和光照充足的环境中使用。而**热红外图像则能够捕捉到物体的温度特征，能够在夜晚、烟雾、多云等恶劣环境中提供有效的信息。**这种模态互补性使得RGB-T跟踪**在各种光照条件下都能保持稳定性**。
2. **鲁棒性增强**：由于目标跟踪的环境通常复杂多变，如光照变化、遮挡、天气干扰等，**单一模态的数据可能无法应对这些挑战。而RGB-T目标跟踪能够充分利用两种模态的特征，提升在各种场景中的鲁棒性和跟踪精度。**

**技术挑战**

1. **多模态信息融合**：如何将RGB和热红外图像中的信息有效地融合是RGB-T目标跟踪的核心挑战之一。要设计合理的融合策略，确保在跟踪过程中充分利用两种模态的优势。
2. **跨模态特征对齐**：由于RGB和热红外图像的成像机制不同，两者可能存在视角、分辨率、尺度等差异。有效地**对齐和匹配**两种模态的特征是实现准确跟踪的重要步骤。
3. **环境适应性**：目标在跟踪过程中可能会经历不同的光照、天气条件和遮挡，如何在这些动态环境下持续保持跟踪精度，要求模型具备较强的环境适应性。

**应用场景**

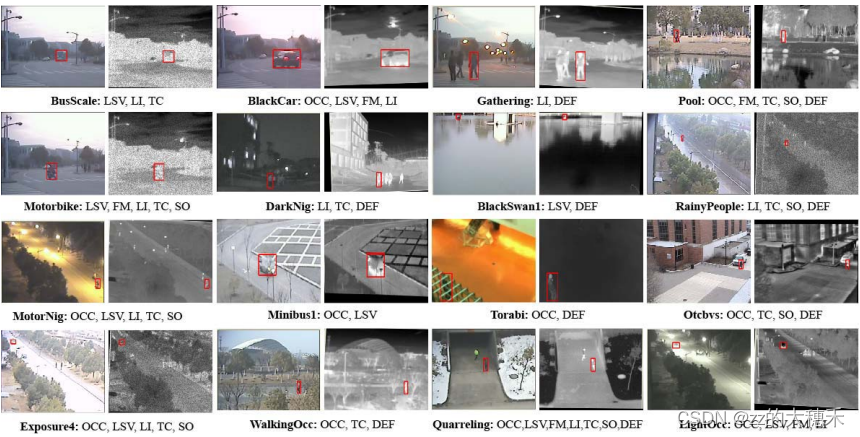
* **安全监控**：在全天候环境中监控目标，例如夜间监控、烟雾环境或复杂光照条件下的监控。
* **自动驾驶**：用于全天候目标检测和跟踪，提升自动驾驶车辆在复杂环境下对行人、车辆等目标的检测能力。
* **无人机跟踪**：在无人机任务中，RGB-T跟踪可以提高在昼夜环境中对目标的跟踪能力。

1.2 RGB-T数据集介绍

**热红外图像**的原理是捕捉图片中高于绝对零度温度的物体的热红外辐射，所以对比可见光图片（RGB），热红外图片可以不受光照条件影响，比如即使很大的雾，热红外图片仍能拍摄出当前场景中的物体轮廓，而可见光图片则是雾蒙蒙一片，啥也看不清。随之而来的挑战也在于：当场景中目标温度与背景相似就会出现Themal Crossover（热交叉）问题。如果这时候只靠热力图，那么很可能误追。

1.2.1 GTOT

* 出自2016 IEEE Transactions on Image Processing (T-IP)的Learning Collaborative Sparse Representation for Grayscale-thermal Tracking 文章，安徽大学李成龙课题组。
* 数据集包括带统计偏差的50个视频对（以灰度图片-热红外图片对的序列）、视频对应的每帧GroundTruth注释、两个评估指标。
* 视频对是**对齐的灰度图片和热红外图**。视频拍摄场景有16个，包括实验室、校园道路、操场、水池等等。



* **评价指标**：  
  ***Precision Score*** **预测框与真值框之间的中心位置距离**。由于是小目标，所以这里阈值也由常用的20像素改成5像素。  
  ***Success plot*** **预测框与真值框之间的交并比**。AUC表示Success plot的曲线下距离。

1.2.2 RGBT234

出自2019 Pattern Recognition (PR)的RGB-T Object Tracking: Benchmark and Baseline. 文章，安徽大学李成龙课题组。

数据集特性：

* 数据集包括234个RGB-T视频序列对及其对应真值GroundTruth。视频序列标注中有12个属性。总的帧数为234K，最长的视频序列有8K。
* 标注序列第一帧的真值格式(x,y,box-width,box-height)
* 一个有5个评估指标，其中包括了VOT中的评估指标。

（1）Maximum Precision Rate (MPR)：Precision 还是那个Precision，只是考虑到对齐的误差问题，所以用MPR代替PR，就是说**RGB图片的预测边界框和热力图的预测边界框的中心距离取更大的那个。**

(2) Maximum Success Rate (MSR)：Success Rate还是那个Success Rate，是考虑到对齐的误差问题，所以用MSR代替SR。再用AUC计算SR的曲线下面积。

(3) Accuracy：衡量预测为真的这些帧的平均IOU。

(4) Robustness：衡量追踪器追到（预测框与真值框之间IOU大于阈值）的频率。  
(5) Expected Average Overlap (EAO)：期望平均IOU结合了追踪器的准确度与鲁棒性两个评估指标。具体公式计算见这篇博客。



图中第一行的RGB图像，下一行是对应的热红外图：

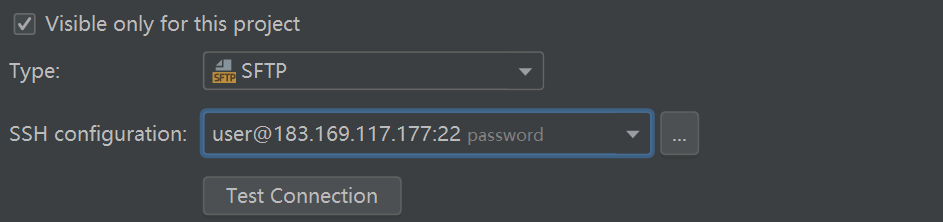
* 由（a）组图片可知，当高光照条件、低分辨率条件下热力图中目标可见度更高，为RGB图的补充了许多目标位置相关信息。
* 由（b）组图片可知，当出现Themal Crossover（热交叉）问题，RGB图像也为热红外图提供更多外观上的信息。

2 实验记录

2.1 连接远程服务器以及初始化环境

首先我们连接远程服务器（Tools-Deployment）；然后上传文件；

映射路径：/home/user/chenzimeng/APFNet/AAAICode



然后我们创建虚拟环境apf

conda create -n apf python==3.8

激活并进入虚拟环境：  
conda activate apf

2.2 阅读Readme文档

**Attribute-Based Progressive Fusion Network for RGBT Tracking**

**This project is created base on**

--MDNet: Real-Time Multi-Domain Convolutional Neural Network Tracker Created by Ilchae Jung, Jeany Son, Mooyeol Baek, and Bohyung Han

**Prerequisites**

1. python>=3.6
2. pytorch>=1.0
3. opencv 3.0+
4. some others library functions

For more detailed packages, refer to [MDNet](<https://github.com/hyeonseobnam/py-MDNet>) .

包（依赖项）安装命令：

1 安装 PyTorch（请根据你的CUDA版本选择合适的命令）

pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118

查看CUDA版本：

nvcc --version OR

nvidia-smi

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2022 NVIDIA Corporation

Built on Wed\_Sep\_21\_10:33:58\_PDT\_2022

Cuda compilation tools, release 11.8, V11.8.89

Build cuda\_11.8.r11.8/compiler.31833905\_0

2 安装 OpenCV 3.0+

pip3 install opencv-python

**Pretrained model for APFNet**

In our tracker, we use **MDNet as our backbone** and extend to multi-modal tracker.

We use **imagenet-vid.pth as our pretrain model**.

Then we use this with the training model in GTOT and RGBT234 models to pre-train our dual-stream MDNet\_RGBT backbone network.And thus we get the **GTOT.pth** and **RGBT234.pth**.

And Then We load the basic model to train Our network and get the final model.

Our model and the pretrain model is available at [pth model] ([https://pan.baidu.com/s/1UmbO7QSt41d4hed4CcTJTg). The](https://pan.baidu.com/s/1UmbO7QSt41d4hed4CcTJTg).%20The) extract code is **mmic**. After downloading the model, you should put it in **AAAICode/models/**

**Run tracker**

In the tracking/Run.py file, you need to change dataset path, model\_path and result\_path.

In the tracking/Run.py file. You can load the model GTOT\_ALL\_Transformer for testing RGBT234 and LasHeR. And use the RGBT234\_ALL\_Transformer for testing the GTOT.

python ./tracking/Run.py

注意修改GPU编号os.environ["CUDA\_VISIBLE\_DEVICES"] = '3'

修改后的相关代码如下：

logger = get\_logger('./log/GTOT\_ALL\_Transformer.log') #Record your log files

parser = argparse.ArgumentParser()

parser.add\_argument('-s', '--seq', default='', help='input seq')

parser.add\_argument('-j', '--json', default='', help='input json')

parser.add\_argument('-f', '--savefig', action='store\_true')

parser.add\_argument('-d', '--display', action='store\_true')

parser.add\_argument("-dataset", default = 'RGBT234' ) #testing dataset RGB\_T234

parser.add\_argument("-model\_path", default ='./models/GTOT\_ALL\_Transformer.pth') # your model GTOT

parser.add\_argument("-result\_path", default ='./Results/RGBT234+GTOT/') #your result path

args = parser.parse\_args()

##your result path

args.result\_path = './Results/RGBT234+GTOT/'+args.dataset+'/'+args.model\_path.split('/')[-1].split('.')[0]+'/'

#assert args.seq != '' or args.json != ''

print(opts)

dataset\_path = os.path.join('./', args.dataset) #dataset path

mylist='./RGBT234/rgbt234.txt' #dataset list path

**Train**

There Stage train:  
At First you should use the GTOT and RGBT234 datasets with challenge tags,you can find the datasets in (<https://github.com/mmic-lcl/Datasets-and-benchmark-code>) run **prepro\_data.py** generate a xxx.pkl file to store the data path.Please note that you should adjust the phased training parameters in the **pretrain\_option.py** when training each stage.

1. In the first stage,you should run the **train\_stage1.py** 5 times because we have five attribute branches. Each time we train the network with specific label data, we load the pre-trained backbone network model parameters and then add the specific branch one by one for training. Note that at this stage we only save each branch model parameters.
2. You have spawned 5 corresponding challenge branches in one phase.In the second stage,you can load the backbone and all branches parameters for training the Attribute-Based Aggregation Fusion model. you should run the **train\_stage2.py**
3. On the basis of the two-stage, only the model parameters generated by the second stage need to be loaded, and the final model can be generated in the three-stage training. you should run the **train\_stage3.py**

3 报错解决记录

3.1 Run过程中

1

OrderedDict([(‘use\_gpu’, True), (‘init\_model\_path’, ‘./models/imagenet-vgg-m.mat’), (‘model\_path’, ‘./models’), (‘batch\_pos’, 32), (‘batch\_neg’, 96), (‘batch\_neg\_cand’, 1024), (‘batch\_test’, 256), (‘n\_samples’, 256), (‘trans’, 0.6), (‘scale’, 1.05), (‘trans\_limit’, 1.5), (‘img\_size’, 107), (‘padding’, 16), (‘trans\_pos’, 0.1), (‘scale\_pos’, 1.3), (‘trans\_neg\_init’, 1), (‘scale\_neg\_init’, 1.6), (‘trans\_neg’, 2), (‘scale\_neg’, 1.3), (‘n\_bbreg’, 1000), (‘overlap\_bbreg’, [0.6, 1]), (‘trans\_bbreg’, 0.3), (‘scale\_bbreg’, 1.6), (‘aspect\_bbreg’, 1.1), (‘lr\_init’, 0.0005), (‘maxiter\_init’, 50), (‘n\_pos\_init’, 500), (‘n\_neg\_init’, 5000), (‘overlap\_pos\_init’, [0.7, 1]), (‘overlap\_neg\_init’, [0, 0.5]), (‘lr\_update’, 0.001), (‘maxiter\_update’, 15), (‘n\_pos\_update’, 50), (‘n\_neg\_update’, 200), (‘overlap\_pos\_update’, [0.7, 1]), (‘overlap\_neg\_update’, [0, 0.3]), (‘long\_interval’, 10), (‘n\_frames\_long’, 100), (‘n\_frames\_short’, 30), (‘lr’, 0.0001), (‘grad\_clip’, 10), (‘lr\_mult’, {‘fc4’: 5, ‘fc5’: 5, ‘fc6’: 10}), (‘ft\_layers’, [‘fc’])])

Traceback (most recent call last):

File “./tracking/Run.py”, line 386, in <module>

img\_list\_v,img\_list\_i,gt=gen\_config(seq\_path,args.dataset)

File “/home/user/chenzimeng/APFNet/AAAICode/tracking/gen\_config.py”, line 22, in gen\_config

if gt.shape[1] == 8:

UnboundLocalError: local variable ‘gt’ referenced before assignment

这个错误提示说明在 gen\_config 函数中，**变量 gt 在被引用之前没有被赋值**。这通常是因为在函数中某些条件下 **gt 没有被正确初始化**。要解决这个问题，可以按照以下步骤进行：

1. **检查 gen\_config 函数**：
   * 找到 gen\_config 函数的定义，查看 gt 变量是如何被初始化的。
   * 确保在所有可能的代码路径中，gt 变量都有被赋值。
2. **确认数据加载**：
   * 检查 gt 的赋值语句，确保相关的数据（例如，文件、数组等）被成功加载。例如，如果 gt 是从文件读取的，请确保文件存在且格式正确。

**3 添加默认值**：

* 如果 gt 在某些情况下可能不会被赋值，可以考虑为 gt 添加一个默认值。例如：
* gt = None # 或者其他合适的默认值

然后在使用 gt 之前检查它是否为 None：

if gt is not None and gt.shape[1] == 8:

# 进行相关操作

else:

# 处理 gt 未被赋值的情况

Gen\_config函数:

def gen\_config(seq\_path, set\_type):

path, seqname = os.path.split(seq\_path)

if 'RGBT' in set\_type:

img\_list\_visible = sorted([seq\_path + '/visible/' + p for p in os.listdir(seq\_path + '/visible') if os.path.splitext(p)[1] in ['.jpg','.png','.bmp']])

img\_list\_infrared = sorted([seq\_path + '/infrared/' + p for p in os.listdir(seq\_path + '/infrared') if os.path.splitext(p)[1] in ['.jpg','.png','.bmp']])

gt = np.loadtxt(seq\_path + '/init.txt', delimiter=',')

elif 'GTOT' in set\_type:

img\_list\_visible = sorted([seq\_path + '/v/' + p for p in os.listdir(seq\_path + '/v') if os.path.splitext(p)[1] in ['.jpg','.png','.bmp']])

img\_list\_infrared = sorted([seq\_path + '/i/' + p for p in os.listdir(seq\_path + '/i') if os.path.splitext(p)[1] in ['.jpg','.png','.bmp']])

gt = np.loadtxt(seq\_path + '/init.txt')

elif 'LasHeR' in set\_type:

img\_list\_visible = sorted([seq\_path + '/visible/' + p for p in os.listdir(seq\_path + '/visible') if os.path.splitext(p)[1] in ['.jpg','.png','.bmp']])

img\_list\_infrared = sorted([seq\_path + '/infrared/' + p for p in os.listdir(seq\_path + '/infrared') if os.path.splitext(p)[1] in ['.jpg','.png','.bmp']])

gt = np.loadtxt(seq\_path + '/init.txt', delimiter=',')

if gt.shape[1] == 8:

x\_min = np.min(gt[:, [0, 2, 4, 6]], axis=1)[:, None]

y\_min = np.min(gt[:, [1, 3, 5, 7]], axis=1)[:, None]

x\_max = np.max(gt[:, [0, 2, 4, 6]], axis=1)[:, None]

y\_max = np.max(gt[:, [1, 3, 5, 7]], axis=1)[:, None]

gt = np.concatenate((x\_min, y\_min, x\_max - x\_min, y\_max - y\_min), axis=1)

return img\_list\_visible,img\_list\_infrared,gt

从 gen\_config 函数来看，出现 UnboundLocalError: local variable 'gt' referenced before assignment 错误的原因是，当 set\_type 不符合 'RGBT'、'GTOT' 或 'LasHeR' 这三个条件时，gt 没有被初始化。

通过检查gen\_config 函数，我发现必须将RGB\_T234数据集命名为RGBT234才可以被该函数识别，因此解决方式为，重命名RGB\_T234数据集文件夹为RGBT234；

命令：mv RGB\_T234 RGBT234

如果遇到缺少Result文件夹报错，使用以下命令新建结果文件夹：

mkdir -p ./Results/RGBT234+GTOT/RGBT234/GTOT\_ALL\_Transformer/

2 无法加载模型到GPU以及GPU运存不足

RuntimeError: No CUDA GPUs are available

(apf) user@user-Super-Server:~/chenzimeng/APFNet/AAAICode$ python

Python 3.8.0 (default, Nov 6 2019, 21:49:08)

[GCC 7.3.0] :: Anaconda, Inc. on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> torch.cuda.is\_available()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'torch' is not defined

>>> import torch

>>> torch.cuda.is\_available()

True

os.environ["CUDA\_VISIBLE\_DEVICES"] = '3'

将这一句代码（Run.py）下修改为可用的GPU编号，0，1，2，3；取决于你有几块可用的GPU；

RuntimeError: CUDA error: out of memory

CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect.

For debugging consider passing CUDA\_LAUNCH\_BLOCKING=1

Compile with `TORCH\_USE\_CUDA\_DSA` to enable device-side assertions.

GPU运存不足，可以通过nvidia-smi检查GPU运行状态，选取可用的GPU，然后修改

os.environ["CUDA\_VISIBLE\_DEVICES"] = '3' ；

到这里就可以成功运行模型了；

./modules/model\_tracking.py:1170: FutureWarning: You are using `torch.load` with `weights\_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights\_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add\_safe\_globals`. We recommend you start setting `weights\_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.

states = torch.load(model\_path)

load LanYangYang model.

load LanYangYang model end.

get\_learnable\_params odict\_keys(['fc4\_weight', 'fc4\_bias', 'fc5\_weight', 'fc5\_bias', 'fc6\_0\_weight', 'fc6\_0\_bias'])

get\_learnable\_params odict\_keys(['fc4\_weight', 'fc4\_bias', 'fc5\_weight', 'fc5\_bias', 'fc6\_0\_weight', 'fc6\_0\_bias'])

get\_learnable\_params odict\_keys(['fc4\_weight', 'fc4\_bias', 'fc5\_weight', 'fc5\_bias', 'fc6\_0\_weight', 'fc6\_0\_bias'])

Frame 1/613, Overlap 0.969, Score 10.696, Time 0.447

Frame 2/613, Overlap 0.963, Score 10.049, Time 0.358

Frame 3/613, Overlap 0.888, Score 9.220, Time 0.358

Frame 4/613, Overlap 0.881, Score 9.561, Time 0.358

另注：def get\_logger(filename, verbosity=1, name=None)函数的作用

这段代码定义了一个**用于创建日志记录器 (logger)** 的函数 get\_logger，它可以**将日志信息输出到文件和控制台，便于跟踪程序的执行状态。**在命令行参数

logger = get\_logger('./log/GTOT\_ALL\_Transformer.log')

中，可以指定所创建的log文件名称。

**代码的主要作用：**

1. **创建和配置日志记录器**：函数创建了一个名为 logger 的日志记录器，通过设置不同的日志级别，可以控制日志信息的详细程度。
2. **输出到文件和控制台**：日志信息会同时输出到指定的文件和控制台，方便用户在开发和调试过程中查看。

3.2 Train过程中

* **预备阶段**：你需要使用GTOT和RGBT234数据集，并且要使用带有“challenge tags”（挑战标签）的数据集。可以从[这个链接](https://github.com/mmic-lcl/Datasets-and-benchmark-code)找到数据集。首先运行prepro\_data.py脚本来生成一个.pkl文件，用于存储数据路径。在进行训练时，**需要在pretrain\_option.py文件中调整阶段训练的参数**。
* **第一阶段：**在第一阶段，你需要运行train\_stage1.py五次，因为我们有五个不同的属性分支。每次训练时，网络将使用特定的标签数据，首先加载预训练好的骨干网络模型参数，然后逐一添加特定的分支进行训练。请注意，这个阶段只需要保存每个分支的模型参数。
* **第二阶段**：在第一阶段中，你已经为五个分支生成了相应的挑战分支。在第二阶段，你可以加载骨干网络的参数以及所有分支的参数，进行属性融合模型（Attribute-Based Aggregation Fusion model）的训练。此时需要运行train\_stage2.py脚本。
* **第三阶段**：基于前两个阶段，只需要加载第二阶段生成的模型参数，进行第三阶段的训练，最终生成完整的模型。在这个阶段，需要运行train\_stage3.py脚本。

3.2.1 prepro\_data.py运行以及参数设置

位置：AAAIcode/pretrain/data/

py文件功能：这段 Python 代码主要实现的是对 GTOT 或 RGBT234 数据集进行预处理，**将图像数据路径及相应的标签信息保存到一个 .pkl 文件中**。该代码基于指定的数据集和挑战类型提取图像数据和标注信息，生成可用于模型训练的数据文件。

1. 设置数据集、挑战类型及路径

set\_type = 'GTOT' # 设置数据集类型为GTOT或RGBT234

seq\_home = '/'+set\_type +'/' # 数据集的根目录路径

challenge\_type = 'ALL' # 设置挑战类型。选项包括: Fast Motion, Occlusion, Scale, Illumination Variation, Thermal Crossover, ALL（完整数据集）

这里根据set\_type选择使用 GTOT 或 RGBT234 数据集，并且根据challenge\_type指定要处理的挑战类型

2. 根据数据集类型设置路径

if set\_type=='GTOT':

seqlist\_path = '/DATA/gtot.txt' # GTOT 数据集的序列列表文件路径

output\_path = '/DATA/yangmengmeng/MyCode/MDNet\_CAT\_SK\_Transformer/pretrain/data/GTOT\_'+challenge\_type+'.pkl' # 输出的 pkl 文件路径

elif set\_type == 'RGBT234':

seqlist\_path = '/DATA/RGBT234.txt' # RGBT234 数据集的序列列表文件路径

output\_path = '/DATA/yangmengmeng/MyCode/MDNet\_CAT\_SK\_Transformer/pretrain/data/RGBT234\_'+challenge\_type+'.pkl'

根据不同的数据集类型设置不同的序列列表文件路径（seqlist\_path）和输出文件路径（output\_path）。

3. 读取序列列表文件

with open(seqlist\_path,'r') as fp:

seq\_list = fp.read().splitlines()

通过读取seqlist\_path中的序列列表，将其逐行读取到seq\_list中，每一行代表一个数据序列的名称。

4 构建数据字典

data = OrderedDict()

for i, seqname in enumerate(seq\_list):

初始化一个有序字典data来存储每个序列的图像路径和标注信息，遍历seq\_list中的序列名称。

5 加载可见光和红外图像及标签

根据数据集类型，加载 GTOT 或 RGBT234 数据集中的可见光图像 (img\_list\_v) 和红外图像 (img\_list\_i)，并读取每个序列的目标检测标签 (gt)。

**GTOT 数据集：**

if set\_type=='GTOT':

seq\_path = seq\_home+seqname

img\_list\_v = sorted([p for p in os.listdir(seq\_path+'/v') if os.path.splitext(p)[1] in ['.jpg','.bmp','.png']])

img\_list\_i = sorted([p for p in os.listdir(seq\_path+'/i') if os.path.splitext(p)[1] in ['.jpg','.bmp','.png']])

img\_list\_v = [os.path.join(seq\_home, seqname, 'v', img) for img in img\_list\_v]

img\_list\_i = [os.path.join(seq\_home, seqname, 'i', img) for img in img\_list\_i]

gt = np.loadtxt(seq\_path + '/init.txt')

这部分代码根据不同的数据集类型选择合适的文件夹来加载可见光和红外图像，并读取目标检测标签gt。GTOT 数据集中的图片存储在/v和/i文件夹中，RGBT234 数据集则存储在/visible和/infrared文件夹中。

6 处理目标标签

if gt.shape[1] == 8:

x\_min = np.min(gt[:, [0, 2, 4, 6]], axis=1)[:, None]

y\_min = np.min(gt[:, [1, 3, 5, 7]], axis=1)[:, None]

x\_max = np.max(gt[:, [0, 2, 4, 6]], axis=1)[:, None]

y\_max = np.max(gt[:, [1, 3, 5, 7]], axis=1)[:, None]

gt = np.concatenate((x\_min, y\_min, x\_max - x\_min, y\_max - y\_min), axis=1)

这段代码将原始的目标边界标签（有8个点表示目标的四个顶点）转换为标准的边界框格式，即(x\_min, y\_min, width, height)格式。

7 挑战标签的处理（可选）

if challenge\_type == 'FM':

challenge\_inf = 'fast\_motion.tag'

elif challenge\_type == 'OCC':

challenge\_inf = 'occlusion.tag'

elif challenge\_type == 'SC':

challenge\_inf = 'size\_change.tag'

elif challenge\_type == 'ILL':

challenge\_inf = 'illum\_change.tag'

elif challenge\_type == 'TC':

challenge\_inf = 'thermal\_crossover.tag'

elif challenge\_type == 'ALL':

data[seqname] = {'images\_v':img\_list\_v, 'images\_i':img\_list\_i, 'gt':gt}

根据challenge\_type（如快速运动、遮挡、大小变化等），选择相应的挑战标签文件。如果challenge\_type设置为ALL，将所有图像路径和目标标签存入数据字典。

8 过滤挑战标签

if challenge\_type!='ALL':

try:

challenge\_label = np.loadtxt(os.path.join(seq\_home, seqname, challenge\_inf))

challenge\_label = challenge\_label.tolist()

assert len(challenge\_label) == len(img\_list\_v), 'len(challenge\_label)!=len(img\_list\_v):'

challenge\_label = np.array(challenge\_label)

idx = np.ones(len(img\_list\_v), dtype=bool)

idx\*=(challenge\_label>0)

img\_list\_v = np.array(img\_list\_v)

img\_list\_i = np.array(img\_list\_i)

gt = gt[idx,:]

img\_list\_v = img\_list\_v[idx]

img\_list\_v = img\_list\_v.tolist()

img\_list\_i = img\_list\_i[idx]

img\_list\_i = img\_list\_i.tolist()

print(seqname,challenge\_type,len(img\_list\_v),len(gt))

if len(img\_list\_v)>0:

data[seqname] = {'images\_v': img\_list\_v, 'images\_i': img\_list\_i,'gt': gt}

else:

print (seqname,'length not enough!')

except:

print (seqname,'no',challenge\_type)

如果指定了挑战标签类型，代码会加载相应的标签文件，并且根据挑战标签进行筛选，保留满足挑战条件的图像和标签。

9 保存处理后的数据

with open(output\_path, 'wb') as fp:

print ('output\_path',output\_path)

pickle.dump(data, fp)

最终，将所有处理后的数据存储到指定的output\_path文件中，并使用pickle进行序列化保存。

运行命令：在(apf) user@user-Super-Server:~/chenzimeng/APFNet/AAAICode$下

python ./pretrain/data/prepro\_data.py

3.2.2 prepro\_data.py运行报错

Traceback (most recent call last):

File "./pretrain/data/prepro\_data.py", line 35, in <module>

assert len(img\_list\_v) == len(gt) == len(img\_list\_i), "Lengths do not match!!"

AssertionError: Lengths do not match!!

出现这个 AssertionError: Lengths do not match!! 错误意味着在你的 GTOT 或 RGBT234 数据集中，img\_list\_v（可见光图像列表）、img\_list\_i（红外图像列表）和 gt（ground truth，标注文件）的长度不一致。

**检查文件夹结构和文件数量是否匹配**

确保每个序列文件夹下的可见光（v 或 visible）、红外（i 或 infrared）图像数量与 init.txt 中的标注文件一致。

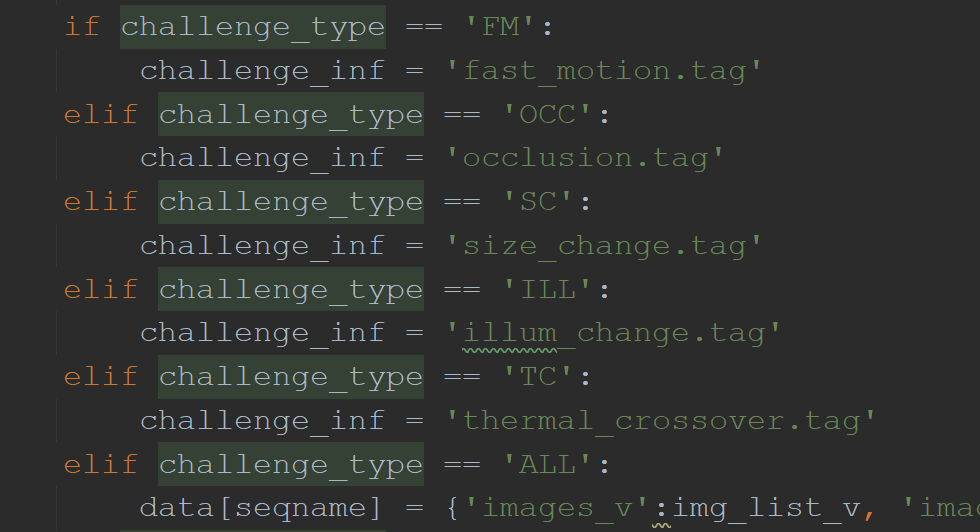
* img\_list\_v 是可见光图像的列表。
* img\_list\_i 是红外图像的列表。
* gt 是标注文件加载的数据，通常是目标检测框的坐标信息。

需要确保每个序列中的这三者长度一致。

解决：通过排查GTOT数据集文件，发现在OccCar-1文件夹下的的35号、36号、37号、38号、42号文件出现重复，可能是由于百度网盘下载时出现的问题，而导致图片数量和标签数量不匹配。删除掉多余文件后，可以运行prepro\_data.py文件了。

注意，需要运行该文件6次，对应6种不同的挑战类型；

challenge\_type = 'ALL' # set challenge type. FM~Fast



3.2.3 train\_stage1.py文件

在第一阶段，你需要运行train\_stage1.py五次，因为我们有五个不同的属性分支。

实际上，我运行了六次，包括 ALL，FM, SC, OCC, ILL,TC；

应该只用运行5次：即FM, SC, OCC, ILL,TC;ALL权重文件应该放在stage2中生成。

# Init dataset

## set image directory

if opts['set\_type'] == 'RGBT234\_ALL':

img\_home = './RGBT234/'

data\_path = './pretrain/data/RGBT234\_ALL.pkl'

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

elif opts['set\_type'] == 'GTOT\_all':

img\_home = './GTOT/'

data\_path = './pretrain/data/GTOT\_ALL.pkl'

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

elif opts['set\_type'] == 'RGBT234\_FM':

img\_home = './RGBT234/'

data\_path = './pretrain/data/RGBT234\_FM.pkl'

elif opts['set\_type'] == 'RGBT234\_SC':

img\_home = './RGBT234/'

data\_path = './pretrain/data/RGBT234\_SC.pkl'

elif opts['set\_type'] == 'RGBT234\_OCC':

img\_home = './RGBT234/'

data\_path = './pretrain/data/RGBT234\_OCC.pkl'

elif opts['set\_type'] == 'RGBT234\_ILL':

img\_home = './RGBT234/'

data\_path = './pretrain/data/RGBT234\_ILL.pkl'

elif opts['set\_type'] == 'RGBT234\_TC':

img\_home = './RGBT234/'

data\_path = './pretrain/data/RGBT234\_TC.pkl'

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

elif opts['set\_type'] == 'GTOT\_FM':

img\_home = './GTOT/'

data\_path = './pretrain/data/GTOT\_FM.pkl'

elif opts['set\_type'] == 'GTOT\_SC':

img\_home = './GTOT/'

data\_path = './pretrain/data/GTOT\_SC.pkl'

elif opts['set\_type'] == 'GTOT\_OCC':

img\_home = './GTOT/'

data\_path = './pretrain/data/GTOT\_OCC.pkl'

elif opts['set\_type'] == 'GTOT\_ILL':

img\_home = './GTOT/'

data\_path = './pretrain/data/GTOT\_ILL.pkl'

elif opts['set\_type'] == 'GTOT\_TC':

img\_home = './GTOT/'

data\_path = './pretrain/data/GTOT\_TC.pkl'

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

另外，我在命令行参数相关部分的设置如下：

#We only save the attribute branches

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

#Change it by yourslef to train different attribute branches

parser.add\_argument("-set\_type", default = 'GTOT\_ALL')

#your path for saving attribute branch

parser.add\_argument("-model\_path", default ="/home/user/chenzimeng/APFNet/AAAICode/models/GTOT\_branches/GTOT\_ALL.pth", help = "model path")

#load the backbone model GTOT.pth use the GTOT datasets pretain, and the RGBT234.pth usze the RGBT234 dataset pretrain the backbone.

parser.add\_argument("-init\_model\_path", default="/home/user/chenzimeng/APFNet/AAAICode/models/GTOT.pth")

parser.add\_argument("-batch\_frames", default = 8, type = int)

parser.add\_argument("-lr", default=0.0001, type = float) #you can set it by yourself

parser.add\_argument("-batch\_pos",default = 32, type = int)

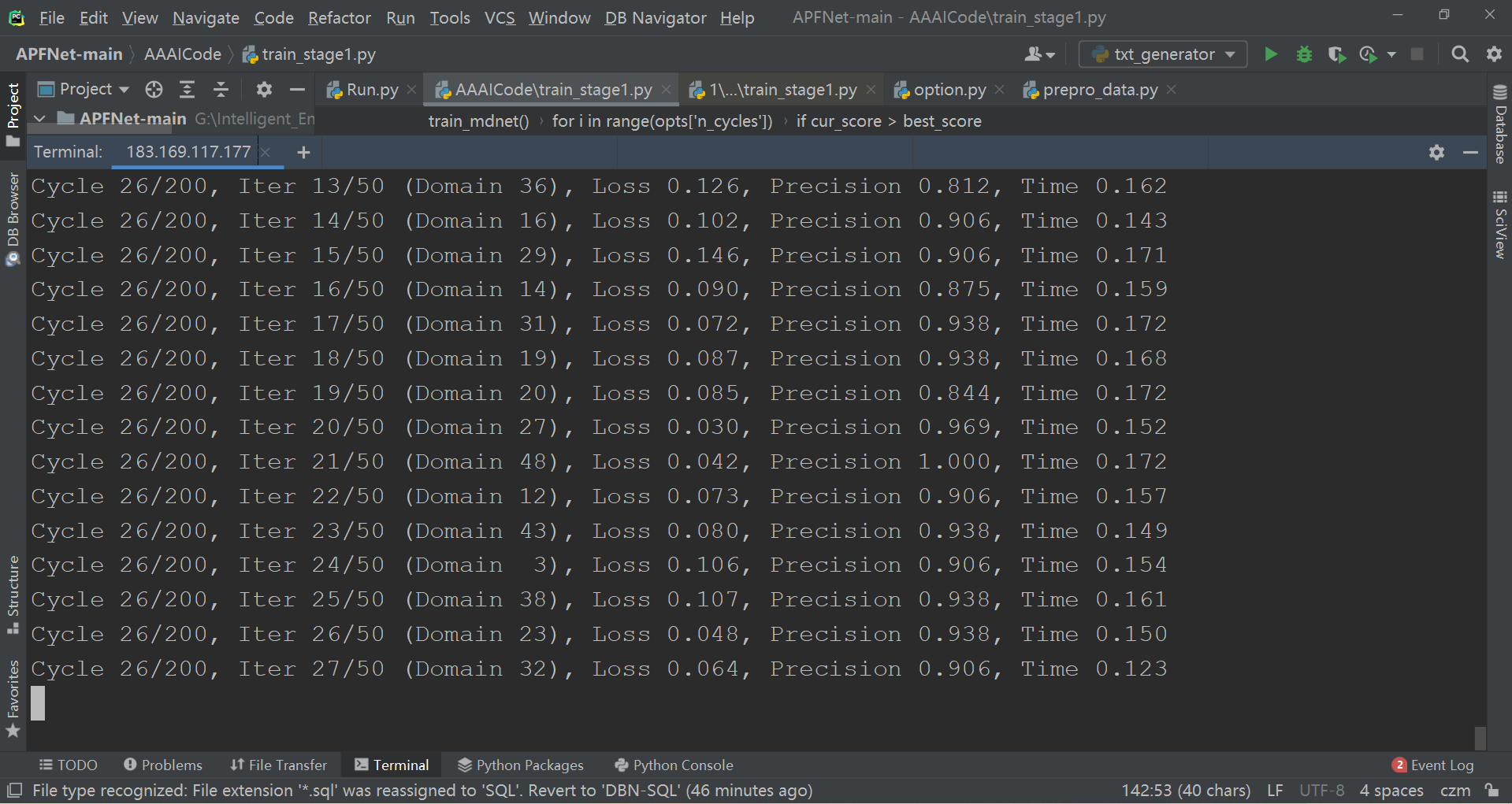
parser.add\_argument("-batch\_neg", default = 96, type = int)

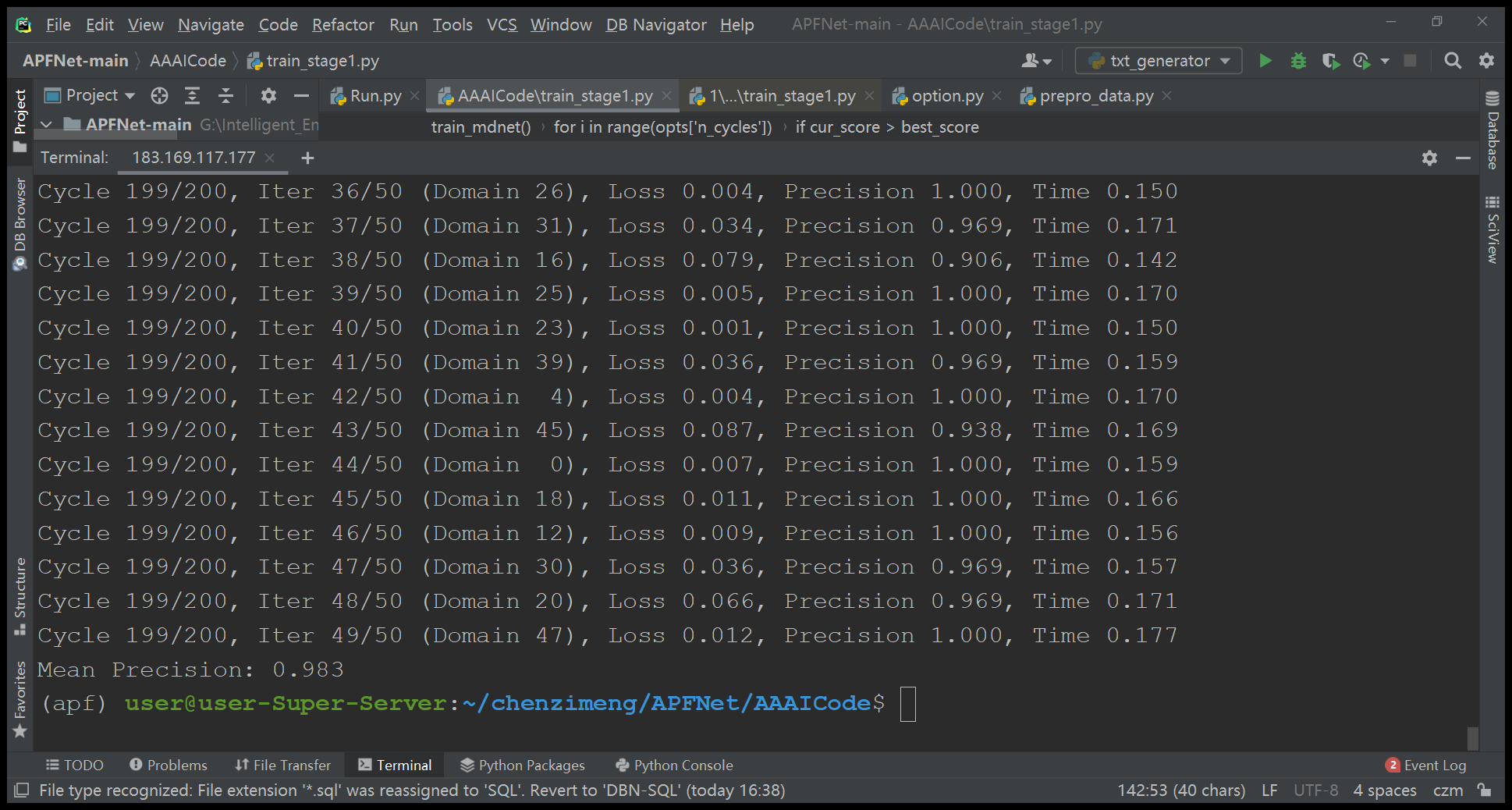
parser.add\_argument("-n\_cycles", default = 200, type = int ) #you can set it by yourself

args = parser.parse\_args()

**需要注意的是，-model\_path在源码中，采用torch.save(states, opts['model\_path'])进行路径读取，这里一定要指定文件路径，并详细到权重文件名和后缀名，而不是只有文件夹的名称；**

**/home/user/chenzimeng/APFNet/AAAICode/models/GTOT\_branches/GTOT\_ALL.pth 不然会出现RuntimeError无法打开文件的报错。**

****

****

训练过程截图

3.2.4 train\_stage2.py文件

You have spawned 5 corresponding challenge branches in one phase.In the second stage,you can load the backbone and all branches parameters for training the Attribute-Based Aggregation Fusion model. you should run the **train\_stage2.py**

注意：在train\_stage2.py文件中的命令行参数设置如下：

#We save the attributes ,the backbone ,the aggregation in one model

if \_\_name\_\_ == "\_\_main\_\_":

logger = get\_logger('./log/trainGTOTSKALL.log')

parser = argparse.ArgumentParser()

#set yourdataset

parser.add\_argument("-set\_type", default = 'GTOT\_ALL')

#Saving model path

parser.add\_argument("-model\_path", default ="/home/user/chenzimeng/APFNet/AAAICode/models/GTOT\_Fusion/GTOT\_ALL.pth", help = "model path")

#your backbone model

parser.add\_argument("-init\_model\_path", default="/home/user/chenzimeng/APFNet/AAAICode/models/GTOT.pth")

parser.add\_argument("-batch\_frames", default = 8, type = int)

parser.add\_argument("-lr", default=0.0001, type = float) #set it bysourself

parser.add\_argument("-batch\_pos",default = 32, type = int)

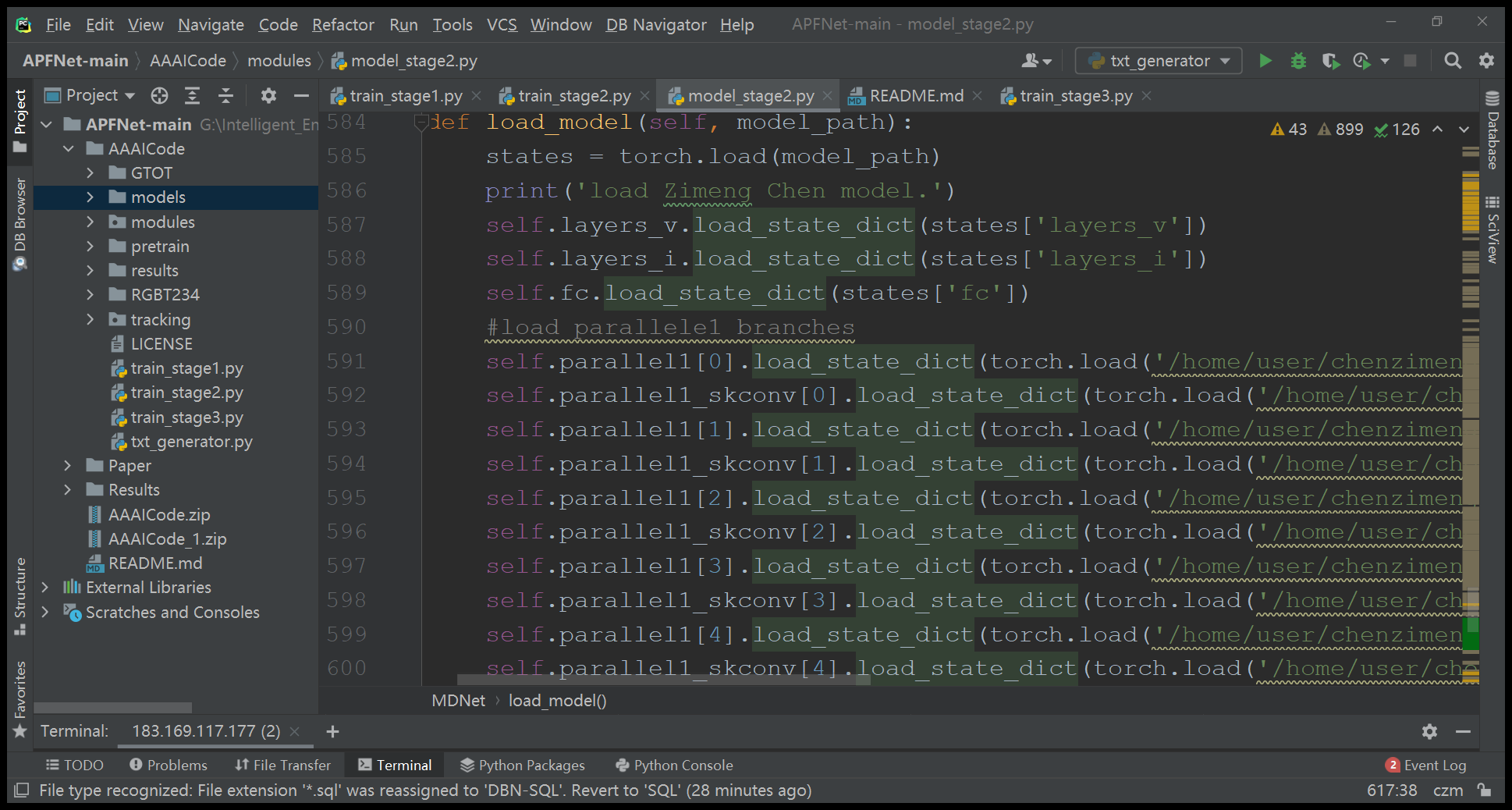
parser.add\_argument("-batch\_neg", default = 96, type = int)

parser.add\_argument("-n\_cycles", default = 500, type = int ) #set it bysourself

args = parser.parse\_args()

另外，module/model\_stage2.py中的代码也要修改如下：

584行： def load\_model(self, model\_path):



修改成自己训练好的分支权重文件路径；

3.2.5 train\_stage3.py文件

On the basis of the two-stage, **only the model parameters generated by the second stage need to be loaded**, and the final model can be generated in the three-stage training. you should run the **train\_stage3.py**

报错解决：

1

states = torch.load(model\_path)

load LanYangYang model.

load LanYangYang model error!

load VID model.

Traceback (most recent call last):

File "./modules/model\_stage3.py", line 1353, in load\_model

self.parallel1.load\_state\_dict(states['parallel1'])

KeyError: 'parallel1'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):

File "train\_stage3.py", line 219, in <module>

train\_mdnet(opts)

File "train\_stage3.py", line 97, in train\_mdnet

model = MDNet(opts['init\_model\_path'], K)

File "./modules/model\_stage3.py", line 356, in \_\_init\_\_

self.load\_model(model\_path)

File "./modules/model\_stage3.py", line 1366, in load\_model

shared\_layers = states['shared\_layers']

KeyError: 'shared\_layers'

**错误分析：**

1. **KeyError: 'parallel1'**： 当程序试图加载模型状态时，期望从 states 中找到 parallel1 这个键（即 states['parallel1']），但在加载的模型文件中找不到对应的键。这可能意味着：
   * 模型文件保存时没有包含 parallel1 的信息。
   * 加载的模型与当前的模型架构不匹配。
2. **KeyError: 'shared\_layers'**： 程序在处理异常时，尝试从 states 中获取 shared\_layers，但是在模型状态文件中也找不到这个键。这同样可能是由于保存的模型文件不包含该部分，或者是模型架构发生了变化。

**可能原因：**

* **模型架构不匹配**：训练模型时和加载模型时的架构不一致，导致模型文件中没有保存或不支持当前需要的层（如 parallel1 和 shared\_layers）。
* **模型文件不完整或错误**：保存的模型文件可能不完整，或者是在某个阶段中只保存了部分网络权重，而非完整的模型结构。

因此，我分析在train\_stage3.py文件中命令行参数部分的设置，发现了以下问题：

if \_\_name\_\_ == "\_\_main\_\_":

logger = get\_logger('./log/Train\_RGBT234\_ALL\_Transformer.log')

parser = argparse.ArgumentParser()

parser.add\_argument("-set\_type", default = 'GTOT\_ALL')

parser.add\_argument("-model\_path", default ="/home/user/chenzimeng/APFNet/AAAICode/models/GTOT\_final/GTOT\_ALL\_Final.pth", help = "model path")

parser.add\_argument("-init\_model\_path", default="/home/user/chenzimeng/APFNet/AAAICode/models/GTOT\_Fusion/GTOT\_ALL.pth.pth")

parser.add\_argument("-batch\_frames", default = 8, type = int)

parser.add\_argument("-lr", default=0.0001, type = float)

parser.add\_argument("-batch\_pos",default = 32, type = int)

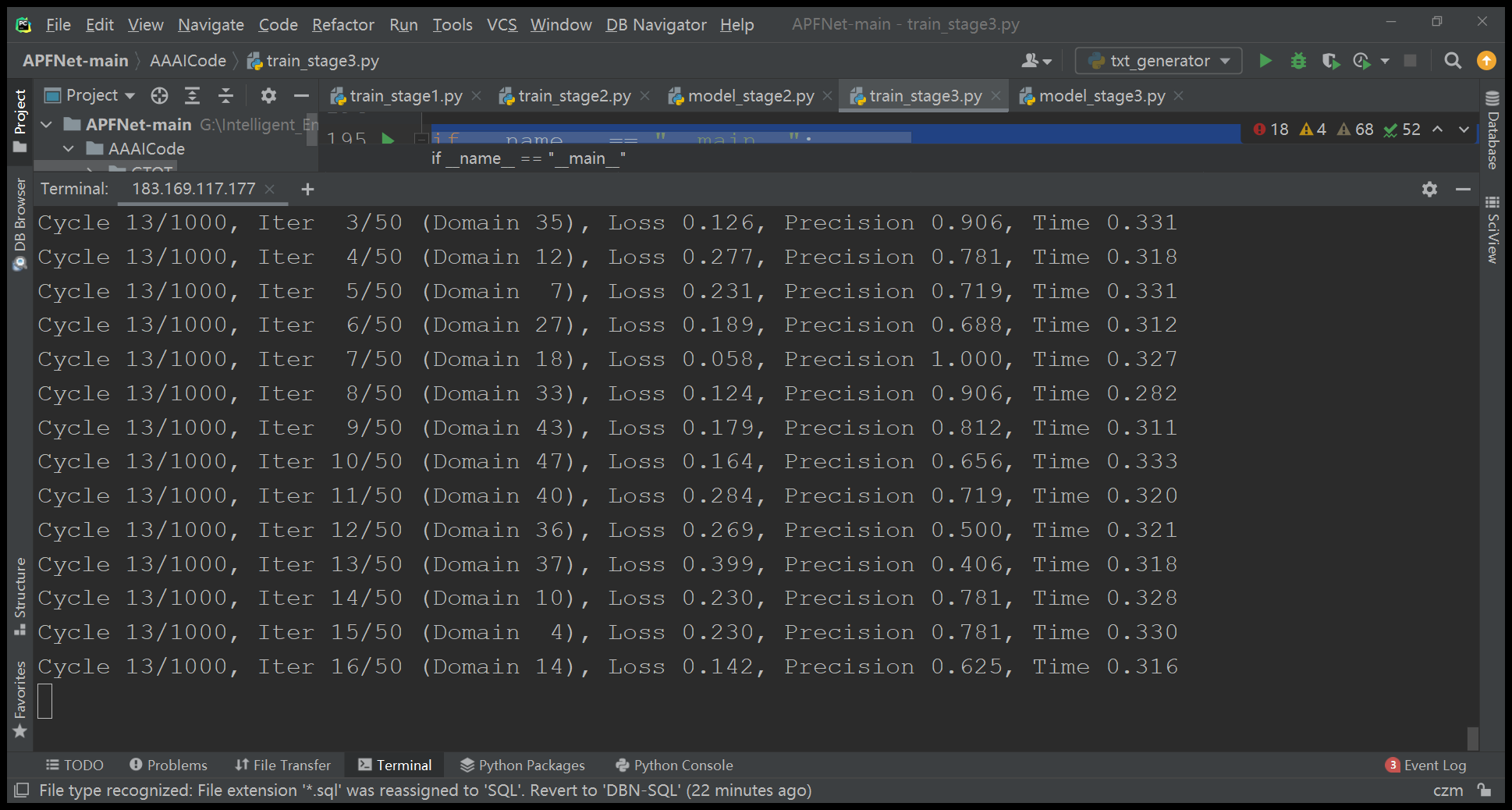
parser.add\_argument("-batch\_neg", default = 96, type = int)

parser.add\_argument("-n\_cycles", default = 1000, type = int )

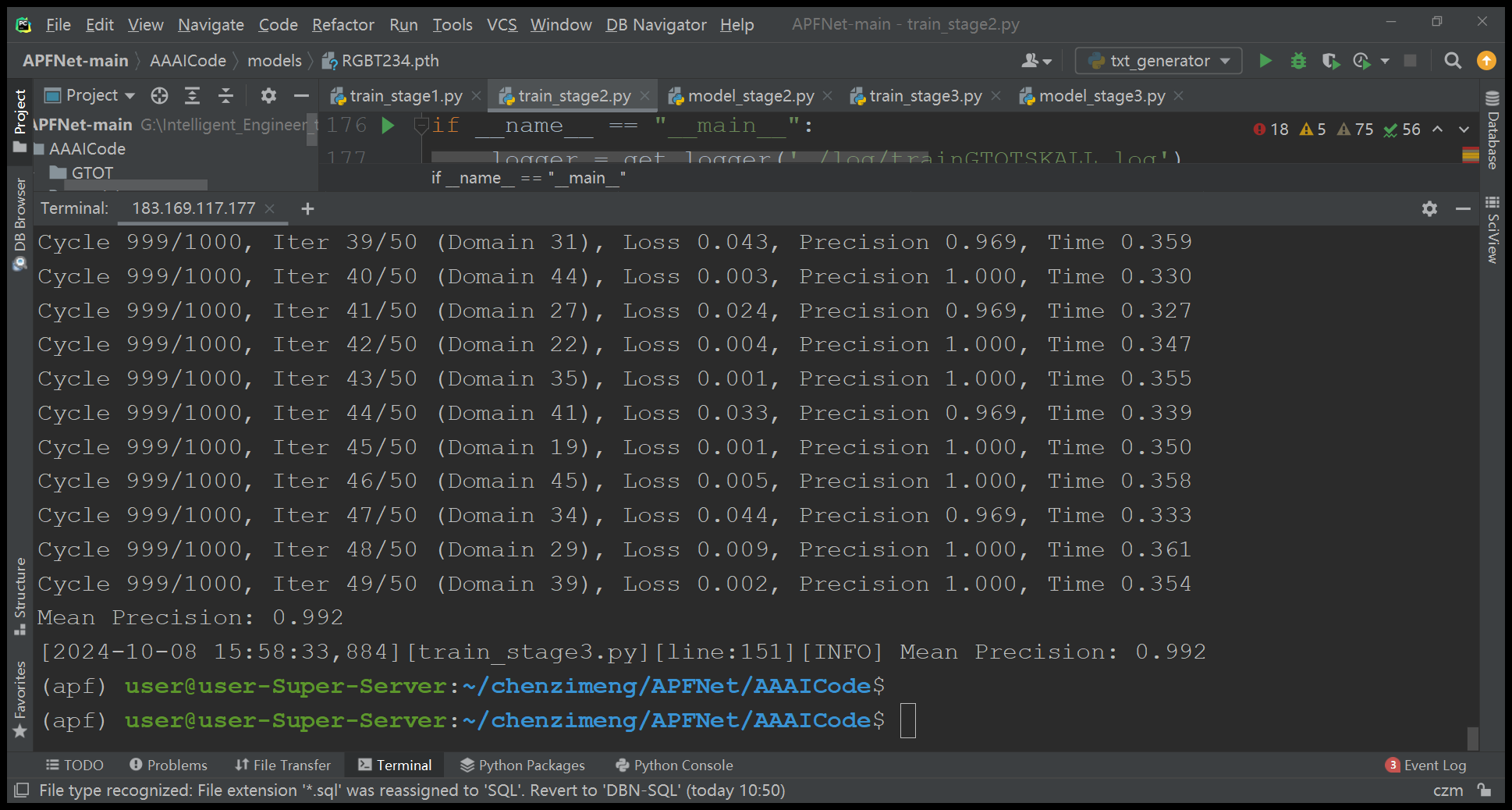
args = parser.parse\_args()

在-init\_model\_path那一栏，应该填入第二阶段生成的训练模型（only the model parameters generated by the second stage need to be loaded），而不是预训练权重文件，另外需要注意，我这里文件的名称是GTOT\_ALL.pth.pth

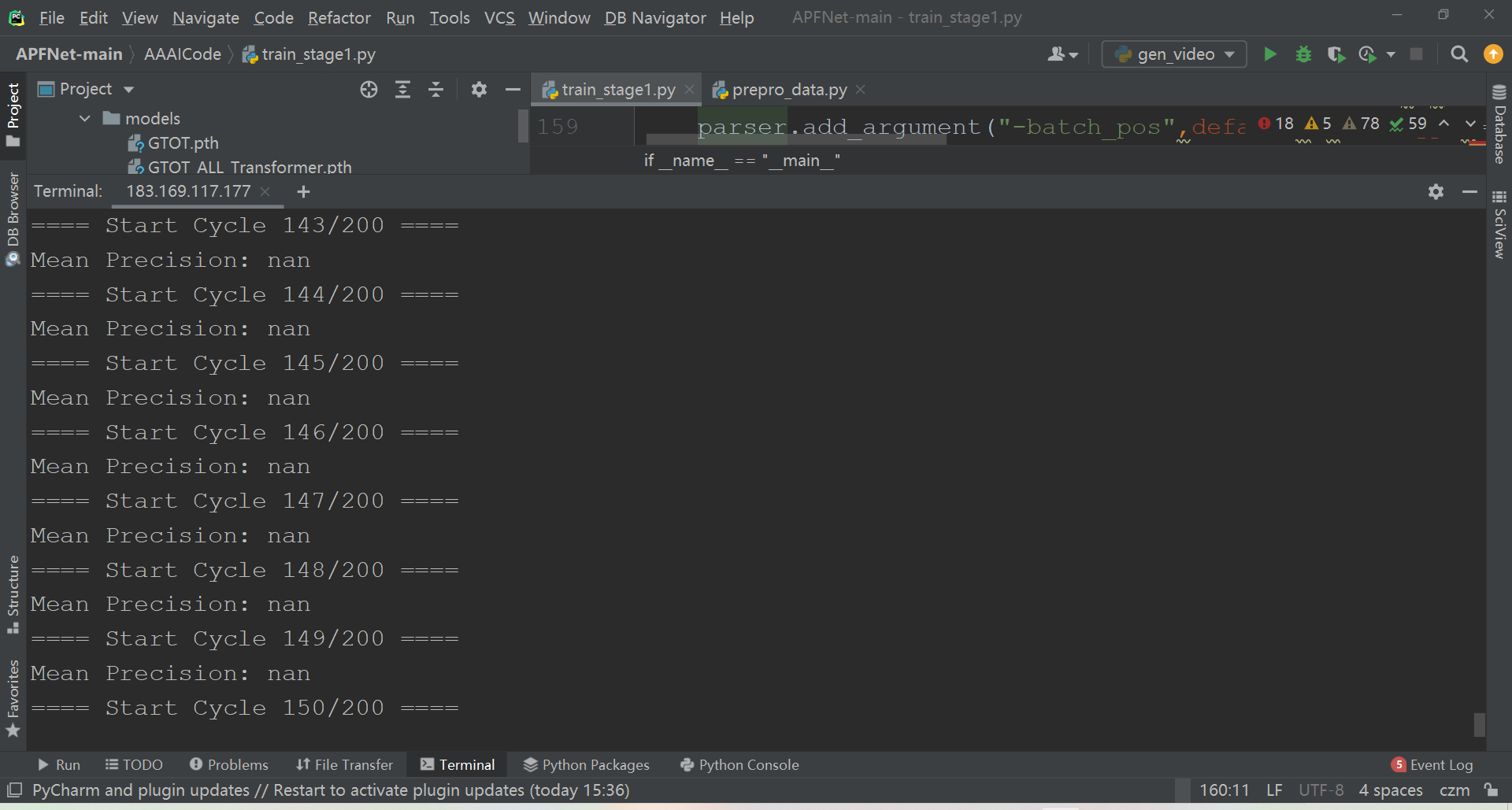
修改之后，可以进行第三阶段的训练了：



阶段三训练过程示意图



附录：RGBT234训练报错：



**利用RGBT234数据集做训练的时候，出现图上错误：Nan**

分析：在训练网络时出现 Mean Precision: nan 的情况，通常是由于计算中出现了无效的数值（例如，零除、无穷大、缺失值等），导致精度无法被计算。

**1. 零除问题**

* 可能在计算 precision（精度）时，分母出现了零值，导致除零错误。例如，precision = TP / (TP + FP)，如果 TP + FP = 0，则会出现零除问题，进而导致 nan。
* **解决方法**：在计算 precision 时，确保分母不为零。如果 TP + FP = 0，可以将 precision 设置为 0，或者跳过该计算：

**2. 数据丢失或无效值**

* 如果输入数据包含无效值（例如 NaN 或 Inf），可能会导致精度的计算无法进行。要检查输入数据是否存在异常值。

**3. 模型输出异常**

* 如果模型在训练过程中参数不稳定（例如，学习率过大导致梯度爆炸），可能会导致模型的输出值异常，精度无法计算。

**解决方法**：

* **降低学习率**：尝试降低学习率，以避免模型的输出发散。
* **梯度剪裁**：在反向传播时对梯度进行裁剪，避免梯度爆炸

**4. 样本不平衡或数据集问题**

* 如果训练数据中某些类别的数据量极少，可能导致精度的计算出现问题，特别是在某个训练周期内该类别没有任何正样本。
* **解决方法**：对数据集进行重新采样或者使用加权损失函数，确保每个类别在训练中的权重合适。

**5. 训练超参数问题**

* 训练过程中某些超参数（如 batch size、学习率、权重初始化等）可能设置不当，导致训练不收敛或者计算结果异常。
* **解决方法**：调整超参数，尤其是学习率和 batch size，查看问题是否得到改善。

**6. 损失函数设计问题**

* 如果损失函数中有不合理的计算或者没有合适的正则化项，可能会导致梯度爆炸或收敛失败，从而导致精度计算为 nan。
* **解决方法**：检查损失函数的设计是否合理，确保其在数值计算中稳定。

在终端指定训练所用的GPU：

CUDA\_VISIBLE\_DEVICES=3 python train\_stage1.py

CUDA\_VISIBLE\_DEVICES=3 python train\_stage2.py

CUDA\_VISIBLE\_DEVICES=3 python train\_stage3.py

python ./pretrain/data/prepro\_data.py

**/home/user/chenzimeng/APFNet/AAAICode**