数据处理
实验报告

作业题目	Python 数据处理方法
学生姓名	卜德华
学　　院	计算机学院
专业班级	大数据 2201 班
老　　师	鲁鸣鸣
学　　号	8208220314

2025 年 1 月 13 日

目录

一、实验描述

（一）课设目的

本次实验将基于船舶视频重识别这一视觉任务，需完成对所给定模型的性能测试。通过此次实验，可了解视频重识别任务的基本背景，可对深度学习视觉任务数据集构建流程建立一定认知，可实践爬虫技术，可实践最新的目标检测技术。

（二）课设开发环境和工具

- 操作系统：Windows11
- 开发语言：python
- IDE：pycharm
- 开发工具：Scrapy、re、PyTorch 等

（三）课设要求

1、课设背景

计算机视觉任务中的目标重识别（Object Re-identification ）任务是在摄像头等视觉设备通过目标识别算法 识别出目标后，判断识别的目标是否是之前出现过的一种视觉任务。为了判断目标是否出现过，需要存储之前识别过的目标。因此，目标重识别任务往往要事先构建已识别目标数据库。这样目标重识别任务就可以在目标识别算法识别出目标后，将当前识别的目标与已识别目标数据库做相似性匹配，从而确定当前识别的目标是之前识别的具体哪个目标。

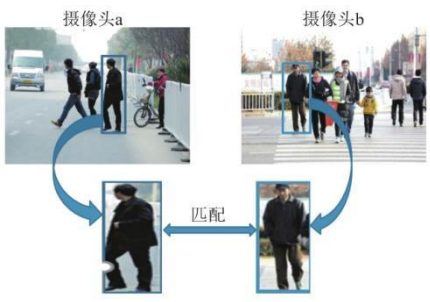

图 1.1 重识别示例

之所以需要目标重识别，是因为在公共安全等场景中，视频监控系统得到了大量普及，通过视频监控系统可以直观的再现目标场景，可作为公安侦破案件的强力辅助。在执法部门的工作中，目标的识别和定位是及其关键的一步。行人/车辆的重识别主要目的是针对出现在监控摄像头内的某个目标行人/车辆，准确快速地从监控网络其他摄像头内的大量行人中将这个目标行人/车辆标识出来。图 1 展示了一个行人重识别的例子。

为了让同学们体验一个重识别任务工作的全流程，我们将针对舰船目标做视频重识别任务。同学们在本次课程设计中需要完成包括测试集构建及重识别模型测试等工作。

2、课程的基本要求

（1）完成爬取舰船视频数据爬取。
（2）完成舰船视频清洗。
（3）完成测试集构建。
（4）使用自建测试集测试所给定重识别模型。

二、课设具体内容

（一）爬取数据集

为了满足模型测试的需要，我们需要大量的船舶视频序列数据。然而，手动搜寻下载数据的工作量非常大。为了更高效完成此项任务，我们使用爬虫来获取原始视频。以下内容将详细介绍如何基于 scrapy 框架编写爬虫程序，并在视觉中国网站上爬取船舶视频。

- 目标：爬取船舶视频数据和标题名称（关键字： 船，舶，舰）
- 爬取网站： http://www.vcg.com
- 语言：python
- 框架：Scrapy

（二）数据清洗

爬取的数据可能有多条船，也可能没有船。为了筛选我们想要的数据，我们可以使用开源库 YOLO8 模型来对爬取的数据进行处理。这个模型可以帮助我们识别图像中的船只，通过使用 YOLO8 模型，我们可以更有效地筛选出包含船只的数据。

（三）测试集构建

在利用 YOLOv8 清洗完船舶视频数据后，便可着手开始构建船舶重识别测试集。若视频文件命名为中文，首先需要将所有视频文件重命名，视频名需全为数字，且每个视频名应当保证唯一。若视频未存放在同一个文件夹中，需将所有视频存放至同一目录下。

在完成上述两个验证后，可正式开始构建测试集。测试集构建将分为两步完成，首先是基于视频数据进行抽帧，从而得到若干个图片序列（可称之为轨迹）；而后便可以基于图片序列构建测试集，包括生成所需文本文件以及重组图片存储架构。

（四）重识别模型测试

在完成测试集的构建之后，便可以尝试将其用于检验所给视频重识别模型的性能，首先需要将示例仓库 3 中的代码 clone 至本地，而后请参考所给仓库中的 readme 所示进行模型性能验证。

三、需求分析

（一）数据爬取需求

1、 使用爬虫技术（Scrapy 框架）爬取视频链接、标题、视频存储路径等数据。
2、 每个爬取的视频需存储为本地文件，文件名需与标题名称一致，且存储路径需要支持中文字符。
3、 爬取数量：至少 500 条船舶视频数据。
4、 爬取的视频需要具备船舶相关内容，标题包含“船”、“舶”、“舰”等关键词。

（二）数据清洗需求

1、 使用 YOLOv8 目标检测模型对每个视频进行检测。
2、 每个视频需通过 YOLOv8 模型识别，筛选出包含船舶的帧。
3、 无效帧或不包含船舶的帧需被去除，只保留有效帧。
4、 需要使用 OpenCV 处理视频帧，确保清洗后的视频内容符合重识别测试的要求。

（三）测试集构建需求

1、 对每个视频进行抽帧处理，每隔 8 帧保存一张图片，形成一条轨迹。
2、 生成每条轨迹对应的图片序列，图片命名格式为 V 船舶 ID_T 轨迹 ID_F 图片 ID.jpg，确保每个图片文件名唯一。
3、 所有图片需存储在统一的文件夹结构中。
4、 生成必要的文本文件：
   - test_names.txt：列出所有测试集中的图片名称，每行一个图片名。
   - name_map.txt：记录图片的原始文件名与新命名之间的映射关系。
   - tracks_test_info.mat：记录每条轨迹的所有图片信息，存储轨迹的起始和结束行数及船舶 ID。
   - query_IDX.mat：记录用于查询的轨迹编号。

（四）重识别模型测试需求

1、 使用给定的重识别模型，加载并测试构建的测试集。
2、 对测试集进行性能评估，包括准确率、召回率、精度等指标。
3、 提供模型的详细测试结果，记录测试过程中可能出现的误识别情况和性能瓶颈。

（五）其他需求

1、环境搭建

完成 Scrapy 的安装，熟悉创建项目、生成爬虫、启动爬虫等等基本命令，在命令行中正确执行这些命令，构建出符合要求的项目目录结构；利用 Anaconda 搭建对应 python 虚拟环境，在其中完成 pytorch 和 cuda 等的安装，为后续模型使用做准备。

2、库安装

安装 ultralytics 及其依赖库，安装 cv2 工具包等，按照给定的链接和命令准确执行操作，确保相关环境配置正确。

3、下载模型

下载指定的已训练好的开源模型，参考官方文档了解其安装和用法，为筛选做准备。

四、概要设计

（一）设计思路

1、进行视频的爬取：修改提供的项目代码，并基于修改后的代码从视觉中国网站爬取船舶视频，提取相关数据（如视频链接、标题等），将视频保存到预设的路径中；接下来，对爬取到的视频进行清洗。

2 在完成 cuda、pytorch 和 yolov8 等相关环境配置后，基于 yolov8 官方项目的代码，添加自定义的 Python 脚本来对爬取的视频进行帧级处理，筛选出有效的视频，并将其保存到指定路径。

3 继续添加 Python 脚本，完成视频重命名、抽帧处理，将每个视频转换成轨迹序列，并生成 txt 文件和 mat 文件。

4 进行重识别模型的测试与分析：从官网下载测试项目，并根据项目中的依赖要求安装相应的 Python 包，确保版本一致，修改测试文件中的路径后运行 test.py，成功运行后，可以在指定路径得到包含测试结果的文本文件。

（二）总计流程图

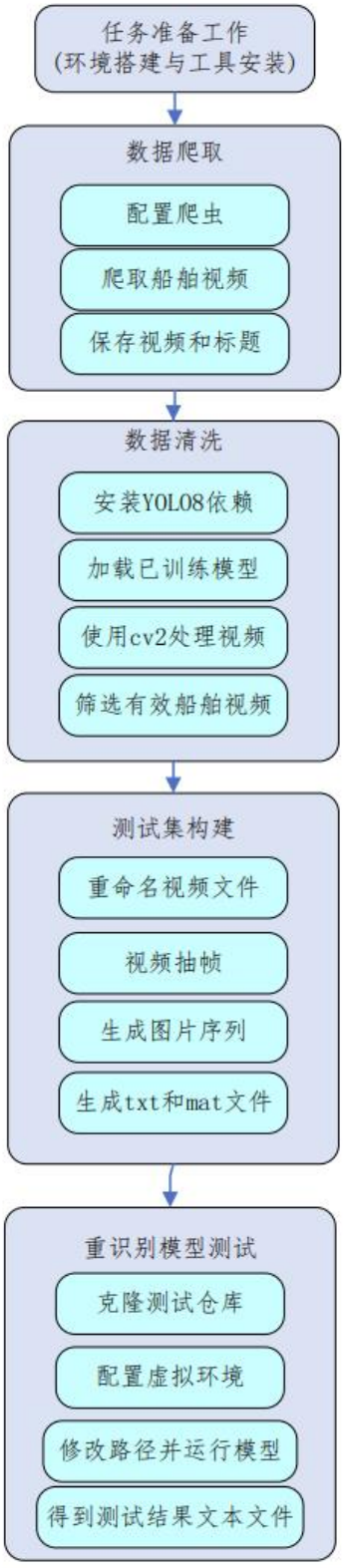

图 2.1 总计流程图

五、详细设计

（一）爬取数据

1、安装 Scrapy

使用 pip install scrapy 安装 Scrapy。

2、创建 Scrapy 项目

（1）使用命令 scrapy startproject getshipVideo 创建项目。
（2）使用命令 scrapy genspider getvideo www.vcg.com 生成爬虫。

3、编写爬虫逻辑

（1）在 getVideo.py 中编写爬虫逻辑

- name：设置爬虫名称，用于在命令行中启动爬虫时识别。
- allowed_domains：限制爬虫爬取的域名，防止爬虫爬取到其他无关网站。
- page 和 url：用于爬取特定页面的视频数据。
- start_urls：将构建的起始 URL 添加到起始 URL 列表中，Scrapy 将从这些 URL 开始爬取。
- parse 方法：Scrapy 爬虫的主要入口点。

（2）parse 方法

使用 XPath 表达式提取页面中的视频列表，每个视频项包含链接和描述。遍历提取的视频列表，对每个视频项进行处理：

- 提取视频链接和描述，并进行简单的文本处理（去除换行符、回车符和空格）。
- 检查描述中是否包含特定关键词（“船”、“舰”、“ship”），如果包含，则发起对视频详情页的请求。
- 使用 scrapy.Request 发起对视频详情页的请求，并将当前视频项和描述传递到详情页处理函数 parse_detail。
- 实现分页爬取，如果当前页码小于 18（500 条以上），则递增页码并发起对下一页的请求，继续爬取。
- parse_detail 方法：定义了处理视频详情页响应的 parse_detail 方法。
- 从响应的 meta 中获取传递过来的视频项和描述。

- 使用 XPath 表达式提取视频文件的 URL。
- 更新视频项的 video 字段为视频文件的 URL，并保留描述信息。
- 使用 yield 关键字返回处理后的视频项，Scrapy 将自动将其传递到项目管道进行进一步处理（如存储）。

4、管道逻辑

在 pipelines.py 中处理视频的存储路径，确保爬取的数据按标题命名。
视频将保存至指定路径，并以视频标题为文件名进行命名。

5、item 逻辑

在 item.py 中，利用 scrapy.Filed（）创建 desc，video 两种 item，分别保存视频名和视频路径，为后续处理做准备。

6、setting 逻辑

在 setting.py 中

- 设置 FILES_STORE=“F:\ApplyData\Python\getshipVideo\Video”
- 将 User-Agent 和 cookie 设置为自己浏览器的，因为这个网站反爬；
- 需要设置 DOWNLOAD_DELAY = 2 及以上，如果不设置会被网站要求滑动验证。

（二）数据清洗

1、安装 YOLOv8

- 安装 pip install ultralytics 来引入 YOLOv8 模型。
- 下载训练好的模型文件 yolov8n.pt。

2、视频清洗

（1）环境和模型准备

- 导入 YOLO 类等库。
- 使用 YOLO('yolo11n.pt')加载预训练的 YOLOv8 模型。

（2）路径和目录准备

- 设置源视频文件夹 source 和目标文件夹 target_path。

- 创建目标文件夹。

（3）视频文件遍历

- 只处理以.mp4、.avi、.mov 结尾的文件，确保处理的是视频文件。

（4）视频帧处理

- 使用 cv2.VideoCapture 打开视频文件，并获取总帧数。
- 如果视频帧数少于 1，释放视频捕获对象并跳过当前视频。
- 使用 video_capture.set 定位到特定帧，之后进行帧的读取，如果读取失败，标记当前帧检查失败并跳过；如果读取成功，使用 YOLOv8 模型对帧进行预测。

（5）视频文件处理

- 如果所有检查的帧都成功读取且至少有一帧检测到船只，将视频文件复制到目标文件夹。
- 删除不包含船舶的帧或视频，确保后续的测试集有效性。

（三）测试集构建

1、视频重命名

- 遍历文件并重命名：通过 enumerate 函数循环遍历排序后的视频文件列表
- 构建新文件名：对于每个视频文件，先通过 os.path.join 函数获取其完整的旧文件路径。
- 按照“索引+扩展名”的格式构建新的文件名，并使用 os.path.join 函数生成新的文件路径。
- 调用 os.rename 函数，将视频文件从旧路径重命名为新路径。这样就实现了按照顺序对视频文件进行批量重命名的功能。

2、视频抽帧

- 使用 cv2.VideoCapture 加载视频文件。
- 每隔 8 帧保存一张图片，使用 cap_image 函数形成图像序列（轨迹），并按轨迹分组存储。
- 每 8 张图片为一组，生成新的文件夹保存一条轨迹的数据。

3、构建图片序列和图片命名与存储：

- 在 build_dataset 函数中，对保存的图片进行重新命名，其中包含船舶类别编号、轨迹编号和帧编号等信息，这种命名方式使得每张图片都能清晰地对应到其所属的船舶和轨迹，便于后续的数据管理和查询。
- 将图片文件命名为 V 船舶 ID_T 轨迹 ID_F 图片 ID.jpg 格式，如 V00001_T00001_F00001.jpg。

4、构建测试集

通过 build_dataset 函数构建测试集。对于测试集中的每艘船舶，将其第一条轨迹作为 query（查询样本），其余轨迹作为 gallery（检索样本）。

在构建测试集时，会将源图片从原始路径复制到新的测试集图片存储路径下，并在 test_name.txt 文件中记录每张图片的新名字。

生成文件如下：

- **test_names.txt**：保存所有测试图片的文件名。
- **name_map.txt**：保存原始文件名与新命名之间的映射关系。
- **tracks_test_info.mat**：记录每条轨迹的图片信息。
- **query_IDX.mat**：存储用于查询的轨迹编号。

（四）重识别模型测试

1、配置测试环境

- 通过 anaconda 创建新的虚拟环境，并通过 requirements.txt 安装必要的依赖包。
- 配置测试所需的文件路径和模型文件。

2、修改测试代码 test.py

（1）参数解析与配置管理

- 通过 argparse 库定义一个命令行参数解析器 parser，用于接收外部传入的参数，如配置文件路径、项目目录、GPU 设备 ID 等。
- 使用 yacs.config.CfgNode 加载并解析 YAML 格式的配置文件，将配置文件中的

内容转换为 CfgNode 实例。然后将命令行参数中指定的配置与默认配置进行合并，生成最终的配置对象 cfg。

（2）其他设置

- 调用 set_random_seed 设置随机种子，确保程序在每次运行时能够产生相同的随机结果。
- 通过 torch.cuda.is_available()检测系统中是否可用 GPU，如果可用，则启用 CUDA 并设置 cudnn.benchmark 为 True 以加速卷积操作。同时，根据是否使用 GPU，设置数据加载时的 pin_memory 参数，以优化数据传输效率。

（3）日志

- 根据配置中的日志路径和当前时间生成具体的日志文件路径。如果开启了开发者模式，则会在日志路径中包含当前时间，以便区分不同时间的运行日志。
- 使用自定义的 Logger 类将标准输出重定向到日志文件中，同时在控制台输出日志信息，方便用户实时查看程序运行状态，同时也便于后续的问题排查和分析。

（4）模型加载与测试

- 根据配置初始化 PSTA 模型，设置模型的类别数、预训练选择、模型名称和序列长度等参数。然后根据是否使用 GPU，将模型包装为 DataParallel 对象并移至 GPU，或者保持在 CPU 上。
- 从配置中指定的模型路径加载预训练模型的参数，并将其加载到初始化的模型中。
- 调用 tester 函数对模型进行测试，传入模型、查询集和图库集的数据加载器、是否使用 GPU 以及测试距离等参数。在测试过程中，模型会对查询集中的每个样本计算其与图库集中所有样本的相似度，并根据相似度进行排名，最终输出测试结果。

3、运行测试

- 运行测试脚本 test.py，获取模型的测试结果。
- 在测试过程中，模型会对查询集中的每个样本计算其与图库集中所有样本的相似度，并根据相似度进行排名，最终输出测试结果。
- 根据模型输出的结果进行分析，评估模型在自建测试集上的性能。

4、结果分析

- 生成性能报告，分析模型的准确率、召回率、F1 得分等指标。

- 根据结果优化模型或数据集。

六、实验结果分析

（一）爬取数据

1、安装 scrapy

```
C:\Users\25322>pip install scrapy -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com
```

图 6.1

```
Installing collected packages: PyDispatcher, zope.interface, w3lib, urllib3, typing-extensions, queuelib, pycparser, pya
sn1, protego, lxml, jmespath, itemadapter, incremental, idna, filelock, defusedxml, cssselect, constantly, charset-norma
lizer, certifi, automat, attrs, requests, pyasn1-modules, parsel, hyperlink, cffi, Twisted, requests-file, itemloaders,
cryptography, tldextract, service-identity, pyOpenSSL, scrapy
Successfully installed PyDispatcher-2.0.7 Twisted-24.11.0 attrs-24.3.0 automat-24.8.1 certifi-2024.12.14 cffi-1.17.1 cha
rset-normalizer-3.4.1 constantly-23.10.4 cryptography-44.0.0 cssselect-1.2.0 defusedxml-0.7.1 filelock-3.16.1 hyperlink-
21.0.0 idna-3.10 incremental-24.7.2 itemadapter-0.10.0 itemloaders-1.3.2 jmespath-1.0.1 lxml-5.3.0 parsel-1.9.1 protego-
0.3.1 pyOpenSSL-24.3.0 pyasn1-0.6.1 pyasn1-modules-0.4.1 pycparser-2.22 queuelib-1.7.0 requests-2.32.3 requests-file-2.1
.0 scrapy-2.12.0 service-identity-24.2.0 tldextract-5.1.3 typing-extensions-4.12.2 urllib3-2.3.0 w3lib-2.2.1 zope.interf
ace-7.2
```

图 6.2

2、创建项目

```
C:\Users\25322>F:

F:\>cd ApplyData

F:\ApplyData>cd Python

F:\ApplyData\Python>scrapy startproject getshipVideo
New Scrapy project 'getshipVideo', using template directory 'F:\Apply\Python\Lib\site-packages\scrapy\templates\project'
, created in:
    F:\ApplyData\Python\getshipVideo

You can start your first spider with:
    cd getshipVideo
    scrapy genspider example example.com

F:\ApplyData\Python>cd getshipVideo

F:\ApplyData\Python\getshipVideo>scrapy genspider getvideo www.vcg.com
Created spider 'getvideo' using template 'basic' in module:
  getshipVideo.spiders.getvideo
```

图 6.3

3、爬取视频

```
2025-01-12 17:50:22 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.vcg.com/creative-video/13155870> (referer: https://www.vcg.com/creat
x: //gossv-vcg.cfp.cn/videos/mts_videos/medium/temp/VCG42N1294543857.mp4
2025-01-12 17:50:22 [scrapy.pipelines.files] DEBUG: File (uptodate): Downloaded file from <GET https://gossv-vcg.cfp.cn/videos/mts_videos/medium/te
2025-01-12 17:50:22 [scrapy.core.scraper] DEBUG: Scraped from <200 https://www.vcg.com/creative-video/13155870>
{'desc': '捕鱼业:在北海的一艘船上捕获大量鲭鱼',
 'video': '//gossv-vcg.cfp.cn/videos/mts_videos/medium/temp/VCG42N1294543857.mp4'}
2025-01-12 17:50:25 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.vcg.com/creative-video/8964601> (referer: https://www.vcg.com/creati
x: //gossv-vcg.cfp.cn/videos/mts_videos/medium/temp/VCG42N1180094093.mp4
2025-01-12 17:50:25 [scrapy.pipelines.files] DEBUG: File (uptodate): Downloaded file from <GET https://gossv-vcg.cfp.cn/videos/mts_videos/medium/te
2025-01-12 17:50:26 [scrapy.core.scraper] DEBUG: Scraped from <200 https://www.vcg.com/creative-video/8964601>
{'desc': '当船快速驶过时，在头顶形成了尾流。',
 'video': '//gossv-vcg.cfp.cn/videos/mts_videos/medium/temp/VCG42N1180094093.mp4'}
```

图 6.4

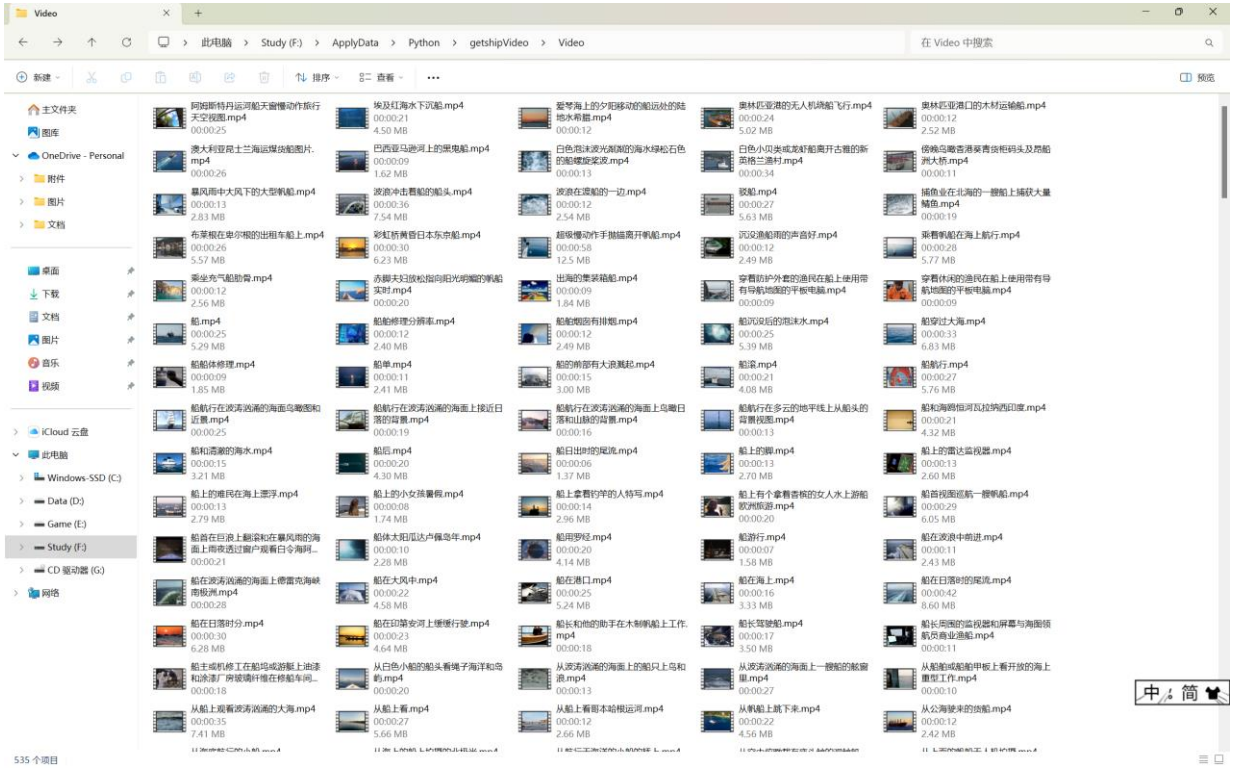

图 6.5

共爬取 535 个视频，符合预期目的。

（二）数据清洗

1、清洗视频

对上一步爬取到的视频进行清洗工作。

```
0: 384x640 (no detections), 16.1ms
Speed: 3.5ms preprocess, 16.1ms inference, 3.1ms postprocess per image at shape (1, 3, 384, 640)
  4%|         | 22/535 [00:08<01:55,  4.45it/s]
0: 384x640 1 boat, 11.0ms
Speed: 7.2ms preprocess, 11.0ms inference, 2.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 boat, 10.3ms
Speed: 1.0ms preprocess, 10.3ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 boat, 11.0ms
Speed: 2.6ms preprocess, 11.0ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)
  4%|         | 23/535 [00:08<01:52,  4.56it/s]
```

图 6.6

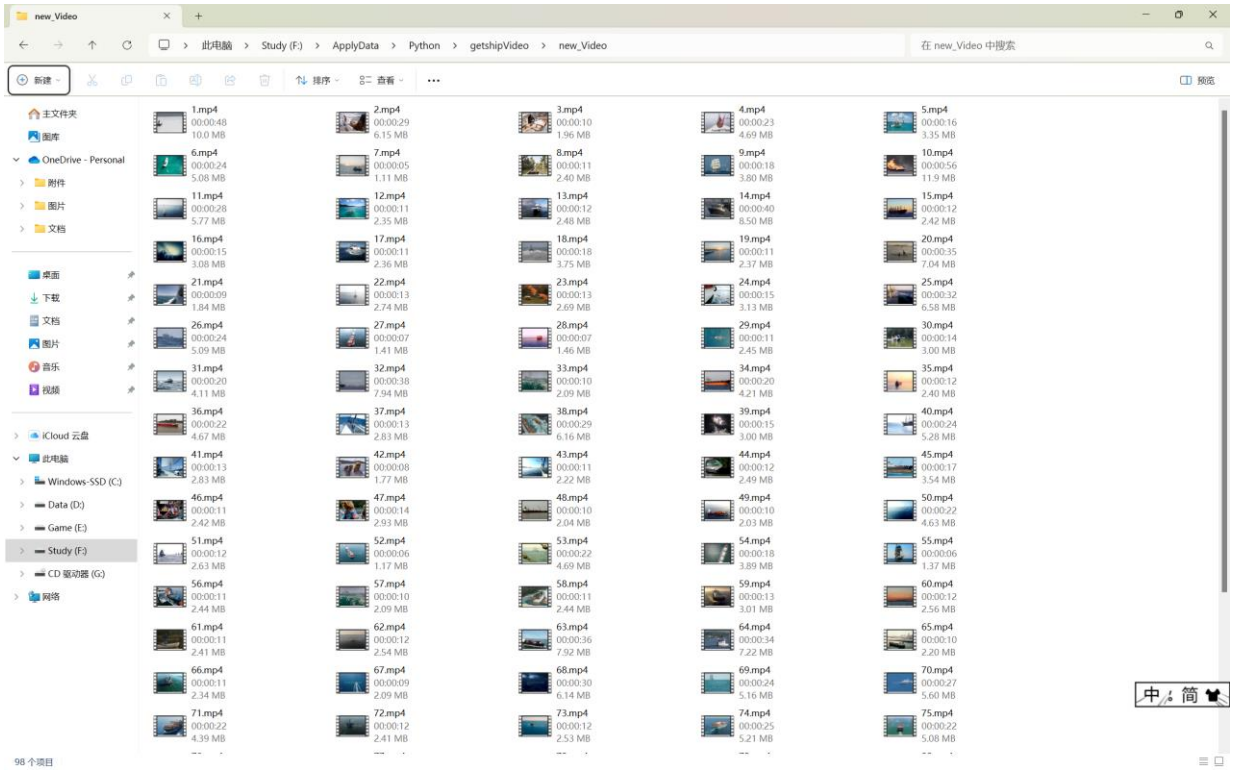

图 6.7

清洗结束后，符合条件的视频共有 98 条，且视频的编号唯一。

（三）测试集构建

1、视频重命名

每个视频具有独一无二的编号，详情见图 6.7。

2、测试集构建

通过抽帧处理构建图片序列，再通过图片序列进一步构建所需的测试集。

```
开始抽帧...
100%|██████████| 102/102 [03:07<00:00,  1.84s/it]

start to build test dataset

当前源图片路径： D:\vessel_videos_frames\1\00001\00001.jpg
当前目标图片路径： D:\vessel_video_reid_dataset\images\V1_T00001_F00001.jpg
  0%|          | 0/102 [00:00<?, ?it/s]当前源图片路径： D:\vessel_videos_frames\1\00001\00002.jpg
当前目标图片路径： D:\vessel_video_reid_dataset\images\V1_T00001_F00002.jpg
当前源图片路径： D:\vessel_videos_frames\1\00001\00003.jpg
当前目标图片路径： D:\vessel_video_reid_dataset\images\V1_T00001_F00003.jpg
当前源图片路径： D:\vessel_videos_frames\1\00001\00004.jpg
当前目标图片路径： D:\vessel_video_reid_dataset\images\V1_T00001_F00004.jpg
当前源图片路径： D:\vessel_videos_frames\1\00001\00005.jpg
当前目标图片路径： D:\vessel_video_reid_dataset\images\V1_T00001_F00005.jpg
```

图 6.8

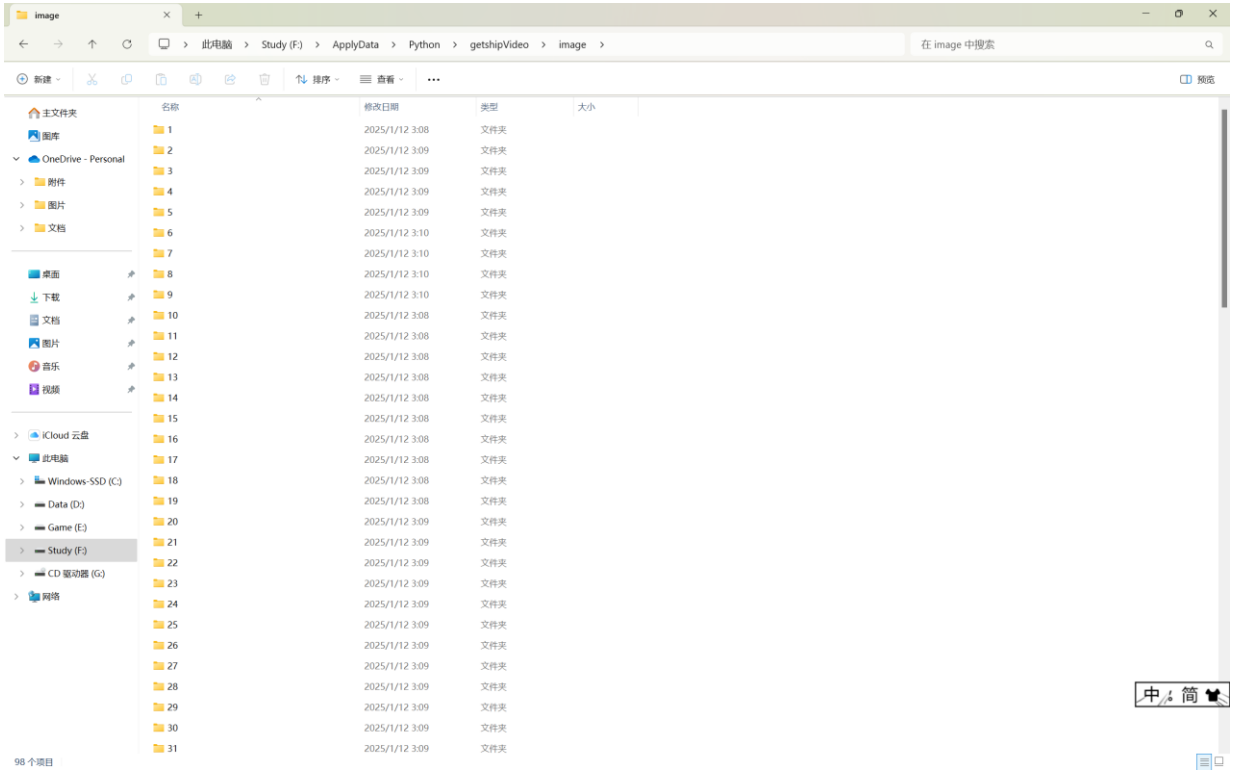

图 6.9

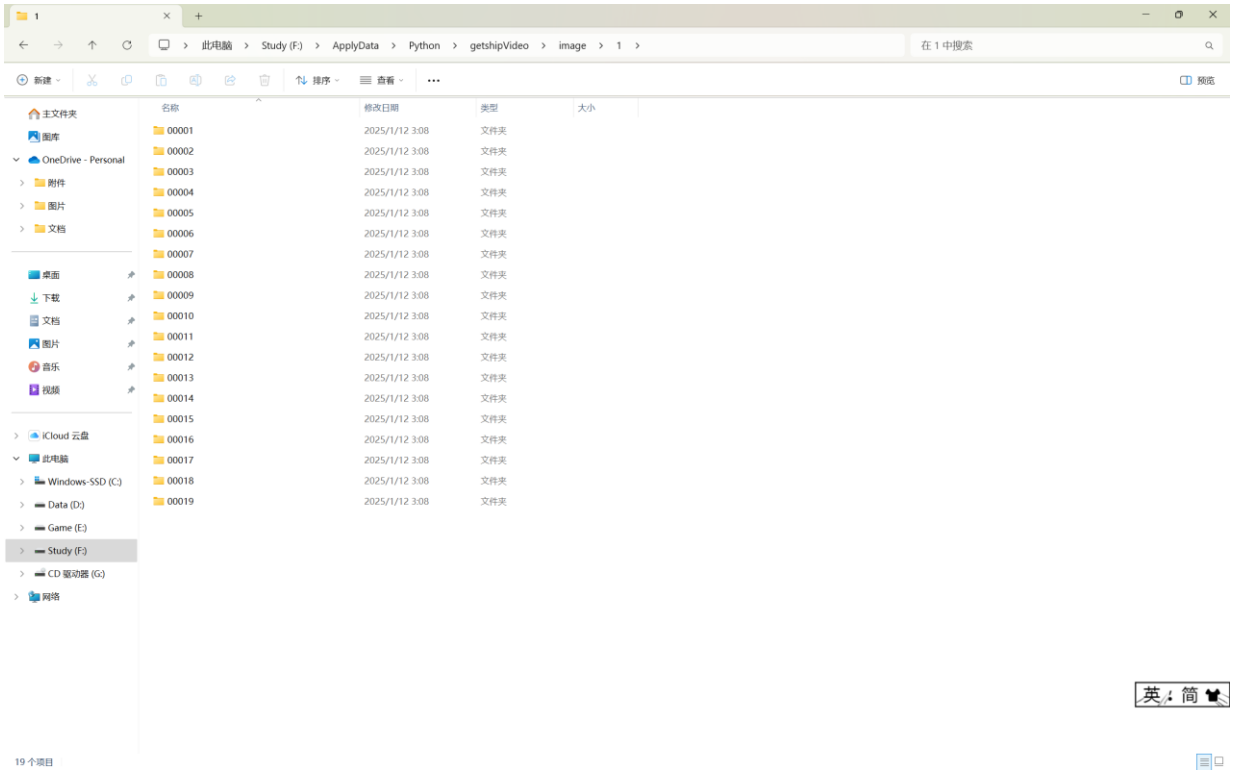

图 6.10

可见每个视频都完成序列图的抽取与命名。

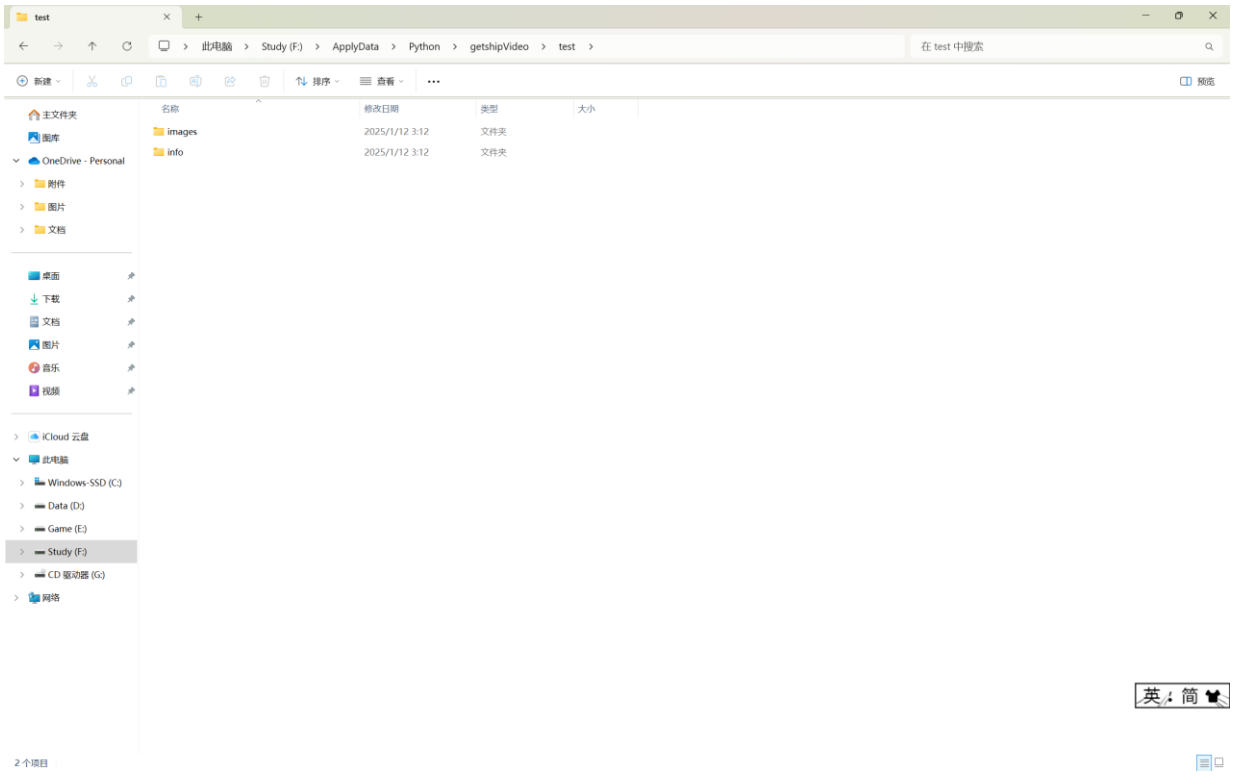

图 6.11

构建完成的测试集中包含图片和文件两部分内容。

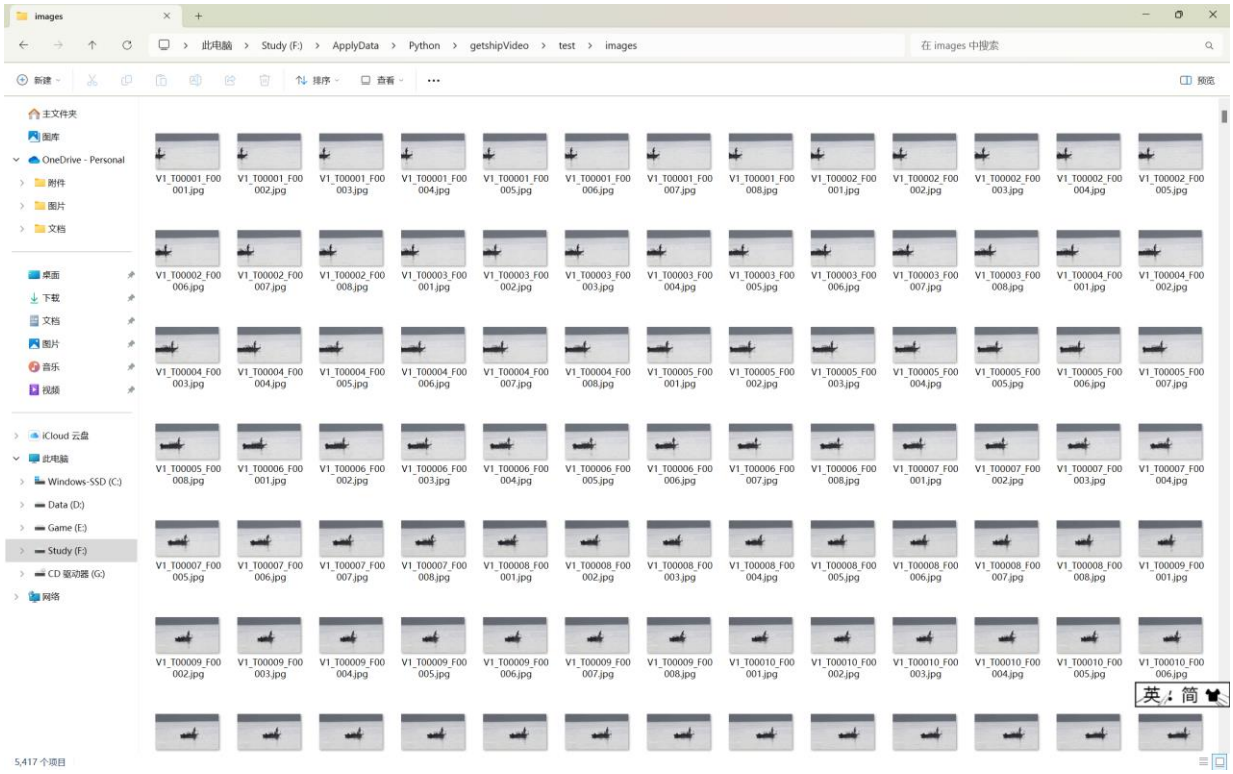

图 6.12

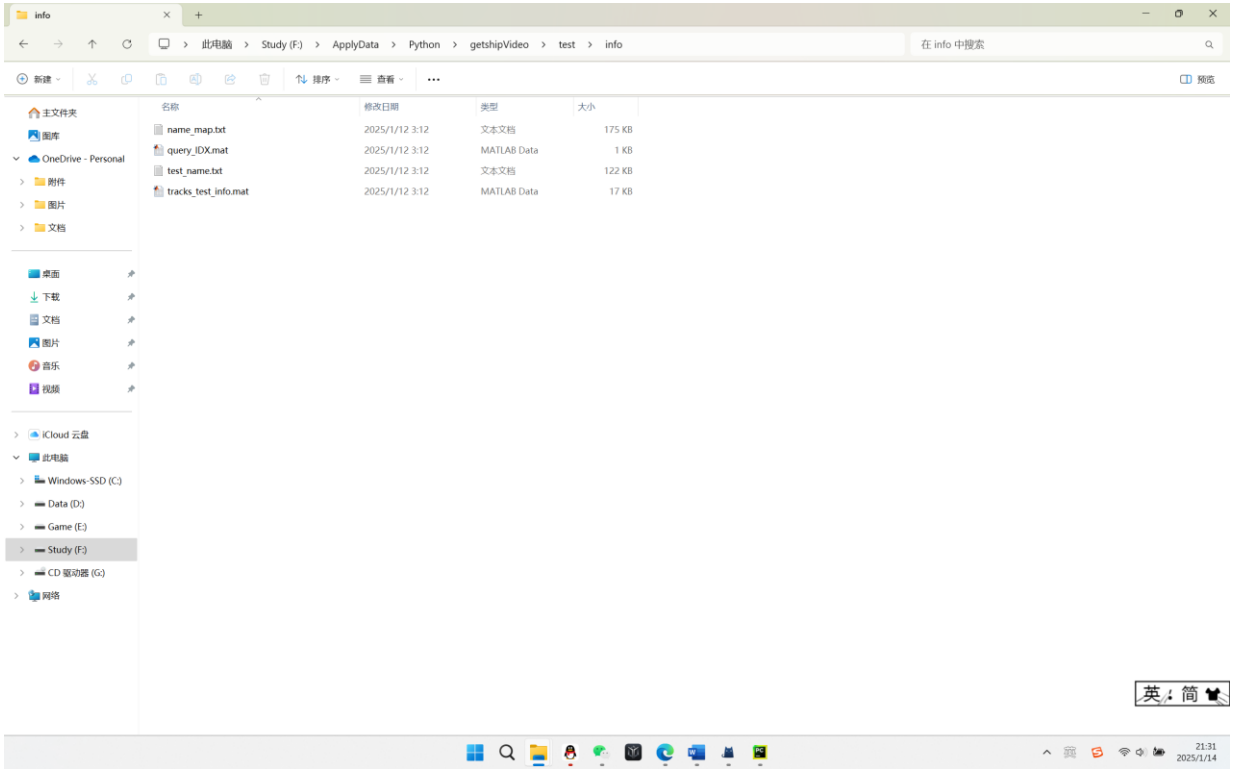

图 6.12

在测试集的图片部分，所有在图片序列中出现的图片已按照实验要求重命名，且做到了同一个轨迹中图片相邻。

在测试集的文件部分，实验要求的四个文件（两个 txt 文件、两个 mat 文件）已完成，各文件所代表的含义在详细设计给出。

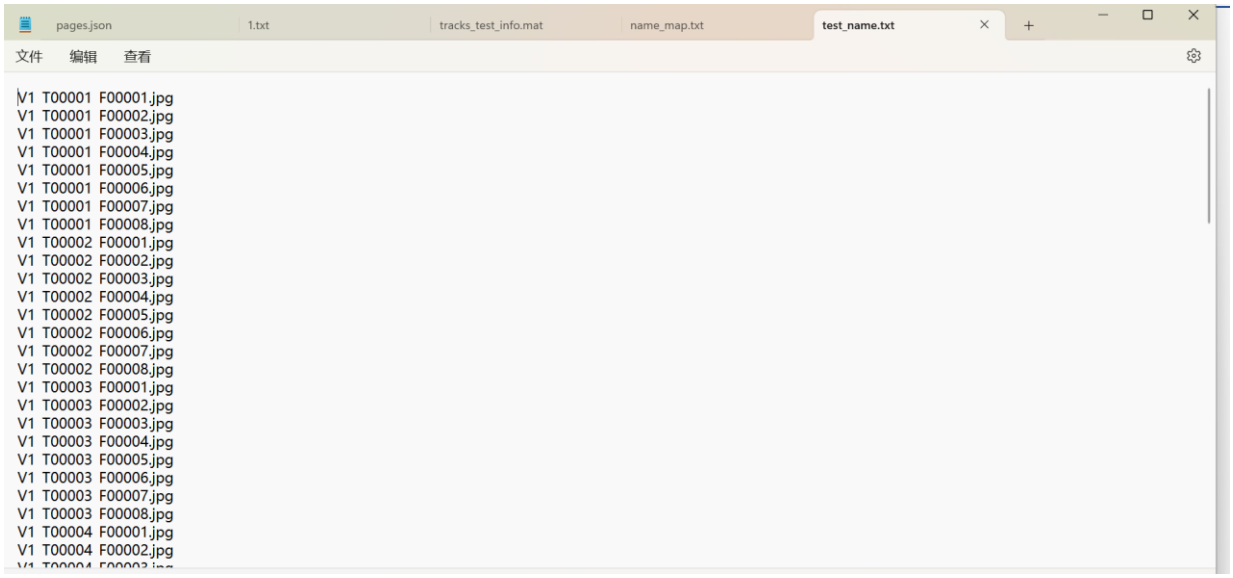

图 6.13 test_name.txt

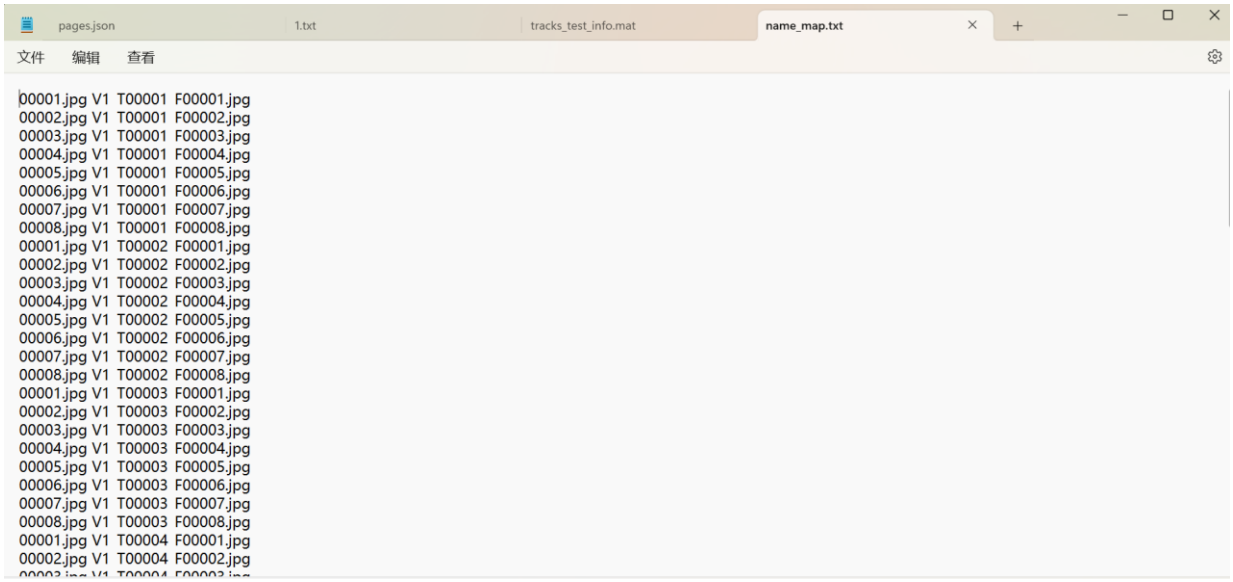

图 6.14 name_map.txt

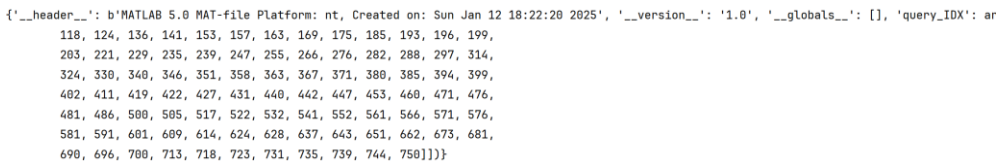

图 6.15 test_name.txt

第一列为该轨迹第一张图片在 test_names.txt 中的行数，第二列为该轨迹最后一张图片在 test_names.txt 中的行数，第三列为该轨迹所对应的船舶 id 编号，每一行是记录的一条轨迹的所有图片。

（四）重识别模型测试

1、虚拟环境搭建及依赖下载

```
(python39) PS F:\ApplyData\Python\2023_lu_data_process_course-master> conda install pytorch==1.10.0 torchvision==0.11.1 cudatoolkit=11.1 -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.

PackagesNotFoundError: The following packages are not available from current channels:
```

图 6.16

```
Downloading and Extracting Packages
urllib3-2.2.3         | 182 KB    | ########################################################################################################################################## | 100%
torchvision-0.20.1    | 6.5 MB    | ########################################################################################################################################## | 100%
libwebp-1.3.2         | 73 KB     | ########################################################################################################################################## | 100%
pyyaml-6.0.2          | 173 KB    | ########################################################################################################################################## | 100%
idna-3.7              | 114 KB    | ########################################################################################################################################## | 100%
pysocks-1.7.1         | 55 KB     | ########################################################################################################################################## | 100%
libuv-1.48.0          | 322 KB    | ########################################################################################################################################## | 100%
mpfr-4.0.2            | 1.5 MB    | ########################################################################################################################################## | 100%
typing_extensions-4.  | 65 KB     | ########################################################################################################################################## | 100%
pytorch-2.5.1         | 147.3 MB  | ##################################9                                                                                                       |  24%
markupsafe-2.1.3      | 34 KB     | ########################################################################################################################################## | 100%
mpir-3.0.0            | 1.3 MB    | ########################################################################################################################################## | 100%
pytorch-mutex-1.0     | 3 KB      | ########################################################################################################################################## | 100%
filelock-3.13.1       | 21 KB     | ########################################################################################################################################## | 100%
requests-2.32.3       | 100 KB    | ########################################################################################################################################## | 100%
torchaudio-2.5.1      | 5.8 MB    | ##################################################4                                                                                      |  31%
charset-normalizer-3  | 44 KB     | ########################################################################################################################################## | 100%
    (more hidden)
```

图 6.17

在 python3.9 环境中下载 require 的依赖。

2、运行 test.py

```
初始化DataManager
=> Dataset loaded
Dataset statistics:
  ------------------------------
  subset   | # ids | # tracklets
  ------------------------------
  query    |    98 |        98
  gallery  |    98 |       618
  ------------------------------
  total    |    98 |       716
  number of images per tracklet: 1 ~ 8, average 7.6
  ------------------------------
```

图 6.18

```
模型初始化
Loading pretrained ImageNet model ......
Build 1 layer TRA!
Build 1 layer SRA!
Build 2 layer TRA!
Build 2 layer SRA!
Build 3 layer TRA!
Build 3 layer SRA!
Model size: 33.503 M
Loading checkpoint from 'F:\ApplyData\Python\2023_lu_data_process_course-master\pth\best_model.pth'
load model...
  0%|          | 0/4 [00:00<?, ?it/s]Evaluate...
100%|██████████| 4/4 [00:25<00:00,  6.45s/it]
  0%|          | 0/20 [00:00<?, ?it/s]Extracted features for query set, obtained 98-by-1024 matrix
100%|██████████| 20/20 [00:25<00:00,  1.28s/it]
Extracted features for gallery set, obtained 618-by-1024 matrix
Computing distance matrix
Computing CMC and mAP
feature Results ----------
mAP: 66.5%
CMC curve
Rank-1  : 87.8%
Rank-5  : 89.8%
Rank-10 : 90.8%
Rank-20 : 93.9%
------------------
```

图 6.19

图 6.18 中，subset 代表数据集的子集，ids 代表每个子集中的 ID 数量，# tracklets 代表每个子集中的 tracklets 数量，query 代表查询子集，包含 98 个 ID 和 98 个 tracklets，gallery 代表图库子集，包含 98 个 ID 和 618 个 tracklets，total 代表总共包含 98 个 ID 和 716 个 tracklets。，number of images per tracklet 代表每个 tracklet 的图像数量范围是 1 到 8，平均数量是 7.6。

图 6.19 中，模型大小为 33.503 M，提取查询集的特征；得到一个 98-by-1024 的矩阵；提取图库集的特征，得到一个 618-by-1024 的矩阵。特征结果中，mAP：66.5%，表示平均精度。CMC 曲线：

- Rank-1：87.8%，表示在第一个匹配中找到正确结果的概率。
- Rank-5：89.8%，表示在前五个匹配中找到正确结果的概率。
- Rank-10：90.8%，表示在前十个匹配中找到正确结果的概率。
- Rank-20：93.9%，表示在前二十个匹配中找到正确结果的概率。

3、分析文件

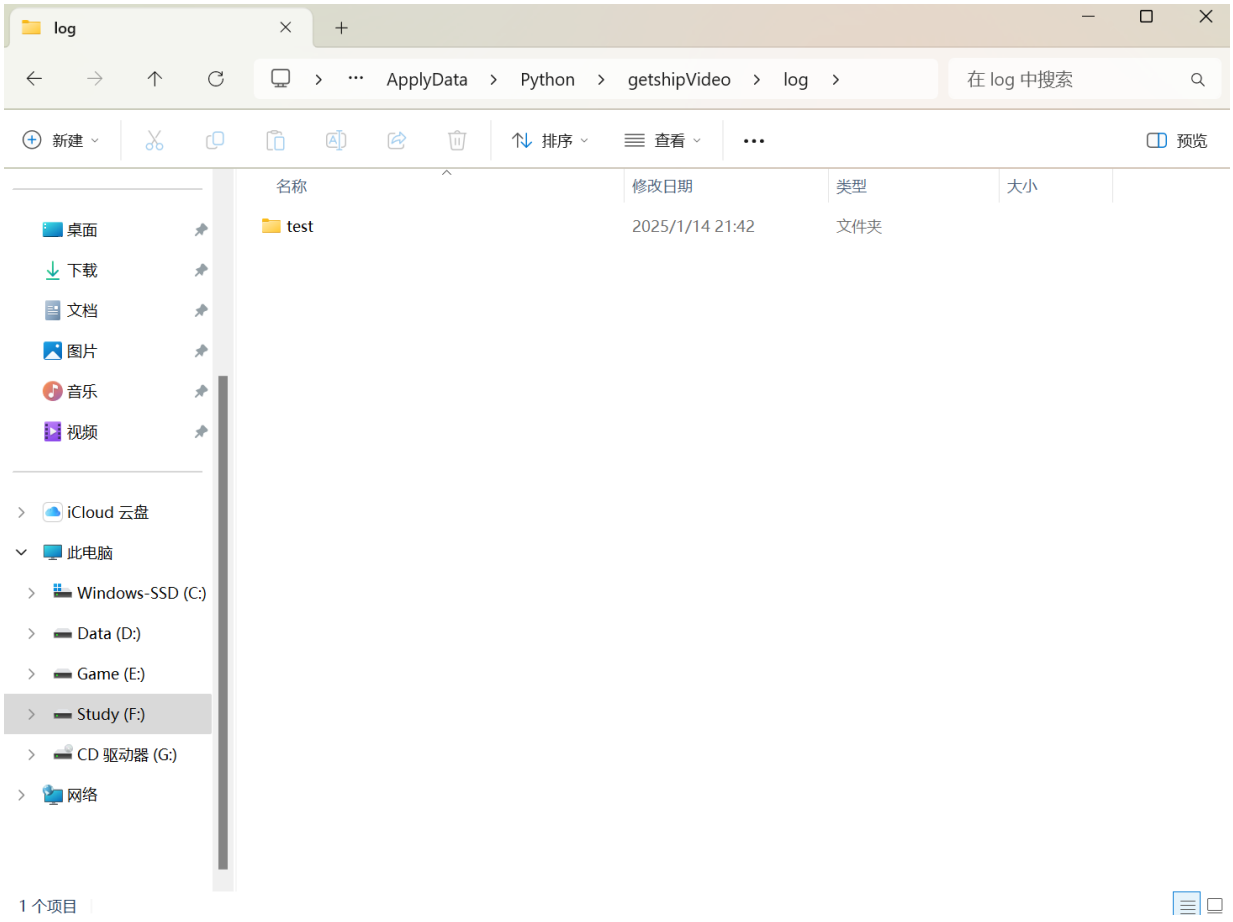
log
ApplyData
Python
getshipVideo
log
在 log 中搜索
新建
排序
查看
预览
名称
修改日期
类型
大小
test
2025/1/14 21:42
文件夹
桌面
下载
文档
图片
音乐
视频
iCloud 云盘
此电脑
Windows-SSD (C:)
Data (D:)
Game (E:)
Study (F:)
CD 驱动器 (G:)
网络
1 个项目

图 6.20

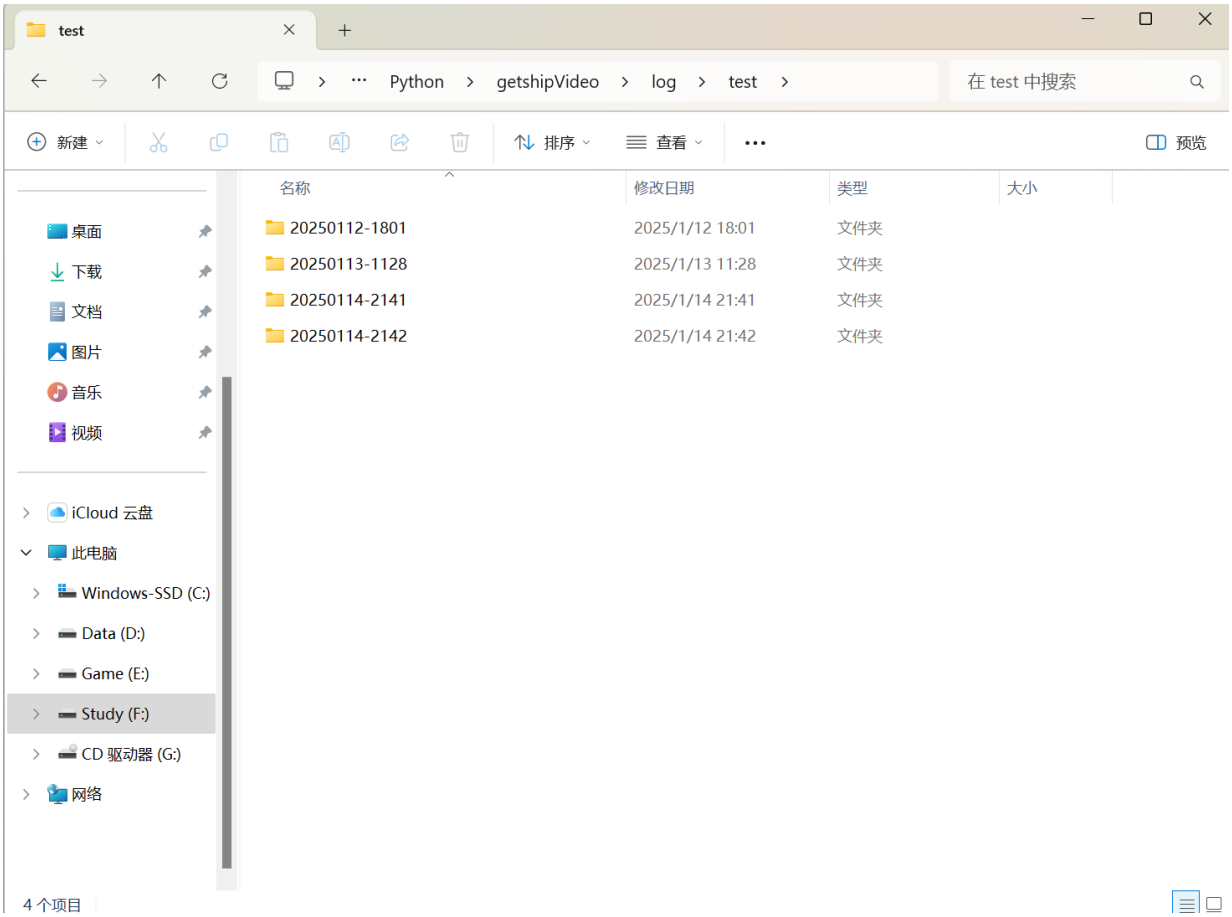

图 6.21

多个文件的原因为我运行多次代码，文件名代表当时运行的时间。

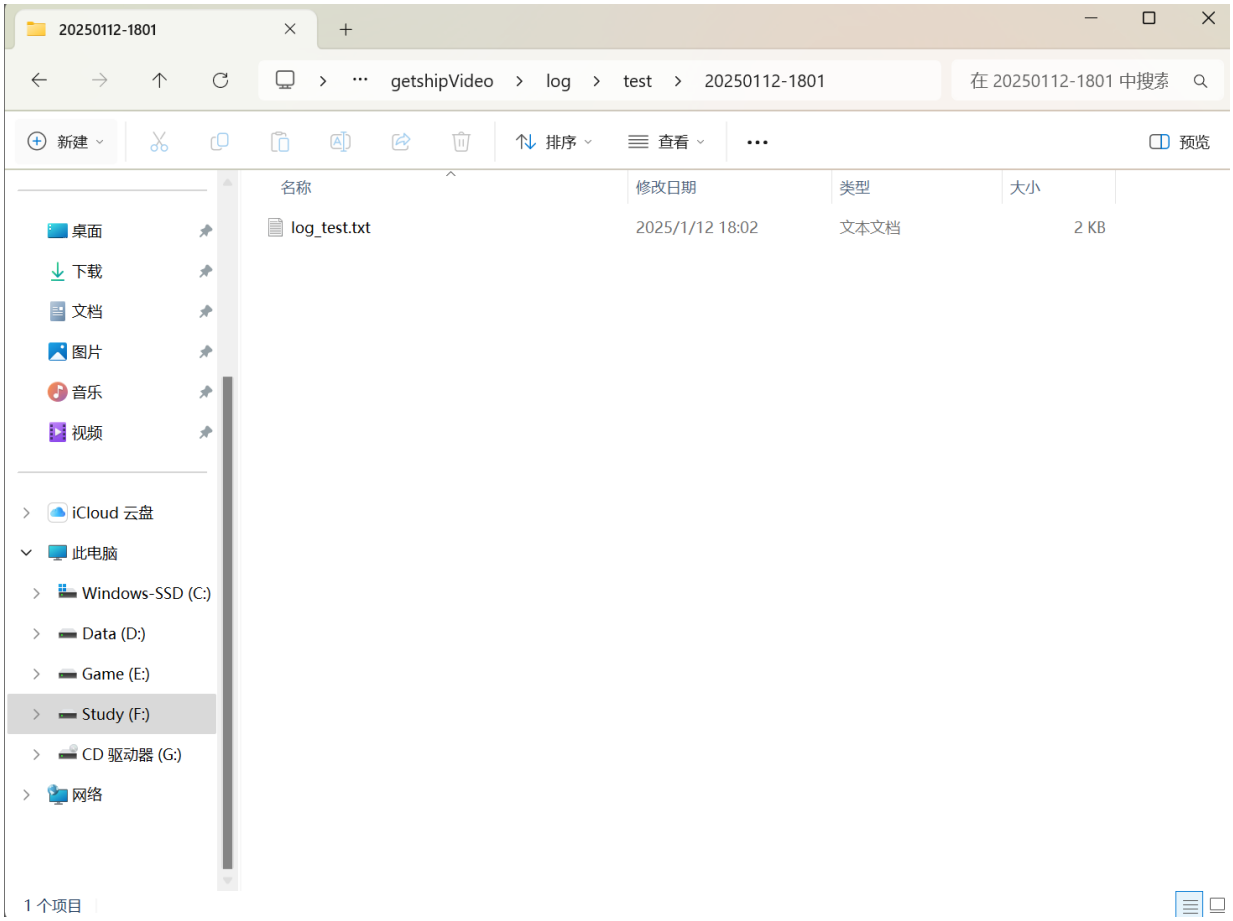

图 6.22

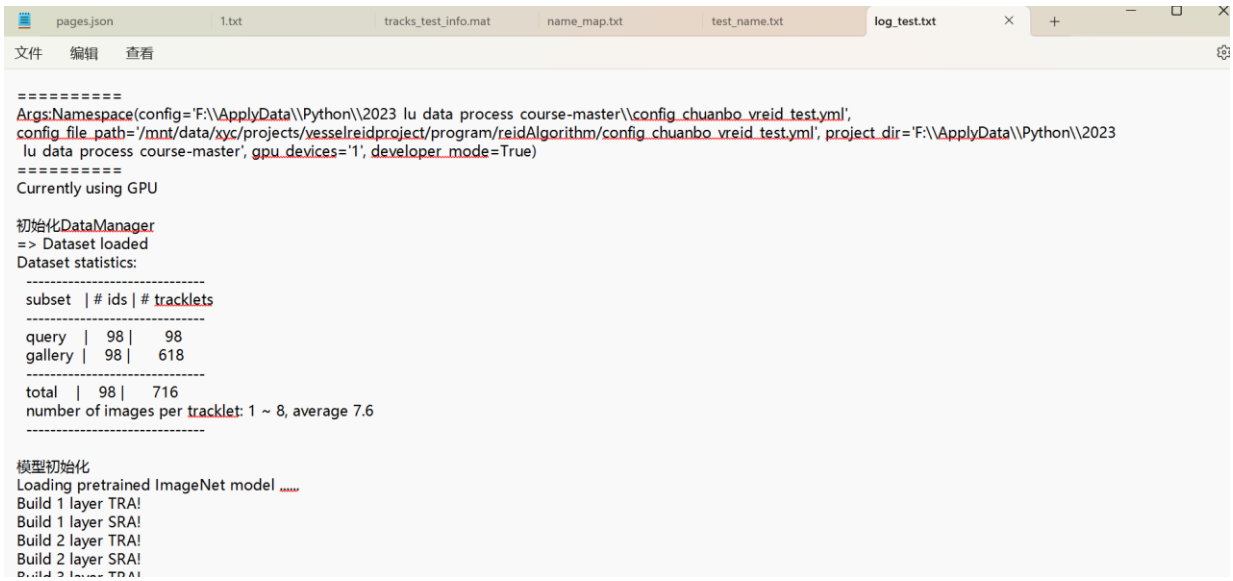

图 6.23 log_test.txt

该文件就是上述命令行的内容，解释在上述已经表达，不再重复说明。

七、实验所遇难题以及解决方法

（一）文件下载问题

```
C:\Users\25322>pip3 install scrapy
Looking in indexes: http://mirrors.aliyun.com/pypi/simple/
WARNING: The repository located at mirrors.aliyun.com is not a trusted or secure host and is being ignored. If this repo
sitory is available via HTTPS we recommend you use HTTPS instead, otherwise you may silence this warning and allow it an
yway with '--trusted-host mirrors.aliyun.com'.
ERROR: Could not find a version that satisfies the requirement scrapy (from versions: none)
ERROR: No matching distribution found for scrapy
WARNING: The repository located at mirrors.aliyun.com is not a trusted or secure host and is being ignored. If this repo
sitory is available via HTTPS we recommend you use HTTPS instead, otherwise you may silence this warning and allow it an
yway with '--trusted-host mirrors.aliyun.com'.
WARNING: There was an error checking the latest version of pip.
```

图 7.1

经过查询资料，了解到了我的电脑对阿里源的证书并不信任，解决方法为在后面加上—trusted-host mirror.aliyun.com

```
C:\Users\25322>pip install scrapy -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com
```

图 7.2

（二）域名问题

一开始 allowed_domains 只包含 www.vcg.com，但是发来视频的域名可能为 gossv-vcg.cfp.cn，导致 scrapy 过滤了视频的下载，在 allowed_domains 加入就可以解决。

```
allowed_domains = ["www.vcg.com", "gossv-vcg.cfp.cn"]  #注意gossv-vcg.cfp.cn，爬取的源域名
```

图 7.3

（三）依赖问题

```
WARNING: The repository located at mirrors.aliyun.com is not a trusted or secure host and is being ignored. If this repository is available via HTTPS we recommend you use HTTPS instead, otherwise you may si
lence this warning and allow it anyway with '--trusted-host mirrors.aliyun.com'.
ERROR: Could not find a version that satisfies the requirement torchvision==0.11.1+cu111 (from versions: 0.1.6, 0.2.0, 0.8.2, 0.8.2+cpu, 0.8.2+cu101, 0.8.2+cu110, 0.9.0, 0.9.0+cpu, 0.9.0+cu101, 0.9.0+cu111,
 0.9.1+cpu, 0.9.1+cu101, 0.9.1+cu102, 0.9.1+cu111, 0.10.0, 0.10.0+cpu, 0.10.0+cu102, 0.10.0+cu111, 0.10.1+cpu, 0.10.1+cu102, 0.10.1+cu111, 0.11.0+cpu, 0.11.0+cu102, 0.11.0+cu113, 0.11.1+cpu, 0.11.1+cu102, 0
.11.1+cu113, 0.11.2+cpu, 0.11.2+cu102, 0.11.2+cu113, 0.11.3+cpu, 0.11.3+cu102, 0.11.3+cu113, 0.12.0+cpu, 0.12.0+cu113, 0.12.0+cu115, 0.13.0+cpu, 0.13.0+cu113, 0.13.0+cu116, 0.13.1+cpu, 0.13.1+cu113, 0.13.1+
cu116, 0.14.0+cpu, 0.14.0+cu116, 0.14.0+cu117, 0.14.1+cpu, 0.14.1+cu116, 0.14.1+cu117, 0.15.0+cpu, 0.15.0+cu117, 0.15.0+cu118, 0.15.1+cpu, 0.15.1+cu117, 0.15.1+cu118, 0.15.2+cpu, 0.15.2+cu117, 0.15.2+cu118,
 0.16.0+cpu, 0.16.0+cu118, 0.16.0+cu121, 0.16.1+cpu, 0.16.1+cu118, 0.16.1+cu121, 0.16.2+cpu, 0.16.2+cu118, 0.16.2+cu121, 0.17.0+cpu, 0.17.0+cu118, 0.17.0+cu121, 0.17.1+cpu, 0.17.1+cu118, 0.17.1+cu121, 0.17.
2+cpu, 0.17.2+cu118, 0.17.2+cu121, 0.18.0+cpu, 0.18.0+cu118, 0.18.0+cu121, 0.18.1+cpu, 0.18.1+cu118, 0.18.1+cu121)
ERROR: No matching distribution found for torchvision==0.11.1+cu111
(python39) PS F:\ApplyData\Python\2023_lu_data_process_course-master> conda pip install torch==1.10.0+cu111 torchvision==0.11.1+cu111 -f https://download.pytorch.org/whl/torch_stable.html
```

图 7.4

一开始采用 python 最新版，下载最新版 pytorch 和其他包，运行所给代码，会不断报错，在经过多次尝试后，发现下载 python3.9 解决一切问题。

八、主要代码

（一）爬取数据

1、getVideo.py

```
import scrapy
from scrapy import cmdline

from getshipVideo.items import GetshipvideoItem

class GetvideoSpider(scrapy.Spider):
    name = "getVideo"
    allowed_domains = ["www.vcg.com", "gossv-vcg.cfp.cn"]  #注意 gossv-vcg.cfp.cn，爬取的源域名
    page = 3
    url = 'https://www.vcg.com/creative-video-search/5578/?page={}'.format(str(page))
    start_urls = [url]
    test = 0
    # 一级页面
    def parse(self, response):
        lists = response.xpath('//article[@class="R3J0t _13pTf"]/div[@class="_2tMKZ"]')
        # 所有 article，class 为 R3J0t_13pTf，的 div，class="_2tMKZ"的节点
        print("lists:", lists)
        item = GetshipvideoItem()
        for i in lists:
            link = i.xpath('./a/@href').get()  # href 属性所带的视频链接
            desc = i.xpath('.//div[@class="_2jWqo"]/span[1]/text()').get().replace('\n', '').replace('\r', '').replace(
                ' ', '')    #名称
```

```
            item['video'] = link
            print("test: ", self.test)
            if type(item["video"]) == str and (desc.find("船") >= 0 or desc.find("
舰") >= 0 or desc.find("ship") >= 0):
                print("link: ", link)
                # 请求详情页
                yield scrapy.Request(
                    url="https://www.vcg.com" + item["video"],
                    callback=self.parse_detail,
                    meta={"item": item, "desc": desc}
                )
            self.test += 1
        if self.page < 18:
            self.page += 1
            url = 'https://www.vcg.com/creative-video-
search/5578/?page={}'.format(str(self.page))
            yield scrapy.Request(url=url, callback=self.parse)

    # 二级页面
    def parse_detail(self, response):
        # item = GetshipvideoItem()
        item = response.meta["item"]
        x = response.xpath('//video/@src').get()
        print("x:", x)
        item['video'] = x
        item['desc'] = response.meta['desc']
        yield item


cmdline.execute('scrapy crawl getVideo'.split())  # 转列表
```

2、item.py

```
import scrapy


class GetshipvideoItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    desc = scrapy.Field()   #desc
    video = scrapy.Field()    #video
    pass
```

3、pipelines.py

```
class GetshipvideoPipeline(FilesPipeline):

    # 依次对视频地址发送请求，meta 用于传递视频的文件名
    def get_media_requests(self, item, info):
        # 依次对视频地址发送请求，meta 用于传递视频的文件名
        yield scrapy.Request(url="https:"+item['video'], meta={'desc':
item['desc']})

    # 返回下载的视频文件名
    def file_path(self, request, response=None, info=None, *, item=None):
        pattern = re.compile(r'[^\u4e00-\u9fa5]')  # 匹配非汉字字符的正则表达式
        text = str(request.meta['desc'])
        text = re.sub(pattern, '', text)
        filename = text+".mp4"  # 获取视频文件名
        return filename  # 返回下载的视频文件名

    def item_completed(self, results, item, info):
        return item
```

4、settings.py

```
class GetshipvideoPipeline(FilesPipeline):


    # 依次对视频地址发送请求，meta 用于传递视频的文件名
    def get_media_requests(self, item, info):
        # 依次对视频地址发送请求，meta 用于传递视频的文件名
        yield scrapy.Request(url="https:"+item['video'], meta={'desc': item['desc']
})


    # 返回下载的视频文件名
    def file_path(self, request, response=None, info=None, *, item=None):
        pattern = re.compile(r'[^\u4e00-\u9fa5]')  # 匹配非汉字字符的正则表达式
        text = str(request.meta['desc'])
        text = re.sub(pattern, '', text)
        filename = text+".mp4"  # 获取视频文件名
        return filename  # 返回下载的视频文件名


    def item_completed(self, results, item, info):
        return item
```

（二）数据清洗

1、demo.pridict.py

```
import os
import shutil
import random
import torch
import cv2
from tqdm import tqdm
from ultralytics import YOLO
```

```
# 加载 YOLOv8 的模型
model = YOLO('yolo11n.pt')

source = r'F:\ApplyData\Python\getshipVideo\Video'
target_path = r'F:\ApplyData\Python\getshipVideo\new_Video'
if not os.path.exists(target_path):
    os.makedirs(target_path)

# 遍历视频文件
for video in tqdm([f for f in os.listdir(source) if f.endswith(('.mp4', '.avi', '.mov'))]):
    video_path = os.path.join(source, video)
    video_capture = cv2.VideoCapture(video_path)
    total_frames = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))

    # 检查帧数是否足够
    if total_frames < 1:
        video_capture.release()
        continue

    random_frames = random.sample(range(total_frames), min(3, total_frames))  # 最多检查 3 帧
    check = [True] * len(random_frames)
    having_ship = False

    for i, frame_number in enumerate(random_frames):
        video_capture.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
        success, frame = video_capture.read()
        if not success:
            check[i] = False
            break
```

```
        results = model.predict(frame, conf=0.5, classes=8)
        for r in results:
            if len(r.boxes.cls) != 1:  # 检查类别数
                check[i] = False
                break
            if r.boxes.cls.item() == 8:  # 检查类别是否为8,代表有船
                having_ship = True

    video_capture.release()

    # 如果检测符合条件，复制文件
    if all(check) and having_ship:
        shutil.copy(video_path, os.path.join(target_path, video))
```

（三）测试集构建

1、video_rename.py

```
import os

def rename_videos(folder_path):
    video_extensions = ('.mp4', '.avi', '.mov', '.mkv', '.flv')  # 可以根据实际情况添加更多视频扩展名
    video_files = [f for f in os.listdir(folder_path) if f.endswith(video_extensions)]
    video_files.sort()  # 对文件列表进行排序，确保顺序稳定

    for index, video_file in enumerate(video_files, start=1):
        old_file_path = os.path.join(folder_path, video_file)
        file_extension = os.path.splitext(video_file)[-1]
        new_file_name = f"{index}{file_extension}"
```

```
        new_file_path = os.path.join(folder_path, new_file_name)
        os.rename(old_file_path, new_file_path)


if __name__ == "__main__":
    folder_path = r"F:\ApplyData\Python\getshipVideo\new_Video"
    rename_videos(folder_path)
```

2、build_image_sequence.py

```
import os

import cv2
from tqdm import tqdm

import shutil

# 将字典写入 mat 文件
def write_mat(dict, fpath):
    from scipy.io import savemat
    savemat(fpath, dict)

def cap_image(video_root_path, image_save_path):
    """
    读取视频文件并抽帧保存为图片
    @param video_root_path: 视频文件根目录
    @param image_save_path: 图片保存根目录
    """
    print("开始抽帧...")
    for video in tqdm(os.listdir(video_root_path)):
        video_path = os.path.join(video_root_path, video)
        # 船舶类别编号文件夹
```

```
        video_name = video.split('.')[0]
        save_path = os.path.join(image_save_path, video_name)
        cap = cv2.VideoCapture(video_path)  # 打开视频文件
        success, frame = cap.read()  # 读取一帧数据
        tracklet = 1
        image_name = 1
        count = 1
        while success:
            # 每 8 帧取一帧
            if count % 8 == 0:
                # 轨迹文件夹
                tracklet_path = os.path.join(save_path, str(tracklet).zfill(5))
                if not os.path.exists(tracklet_path):
                    os.makedirs(tracklet_path)
                cv2.imwrite(os.path.join(tracklet_path, '{}.jpg'.format(str(image_n
ame).zfill(5))), frame)
                image_name += 1
                # 每 8 张图片作为一条轨迹
                if image_name % 9 == 0:
                    tracklet += 1
                    image_name = 1
            count += 1
            success, frame = cap.read()  # 继续读取下一帧 这段代码实现了什么功能


def build_dataset(paths, ids, mode):
    """
    用于构建训练集，测试集和基准库，并生成对应的 txt 文件和 mat 文件
    Args:
        paths:
        ids:
        mode:
```

```
    Returns:

    """
    assert mode in ["train", "test", "baselib"]
    print("\nstart to build {} dataset\n".format(mode))
    txt_path = os.path.join(paths["info_root"], mode + "_name.txt")
    # 存储每个视频的开始帧和结束帧索引，以及对应的 vessel_id
    mat_contents = []
    image_nums = 1
    # 测试集的 query 索引，每艘船舶的第一条轨迹作为 query
    query_IDX = []
    num = 1
    for vessel_id in tqdm(ids):
        vessel_path = os.path.join(paths["image_source"], vessel_id)
        first = True
        for track_id in os.listdir(vessel_path):
            track_path = os.path.join(vessel_path, track_id)
            start_num = image_nums
            for image_name in os.listdir(track_path):
                new_image_name = "V{}_T{}_F{}.jpg".format(vessel_id, track_id, imag
e_name.split(".")[0])  #V{}_T{}_F{}.jpg 格式
                image_path = os.path.join(track_path, image_name)
                new_image_path = os.path.join(paths["image_root"], new_image_name)
                if mode != "baselib":
                    shutil.copy(image_path, new_image_path)
                    print("当前源图片路径：", image_path)
                    print("当前目标图片路径：", new_image_path)
                with open(txt_path, "a") as f:
                    f.write("{}\n".format(new_image_name))
                with open(os.path.join(paths["info_root"], "name_map.txt"), "a") as
 f_map:
                    f_map.write("{} {}\n".format(image_name, new_image_name))
```

```
                image_nums += 1
            end_num = image_nums - 1
            mat_contents.append([start_num, end_num, int(vessel_id)])
            if first and mode == "test":
                first = False
                query_IDX.append(num)
            num += 1
    mat_file = {"tracks_{}_info".format(mode): mat_contents}
    mat_file_path = os.path.join(paths["info_root"], "tracks_" + mode + "_info.mat"
)
    write_mat(mat_file, mat_file_path)
    if mode == "test":
        query_IDX_mat = {"query_IDX": query_IDX}
        query_IDX_mat_path = os.path.join(paths["info_root"], "query_IDX.mat")
        write_mat(query_IDX_mat, query_IDX_mat_path)


def split_dataset(paths):
    """
    切分数据集，分为训练集、测试集和基准库
    Args:
        paths: 字典，包含各种数据集路径

    Returns:

    """
    vessel_num = len(os.listdir(paths["image_source"]))
    # 70%作为训练集，30%作为测试集
    train_num = int(vessel_num * 0.7)
    train_vessel_ids = os.listdir(paths["image_source"])[:train_num]
    test_vessel_ids = os.listdir(paths["image_source"])[train_num:]
    build_dataset(paths, train_vessel_ids, mode="train")
    build_dataset(paths, test_vessel_ids, mode="test")
```

```
    build_dataset(paths, os.listdir(paths["image_source"]), mode="baselib")


def main():
    video_root_path = r"F:\ApplyData\Python\getshipVideo\new_Video"  # 清洗后视频文件根目录
    save_root_path = r"F:\ApplyData\Python\getshipVideo\image"  # 抽帧图片保存根目录
    save_dataset_root_path = r"F:\ApplyData\Python\getshipVideo\test"  # 测试集保存根目录
    os.makedirs(save_root_path, exist_ok=True)
    cap_image(video_root_path, save_root_path)  # 抽帧
    paths = {
        "image_source": save_root_path,
        "image_root": os.path.join(save_dataset_root_path, "images"),
        "info_root": os.path.join(save_dataset_root_path, "info"),
    }
    os.makedirs(paths["image_root"], exist_ok=True)
    os.makedirs(paths["info_root"], exist_ok=True)
    # split_dataset(paths)
    build_dataset(paths, os.listdir(paths["image_source"]), mode="test")  # 构建测试集


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

（四）重识别模型测试

1、config_chuanbo_vreid_test.yml

```
# 模型测试
scriptPath:
  python -u reidAlgorithm/reid/test.py # 测试模型训练效果
params:
```

```
  BATCH_SIZE: 128
  TEST_FILE_PATH: r"F:\ApplyData\Python\getshipVideo\test" # 测试数据集标签文件
  MODEL_PATH: r"F:\ApplyData\Python\2023_lu_data_process_course-
master\pth\best_model.pth"  # 存放模型训练过程中所保存的性能最佳的权重文件 <必需，不可修
改>
  LOG_PATH: r"F:\ApplyData\Python\getshipVideo\log" # 存放模型测试日志放模型测试日志
```

2、test.py

```
from __future__ import division
from __future__ import print_function

import argparse
import datetime
import os
import os.path as osp
import sys
import time
import warnings
from collections import OrderedDict

import torch
import torch.backends.cudnn as cudnn
import torch.nn as nn
from torch.utils.data import DataLoader

from reid import data_manager
from reid.engine import tester
from reid.models.PSTA import PSTA
from reid.tools.generaltools import get_cfg, set_random_seed
from reid.tools.loggers import Logger
from reid.tools.torchtools import count_num_param
```

```
from reid.utils import transforms as T
from reid.utils.video_loader import VideoDataset

sys.path.append(os.getcwd())

from yacs.config import CfgNode as CN
def main(args):
    torch.cuda.empty_cache()
    cfg = get_cfg()

    # 正常加载配置文件
    print("Loading config from file: {}".format(args.config))
    with open(args.config, 'r', encoding='utf-8') as file:
        config_str = file.read()

        # 将配置文件内容转换为 CfgNode 实例
    config_node = CN.load_cfg(config_str)

    # 合并配置
    cfg.merge_from_other_cfg(config_node)

    cfg.MODE = "test"
    cfg.params.RESUME = True

    # 中南大学开发时需配置 pycharm 中的运行参数 --developer_mode True
    if args.developer_mode:
        time_now = time.strftime("%Y%m%d-%H%M", time.localtime())  # 使用 %H%M 来避免使用冒号
        cfg.params.LOG_PATH = os.path.join(cfg.params.LOG_PATH, cfg.MODE, time_now)

    cfg.freeze()
    set_random_seed(cfg.RANDOM_SEED)
```

```
use_gpu = torch.cuda.is_available()
log_name = 'log_test.txt'
logdir = osp.join(cfg.params.LOG_PATH, log_name)
os.makedirs(osp.dirname(logdir), exist_ok=True)
sys.stdout = Logger(logdir)

print('==========\nArgs:{}\n=========='.format(args))

if use_gpu:
    print('Currently using GPU')
    cudnn.benchmark = True
else:
    warnings.warn('Currently using CPU, however, GPU is highly recommended')

print('\n 初始化 DataManager')
dataset = data_manager.init_dataset(mode=cfg.MODE, cfg=cfg)

print('\n 模型初始化')
model = PSTA(num_classes=625,
             pretrain_choice=cfg.MODEL.PRETRAIN_CHOICE,
             model_name=cfg.MODEL.NAME,
             seq_len=cfg.MODEL.SEQ_LEN)
print('Model size: {:.3f} M'.format(count_num_param(model)))
model = nn.DataParallel(model).cuda() if use_gpu else model

transform_test = T.Compose([
    T.Resize(cfg.INPUT.SIZE_TEST),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
pin_memory = True if use_gpu else False
```

```
if cfg.params.TEST_SAMPLER == 'dense':
    print('Build dense sampler')
    queryloader = DataLoader(
        dataset=VideoDataset(dataset=dataset.query,
                             seq_len=cfg.MODEL.SEQ_LEN,
                             sample=cfg.params.TEST_SAMPLER,
                             transform=transform_test,
                             max_seq_len=cfg.params.TEST_MAX_SEQ_NUM),
        batch_size=1,
        shuffle=False,
        num_workers=cfg.DATALOADER.NUM_WORKERS,
        pin_memory=pin_memory,
        drop_last=False
    )

    galleryloader = DataLoader(
        dataset=VideoDataset(dataset=dataset.gallery,
                             seq_len=cfg.MODEL.SEQ_LEN,
                             sample=cfg.params.TEST_SAMPLER,
                             transform=transform_test,
                             max_seq_len=cfg.params.TEST_MAX_SEQ_NUM),
        batch_size=1,
        shuffle=False,
        num_workers=cfg.DATALOADER.NUM_WORKERS,
        pin_memory=pin_memory,
        drop_last=False,
    )
else:
    queryloader = DataLoader(
        dataset=VideoDataset(dataset=dataset.query,
                             seq_len=cfg.MODEL.SEQ_LEN,
                             sample=cfg.params.TEST_SAMPLER,
```

```
                                transform=transform_test,
                                max_seq_len=cfg.params.TEST_MAX_SEQ_NUM),
            batch_size=cfg.params.SEQS_PER_BATCH,
            shuffle=False,
            num_workers=cfg.DATALOADER.NUM_WORKERS,
            pin_memory=pin_memory,
            drop_last=False
        )

        galleryloader = DataLoader(
            dataset=VideoDataset(dataset=dataset.gallery,
                                seq_len=cfg.MODEL.SEQ_LEN,
                                sample=cfg.params.TEST_SAMPLER,
                                transform=transform_test,
                                max_seq_len=cfg.params.TEST_MAX_SEQ_NUM),
            batch_size=cfg.params.SEQS_PER_BATCH,
            shuffle=False,
            num_workers=cfg.DATALOADER.NUM_WORKERS,
            pin_memory=pin_memory,
            drop_last=False
        )

    start_time = time.time()

    print("Loading checkpoint from '{}'".format(cfg.params.MODEL_PATH))
    print("load model... ")
    checkpoint = torch.load(cfg.params.MODEL_PATH)
    new_state_dict = OrderedDict()
    for k, v in checkpoint['state_dict'].items():
        name = k[7:]
        new_state_dict[name] = v
    model.load_state_dict(new_state_dict, strict=False)
```

```
    print("Evaluate...")
    tester(model, queryloader, galleryloader, use_gpu, test_distance=cfg.params.TEST_DISTANCE)

    elapsed = round(time.time() - start_time)
    elapsed = str(datetime.timedelta(seconds=elapsed))
    print("Finished. Total elapsed time (h:m:s): {}".format(elapsed))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="vesselreid Testing")
    parser.add_argument('--config', type=str, default="config_chuanbo_vreid_test.yml",
                        help='the file of train/test/infer')
    parser.add_argument('--config_file_path', type=str,
                        default="/mnt/data/xyc/projects/vesselreidproject/program/reidAlgorithm/config_chuanbo_vreid_test.yml",
                        help='the config file of train/test/infer from the AI plate!')
    parser.add_argument('--project_dir', type=str, default="",
                        help='project\'s root direction')
    parser.add_argument('--gpu-devices', default='1', type=str,
                        help='gpu device ids for CUDA_VISIBLE_DEVICES')
    parser.add_argument('--developer_mode', default=True, type=bool,
                        help='true：中南用的开发者模式 false：提交给远望时的模式')
    args = parser.parse_args()
    args.project_dir = os.path.dirname(os.path.dirname(__file__))
    args.config = os.path.join(args.project_dir, args.config)
    # args.config = args.config_file_path  # 自己调试时注释掉该行
    main(args)
```

九、实验总结

在完成本次船舶视频重识别实验的过程中，我深刻地感受到这一视觉任务的复杂性和多样性。整个实验不仅让我了解了目标重识别任务的背景，也通过实际操作加深了对数据处理和深度学习技术的理解，为我后续读写相关领域论文打下了坚实的基础。

爬虫技术的使用让我熟悉了如何自动化地从网站获取大量的数据。通过使用 Scrapy 框架来爬取视频数据，我学会了如何处理爬虫中的常见问题，如反爬虫机制的绕过和数据清洗。尽管在实际操作中遇到了一些技术难点，比如如何正确配置请求头和解决滑动验证的问题，但通过查阅老师给我的相关资料和尝试解决方案，我成功克服了这些问题，并顺利完成了视频的爬取，给我带来了很大的新鲜感。

数据清洗环节的应用让我理解到视频数据的处理不仅仅是提取出有效的信息，还需要借助 YOLO8 等目标检测算法筛选出船舶相关的内容。这一过程让我学会了如何结合计算机视觉技术和深度学习模型进行数据的筛选和处理，是对我的实践能力的一次提升，让我有了这门学科跟多的认识，为后续的学习提供了不一样的思路。

在测试集的构建阶段，我学会了如何根据抽帧生成图片序列并重新命名图片，这一过程对后续的模型测试至关重要。通过构建测试集和使用自建数据进行模型验证，我更好地理解了模型的训练和测试流程，也体会到如何根据实验结果进行性能评估和模型调优。

通过本次课设实验，我不仅掌握了目标重识别任务的基本操作和流程，还加深了对数据处理和模型测试的理解。实验中的技术挑战和解决方案，使我在深度学习与 cv 领域获得了更加全面的实践经验，也增强了我面对复杂任务时解决问题的能力。

最后通过这次课程设计，我充分认识到自己的不足之处，明白了对于计算机科学的探索之路还道阻且长，但同时也让我在实验中学到了很多知识，对深度学习有了相对于自己能够认知理解的知识，并且激发了我强烈的学习兴趣

参考文献

[1] Scrapy Documentation: https://docs.scrapy.org/en/latest/

[2] CSDN 博客：Scrapy 入门教程，Scrapy 爬虫案例 - 详细讲解：

https://blog.csdn.net/ck784101777/article/details/104291634

[3] Ultralytics YOLOv8 Documentation: https://docs.ultralytics.com/

[4] "Scrapy 爬虫框架，入门案例" CSDN 博客，链接：

https://blog.csdn.net/ck784101777/article/details/104291634

[5] PyTorch Documentation: https://pytorch.org/docs/stable/

[6] cv2.VideoCapture: OpenCV API Reference, OpenCV documentation for video processing.