郑 州 大 学

生 产 实 习 报 告

专　业	计算机科学与技术
班　级	4 班
学　号	202101001624
姓　名	杨浩晨

计算机与人工智能学院

2024 年 7 月

实习时间	2024 年 6 月 27 日至 7 月 6 日	实习地点	3 楼机房
实习单位	东方瑞通（郑州）咨询服务有限公司		
指导教师	梁鹏		
项目名称	VisionVoyage-基于鱼眼相机与其他感知技术的自动驾驶仿真系统		

实习目的：

1. 提升团队协作能力：在团队中负责分工合作，利用版本控制工具（例如 Git），并采用敏捷开发模式（如 Scrum）。通过快速迭代、持续反馈和改进方法，加强学生在动态环境下开发人工智能项目的能力，提高团队协作和沟通技能。
2. 深入了解人工智能领域的应用：通过具体实验，熟悉 Python 在人工智能中的广泛应用，包括数据处理、模型训练和结果评估等方面。
3. 掌握基本的人工智能算法和模型：学习并实现常见的人工智能算法，如线性回归、决策树和神经网络等，深入理解它们的原理和适用场景。
4. 提升 Python 编程能力：加强对 Python 语言的掌握，特别是在数据分析、模型构建和优化方面的应用能力，熟悉相关的库和工具，如 NumPy、Pandas、Scikit-learn、TensorFlow 和 Keras 等。
5. 熟悉人工智能项目的开发流程：从数据预处理、模型选择与训练，到模型评估与优化，最终到结果解释与应用，全面了解人工智能项目的完整开发流程。
6. 培养解决实际问题的能力：通过实验项目中的实际挑战，学习问题分析、解决方案设计、实施验证等关键能力，提升在实际场景中解决问题的技能。
7. 关注人工智能领域的前沿技术和发展趋势：了解当前人工智能领域的最新研究成果和技术进展，培养创新意识和行业洞察力。
8. 提升职业素养和专业能力：通过严格的实验规范和要求，培养职业操守和专业技能，为未来工作奠定坚实基础。
9. 激发创新思维：在实验过程中探索新方法、新技术，提出创新性解决方案，培养独立思考和创新能力。

项目简介：

在环视场景中，鱼眼相机通过将大角度范围（180°）内的光线，进行 压缩、扭曲之后提供大角度范围内车辆周边环境数据，并通过将多个相机 图像的拼接，去扭和变形之后实现低速场景下非常实用的 360° 全景影像 功能。而在泊车场景下，通过和超声波雷达融合，完成对停车线的识别， 障碍物的检测，从而实现不同等级的泊车功能。在自动驾驶系统中，鱼眼 相机还可以用于实现车道线识别、交通信号灯识别、路况检测等功能。结 合计算机视觉和深度学习技术，可以对鱼眼相机获取的图像进行实时处理 和分析，从而使车辆能够更准确地理解并适应不同的交通场景。此外，鱼 眼相机在自动驾驶系统中还可以用于实现车辆周围的盲区监测，提高行车 安全性。通过将鱼眼相机获取的图像与高精度地图数据进行融合，还可以 实现更精准的定位和路径规划，为自动驾驶系统提供更可靠的导航和控制 支持。 纵使鱼眼相机有诸多好处，但是公开发布的鱼眼数据集很少，除了仅 有的 WoodScape 数据集，几乎没有其他数据集提供真正的语义分割注释。 此外，目前针对鱼眼摄像头，还没有一个量产的，基于多摄像头组合的语 义分割算法，大部分工作也都集中在独立解决个别任务上。目前，编码器 是共享的，但解码器之间没有协同作用。同时，现有的数据集主要是为了 方便特定于任务的学习而设计的，并没有为所有的任务提供同时注释。且 鱼眼图像存在畸变。传统的语义分割模型分割效果不好。需要针对鱼眼图 像中的畸变设计特定的处理模块。

且目前大众对于自动驾驶的呼声越来越高，在这样的背景下，我们公 司开发 VisionVoyage 软件是为了满足自动驾驶领域对于鱼眼相机和感知技 术的需求，并推动自动驾驶技术的发展。随着自动驾驶技术的不断成熟和 市场需求的增加，人们对于自动驾驶的期待越来越高，希望能够通过先进 的技术实现更安全、高效的交通出行方式。 VisionVoyage 软件的开发将有助于解决目前自动驾驶领域存在的一些 挑战和问题，例如鱼眼数据集稀缺、鱼眼图像畸变处理、多摄像头组合的 语义分割算法等方面的技术瓶颈。通过提供普通图像转鱼眼图像、仿真环 境下各种传感器的图像获取与处理、虚拟驾驶体验和自动驾驶仿真等功能， 我们的软件可以帮助研究人员和工程师更好地理解和应用鱼眼相机和其他 传感器数据，从而提升自动驾驶系统的感知能力和安全性。 此外，由于当前市场上公开发布的鱼眼数据集相对较少，VisionVoyage 的开发填补了这一空白，为研究人员和开发者提供了更多的实验和模拟环

境，促进了自动驾驶领域的创新和进步。我们公司致力于通过 VisionVoyage 软件的推出，为自动驾驶技术的发展做出贡献，并满足大众对于更先进、 更安全交通技术的迫切需求。

个人承担工作：

我在项目中负责软件开发，主要承担了部分算法、功能的实现，包括但不限于：

图像处理与拼接：

1. 开发图像预处理算法，处理鱼眼图像的畸变。
2. 实现多个相机图像的拼接算法，生成 360° 全景图像。

感知算法：

3. 开发车道线识别、交通信号灯识别、路况检测等算法。
4. 结合深度学习技术，实现实时图像分析和处理。

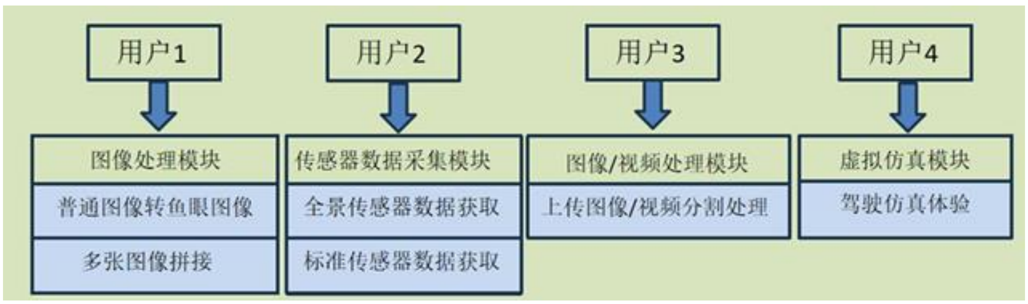
语义分割：

5. 设计针对鱼眼图像的语义分割算法。
6. 实现多摄像头组合的语义分割模型，解决图像畸变问题。

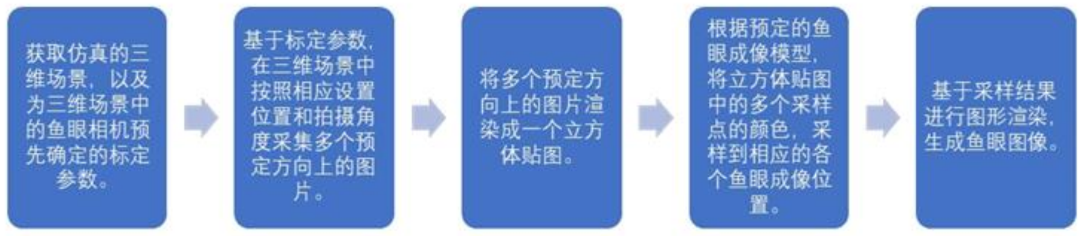
数据融合：

7. 将鱼眼相机图像与超声波雷达数据进行融合，实现精确的泊车辅助功能。
8. 结合高精度地图数据，实现精准定位与路径规划。

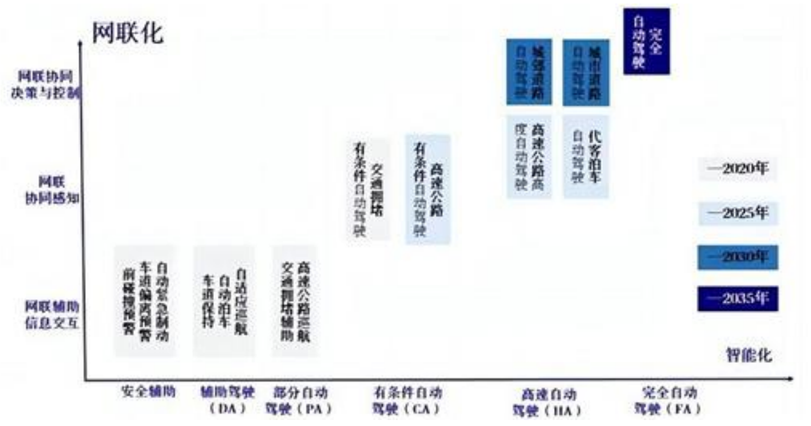
功能图：



逻辑结构图：


获取仿真的三维场景，以及为三维场景中的鱼眼相机预先确定的标定参数。
基于标定参数，在三维场景中按照相应设置位置和拍摄角度采集多个预定方向上的图片。
将多个预定方向上的图片渲染成一个立方体贴图。
根据预定的鱼眼成像模型，将立方体贴图中的多个采样点的颜色，采样到相应的各个鱼眼成像位置。
基于采样结果进行图形渲染，生成鱼眼图像。

需求分析图:



重要代码以及主要程序接口:

图像拼接的核心代码:

```python
import cv2
import numpy as np

def stitch_images(image1, image2):
    # 使用 OpenCV 的 Stitcher 类进行图像拼接
    stitcher = cv2.Stitcher_create()
    status, stitched_image = stitcher.stitch((image1, image2))

    if status == cv2.Stitcher_OK:
        return stitched_image
    else:
        print("图像拼接失败!")
        return None

# 示例图像路径（示例用，实际应用中需要替换为实际图像路径）
image1_path = 'image1.jpg'
image2_path = 'image2.jpg'

# 读取图像
image1 = cv2.imread(image1_path)
```

```python
image2 = cv2.imread(image2_path)

# 调整图像大小（如果需要）
# image1 = cv2.resize(image1, (new_width, new_height))
# image2 = cv2.resize(image2, (new_width, new_height))

# 进行图像拼接
result_image = stitch_images(image1, image2)

# 显示结果
if result_image is not None:
    cv2.imshow('Stitched Image', result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```python
# Copyright (c) OpenMMLab. All rights reserved.
import os
import os.path as osp
import platform
import shutil
import sys
import warnings
from setuptools import find_packages, setup


def readme():
    with open('README.md', encoding='utf-8') as f:
        content = f.read()
    return content


version_file = 'mmseg/version.py'


def get_version():
    with open(version_file) as f:
        exec(compile(f.read(), version_file, 'exec'))
    return locals()['__version__']


def parse_requirements(fname='requirements.txt', with_version=True):
    """Parse the package dependencies listed in a requirements file but strips
    specific versioning information.

    Args:
        fname (str): path to requirements file
        with_version (bool, default=False): if True include version specs

    Returns:
        List[str]: list of requirements items

    CommandLine:
        python -c "import setup; print(setup.parse_requirements())"
    """
    import re
    import sys
    from os.path import exists
    require_fpath = fname

    def parse_line(line):
        """Parse information from a line in a requirements text file."""
        if line.startswith('-r '):
            # Allow specifying requirements in other files
            target = line.split(' ')[1]
            for info in parse_require_file(target):
                yield info
        else:
            info = {'line': line}
            if line.startswith('-e '):
                info['package'] = line.split('#egg=')[1]
            else:
                # Remove versioning from the package
                pat = '(' + '|'.join(['>=', '==', '>']) + ')'
                parts = re.split(pat, line, maxsplit=1)
                parts = [p.strip() for p in parts]

                info['package'] = parts[0]
                if len(parts) > 1:
                    op, rest = parts[1:]
                    if ';' in rest:
                        # Handle platform specific dependencies
                        #
http://setuptools.readthedocs.io/en/latest/setuptools.html#declaring-platform-specif
ic-dependencies
                        version, platform_deps = map(str.strip,
                                                     rest.split(';'))
                        info['platform_deps'] = platform_deps
                    else:
                        version = rest  # NOQA
                    info['version'] = (op, version)
            yield info

    def parse_require_file(fpath):
        with open(fpath) as f:
            for line in f.readlines():
                line = line.strip()
                if line and not line.startswith('#'):
                    yield from parse_line(line)

    def gen_packages_items():
        if exists(require_fpath):
            for info in parse_require_file(require_fpath):
                parts = [info['package']]
                if with_version and 'version' in info:
                    parts.extend(info['version'])
                if not sys.version.startswith('3.4'):
                    # apparently package_deps are broken in 3.4
                    platform_deps = info.get('platform_deps')
                    if platform_deps is not None:
                        parts.append(';' + platform_deps)
                item = ''.join(parts)
                yield item

    packages = list(gen_packages_items())
    return packages


def add_mim_extension():
    """Add extra files that are required to support MIM into the package.

    These files will be added by creating a symlink to the originals if the
    package is installed in `editable` mode (e.g. pip install -e .), or by
    copying from the originals otherwise.
    """

    # parse installment mode
    if 'develop' in sys.argv:
        # installed by `pip install -e .`
        if platform.system() == 'Windows':
            # set `copy` mode here since symlink fails on Windows.
            mode = 'copy'
        else:
            mode = 'symlink'
    elif 'sdist' in sys.argv or 'bdist_wheel' in sys.argv or \
            platform.system() == 'Windows':
        # installed by `pip install .`
        # or create source distribution by `python setup.py sdist`
        # set `copy` mode here since symlink fails with WinError on Windows.
        mode = 'copy'
    else:
        return

    filenames = ['tools', 'configs', 'model-index.yml', 'dataset-index.yml']
    repo_path = osp.dirname(__file__)
    mim_path = osp.join(repo_path, 'mmseg', '.mim')
    os.makedirs(mim_path, exist_ok=True)

    for filename in filenames:
        if osp.exists(filename):
            src_path = osp.join(repo_path, filename)
            tar_path = osp.join(mim_path, filename)

            if osp.isfile(tar_path) or osp.islink(tar_path):
                os.remove(tar_path)
            elif osp.isdir(tar_path):
                shutil.rmtree(tar_path)

            if mode == 'symlink':
                src_relpath = osp.relpath(src_path, osp.dirname(tar_path))
                try:
                    os.symlink(src_relpath, tar_path)
                except OSError:
                    # Creating a symbolic link on windows may raise an
                    # `OSError: [WinError 1314]` due to privilege. If
                    # the error happens, the src file will be copied
                    mode = 'copy'
                    warnings.warn(
                        f'Failed to create a symbolic link for {src_relpath}, '
                        f'and it will be copied to {tar_path}')
                else:
                    continue

            if mode == 'copy':
                if osp.isfile(src_path):
                    shutil.copyfile(src_path, tar_path)
                elif osp.isdir(src_path):
                    shutil.copytree(src_path, tar_path)
            else:
                warnings.warn(f'Cannot copy file {src_path}.')
        else:
            raise ValueError(f'Invalid mode {mode}')


if __name__ == '__main__':
    add_mim_extension()
    setup(
        name='mmsegmentation',
        version=get_version(),
        description='Open MMLab Semantic Segmentation Toolbox and Benchmark',
        long_description=readme(),
        long_description_content_type='text/markdown',
        author='MMSegmentation Contributors',
        author_email='openmmlab@gmail.com',
        keywords='computer vision, semantic segmentation',
        url='https://github.com/open-mmlab/mmsegmentation',
        packages=find_packages(exclude=('configs', 'tools', 'demo')),
        include_package_data=True,
        classifiers=[
            'Development Status :: 4 - Beta',
            'License :: OSI Approved :: Apache Software License',
            'Operating System :: OS Independent',
            'Programming Language :: Python :: 3.6',
            'Programming Language :: Python :: 3.7',
            'Programming Language :: Python :: 3.8',
            'Programming Language :: Python :: 3.9',
        ],
        license='Apache License 2.0',
        install_requires=parse_requirements('requirements/runtime.txt'),
        extras_require={
            'all': parse_requirements('requirements.txt'),
            'tests': parse_requirements('requirements/tests.txt'),
            'optional': parse_requirements('requirements/optional.txt'),
            'mim': parse_requirements('requirements/mminstall.txt'),
            'multimodal': parse_requirements('requirements/multimodal.txt'),
        },
        ext_modules=[],
        zip_safe=False)
```

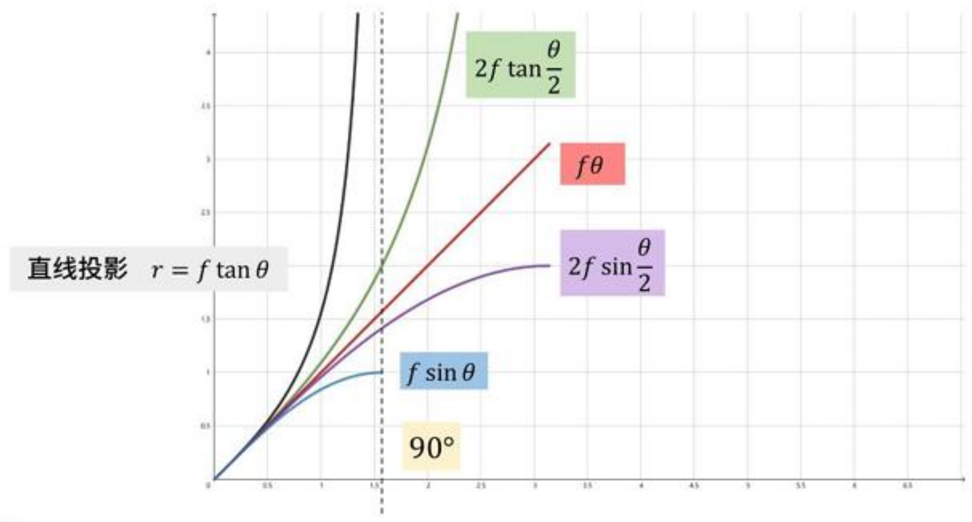


图 1-30 直线投影与四种经典的鱼眼投影模型公式

```python
import cv2
import numpy as np

def lane_detection(image):
    # 图像预处理：灰度化、高斯模糊、Canny 边缘检测
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)

    # 霍夫直线检测
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=20, minLineLength
=30, maxLineGap=100)

    # 绘制检测到的线段
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)

    return image

# 读取图像并应用车道线检测算法
image_path = 'lane_image.jpg'
image = cv2.imread(image_path)
result_image = lane_detection(image)

# 显示结果
cv2.imshow('Lane Detection Result', result_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```python

def traffic_light_detection(image):
    # 图像预处理和颜色分割（示例中简化处理）
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_red = np.array([0, 50, 50])
    upper_red = np.array([10, 255, 255])
    mask = cv2.inRange(hsv, lower_red, upper_red)

    # 轮廓检测
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # 绘制检测到的信号灯轮廓
    cv2.drawContours(image, contours, -1, (0, 255, 0), 2)

    return image

# 读取图像并应用交通信号灯检测算法
image_path = 'traffic_light_image.jpg'
image = cv2.imread(image_path)
result_image = traffic_light_detection(image)

# 显示结果
cv2.imshow('Traffic Light Detection Result', result_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
import argparse
import logging
import math
import os
import random
import time
from copy import deepcopy
from pathlib import Path
from threading import Thread

import numpy as np
import torch.distributed as dist
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.optim.lr_scheduler as lr_scheduler
import torch.utils.data
import yaml
```

```python
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

import test  # import test.py to get mAP after each epoch
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path, labels_to_image_weights, init_seeds, \
    fitness, fitness2, strip_optimizer, get_latest_run, check_dataset, check_file, check_git_status, check_img_size, \
    check_requirements, print_mutation, set_logging, one_cycle, colorstr
from utils.google_utils import attempt_download
from utils.loss import ComputeLoss, SegmentationLosses, SegFocalLoss, OhemCELoss, ProbOhemCrossEntropy2d
from utils.plots import plot_images, plot_labels, plot_results, plot_evolution
from utils.torch_utils import ModelEMA, select_device, intersect_dicts, torch_distributed_zero_first, is_parallel
from utils.wandb_logging.wandb_utils import WandbLogger, check_wandb_resume

import SegmentationDataset
import torch.backends.cudnn as cudnn


logger = logging.getLogger(__name__)


def train(hyp, opt, device, tb_writer=None):
    logger.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))  # 打印超参数
    save_dir, epochs, batch_size, total_batch_size, weights, rank = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.total_batch_size, opt.weights, opt.global_rank

    # Directories
    wdir = save_dir / 'weights'
    wdir.mkdir(parents=True, exist_ok=True)  # make dir
    last = wdir / 'last.pt'
    best = wdir / 'best.pt'
    results_file = save_dir / 'results.txt'

    # Save run settings # 存超参数和优化器参数，优化器参数,可用于 resume
    with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.dump(vars(opt), f, sort_keys=False)

    # Configure
    plots = not opt.evolve  # create plots  不进化就画图
    cuda = device.type != 'cpu'
    init_seeds(2 + rank)
    with open(opt.data) as f:
        data_dict = yaml.load(f, Loader=yaml.SafeLoader)  # data dict
    is_coco = opt.data.endswith('coco.yaml')

    # Logging- Doing this before checking the dataset. Might update data_dict
    loggers = {'wandb': None}  # loggers dict
    if rank in [-1, 0]:  # -1 不开 DDP, 0 是 DDP 主进程
        opt.hyp = hyp  # add hyperparameters
        run_id = torch.load(weights).get('wandb_id') if weights.endswith('.pt') and os.path.isfile(weights) else None
        wandb_logger = WandbLogger(opt, Path(opt.save_dir).stem, run_id, data_dict)
        loggers['wandb'] = wandb_logger.wandb
        data_dict = wandb_logger.data_dict
        if wandb_logger.wandb:
            weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp  # WandbLogger might update weights, epochs if resuming

    nc = 1 if opt.single_cls else int(data_dict['nc'])  # number of classes
    names = ['item'] if opt.single_cls and len(data_dict['names']) != 1 else data_dict['names']  # class names
    assert len(names) == nc, '%g names found for nc=%g dataset in %s' % (len(names), nc, opt.data)  # check

    # Model
    pretrained = weights.endswith('.pt')  # 有 weights 输入就用其初始化
    if pretrained:  # 有预训练参数
        with torch_distributed_zero_first(rank):
            attempt_download(weights)  # download if not found locally
        ckpt = torch.load(weights, map_location=device)  # load checkpoint
```

```python
        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)  # create
        exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and not opt.resume else []  # exclude keys 初始化时候不使用的参数(非resume且有配置时按cfg和model初始化来指定anchor, 否则延用pretrained weights的anchor)
        state_dict = ckpt['model'].float().state_dict()  # to FP32 ckpt里model键对应的值才是模型,训练结束后保存的是float16(中间保存的float32),模型的state_dict是参数,包括可训练参数和register_buffer保存的buffer parameter(Detect的anchor和gird等)
        state_dict = intersect_dicts(state_dict, model.state_dict(), exclude=exclude)  # intersect 赋值参数,预训练的参数赋值到新建的模型,exclude除外(即anchor使用cfg的而不是预训练)
        model.load_state_dict(state_dict, strict=False)  # load 调整好的预训练参数加载到模型
        logger.info('Transferred %g/%g items from %s' % (len(state_dict), len(model.state_dict()), weights))  # report
    else:  # 无预训练参数,只建模型
        model = Model(opt.cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)  # create
    with torch_distributed_zero_first(rank):
        check_dataset(data_dict)  # check
    train_path = data_dict['train']
    test_path = data_dict['val']
    segtrain_path = data_dict['segtrain']
    segval_path = data_dict['segval']

    # Freeze 要冻结的参数 似乎只能在此代码处手动设置列表
    freeze = []  # parameter names to freeze (full or partial)
    for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers 全部参数可导
        if any(x in k for x in freeze):  # 碰见freeze列表的层使其参数不可导
            print('freezing %s' % k)
            v.requires_grad = False

    # Optimizer
    nbs = 64  # nominal batch size
    accumulate = max(round(nbs / total_batch_size), 1)  # accumulate loss before optimizing
    hyp['weight_decay'] *= total_batch_size * accumulate / nbs  # scale weight_decay 权重衰减系数
    logger.info(f"Scaled weight_decay = {hyp['weight_decay']}")

    # 参数分组,pg0是BN,pg1是权重,pg2是偏置
    pg0, pg1, pg2 = [], [], []  # optimizer parameter groups
    for k, v in model.named_modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):
            pg2.append(v.bias)  # biases
        if isinstance(v, nn.BatchNorm2d):
            pg0.append(v.weight)  # no decay
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):
            pg1.append(v.weight)  # apply decay
    # 优化器初始化，BN 层参数不带权重衰减
    if opt.adam:
        optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  # adjust beta1 to momentum
    else:
        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True)
    # 权重参数，带权重衰减
    optimizer.add_param_group({'params': pg1, 'weight_decay': hyp['weight_decay']})  # add pg1 with weight_decay
    # 偏置参数，同样不带衰减
    optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
    logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' % (len(pg2), len(pg1), len(pg0)))
    del pg0, pg1, pg2

    # Scheduler https://arxiv.org/pdf/1812.01187.pdf 设置好优化器后用其设置 Learning Scheduler
    # https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#OneCycleLR
    if opt.linear_lr:
        lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf']) + hyp['lrf']  # linear
    else:
        lf = one_cycle(1, hyp['lrf'], epochs)  # cosine 1->hyp['lrf']
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)  # 自定义 lambda 函数学习率衰减策略
    # plot_lr_scheduler(optimizer, scheduler, epochs)

    # EMA 指数滑动平均
    ema = ModelEMA(model) if rank in [-1, 0] else None

    # Resume
    start_epoch, best_fitness = 0, 0.0
    if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
            optimizer.load_state_dict(ckpt['optimizer'])
            best_fitness = ckpt['best_fitness']

        # EMA 指数平均
        if ema and ckpt.get('ema'):
            ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
            ema.updates = ckpt['updates']

        # Results
        if ckpt.get('training_results') is not None:
            results_file.write_text(ckpt['training_results'])  # write results.txt

        # Epochs
        start_epoch = ckpt['epoch'] + 1  # 预训练模型的 epoch 是-1
        if opt.resume:  # resume 参数 epoch 应该大于 0
            assert start_epoch > 0, '%s training to %g epochs is finished, nothing to resume.' % (weights, epochs)
        if epochs < start_epoch:  # 总轮数比开始轮还小, 总轮数加上已训练轮(即再训练总轮数次而不是通常的 总轮数-开始轮 次)
            logger.info('%s has been trained for %g epochs. Fine-tuning for %g additional epochs.' %
                        (weights, ckpt['epoch'], epochs))
            epochs += ckpt['epoch']  # finetune additional epochs

        del ckpt, state_dict

    # Image sizes
    gs = max(int(model.stride.max()), 32)  # grid size (max stride)  至少 32
    nl = model.model[-1].nl  # number of detection layers (used for scaling hyp['obj']) model 最后一层是 Detect, nl 是其输出层数量
    imgsz, imgsz_test = [check_img_size(x, gs) for x in opt.img_size]  # verify imgsz are gs-multiples 检查图片尺寸是否合法,不合法就自动替换

    # DP mode DP 多线程数据并行模式, 不使用, 并行推荐 DDP 多进程
    if cuda and rank == -1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)

    # SyncBatchNorm 跨卡 BN, 仅支持 DDP
    if opt.sync_bn and cuda and rank != -1:
        model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
        logger.info('Using SyncBatchNorm()')
```

```python

    # 检测 Trainloader
    dataloader, dataset = create_dataloader(train_path, imgsz, batch_size, gs, opt,
                                            hyp=hyp, augment=True, cache=opt.cache_images, rect=opt.rect, rank=rank,
                                            world_size=opt.world_size, workers=opt.workers,
                                            image_weights=opt.image_weights, quad=opt.quad, prefix=colorstr('train: '))
    mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # max label class 纵向连接了标签后找第一列最大值, mlc 的值就是类别数-1
    nb = len(dataloader)  # number of batches
    # mlc=实际标签类别数-1 应该小于 nc 模型结构支持的前景类别数 (不用等式关系, 因为结构类别多的模型可以支持训练标签类别少的数据, 反之不成立)
    assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible class labels are 0-%g' % (mlc, nc, opt.data, nc - 1)

    # Process 0  非 DDP 或 DDP 中的主进程
    if rank in [-1, 0]:
        testloader = create_dataloader(test_path, imgsz_test, batch_size * 2, gs, opt,  # testloader batch_size 翻倍
                                       hyp=hyp, cache=opt.cache_images and not opt.notest, rect=True, rank=-1,
                                       world_size=opt.world_size, workers=opt.workers,
                                       pad=0.5, prefix=colorstr('val: '))[0]  # [0]只要了 loader 没要 dataset, 和 train 处理不一样

        if not opt.resume:  # 常规, 非 resume
            labels = np.concatenate(dataset.labels, 0)
            c = torch.tensor(labels[:, 0])  # classes 所有对象类别(包括所有目标,不是图像)
            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # frequency
            # model._initialize_biases(cf.to(device))
            if plots:
                plot_labels(labels, names, save_dir, loggers)
                if tb_writer:
                    tb_writer.add_histogram('classes', c, 0)  #

            # Anchors
            if not opt.noautoanchor:  # 用 train dataset 自动聚类选取最好 anchor
```

```python
                check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)  # anchor_t 是最大放大倍数,yolov5 公式不同于 v3v4, 见核心 Model 推理时 anchor 偏移放缩公式和 issue
            model.half().float()  # pre-reduce anchor precision 先转 float16 再转回 32,虽然 type 是 32,但此时参数的数值范围限到 16 了

    # 分割 loader   citys 和 bdd 的 basesize 都取 1024,crop_size 长边取 imgsz,短边 imgsz 砍半
        seg_valloader = SegmentationDataset.get_citys_loader(root=segval_path, batch_size=4,
                                                         split="val", mode="testval",  # 旧版为 val 新版训练中验证也用 testval 模式
                                                         base_size=1024,  # 对 cityscapes, 原图 resize 到(1024, 512)输入后双线性插值到原图尺寸计算精度
                                                         # crop_size=640,  # testval 时候 cropsize 不起作用
                                                         workers=4, pin=True)  # 验证 batch_size 和 workers 得配合, 都太大会导致子进程死亡, 单进程龟速加载数据
                                                                     # 我电脑上(4,4)是最快的, 更大子进程会挂(现在图大了,怎么设都会挂, BUG)
    seg_trainloader = SegmentationDataset.get_citys_loader(root=segtrain_path,
                                                           split="train", mode="train",
                                                           base_size=1024, crop_size=(imgsz, imgsz//2),  # cropsize 长边和检测同, 短边砍半(针对 cityscapes,bdd 也行)
                                                           batch_size=batch_size,
                                                           workers=opt.workers, pin=True)

    segnb = len(seg_trainloader)
    # DDP mode
    if cuda and rank != -1:  # 没禁用(-1)就开 DDP 模型
        model = DDP(model, device_ids=[opt.local_rank], output_device=opt.local_rank,
                    # nn.MultiheadAttention incompatibility with DDP https://github.com/pytorch/pytorch/issues/26698
                    find_unused_parameters=any(isinstance(layer, nn.MultiheadAttention) for layer in model.modules()))

    # Model parameters 根据输出层数,类别数等调整损失增益,模型超参数
    hyp['box'] *= 3. / nl  # scale to layers
    hyp['cls'] *= nc / 80. * 3. / nl  # scale to classes and layers
    hyp['obj'] *= (imgsz / 640) ** 2 * 3. / nl  # scale to image size and layers
    hyp['label_smoothing'] = opt.label_smoothing
    model.nc = nc  # attach number of classes to model
    model.hyp = hyp  # attach hyperparameters to model
    model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
    model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device) * nc  # attach class weights
    model.names = names

    # Start training
    t0 = time.time()
    nw = max(round(hyp['warmup_epochs'] * nb), 800)  # number of warmup iterations, max(3 epochs, 1k iterations) 最少warmup三轮或500batch(原版1000,800就够了)
    # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup to < 1/2 of training
    maps = np.zeros(nc)  # mAP per class
    results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
    scheduler.last_epoch = start_epoch - 1  # do not move 配置lr_scheduler起始位置
    scaler = amp.GradScaler(enabled=cuda)  # 说明不是float16训练,而是16和32混合精度训练. 训练前初始化loss scaler 用于float16放大梯度后backward, optimizer.step之前自动转float32再缩回来
    compute_loss = ComputeLoss(model)  # init loss class 初始化检测criteria

#----------------------------------------------------------------------------------------------------------
    # deeplab早期版本(无ohem)中对cityscapes数据集设定的weights,可用,非必要,现代更常用ohem, 但目前ohem实验略差一点
    # citys_class_weight=torch.tensor([0.8373, 0.9180, 0.8660, 1.0345, 1.0166,
    #                                  0.9969, 0.9754, 1.0489, 0.8786, 1.0023,
    #                                  0.9539, 0.9843, 1.1116, 0.9037, 1.0865,
    #                                  1.0955, 1.0865, 1.1529, 1.0507])
    citys_class_weight = None

    # 无aux模型输出不用[],有aux几个结果输出用[]包装
    # Base，PSP 和 Lab 用这个，无 aux
    compute_seg_loss = SegmentationLosses(aux=False, ignore_index=-1, weight=citys_class_weight).cuda()#SegFocalLoss(ignore_index=-1, gamma=1, reduction="mean").cuda()
    # BiSe 用这个  两个 aux
    # compute_seg_loss = SegmentationLosses(nclass=19, aux=True, aux_num=2, aux_weight=0.1, ignore_index=-1, weight=citys_class_weight).cuda()
    # 一个 aux，没有用这个
    # compute_seg_loss = SegmentationLosses(nclass=19, aux=True, aux_num=1, aux_weight=0.1, ignore_index=-1, weight=citys_class_weight).cuda()
#------------------------------------------------------------------------------------------------------------

    # focalloss 别用，cityscapes 效果不行
    # OHEM 能用，理论上应该超过 CE，但是目前实验效果不如 CE(设成默认 0.7 收敛蛮快的但最终值不够好)，认为与计算像素个数和学习率有关，用的话循环损失计算的语句得改一下，接口和 CE 还没来得及保持一致
    # compute_seg_loss = OhemCELoss(thresh=0.7, ignore_index=-1, aux=False)
    # compute_seg_loss = OhemCELoss(thresh=0.7, ignore_index=-1, aux=True, aux_weight=[0.15, 0.1])

    detgain, seggain = 0.6, 0.35  # 检测, 分割比例
    # CE、1/8 单输入、batchsize13 用 0.65,0.35 左右,注意 64 向下取整的梯度积累，比 13*4=52 大(12*5=64)通常应该降低分割损失比例或调小学习率



    logger.info(f'Image sizes {imgsz} train, {imgsz_test} test\n'
                f'Using {dataloader.num_workers} dataloader workers\n'
                f'Logging results to {save_dir}\n'
                f'Starting training for {epochs} epochs...')
    for epoch in range(start_epoch, epochs):  # epoch ------------------------------------------------------------------
        mIoU = 0  # 每轮开始 mIoU 设置成 0 ，因为选模型按 m IoU 选，为了加速训练可能 n 轮才测一次 mIoU，对没测 mIoU 的模型不会存为 best.pt
        print(f'accumulate: {accumulate}')  # 显示 epoch 开始时梯度积累次数(第一个值忽略, 注意 warmup 期间按 batch 变化, 此处只是辅助观察防梯度爆炸)
        model.train()  # epoch 开始, 确保 train 模式 注意 validation 时候可能会把模型.eval()因此开始的 train()很有必要

        # Update image weights (optional) 更新 image_weights 权重, 默认不开 image_weights 忽略此块代码
        if opt.image_weights:
            # Generate indices
            if rank in [-1, 0]:
                cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc  # class weights
                iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)  # image weights
                dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)  # rand weighted idx
            # Broadcast if DDP
            if rank != -1:
                indices = (torch.tensor(dataset.indices) if rank == 0 else torch.zeros(dataset.n)).int()
                dist.broadcast(indices, 0)
                if rank != 0:
                    dataset.indices = indices.cpu().numpy()

        # Update mosaic border
        # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
        # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

        mloss = torch.zeros(4, device=device)  # 检测 mean losses
        msegloss = torch.zeros(1, device=device)  # 混合的 mean losses, 两者计算也可知分割 loss
        if rank != -1:
            dataloader.sampler.set_epoch(epoch)  # shuffle 时, 保证每个 epoch 顺序不同
        pbar = enumerate(dataloader)
        segpbar = enumerate(seg_trainloader)
        logger.info(('\n' + '%10s' * 9) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls', 'total', 'seg', 'labels', 'img_size'))
        if rank in [-1, 0]:
            pbar = tqdm(pbar, total=min(nb, segnb))  # progress bar # tqdm 进度条迭代
            segpbar = tqdm(segpbar, total=min(nb, segnb))
        optimizer.zero_grad()  # 每轮前清空梯度

        # 暂时用 zip, 每轮 batch 数以数量少的为准
        for det_batch, seg_batch in zip(pbar, segpbar):  # batch -------------------------------------------------------------
            i, (imgs, targets, paths, _) = det_batch  # 检测
            _, (segimgs, segtargets) = seg_batch   # 分割
            if len(imgs)==1 or len(segimgs)==1:  # 手动 droplast,SE 或者
```

```
gloablpool 后的 bn 不支持单个样本，检测 loader 调用地方太多不好 droplast，这里手动
                    continue
            # warmup 等参数变化以检测为准
            ni = i + nb * epoch  # number integrated batches (since train start) 记录总 iterations, 可以用于停止 warmup
            imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to float32, 0-255 to 0.0-1.0
            # Warmup
            if ni <= nw:
                xi = [0, nw]  # x interp
                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou loss ratio (obj_loss = 1.0 or iou)  # 修改了 accumulate 上限,使其不超过 nbs(防止 Nan)
                accumulate = max(1, np.interp(ni, xi, [1, math.floor(nbs / total_batch_size)]).round())  # 梯度积累 线性插值 xi=[0, 1000], yi=[1, 64/batchsize], 插入点 x=ni, 之后取整, 最小限 1. warmup 时 accumulate 会逐渐从 1 按整数增大到目标, warmup 结束后稳定在目标值 round(nbs/accumulate), 例如 batchsize32 实际上两 batch 才更新一次,等效于 64
                for j, x in enumerate(optimizer.param_groups):  # warmup 过程中逐渐把三组参数的 lr 调到 lr0
                    # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to lr0
                    x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else 0.0, x['initial_lr'] * lf(epoch)])
                    if 'momentum' in x:
                        x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'], hyp['momentum']])
            # Multi-scale 默认关 multi scale
            if opt.multi_scale:
                sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs  # size
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                    ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]]  # new shape (stretched to gs-multiple)
                    imgs = F.interpolate(imgs, size=ns, mode='bilinear', align_corners=False)

            # Forward and Backward 对比原版 yolov5 此处修改, 否则 batchsize 只能取单检测时候的一半, 这种写法可以更大一点

            with amp.autocast(enabled=cuda):  # 混合精度训练中用来代替 autograd
                pred = model(imgs)  # forward
                loss, loss_items = compute_loss(pred[0], targets.to(device)
)  # loss scaled by batch_size
                if rank != -1:  # DDP 中 loss * GPU 数
                    loss *= opt.world_size  # gradient averaged between devices in DDP mode
                if opt.quad:
                    loss *= 4.
                loss *= detgain  # 检测 loss 比例
            scaler.scale(loss).backward()
            imgshape = imgs.shape[-1]
            if plots and ni >= 3:
                del imgs  # 前三个 batch 画图不能 del
            else:
                imgs = imgs.to(torch.device('cpu'), non_blocking=True)  # 释放 segimgs 输入后就没被调用会被 pytorch 自动回收不用手动释放(img 后续有被调用要手动释放)

            segimgs = segimgs.to(device, non_blocking=True)  # 分割已经做过 totensor 了, 不用/255

            with amp.autocast(enabled=cuda):  # 混合精度训练中用来代替 autograd
                pred = model(segimgs)
 #-----------------------------------------------------------------------------------------------------------
                # 无 aux 模型输出不用[],有 aux 模型几个结果输出用[]包装
                # Base,PSP 和 Lab 用这个,无 aux
                segloss = compute_seg_loss(pred[1], segtargets.to(device)) * batch_size # 分割 loss CE 是平均 loss, 配合检测做梯度积累, 因此乘以 batchsize(注意有梯度积累其真实 batchsize 约是 nbs=64)
                # Bise 用这个,两个 aux
                # segloss = compute_seg_loss(pred[1][0], pred[1][1], pred[1][2], segtargets.to(device)) * batch_size
                # 一个 aux, 没有用这个
                # segloss = compute_seg_loss(pred[1][0], pred[1][1], segtargets.to(device)) * batch_size
 #-----------------------------------------------------------------------------------------------------------
                segloss *= seggain
            scaler.scale(segloss).backward()
            del segimgs

            # Optimize
            if ni % accumulate == 0:  # 梯度积累 accumulate 次后才优化,
                scaler.step(optimizer)  # optimizer.step  # 混合精度训练优化
```

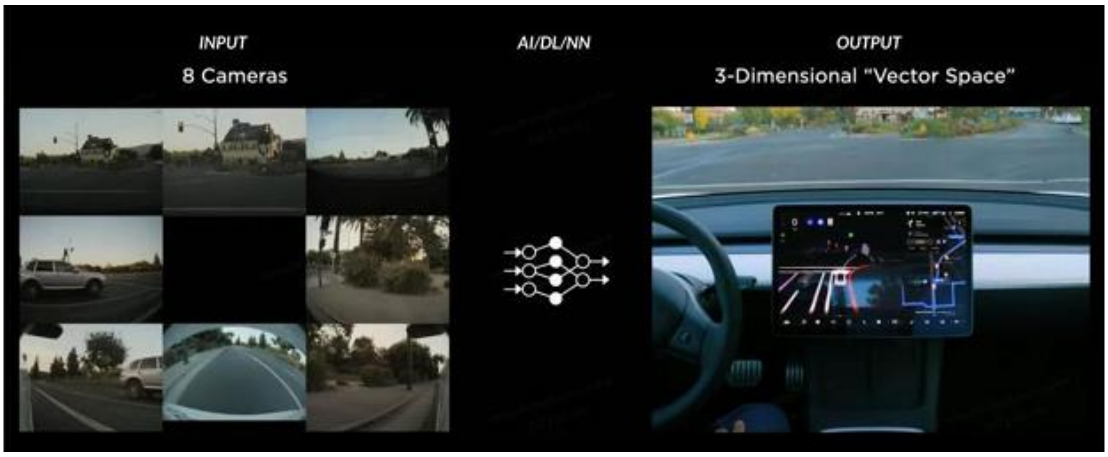
```python
时用 scaler
                scaler.update()
                optimizer.zero_grad()  # 每次更新完参数才清空梯度，不更新时累计
                if ema:  # 不开 DDP 和 DDP 主进程中 ema 开启，每次更新 ema
                    ema.update(model)

            # Print
            if rank in [-1, 0]:
                mloss = (mloss * i + loss_items) / (i + 1)  # update mean losses
                msegloss = (msegloss * i + segloss.detach()/total_batch_size) / (i + 1)
                mem = '%.3gG' % (torch.cuda.memory_reserved() / 1E9 if torch.cuda.is_available() else 0)  # (GB)
                s = ('%10s' * 2 + '%10.4g' * 7) % (
                    '%g/%g' % (epoch, epochs - 1), mem, *mloss, msegloss, targets.shape[0], imgshape)
                pbar.set_description(s)

                # Plot
                if plots and ni < 3:
                    f = save_dir / f'train_batch{ni}.jpg'  # filename
                    Thread(target=plot_images, args=(imgs, targets, paths, f), daemon=True).start()
                    # if tb_writer:
                    #     tb_writer.add_image(f, result, dataformats='HWC', global_step=epoch)
                    #     tb_writer.add_graph(torch.jit.trace(model, imgs, strict=False), [])  # add model graph
                elif plots and ni == 10 and wandb_logger.wandb:
                    wandb_logger.log({"Mosaics": [wandb_logger.wandb.Image(str(x), caption=x.name) for x in
                                                  save_dir.glob('train*.jpg') if x.exists()]})

            # end batch ------------------------------------------------------------------------------------------------
        # end epoch ----------------------------------------------------------------------------------------------------

        # Scheduler
        lr = [x['lr'] for x in optimizer.param_groups]  # for tensorboard
        scheduler.step()  # 更新 Scheduler


        # DDP process 0 or single-GPU
        if rank in [-1, 0]:
            ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'gr', 'names', 'stride', 'class_weights'])
            # pixACC, mIoU
            if epoch % 10 == 0 or (epochs - epoch) < 40:
                mIoU = test.seg_validation(model=ema.ema, valloader=seg_valloader, device=device, n_segcls=19,
                                half_precision=True)
            # mAP
            final_epoch = epoch + 1 == epochs  # 是否是最后一轮
            if not opt.notest or final_epoch:  # Calculate mAP
                wandb_logger.current_epoch = epoch + 1
                results, maps, times = test.test(data_dict,
                                                 batch_size=batch_size * 2,

                                                 imgsz=imgsz_test,
                                                 model=ema.ema,
                                                 single_cls=opt.single_cls,

                                                 dataloader=testloader,
                                                 save_dir=save_dir,
                                                 verbose=nc < 50 and final_epoch,
                                                 plots=plots and final_epoch,
                                                 wandb_logger=wandb_logger,

                                                 compute_loss=compute_loss,

                                                 is_coco=is_coco)

            # Write
            with open(results_file, 'a') as f:
                f.write(s + '%10.4g' * 7 % results + '\n')  # append metrics, val_loss
            if len(opt.name) and opt.bucket:
                os.system('gsutil cp %s gs://%s/results/results%s.txt' % (results_file, opt.bucket, opt.name))

            # Log
            tags = ['train/box_loss', 'train/obj_loss', 'train/cls_loss',
# train loss
                    'metrics/precision', 'metrics/recall', 'metrics/mAP_0.5', 'metrics/mAP_0.5:0.95',
                    'val/box_loss', 'val/obj_loss', 'val/cls_loss',  # val loss
                    'x/lr0', 'x/lr1', 'x/lr2']  # params
            for x, tag in zip(list(mloss[:-1]) + list(results) + lr, tags):  # 写 tensorboard
                if tb_writer:
                    tb_writer.add_scalar(tag, x, epoch)  # tensorboard
                if wandb_logger.wandb:
                    wandb_logger.log({tag: x})  # W&B

            # Update best mIoU  #mAP
            # fi = fitness(np.array(results).reshape(1, -1))  # weighted combination of [P, R, mAP@.5, mAP@.5-.95] 按 0.1*AP.5+0.9*AP.5:.95 指标衡量模型
            fi = fitness2(np.array(results).reshape(1, -1), mIoU)  # weighted combination of [P, R, mAP@.5, mAP@.5-.95] 按 0.1*AP.5+0.9*AP.5:.95 指标衡量模型

            if fi > best_fitness:
                best_fitness = fi
            wandb_logger.end_epoch(best_result=best_fitness == fi)

            # Save model
            if (not opt.nosave) or (final_epoch and not opt.evolve):  # if save
                ckpt = {'epoch': epoch,
                        'best_fitness': best_fitness,
                        'training_results': results_file.read_text(),
                        'model': deepcopy(model.module if is_parallel(model) else model).half(),
                        'ema': deepcopy(ema.ema).half(),
                        'updates': ema.updates,
                        'optimizer': optimizer.state_dict(),
                        'wandb_id': wandb_logger.wandb_run.id if wandb_logger.wandb else None}

                # Save last, best and delete
                torch.save(ckpt, last)
                if best_fitness == fi:
                    torch.save(ckpt, best)
                if wandb_logger.wandb:
                    if ((epoch + 1) % opt.save_period == 0 and not final_ep
```

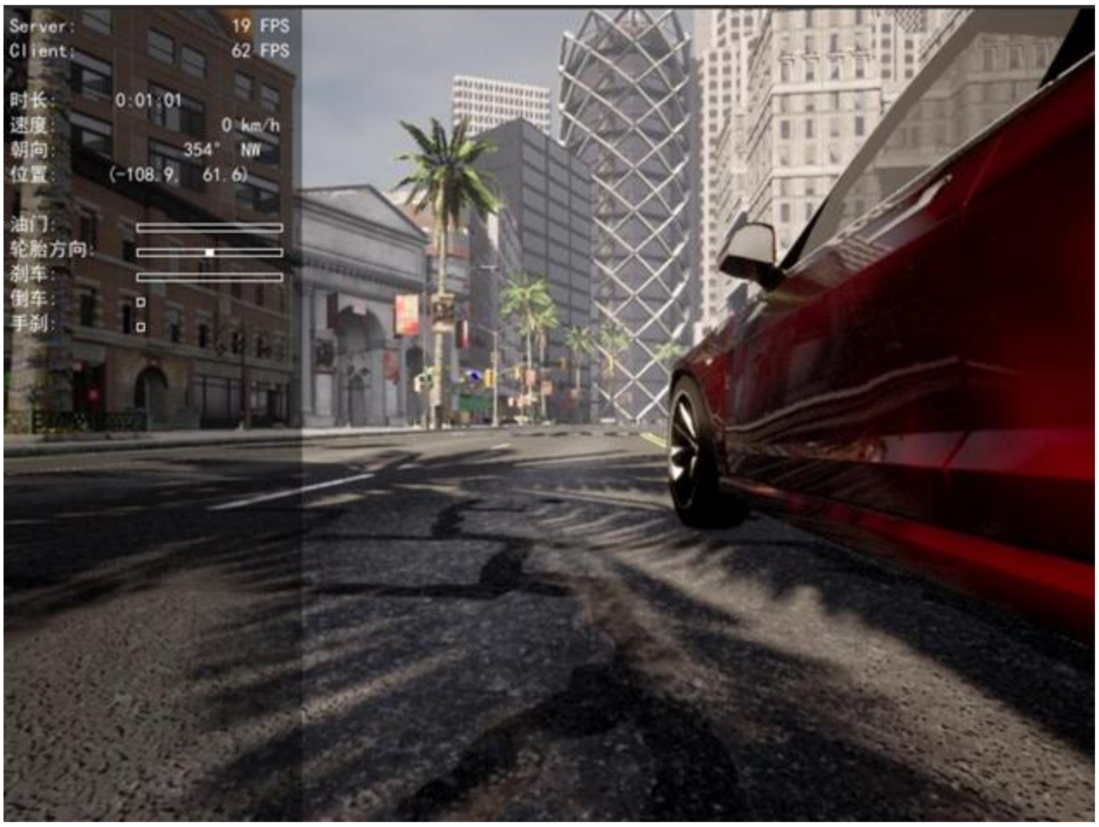
```python
och) and opt.save_period != -1:
                        wandb_logger.log_model(
                            last.parent, opt, epoch, fi, best_model=best_fitness == fi)
                del ckpt

        # end epoch ----------------------------------------------------------------------------------------------------
    # end training
    if rank in [-1, 0]:
        # Plots
        if plots:  # 不进化就画图
            plot_results(save_dir=save_dir)  # save as results.png
            if wandb_logger.wandb:
                files = ['results.png', 'confusion_matrix.png', *[f'{x}_curve.png' for x in ('F1', 'PR', 'P', 'R')]]
                wandb_logger.log({"Results": [wandb_logger.wandb.Image(str(save_dir / f), caption=f) for f in files
                                              if (save_dir / f).exists()]})

        # Test best.pt
        logger.info('%g epochs completed in %.3f hours.\n' % (epoch - start_epoch + 1, (time.time() - t0) / 3600))
        if opt.data.endswith('coco.yaml') and nc == 80:  # if COCO
            for m in (last, best) if best.exists() else (last):  # speed, mAP tests
                results, _, _ = test.test(opt.data,
                                          batch_size=batch_size * 2,
                                          imgsz=imgsz_test,
                                          conf_thres=0.001,
                                          iou_thres=0.7,
                                          model=attempt_load(m, device).half(),
                                          single_cls=opt.single_cls,
                                          dataloader=testloader,
                                          save_dir=save_dir,
                                          save_json=True,
                                          plots=False,
                                          is_coco=is_coco)

        # Strip optimizers
        final = best if best.exists() else last  # final model
        for f in last, best:
            if f.exists():
                strip_optimizer(f)  # strip optimizers
        if opt.bucket:
            os.system(f'gsutil cp {final} gs://{opt.bucket}/weights')  # upload
        if wandb_logger.wandb and not opt.evolve:  # Log the stripped model
            wandb_logger.wandb.log_artifact(str(final), type='model',
                                            name='run_' + wandb_logger.wandb_run.id + '_model',
                                            aliases=['last', 'best', 'stripped'])
        wandb_logger.finish_run()
    else:
        dist.destroy_process_group()
    torch.cuda.empty_cache()
    return results


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml', help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640], help='[train, test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyper
```

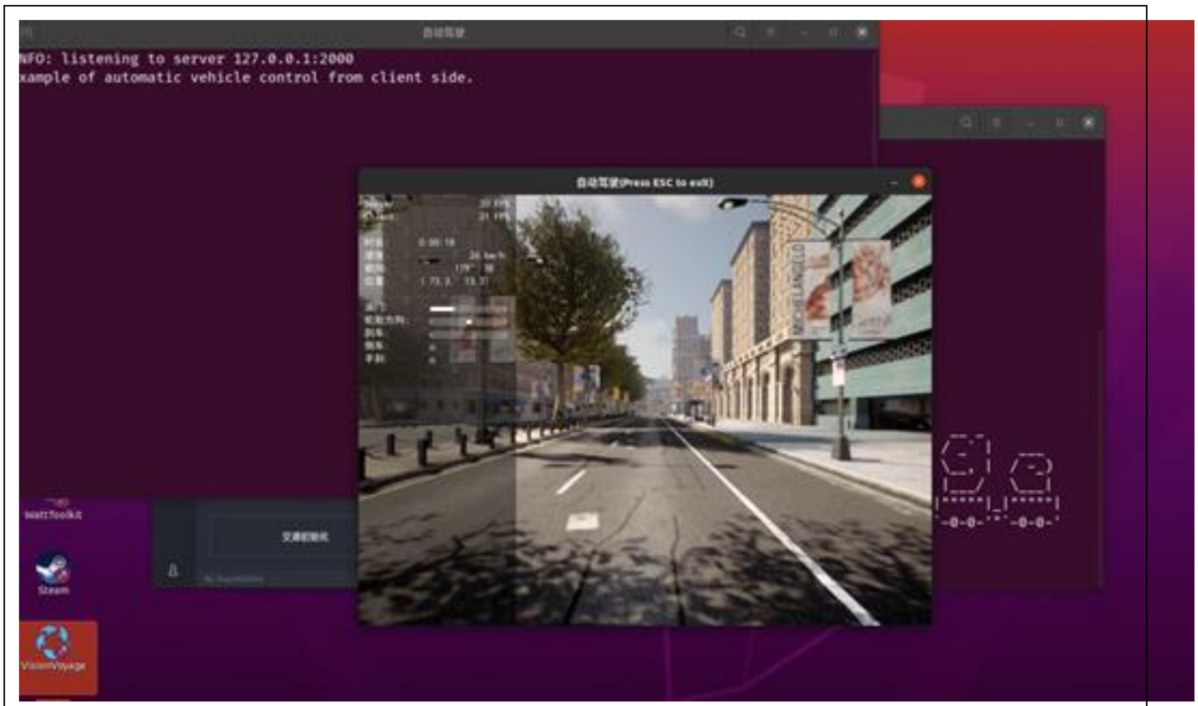
```python
parameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
    parser.add_argument('--workers', type=int, default=8, help='maximum number of dataloader workers')
    parser.add_argument('--project', default='runs/train', help='save to project/name')
    parser.add_argument('--entity', default=None, help='W&B entity')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--linear-lr', action='store_true', help='linear LR')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
    parser.add_argument('--upload_dataset', action='store_true', help='Upload dataset as W&B artifact table')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='Set bounding-box image logging interval for W&B')
    parser.add_argument('--save_period', type=int, default=-1, help='Log model after every "save_period" epoch')
    parser.add_argument('--artifact_alias', type=str, default="latest", help='version of dataset artifact to be used')

    opt = parser.parse_args()

    # Set DDP variables  DDP 常规初始化
    opt.world_size = int(os.environ['WORLD_SIZE']) if 'WORLD_SIZE' in os.environ else 1  # 获取总进程数 world_size
    opt.global_rank = int(os.environ['RANK']) if 'RANK' in os.environ else -1  # global_rank 是所有进程可用的 GPU 号, local_rank 是当前进程对应 GPU 号
    set_logging(opt.global_rank)
    if opt.global_rank in [-1, 0]:
        # check_git_status()  # 检测 git 版本,网络不好会卡住,手动关闭
        check_requirements()

    # Resume
    wandb_run = check_wandb_resume(opt)  # wandb 有 bug,没装
    # 断点重续且没有 wandb 库
    if opt.resume and not wandb_run:  # resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()  # specified or most recent path 找要续的模型 pt
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
        apriori = opt.global_rank, opt.local_rank
        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:  # 找 优化器 配置文件
            opt = argparse.Namespace(**yaml.load(f, Loader=yaml.SafeLoader))  # replace
        opt.cfg, opt.weights, opt.resume, opt.batch_size, opt.global_rank, opt.local_rank = '', ckpt, True, opt.total_batch_size, *apriori  # reinstate

        logger.info('Resuming training from %s' % ckpt)
    else:
        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights else 'hyp.scratch.yaml')
        opt.data, opt.cfg, opt.hyp = check_file(opt.data), check_file(opt.cfg), check_file(opt.hyp)  # check files
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be specified'  # cfg 和 weights 至少有一个
        opt.img_size.extend([opt.img_size[-1]] * (2 - len(opt.img_size)))  # extend to 2 sizes (train, test)
        opt.name = 'evolve' if opt.evolve else opt.name  # project 名字,用于保存文件夹
        opt.save_dir = increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok | opt.evolve)  # increment run

    # DDP mode 数据多进程并行
    opt.total_batch_size = opt.batch_size  # 总 batchsize
    device = select_device(opt.device, batch_size=opt.batch_size)  # 设备数
    if opt.local_rank != -1:  # 默认是-1 不开启 DDP
        assert torch.cuda.device_count() > opt.local_rank
        torch.cuda.set_device(opt.local_rank)
        device = torch.device('cuda', opt.local_rank)
        dist.init_process_group(backend='nccl', init_method='env://')  # distributed backend DDP 初始化进程组
        assert opt.batch_size % opt.world_size == 0, '--batch-size must be multiple of CUDA device count'  # 一般一卡开一进程, batchsize 可被进程数整除
        opt.batch_size = opt.total_batch_size // opt.world_size  # 每个进程 batchsize

    # Hyperparameters 配置超参数
    with open(opt.hyp) as f:
        hyp = yaml.load(f, Loader=yaml.SafeLoader)  # load hyps

    # Train
    logger.info(opt)
    if not opt.evolve:  # 没有用进化算法(默认)
        tb_writer = None  # init loggers
        if opt.global_rank in [-1, 0]:
            prefix = colorstr('tensorboard: ')
            logger.info(f"{prefix}Start with 'tensorboard --logdir {opt.project}', view at http://localhost:6006/")
            tb_writer = SummaryWriter(opt.save_dir)  # Tensorboard
        train(hyp, opt, device, tb_writer)

    # Evolve hyperparameters (optional)
    else:
        # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit, upper_limit)
        meta = {'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
                'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
                'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
                'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
                'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
                'warmup_momentum': (1, 0.0, 0.95),  # warmup initial momentum
                'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial bias lr
                'box': (1, 0.02, 0.2),  # box loss gain
                'cls': (1, 0.2, 4.0),  # cls loss gain
                'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_weight
                'obj': (1, 0.2, 4.0),  # obj loss gain (scale with pixels)
                'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
                'iou_t': (0, 0.1, 0.7),  # IoU training threshold
                'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
                'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
                'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default gamma=1.5)
                'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
                'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation (fraction)
                'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
                'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
                'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
                'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
                'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
                'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction), range 0-0.001
                'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
                'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
                'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
                'mixup': (1, 0.0, 1.0)}  # image mixup (probability)

        assert opt.local_rank == -1, 'DDP mode not implemented for --evolve'
        opt.notest, opt.nosave = True, True  # only test/save final epoch
        # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # evolvable indices
        yaml_file = Path(opt.save_dir) / 'hyp_evolved.yaml'  # save best result here
        if opt.bucket:
            os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket)  # download evolve.txt if exists

        for _ in range(300):  # generations to evolve
            if Path('evolve.txt').exists():  # if evolve.txt exists: select best hyps and mutate
                # Select parent(s)
                parent = 'single'  # parent selection method: 'single' or 'weighted'
                x = np.loadtxt('evolve.txt', ndmin=2)
                n = min(5, len(x))  # number of previous results to consider
                x = x[np.argsort(-fitness(x))][:n]  # top n mutations
                w = fitness(x) - fitness(x).min()  # weights
                if parent == 'single' or len(x) == 1:
                    # x = x[random.randint(0, n - 1)]  # random selection
                    x = x[random.choices(range(n), weights=w)[0]]  # weighted selection
                elif parent == 'weighted':
                    x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # weighted combination

                # Mutate
                mp, s = 0.8, 0.2  # mutation probability, sigma
                npr = np.random
                npr.seed(int(time.time()))
                g = np.array([x[0] for x in meta.values()])  # gains 0-1
                ng = len(meta)
                v = np.ones(ng)
                while all(v == 1):  # mutate until a change occurs (prevent duplicates)
                    v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() * s + 1).clip(0.3, 3.0)
                for i, k in enumerate(hyp.keys()):  # plt.hist(v.ravel(), 300)
                    hyp[k] = float(x[i + 7] * v[i])  # mutate

            # Constrain to limits
            for k, v in meta.items():
                hyp[k] = max(hyp[k], v[1])  # lower limit
                hyp[k] = min(hyp[k], v[2])  # upper limit
                hyp[k] = round(hyp[k], 5)  # significant digits

            # Train mutation
            results = train(hyp.copy(), opt, device)

            # Write mutation results
            print_mutation(hyp.copy(), results, yaml_file, opt.bucket)

        # Plot results
        plot_evolution(yaml_file)
        print(f'Hyperparameter evolution complete. Best results saved as: {yaml_file}\n'
              f'Command to train a new model with these hyperparameters: $ python train.py --hyp {yaml_file}')
```

```python
import tensorflow as tf
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# 加载预训练的 DeepLabv3+模型
model = tf.keras.applications.DeepLabV3(
    weights='cityscapes',  # 使用 Cityscapes 数据集预训练的权重
    input_tensor=tf.keras.layers.Input(shape=(None, None, 3))
)

# 加载和预处理输入图像
def load_and_preprocess_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, (512, 512))  # 调整图像大小
    image = tf.expand_dims(image, 0)  # 扩展维度以匹配模型输入要求
    return image

# 进行语义分割
def semantic_segmentation(image_path):
    image = load_and_preprocess_image(image_path)
    preds = model.predict(image)
    mask = np.argmax(preds.squeeze(), axis=-1)  # 取最大概率对应的类别作为预测结果
    return mask

# 示例：加载图像并进行语义分割
image_path = 'semantic_segmentation_image.jpg'
mask = semantic_segmentation(image_path)

# 可视化结果
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(Image.open(image_path))
plt.title('Input Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(mask)
plt.title('Semantic Segmentation Mask')
plt.axis('off')

plt.tight_layout()
plt.show()
```

项目效果图：







```
NFO: listening to server 127.0.0.1:2000
xample of automatic vehicle control from client side.
```

个人遇到的主要问题及解决方法：

I. **问题：** 数据集稀缺、质量不高，无法满足模型训练需求。

解决方法：

- **数据增强：** 使用数据增强技术如旋转、翻转、缩放等，扩充现有数据集。
- **迁移学习：** 使用预训练模型和迁移学习，通过在其他数据集上预训练模型来提高性能。
- **数据合成：** 使用合成数据技术，通过模拟生成数据来扩充数据集。

II. **问题：** 训练时间长，资源消耗大，性能不稳定。

解决方法：

- **优化模型结构：** 精简模型、减少参数量，提高模型效率。
- **硬件加速：** 使用 GPU 或者云端计算资源进行加速训练。
- **分布式训练：** 将训练任务分布到多个设备或者多台机器上，加快训练速度。

III. **问题：** 算法实现复杂，难以理解和调试。

解决方法：

- **学习和实践：** 深入学习算法原理，多做实验和练习。
- **代码复用：** 查阅文档、示例代码和开源项目，借鉴和复用优秀的实现。

- **团队协作：** 与同事或者论坛社区交流，寻求帮助和解答疑惑。

IV. **问题：** 技术更新快，需要持续学习但时间有限。

解决方法：

- **制定学习计划：** 根据实际情况制定合理的学习计划，分阶段、有重点地学习相关技术。
- **选择有效资源：** 选择高效的学习资源，如优质的书籍、在线课程、教程和社区论坛等。
- **持续实践：** 学习的同时保持实践，通过项目和练习巩固和应用所学知识。

个人实习体会及收获：

项目开发过程中的团队合作与经验总结

在整个项目开发过程中，我们团队成员分工明确，各自承担不同职责，形成了高效的工作机制。通过精心设计、严谨开发、全面测试和持续优化，我们持续提升项目质量，确保实现预期目标。这一过程中，我们不仅积累了丰富的专业知识，还深入理解并应用了敏捷开发的方法论，包括团队协作、计划制定、文档撰写等关键环节。这些宝贵的经验为我们今后在开发岗位上打下了坚实基础。

团队合作与创新

在项目实施过程中，我们深刻认识到了团队合作的重要性。团队精神让我们目标一致，形成了强大的凝聚力和战斗力。通过明确分工，我们将团队的整体目标细化为具体的小目标，并落实到每位成员身上。正是这种协作精神，使我们在项目中发现并实现创新，并最终达成共识。

挑战与成长

通过这次项目研发，我们更加深刻地认识到，软件开发从来不会一帆风顺。每位成员都是独立的个体，拥有自己独特的思维和判断。作为一个团队，我们需要学会包容，学会“求同存异”，通过充分讨论和协商找到最佳解决方案。这次项目的成功不仅仅是满足了用户的核心需求，同时也为我们积累了宝贵的经验，为未来的发展奠定了坚实的基础。

总结

综上所述，VisionVoyage 项目的研发对我们团队的每位成员来说，都是一次宝贵的学习和成长机会。我们不仅在专业技能上有了显著提升，同时在团队合作、创新思维和问题解决能力等方面也取得了重要进步。我们相信，这次项目成功的经验将为我们未来的发展提供强大的动力和支持。

生产实习成绩汇总

生产实习分数	**平时成绩** （20 分）		**项目成绩** （60 分）		**报告成绩** （20 分）		**总分**
	出勤情况 **(10 分)**	**课堂表现** **(10 分)**	**基本功能** **(30 分)**	**答辩情况** **(30 分)**	**报告内容** **(10 分)**	**规范程度** **(10 分)**	

总体评价：

实习导师签名：

企业导师签名：