尝试移植YOLOp

Created	@December 11, 2021 11:36 PM
开始日期	@December 12, 2021
最迟结束时间	@December 13, 2021

github代码：

> https://github.com/hustvl/YOLOP

1、先用conda创建虚拟环境，将他的代码clone下来跑一下，看一下。

Conda 创建虚拟环境

YOLO移植寒武纪测试

1、安装git，git clone代码

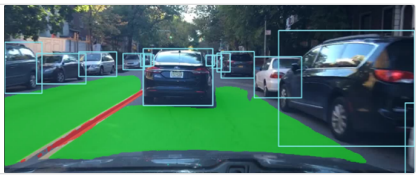
```
apt install git
git clone
```

2、下载数据集，可以在GitHub上将处理好的数据集进行下载，也可以将使用部分数据集，我这里仅仅使用了部分数据集（来自CSDN博客上）

YOLOP 训练+测试+模型评估_Dora_blank的博客-CSDN博客

本文将详细介绍YOLOP 训练+测试+模型评估的过程和个人遇到的所有报错。 必要环境：Windows10+python3.7+CUDA10.1+CUDNN7.6.5 运行 python tools/demo.py --source ./inference/videos/1.mp4 测试图片：-source 图片路径 (或存放图片的文件夹路径) 测试视频：-source 视频路径 (或存放视频的文件夹路径)

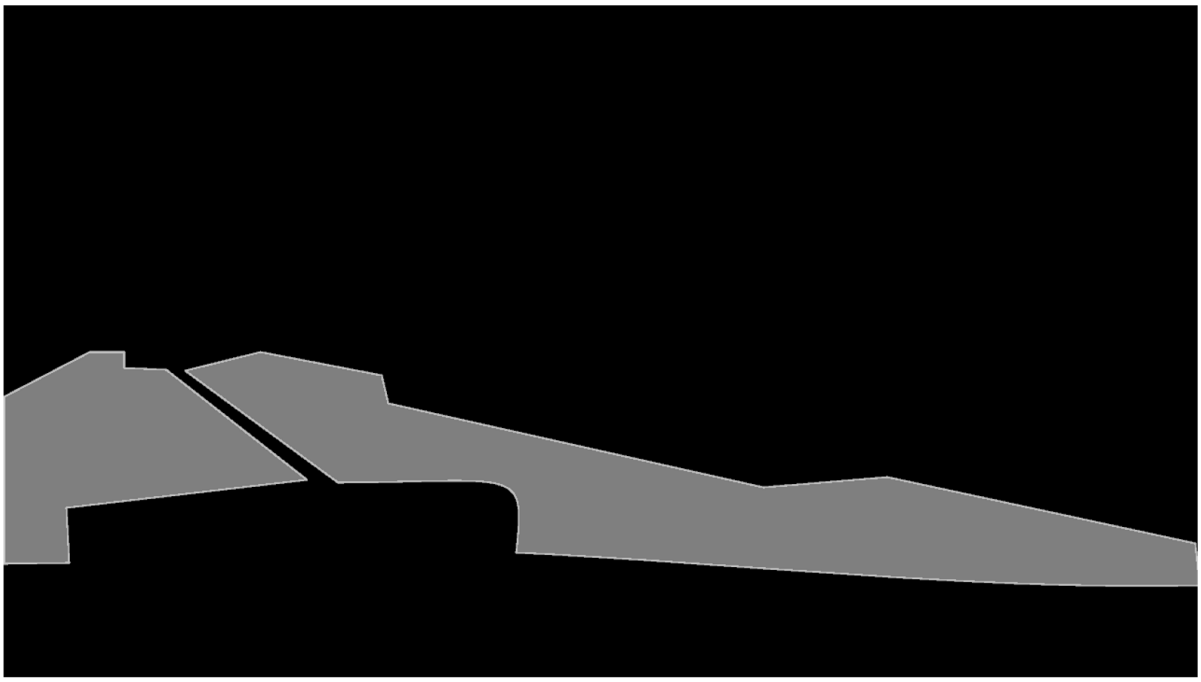
https://blog.csdn.net/Dora_blank/article/details/120070490



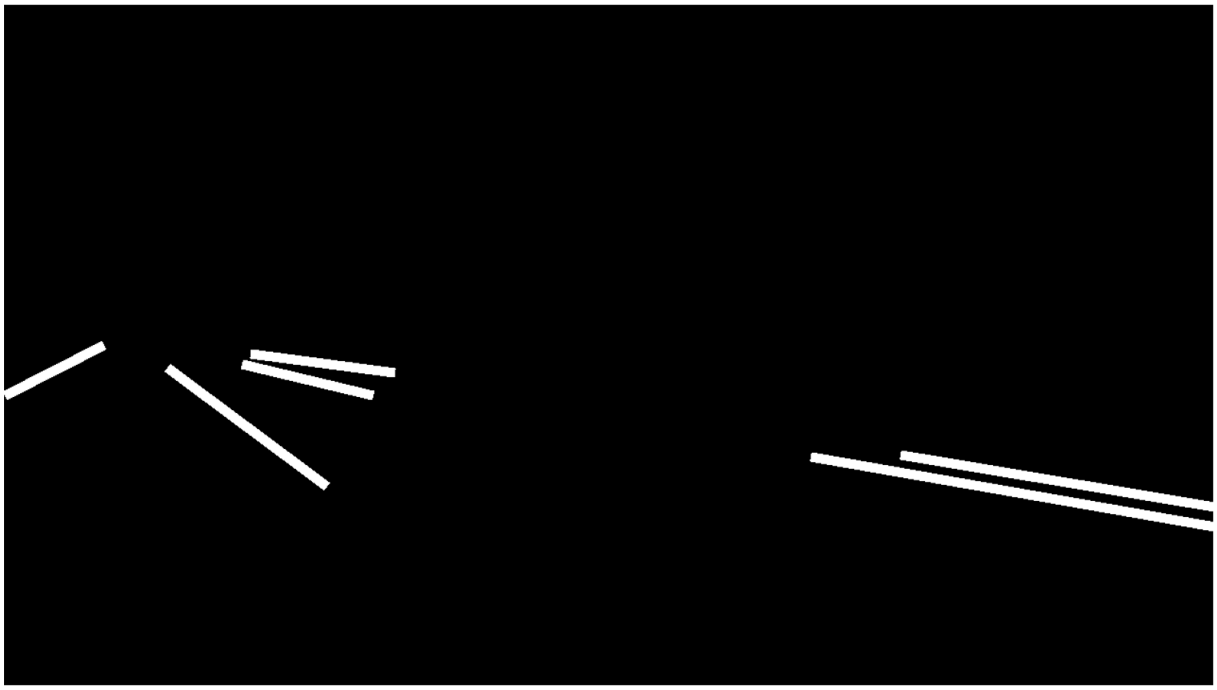
⚠️ 对于数据集来看，如下面的介绍

```
├─dataset root
| ├─images
| | ├─train 存放原图片
| | ├─val
| ├─det_annotations
| | ├─train 存放json文件，即一些框的数据
| | ├─val
| ├─da_seg_annotations
| | ├─train 存放可行驶区域的照片
| | ├─val
| ├─ll_seg_annotations
| | ├─train 存放车道线的照片
| | ├─val
```

▼ 可行驶区域照片



▼ 车道线训练照片



▼ test.py代码 Yolov5模型推理调试

```
import argparse
import os, sys
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.append(BASE_DIR)
```

```
import pprint
import torch
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import numpy as np
from lib.utils import DataLoaderX
from tensorboardX import SummaryWriter

import lib.dataset as dataset
from lib.config import cfg
from lib.config import update_config
from lib.core.loss import get_loss
from lib.core.function import validate
from lib.core.general import fitness
from lib.models import get_net
from lib.utils.utils import create_logger, select_device

def parse_args():
    parser = argparse.ArgumentParser(description='Test Multitask network')

    # philly
    parser.add_argument('--modelDir',
                        help='model directory',
                        type=str,
                        default='')
    parser.add_argument('--logDir',
                        help='log directory',
                        type=str,
                        default='runs/')
    parser.add_argument('--weights', nargs='+', type=str, default='/data2/zwt/wd/YOLOP/runs/BddDataset/detect_and_segbranch_whole/epoch
    parser.add_argument('--conf_thres', type=float, default=0.001, help='object confidence threshold')
    parser.add_argument('--iou_thres', type=float, default=0.6, help='IOU threshold for NMS')
    args = parser.parse_args()

    return args

def main():
    # set all the configurations
    args = parse_args()
    update_config(cfg, args)

    # TODO: handle distributed training logger
    # set the logger, tb_log_dir means tensorboard logdir
    # 模型日志，输出日志位置
    logger, final_output_dir, tb_log_dir = create_logger(
        cfg, cfg.LOG_DIR, 'test')

    logger.info(pprint.pformat(args))
    logger.info(cfg)

    writer_dict = {
        'writer': SummaryWriter(log_dir=tb_log_dir),
        'train_global_steps': 0,
        'valid_global_steps': 0,
    }

    # bulid up model
    # start_time = time.time()
    print("begin to bulid up model...")
    # DP mode
    device = select_device(logger, batch_size=cfg.TEST.BATCH_SIZE_PER_GPU* len(cfg.GPUS)) if not cfg.DEBUG \
        else select_device(logger, 'cpu')
    # device = select_device(logger, 'cpu')
    # 加载模型
    model = get_net(cfg)
    print("finish build model")

    # define loss function (criterion) and optimizer
    #尝试下传入device为ct.mlu_device()或者直接不修改，让它在CPU上算
    criterion = get_loss(cfg, device=device)

    # 可以将模型量化加在这里
if opt.quantization:
    #加载模型原权重
    state_dict = torch.load("../models/yolov5x.pt",map_location='cpu')['model'].state_dict()
    model.load_state_dict(state_dict)
    model.eval()
```

```
    # 在这里设置firstconv参数为False,因为该模型首层为focus算子，非卷积，无法开启first_conv
    qconfig = {'use_avg':False, 'data_scale':1.0, 'firstconv':False, 'per_channel':False}
    # 进行模型量化，里面包含量化后权重和比例因子
    quantized_model = mlu_quantize.quantize_dynamic_mlu(model, qconfig, dtype="int8",gen_quant=True)
    #保存量化权重
    torch.save(quantized_model.state_dict(), "yolov5x_int8.pth")

    # load checkpoint model
    # det_idx_range = [str(i) for i in range(0,25)]
    model_dict = model.state_dict()
    checkpoint_file = args.weights
    logger.info("=> loading checkpoint '{}'".format(checkpoint_file))
    checkpoint = torch.load(checkpoint_file)
    checkpoint_dict = checkpoint['state_dict']
    # checkpoint_dict = {k: v for k, v in checkpoint['state_dict'].items() if k.split(".")[1] in det_idx_range}
    model_dict.update(checkpoint_dict)
    model.load_state_dict(model_dict)
    logger.info("=> loaded checkpoint '{}' ".format(checkpoint_file))

    model = model.to(device)
    model.gr = 1.0
    model.nc = 1
    print('bulid model finished')

    print("begin to load data")
    # Data loading
    normalize = transforms.Normalize(
        mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
    )

    valid_dataset = eval('dataset.' + cfg.DATASET.DATASET)(
        cfg=cfg,
        is_train=False,
        inputsize=cfg.MODEL.IMAGE_SIZE,
        transform=transforms.Compose([
            transforms.ToTensor(),
            normalize,
        ])
    )

    # valid_loader = DataLoaderX(
    #     valid_dataset,
    #     batch_size=cfg.TEST.BATCH_SIZE_PER_GPU * len(cfg.GPUS),
    #     shuffle=False,
    #     num_workers=cfg.WORKERS,
    #     pin_memory=cfg.PIN_MEMORY,
    #     collate_fn=dataset.AutoDriveDataset.collate_fn
    # )
    valid_loader = DataLoaderX(
        valid_dataset,
        batch_size=cfg.TEST.BATCH_SIZE_PER_GPU * len(cfg.GPUS),
        shuffle=False,
        num_workers=cfg.WORKERS,
        pin_memory=False,
        collate_fn=dataset.AutoDriveDataset.collate_fn
    )
    print('load data finished')

    epoch = 0 #special for test
    da_segment_results,ll_segment_results,detect_results, total_loss,maps, times = validate(
        epoch,cfg, valid_loader, valid_dataset, model, criterion,
        final_output_dir, tb_log_dir, writer_dict,
        logger, device
    )
    fi = fitness(np.array(detect_results).reshape(1, -1))
    msg =   'Test:    Loss({loss:.3f})\n' \
            'Driving area Segment: Acc({da_seg_acc:.3f})    IOU ({da_seg_iou:.3f})    mIOU({da_seg_miou:.3f})\n' \
                      'Lane line Segment: Acc({ll_seg_acc:.3f})    IOU ({ll_seg_iou:.3f})  mIOU({ll_seg_miou:.3f})\n' \
                      'Detect: P({p:.3f})  R({r:.3f})  mAP@0.5({map50:.3f})  mAP@0.5:0.95({map:.3f})\n'\
                      'Time: inference({t_inf:.4f}s/frame)  nms({t_nms:.4f}s/frame)'.format(
                          loss=total_loss, da_seg_acc=da_segment_results[0],da_seg_iou=da_segment_results[1],da_seg_miou=da_segment_res
                          ll_seg_acc=ll_segment_results[0],ll_seg_iou=ll_segment_results[1],ll_seg_miou=ll_segment_results[2],
                          p=detect_results[0],r=detect_results[1],map50=detect_results[2],map=detect_results[3],
                          t_inf=times[0], t_nms=times[1])
    logger.info(msg)
    print("test finish")


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

环境修改

问题一：不存在**prefetch_generator，将prefetch_generator安装到对应的目录下用pip**

问题二：

原因.只有PyTorch1.6版本以上才可以从torch.cuda中import amp；

前往环境包存储的目录下

```
git clone https://github.com/NVIDIA/apex
cd apex
pip3 install -v --no-cache-dir ./
```

修改/workspace/volume/private/YOLOP/lib/core/function.py中的

from torch.cuda import amp

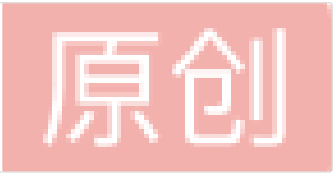
变为：

from apex import amp

在pytorch中使用from apex import amp报错。pytorch安装了cuda但是没有安装nvcc_Eraaaa的博客-CSDN博客

报错 raise RuntimeError("--cuda_ext was requested, but nvcc was not found. Are you sure your environment has nvcc available?

https://blog.csdn.net/weixin_38215769/article/details/106568368?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.nonecase

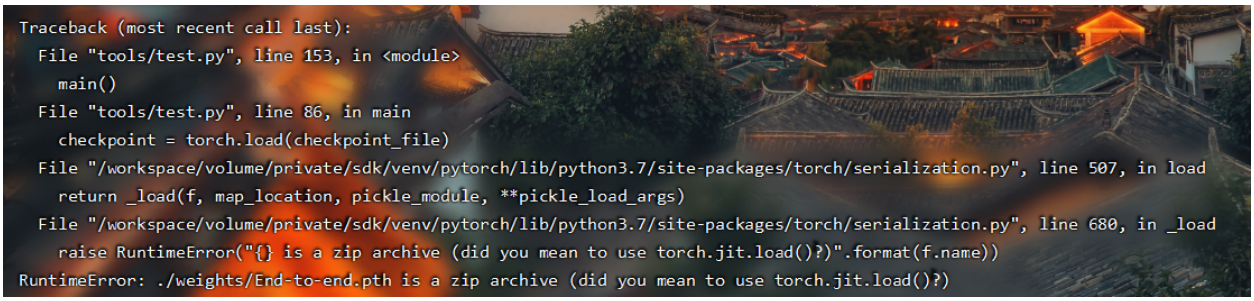


改不太好，直接暴力注释掉，/workspace/volume/private/YOLOP/lib/core/function.py用到amp的注释掉。

```
#         if not cfg.DEBUG:
#             input = input.to(device, non_blocking=True)
#             assign_target = []
#             for tgt in target:
#                 assign_target.append(tgt.to(device))
#             target = assign_target
#         with amp.autocast(enabled=device.type != 'cpu'):
#             outputs = model(input)
#             total_loss, head_losses = criterion(outputs, target, shapes,model)

            # print(head_losses)
```

问题三，读取权重出现问题：



修改加载模型权重部分

改成CPU的也不太行

下面代码去掉了 parser.add_argument('--weights', type=str, default='/data2/zwt/wd/YOLOP/runs/BddDataset/detect_and_segbranch_whole/epoch-169.pth', help='model.pth path(s)')中的一个+号参数

▼ test.py

```
import argparse
import os, sys
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.append(BASE_DIR)

import pprint
import torch
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import numpy as np
from lib.utils import DataLoaderX
from tensorboardX import SummaryWriter

import lib.dataset as dataset
from lib.config import cfg
from lib.config import update_config
from lib.core.loss import get_loss
from lib.core.function import validate
from lib.core.general import fitness
from lib.models import get_net
from lib.utils.utils import create_logger, select_device

def parse_args():
    parser = argparse.ArgumentParser(description='Test Multitask network')

    # philly
    parser.add_argument('--modelDir',
                        help='model directory',
                        type=str,
                        default='')
    parser.add_argument('--logDir',
                        help='log directory',
                        type=str,
                        default='runs/')
    parser.add_argument('--weights', type=str, default='/data2/zwt/wd/YOLOP/runs/BddDataset/detect_and_segbranch_whole/epoch-169.pth',
    parser.add_argument('--conf_thres', type=float, default=0.001, help='object confidence threshold')
    parser.add_argument('--iou_thres', type=float, default=0.6, help='IOU threshold for NMS')
    args = parser.parse_args()

    return args

def main():
    # set all the configurations
    args = parse_args()
    update_config(cfg, args)

    # TODO: handle distributed training logger
    # set the logger, tb_log_dir means tensorboard logdir
```

```
    logger, final_output_dir, tb_log_dir = create_logger(
        cfg, cfg.LOG_DIR, 'test')

    logger.info(pprint.pformat(args))
    logger.info(cfg)

    writer_dict = {
        'writer': SummaryWriter(log_dir=tb_log_dir),
        'train_global_steps': 0,
        'valid_global_steps': 0,
    }

    # bulid up model
    # start_time = time.time()
    print("begin to bulid up model...")
    # DP mode
    device = select_device(logger, batch_size=cfg.TEST.BATCH_SIZE_PER_GPU* len(cfg.GPUS)) if not cfg.DEBUG \
        else select_device(logger, 'cpu')
    # device = select_device(logger, 'cpu')

    model = get_net(cfg)
    print("finish build model")

    # define loss function (criterion) and optimizer
    criterion = get_loss(cfg, device=device)

    # load checkpoint model

    # det_idx_range = [str(i) for i in range(0,25)]
    model_dict = model.state_dict()
    checkpoint_file = args.weights
    logger.info("=> loading checkpoint '{}'".format(checkpoint_file))
    checkpoint = torch.load(checkpoint_file)
    checkpoint_dict = checkpoint['state_dict']
    # checkpoint_dict = {k: v for k, v in checkpoint['state_dict'].items() if k.split(".")[1] in det_idx_range}
    model_dict.update(checkpoint_dict)
    model.load_state_dict(model_dict)
    logger.info("=> loaded checkpoint '{}' ".format(checkpoint_file))

    model = model.to(device)
    model.gr = 1.0
    model.nc = 1
    print('bulid model finished')

    print("begin to load data")
    # Data loading
    normalize = transforms.Normalize(
        mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
    )

    valid_dataset = eval('dataset.' + cfg.DATASET.DATASET)(
        cfg=cfg,
        is_train=False,
        inputsize=cfg.MODEL.IMAGE_SIZE,
        transform=transforms.Compose([
            transforms.ToTensor(),
            normalize,
        ])
    )

    # valid_loader = DataLoaderX(
    #     valid_dataset,
    #     batch_size=cfg.TEST.BATCH_SIZE_PER_GPU * len(cfg.GPUS),
    #     shuffle=False,
    #     num_workers=cfg.WORKERS,
    #     pin_memory=cfg.PIN_MEMORY,
    #     collate_fn=dataset.AutoDriveDataset.collate_fn
    # )
    valid_loader = DataLoaderX(
        valid_dataset,
        batch_size=cfg.TEST.BATCH_SIZE_PER_GPU * len(cfg.GPUS),
        shuffle=False,
        num_workers=cfg.WORKERS,
        pin_memory=False,
        collate_fn=dataset.AutoDriveDataset.collate_fn
    )
    print('load data finished')

    epoch = 0 #special for test
    da_segment_results,ll_segment_results,detect_results, total_loss,maps, times = validate(
        epoch,cfg, valid_loader, valid_dataset, model, criterion,
```

```
        final_output_dir, tb_log_dir, writer_dict,
        logger, device
    )
    fi = fitness(np.array(detect_results).reshape(1, -1))
    msg =   'Test:    Loss({loss:.3f})\n' \
            'Driving area Segment: Acc({da_seg_acc:.3f})    IOU ({da_seg_iou:.3f})    mIOU({da_seg_miou:.3f})\n' \
                      'Lane line Segment: Acc({ll_seg_acc:.3f})    IOU ({ll_seg_iou:.3f})  mIOU({ll_seg_miou:.3f})\n' \
                      'Detect: P({p:.3f})  R({r:.3f})  mAP@0.5({map50:.3f})  mAP@0.5:0.95({map:.3f})\n'\
                      'Time: inference({t_inf:.4f}s/frame)  nms({t_nms:.4f}s/frame)'.format(
                          loss=total_loss, da_seg_acc=da_segment_results[0],da_seg_iou=da_segment_results[1],da_seg_miou=da_segment_res
                          ll_seg_acc=ll_segment_results[0],ll_seg_iou=ll_segment_results[1],ll_seg_miou=ll_segment_results[2],
                          p=detect_results[0],r=detect_results[1],map50=detect_results[2],map=detect_results[3],
                          t_inf=times[0], t_nms=times[1])
    logger.info(msg)
    print("test finish")


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

MLU逐层模式跑模型时候,在运行模型的时候出现的错误：
简单推断是由于Op没有

YOLOP（YOLOP.py文件中）尝试修改后处理算子：

▼ yolop.py

```
import torch
from torch import tensor
import torch.nn as nn
import sys,os
import math
import sys


sys.path.append(os.getcwd())
#sys.path.append("lib/models")
#sys.path.append("lib/utils")
#sys.path.append("/workspace/wh/projects/DaChuang")
from lib.utils import initialize_weights
# from lib.models.common2 import DepthSeperabelConv2d as Conv
# from lib.models.common2 import SPP, Bottleneck, BottleneckCSP, Focus, Concat, Detect
from lib.models.common import Conv, SPP, Bottleneck, BottleneckCSP, Focus, Concat, Detect, SharpenConv
from torch.nn import Upsample
from lib.utils import check_anchor_order
from lib.core.evaluate import SegmentationMetric
from lib.utils.utils import time_synchronized

"""
MCnet_SPP = [
[ -1, Focus, [3, 32, 3]],
[ -1, Conv, [32, 64, 3, 2]],
[ -1, BottleneckCSP, [64, 64, 1]],
[ -1, Conv, [64, 128, 3, 2]],
```

```
[ -1, BottleneckCSP, [128, 128, 3]],
[ -1, Conv, [128, 256, 3, 2]],
[ -1, BottleneckCSP, [256, 256, 3]],
[ -1, Conv, [256, 512, 3, 2]],
[ -1, SPP, [512, 512, [5, 9, 13]]],
[ -1, BottleneckCSP, [512, 512, 1, False]],
[ -1, Conv,[512, 256, 1, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1, 6], Concat, [1]],
[ -1, BottleneckCSP, [512, 256, 1, False]],
[ -1, Conv, [256, 128, 1, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1,4], Concat, [1]],
[ -1, BottleneckCSP, [256, 128, 1, False]],
[ -1, Conv, [128, 128, 3, 2]],
[ [-1, 14], Concat, [1]],
[ -1, BottleneckCSP, [256, 256, 1, False]],
[ -1, Conv, [256, 256, 3, 2]],
[ [-1, 10], Concat, [1]],
[ -1, BottleneckCSP, [512, 512, 1, False]],
# [ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]],
[ [17, 20, 23], Detect,  [13, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]],
[ 17, Conv, [128, 64, 3, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1,2], Concat, [1]],
[ -1, BottleneckCSP, [128, 64, 1, False]],
[ -1, Conv, [64, 32, 3, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ -1, Conv, [32, 16, 3, 1]],
[ -1, BottleneckCSP, [16, 8, 1, False]],
[ -1, Upsample, [None, 2, 'nearest']],
[ -1, SPP, [8, 2, [5, 9, 13]]] #segmentation output
]
# [2,6,3,9,5,13], [7,19,11,26,17,39], [28,64,44,103,61,183]

MCnet_0 = [
[ -1, Focus, [3, 32, 3]],
[ -1, Conv, [32, 64, 3, 2]],
[ -1, BottleneckCSP, [64, 64, 1]],
[ -1, Conv, [64, 128, 3, 2]],
[ -1, BottleneckCSP, [128, 128, 3]],
[ -1, Conv, [128, 256, 3, 2]],
[ -1, BottleneckCSP, [256, 256, 3]],
[ -1, Conv, [256, 512, 3, 2]],
[ -1, SPP, [512, 512, [5, 9, 13]]],
[ -1, BottleneckCSP, [512, 512, 1, False]],
[ -1, Conv,[512, 256, 1, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1, 6], Concat, [1]],
[ -1, BottleneckCSP, [512, 256, 1, False]],
[ -1, Conv, [256, 128, 1, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1,4], Concat, [1]],
[ -1, BottleneckCSP, [256, 128, 1, False]],
[ -1, Conv, [128, 128, 3, 2]],
[ [-1, 14], Concat, [1]],
[ -1, BottleneckCSP, [256, 256, 1, False]],
[ -1, Conv, [256, 256, 3, 2]],
[ [-1, 10], Concat, [1]],
[ -1, BottleneckCSP, [512, 512, 1, False]],
[ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]], #Detect output 24

[ 16, Conv, [128, 64, 3, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1,2], Concat, [1]],
[ -1, BottleneckCSP, [128, 64, 1, False]],
[ -1, Conv, [64, 32, 3, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ -1, Conv, [32, 16, 3, 1]],
[ -1, BottleneckCSP, [16, 8, 1, False]],
[ -1, Upsample, [None, 2, 'nearest']],
[ -1, Conv, [8, 2, 3, 1]], #Driving area segmentation output

[ 16, Conv, [128, 64, 3, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ [-1,2], Concat, [1]],
[ -1, BottleneckCSP, [128, 64, 1, False]],
[ -1, Conv, [64, 32, 3, 1]],
[ -1, Upsample, [None, 2, 'nearest']],
[ -1, Conv, [32, 16, 3, 1]],
[ -1, BottleneckCSP, [16, 8, 1, False]],
```

```
[ -1, Upsample, [None, 2, 'nearest']],
[ -1, Conv, [8, 2, 3, 1]], #Lane line segmentation output
]


# The lane line and the driving area segment branches share information with each other
MCnet_share = [
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16
[ -1, BottleneckCSP, [256, 128, 1, False]],     #17
[ -1, Conv, [128, 128, 3, 2]],      #18
[ [-1, 14], Concat, [1]],       #19
[ -1, BottleneckCSP, [256, 256, 1, False]],     #20
[ -1, Conv, [256, 256, 3, 2]],      #21
[ [-1, 10], Concat, [1]],   #22
[ -1, BottleneckCSP, [512, 512, 1, False]],     #23
[ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]], #Detect output 24

[ 16, Conv, [256, 64, 3, 1]],   #25
[ -1, Upsample, [None, 2, 'nearest']],  #26
[ [-1,2], Concat, [1]],  #27
[ -1, BottleneckCSP, [128, 64, 1, False]],  #28
[ -1, Conv, [64, 32, 3, 1]],    #29
[ -1, Upsample, [None, 2, 'nearest']],  #30
[ -1, Conv, [32, 16, 3, 1]],    #31
[ -1, BottleneckCSP, [16, 8, 1, False]],    #32 driving area segment neck

[ 16, Conv, [256, 64, 3, 1]],   #33
[ -1, Upsample, [None, 2, 'nearest']],  #34
[ [-1,2], Concat, [1]], #35
[ -1, BottleneckCSP, [128, 64, 1, False]],  #36
[ -1, Conv, [64, 32, 3, 1]],    #37
[ -1, Upsample, [None, 2, 'nearest']],  #38
[ -1, Conv, [32, 16, 3, 1]],    #39
[ -1, BottleneckCSP, [16, 8, 1, False]],    #40 lane line segment neck

[ [31,39], Concat, [1]],    #41
[ -1, Conv, [32, 8, 3, 1]],     #42    Share_Block


[ [32,42], Concat, [1]],     #43
[ -1, Upsample, [None, 2, 'nearest']],  #44
[ -1, Conv, [16, 2, 3, 1]], #45 Driving area segmentation output


[ [40,42], Concat, [1]],    #46
[ -1, Upsample, [None, 2, 'nearest']],  #47
[ -1, Conv, [16, 2, 3, 1]] #48Lane line segmentation output
]

# The lane line and the driving area segment branches without share information with each other
MCnet_no_share = [
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
```

```
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16
[ -1, BottleneckCSP, [256, 128, 1, False]],     #17
[ -1, Conv, [128, 128, 3, 2]],      #18
[ [-1, 14], Concat, [1]],       #19
[ -1, BottleneckCSP, [256, 256, 1, False]],     #20
[ -1, Conv, [256, 256, 3, 2]],      #21
[ [-1, 10], Concat, [1]],   #22
[ -1, BottleneckCSP, [512, 512, 1, False]],     #23
[ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]], #Detect output 24

[ 16, Conv, [256, 64, 3, 1]],   #25
[ -1, Upsample, [None, 2, 'nearest']],  #26
[ [-1,2], Concat, [1]],  #27
[ -1, BottleneckCSP, [128, 64, 1, False]],  #28
[ -1, Conv, [64, 32, 3, 1]],    #29
[ -1, Upsample, [None, 2, 'nearest']],  #30
[ -1, Conv, [32, 16, 3, 1]],    #31
[ -1, BottleneckCSP, [16, 8, 1, False]],    #32 driving area segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #33
[ -1, Conv, [8, 3, 3, 1]], #34 Driving area segmentation output

[ 16, Conv, [256, 64, 3, 1]],   #35
[ -1, Upsample, [None, 2, 'nearest']],  #36
[ [-1,2], Concat, [1]], #37
[ -1, BottleneckCSP, [128, 64, 1, False]],  #38
[ -1, Conv, [64, 32, 3, 1]],    #39
[ -1, Upsample, [None, 2, 'nearest']],  #40
[ -1, Conv, [32, 16, 3, 1]],    #41
[ -1, BottleneckCSP, [16, 8, 1, False]],    #42 lane line segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #43
[ -1, Conv, [8, 2, 3, 1]] #44 Lane line segmentation output
]

MCnet_feedback = [
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16
[ -1, BottleneckCSP, [256, 128, 1, False]],     #17
[ -1, Conv, [128, 128, 3, 2]],      #18
[ [-1, 14], Concat, [1]],       #19
[ -1, BottleneckCSP, [256, 256, 1, False]],     #20
[ -1, Conv, [256, 256, 3, 2]],      #21
[ [-1, 10], Concat, [1]],   #22
[ -1, BottleneckCSP, [512, 512, 1, False]],     #23
[ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]], #Detect output 24

[ 16, Conv, [256, 128, 3, 1]],   #25
[ -1, Upsample, [None, 2, 'nearest']],  #26
[ -1, BottleneckCSP, [128, 64, 1, False]],  #28
[ -1, Conv, [64, 32, 3, 1]],    #29
[ -1, Upsample, [None, 2, 'nearest']],  #30
[ -1, Conv, [32, 16, 3, 1]],    #31
[ -1, BottleneckCSP, [16, 8, 1, False]],    #32 driving area segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #33
[ -1, Conv, [8, 2, 3, 1]], #34 Driving area segmentation output

[ 16, Conv, [256, 128, 3, 1]],   #35
[ -1, Upsample, [None, 2, 'nearest']],  #36
[ -1, BottleneckCSP, [128, 64, 1, False]],  #38
[ -1, Conv, [64, 32, 3, 1]],    #39
[ -1, Upsample, [None, 2, 'nearest']],  #40
[ -1, Conv, [32, 16, 3, 1]],    #41
[ -1, BottleneckCSP, [16, 8, 1, False]],    #42 lane line segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #43
[ -1, Conv, [8, 2, 3, 1]] #44 Lane line segmentation output
```

```
]


MCnet_Da_feedback1 = [
[46, 26, 35],   #Det_out_idx, Da_Segout_idx, LL_Segout_idx
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16     backbone+fpn
[ -1,Conv,[256,256,1,1]],   #17


[ 16, Conv, [256, 128, 3, 1]],   #18
[ -1, Upsample, [None, 2, 'nearest']],  #19
[ -1, BottleneckCSP, [128, 64, 1, False]],  #20
[ -1, Conv, [64, 32, 3, 1]],    #21
[ -1, Upsample, [None, 2, 'nearest']],  #22
[ -1, Conv, [32, 16, 3, 1]],    #23
[ -1, BottleneckCSP, [16, 8, 1, False]],    #24 driving area segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #25
[ -1, Conv, [8, 2, 3, 1]], #26 Driving area segmentation output


[ 16, Conv, [256, 128, 3, 1]],   #27
[ -1, Upsample, [None, 2, 'nearest']],  #28
[ -1, BottleneckCSP, [128, 64, 1, False]],  #29
[ -1, Conv, [64, 32, 3, 1]],    #30
[ -1, Upsample, [None, 2, 'nearest']],  #31
[ -1, Conv, [32, 16, 3, 1]],    #32
[ -1, BottleneckCSP, [16, 8, 1, False]],    #33 lane line segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #34
[ -1, Conv, [8, 2, 3, 1]], #35Lane line segmentation output


[ 23, Conv, [16, 16, 3, 2]],     #36
[ -1, Conv, [16, 32, 3, 2]],    #2 times 2xdownsample    37

[ [-1,17], Concat, [1]],       #38
[ -1, BottleneckCSP, [288, 128, 1, False]],    #39
[ -1, Conv, [128, 128, 3, 2]],      #40
[ [-1, 14], Concat, [1]],       #41
[ -1, BottleneckCSP, [256, 256, 1, False]],     #42
[ -1, Conv, [256, 256, 3, 2]],      #43
[ [-1, 10], Concat, [1]],   #44
[ -1, BottleneckCSP, [512, 512, 1, False]],     #45
[ [39, 42, 45], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]] #Detect output 46
]



# The lane line and the driving area segment branches share information with each other and feedback to det_head
MCnet_Da_feedback2 = [
[47, 26, 35],   #Det_out_idx, Da_Segout_idx, LL_Segout_idx
[25, 28, 31, 33],   #layer in Da_branch to do SAD
[34, 37, 40, 42],   #layer in LL_branch to do SAD
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
```

```
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16     backbone+fpn
[ -1,Conv,[256,256,1,1]],   #17


[ 16, Conv, [256, 128, 3, 1]],   #18
[ -1, Upsample, [None, 2, 'nearest']],  #19
[ -1, BottleneckCSP, [128, 64, 1, False]],  #20
[ -1, Conv, [64, 32, 3, 1]],    #21
[ -1, Upsample, [None, 2, 'nearest']],  #22
[ -1, Conv, [32, 16, 3, 1]],    #23
[ -1, BottleneckCSP, [16, 8, 1, False]],    #24 driving area segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #25
[ -1, Conv, [8, 2, 3, 1]], #26 Driving area segmentation output


[ 16, Conv, [256, 128, 3, 1]],   #27
[ -1, Upsample, [None, 2, 'nearest']],  #28
[ -1, BottleneckCSP, [128, 64, 1, False]],  #29
[ -1, Conv, [64, 32, 3, 1]],    #30
[ -1, Upsample, [None, 2, 'nearest']],  #31
[ -1, Conv, [32, 16, 3, 1]],    #32
[ -1, BottleneckCSP, [16, 8, 1, False]],    #33 lane line segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #34
[ -1, Conv, [8, 2, 3, 1]], #35Lane line segmentation output


[ 23, Conv, [16, 64, 3, 2]],     #36
[ -1, Conv, [64, 256, 3, 2]],    #2 times 2xdownsample    37

[ [-1,17], Concat, [1]],       #38

[-1, Conv, [512, 256, 3, 1]],     #39
[ -1, BottleneckCSP, [256, 128, 1, False]],    #40
[ -1, Conv, [128, 128, 3, 2]],      #41
[ [-1, 14], Concat, [1]],       #42
[ -1, BottleneckCSP, [256, 256, 1, False]],     #43
[ -1, Conv, [256, 256, 3, 2]],      #44
[ [-1, 10], Concat, [1]],   #45
[ -1, BottleneckCSP, [512, 512, 1, False]],     #46
[ [40, 42, 45], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]] #Detect output 47
]

MCnet_share1 = [
[24, 33, 45],   #Det_out_idx, Da_Segout_idx, LL_Segout_idx
[25, 28, 31, 33],   #layer in Da_branch to do SAD
[34, 37, 40, 42],   #layer in LL_branch to do SAD
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16
[ -1, BottleneckCSP, [256, 128, 1, False]],     #17
[ -1, Conv, [128, 128, 3, 2]],      #18
[ [-1, 14], Concat, [1]],       #19
[ -1, BottleneckCSP, [256, 256, 1, False]],     #20
[ -1, Conv, [256, 256, 3, 2]],      #21
[ [-1, 10], Concat, [1]],   #22
[ -1, BottleneckCSP, [512, 512, 1, False]],     #23
[ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]], #Detect output 24

[ 16, Conv, [256, 128, 3, 1]],   #25
[ -1, Upsample, [None, 2, 'nearest']],  #26
[ -1, BottleneckCSP, [128, 64, 1, False]],  #27
[ -1, Conv, [64, 32, 3, 1]],    #28
[ -1, Upsample, [None, 2, 'nearest']],  #29
[ -1, Conv, [32, 16, 3, 1]],    #30
```

```
[ -1, BottleneckCSP, [16, 8, 1, False]],    #31 driving area segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #32
[ -1, Conv, [8, 2, 3, 1]], #33 Driving area segmentation output

[ 16, Conv, [256, 128, 3, 1]],   #34
[ -1, Upsample, [None, 2, 'nearest']],  #35
[ -1, BottleneckCSP, [128, 64, 1, False]],  #36
[ -1, Conv, [64, 32, 3, 1]],    #37
[ -1, Upsample, [None, 2, 'nearest']],  #38
[ -1, Conv, [32, 16, 3, 1]],    #39

[ 30, SharpenConv, [16,16, 3, 1]], #40
[ -1, Conv, [16, 16, 3, 1]], #41
[ [-1, 39], Concat, [1]],   #42
[ -1, BottleneckCSP, [32, 8, 1, False]],    #43 lane line segment neck
[ -1, Upsample, [None, 2, 'nearest']],  #44
[ -1, Conv, [8, 2, 3, 1]] #45 Lane line segmentation output
]"""


# The lane line and the driving area segment branches without share information with each other and without link
YOLOP = [
[24, 33, 42],   #Det_out_idx, Da_Segout_idx, LL_Segout_idx
[ -1, Focus, [3, 32, 3]],   #0
[ -1, Conv, [32, 64, 3, 2]],    #1
[ -1, BottleneckCSP, [64, 64, 1]],  #2
[ -1, Conv, [64, 128, 3, 2]],   #3
[ -1, BottleneckCSP, [128, 128, 3]],    #4
[ -1, Conv, [128, 256, 3, 2]],  #5
[ -1, BottleneckCSP, [256, 256, 3]],    #6
[ -1, Conv, [256, 512, 3, 2]],  #7
[ -1, SPP, [512, 512, [5, 9, 13]]],     #8
[ -1, BottleneckCSP, [512, 512, 1, False]],     #9
[ -1, Conv,[512, 256, 1, 1]],   #10
[ -1, Upsample, [None, 2, 'nearest']],  #11
[ [-1, 6], Concat, [1]],    #12
[ -1, BottleneckCSP, [512, 256, 1, False]], #13
[ -1, Conv, [256, 128, 1, 1]],  #14
[ -1, Upsample, [None, 2, 'nearest']],  #15
[ [-1,4], Concat, [1]],     #16         #Encoder

[ -1, BottleneckCSP, [256, 128, 1, False]],     #17
[ -1, Conv, [128, 128, 3, 2]],      #18
[ [-1, 14], Concat, [1]],       #19
[ -1, BottleneckCSP, [256, 256, 1, False]],     #20
[ -1, Conv, [256, 256, 3, 2]],      #21
[ [-1, 10], Concat, [1]],   #22
[ -1, BottleneckCSP, [512, 512, 1, False]],     #23
[ [17, 20, 23], Detect,  [1, [[3,9,5,11,4,20], [7,18,6,39,12,31], [19,50,38,81,68,157]], [128, 256, 512]]], #Detection head 24

[ 16, Conv, [256, 128, 3, 1]],   #25
[ -1, Upsample, [None, 2, 'nearest']],  #26
[ -1, BottleneckCSP, [128, 64, 1, False]],  #27
[ -1, Conv, [64, 32, 3, 1]],    #28
[ -1, Upsample, [None, 2, 'nearest']],  #29
[ -1, Conv, [32, 16, 3, 1]],    #30
[ -1, BottleneckCSP, [16, 8, 1, False]],    #31
[ -1, Upsample, [None, 2, 'nearest']],  #32
[ -1, Conv, [8, 2, 3, 1]], #33 Driving area segmentation head

[ 16, Conv, [256, 128, 3, 1]],   #34
[ -1, Upsample, [None, 2, 'nearest']],  #35
[ -1, BottleneckCSP, [128, 64, 1, False]],  #36
[ -1, Conv, [64, 32, 3, 1]],    #37
[ -1, Upsample, [None, 2, 'nearest']],  #38
[ -1, Conv, [32, 16, 3, 1]],    #39
[ -1, BottleneckCSP, [16, 8, 1, False]],    #40
[ -1, Upsample, [None, 2, 'nearest']],  #41
[ -1, Conv, [8, 2, 3, 1]] #42 Lane line segmentation head
]


class MCnet(nn.Module):
    def __init__(self, block_cfg, **kwargs):
        super(MCnet, self).__init__()
        layers, save= [], []
        self.nc = 1
        self.detector_index = -1
        self.det_out_idx = block_cfg[0][0]
        self.seg_out_idx = block_cfg[0][1:]
```

```
        # Build model
        for i, (from_, block, args) in enumerate(block_cfg[1:]):
            block = eval(block) if isinstance(block, str) else block  # eval strings
            if block is Detect:
                self.detector_index = i
            block_ = block(*args)
            block_.index, block_.from_ = i, from_
            layers.append(block_)
            save.extend(x % i for x in ([from_] if isinstance(from_, int) else from_) if x != -1)  # append to savelist
        assert self.detector_index == block_cfg[0][0]

        self.model, self.save = nn.Sequential(*layers), sorted(save)
        self.names = [str(i) for i in range(self.nc)]

        # set stride·anchor for detector
        Detector = self.model[self.detector_index]  # detector
        if isinstance(Detector, Detect):
            s = 128  # 2x min stride
            # for x in self.forward(torch.zeros(1, 3, s, s)):
            #     print (x.shape)
            with torch.no_grad():
                model_out = self.forward(torch.zeros(1, 3, s, s))
                detects, _, _= model_out
                Detector.stride = torch.tensor([s / x.shape[-2] for x in detects])  # forward
            # print("stride"+str(Detector.stride ))
            Detector.anchors /= Detector.stride.view(-1, 1, 1)  # Set the anchors for the corresponding scale
            check_anchor_order(Detector)
            self.stride = Detector.stride
            self._initialize_biases()

        initialize_weights(self)


    def forward(self, x):
        cache = []
        out = []
        det_out = None
        Da_fmap = []
        LL_fmap = []
        for i, block in enumerate(self.model):
            if block.from_ != -1:
                x = cache[block.from_] if isinstance(block.from_, int) else [x if j == -1 else cache[j] for j in block.from_]       #ca
            x = block(x)
            if i in self.seg_out_idx:     #save driving area segment result
                m=nn.Sigmoid()
                out.append(m(x))
            if i == self.detector_index:
                det_out = x
            cache.append(x if block.index in self.save else None)
        out.insert(0,det_out)
#        return out
#        out_boxes = out[0]
#        anchors_list=[10,13,16,30,33,23,30,61,62,45,59,119,116,90,156,198,373,326]
#        num_anchors = len(anchors_list)
#        img_h = 640
#        img_w = 640
#        conf_thres = 0.01  #0.25
#        iou_thres = 0.6   #0.45
#        maxboxnum = 1024
#        num_classes = 10
#        detect_out = torch.ops.torch_mlu.yolov5_detection_output(out_boxes[0],out_boxes[1],out_boxes[2],anchors_list,num_classes,num_a
#        return detect_out
        print("out_boxes[0].shape",out[0][0].shape)
        print("out_boxes:len",len(out[0]))
        print("out_boxes[0].shape",out[0][1].shape)
#        anchors=[12,16,19,36,40,28,36,75,76,55,72,146,142,110,192,243,459,401]
#        num_classes = 80
#        img_size = 640
#        conf_thres = 0.01
#        nms_thres = 0.6
#        maxBoxNum = 1024
#        num_anchors = 18
#        detect_out = torch.ops.torch_mlu.yolov3_detection_output(out_boxes[0], out_boxes[1], out_boxes[2], tuple(anchors), num_classes
#        return detect_out
        return out

    def _initialize_biases(self, cf=None):  # initialize biases into Detect(), cf is class frequency
        # https://arxiv.org/abs/1708.02002 section 3.3
        # cf = torch.bincount(torch.tensor(np.concatenate(dataset.labels, 0)[:, 0]).long(), minlength=nc) + 1.
```

```
        # m = self.model[-1]  # Detect() module
        m = self.model[self.detector_index]  # Detect() module
        for mi, s in zip(m.m, m.stride):  # from
            b = mi.bias.view(m.na, -1)  # conv.bias(255) to (3,85)
            b.data[:, 4] += math.log(8 / (640 / s) ** 2)  # obj (8 objects per 640 image)
            b.data[:, 5:] += math.log(0.6 / (m.nc - 0.99)) if cf is None else torch.log(cf / cf.sum())  # cls
            mi.bias = torch.nn.Parameter(b.view(-1), requires_grad=True)

def get_net(cfg, **kwargs):
    m_block_cfg = YOLOP
    model = MCnet(m_block_cfg, **kwargs)
    return model


if __name__ == "__main__":
    from torch.utils.tensorboard import SummaryWriter
    model = get_net(False)
    input_ = torch.randn((1, 3, 256, 256))
    gt_ = torch.rand((1, 2, 256, 256))
    metric = SegmentationMetric(2)
    model_out,SAD_out = model(input_)
    detects, dring_area_seg, lane_line_seg = model_out
    Da_fmap, LL_fmap = SAD_out
    for det in detects:
        print(det.shape)
    print(dring_area_seg.shape)
    print(lane_line_seg.shape)
```