

# Efficient RGB-D Semantic Segmentation for Indoor Scene Analysis

Daniel Seichter, Mona Köhler, Benjamin Lewandowski

May 2021

## 摘要

在不同环境中行动的移动机器人对场景的彻底分析至关重要。语义分割可以增强各种后续任务，如（语义辅助的）人物感知、（语义的）自由空间检测、（语义的）地图构建和（语义的）导航。本文提出了一种高效且稳健的 RGB-D 分割方法，可以利用 NVIDIA TensorRT 进行高度优化，因此非常适合作为移动机器人场景分析复杂系统中的公共初始处理步骤。我们展示了 RGB-D 分割优于仅处理 RGB 图像，并且在网络架构经过精心设计的情况下仍然可以实时执行。在常见的室内数据集 NYUv2 和 SUNRGB-D 上评估了我们提出的高效场景分析网络 (ESANet)，并展示了结果达到了最先进的性能，同时实现了更快的推断速度。此外，在室外数据集 Cityscapes 上的评估结果表明，该方法也适用于其他应用领域。最后，论文不仅提供了基准结果，还展示了在一个室内应用场景中的定性结果。

**关键词：**图像分割; 网络架构; 实时系统; 导航

## 1 引言

这篇论文 [17] 介绍了一种高效的 RGB-D 语义分割方法，可以作为复杂系统中用于移动机器人场景分析的常见初始处理步骤。作者提出的方法被称为 ESANet，其特点是两个增强的基于 Resnet 的编码器，利用 Non-Bottleneck-1D 块，一个基于注意力的融合来合并深度信息，一个利用一种新的学习上采样的解码器。作者在常见的室内数据集 NYUv2 和 SUNRGB-D 上评估了他们的方法，并展示了达到最先进性能的同时实现更快的推理。此外，作者还在室内应用场景之一中展示了定性结果，证明了该方法的适用性和鲁棒性。本文的主要贡献包括高效的 RGB-D 分割方法、对方法的基本部分进行的详细消融研究以及在复杂机器人场景分析系统中的定性结果。

## 2 相关工作

语义分割网络架构遵循编码器-解码器设计。该编码器从输入中提取语义丰富的特征，并执行降采样，以减少计算工作量。解码器恢复输入分辨率，并最后为每个输入像素分配一个语义类。

## 2.1 RGB-D 语义分割

将深度信息合并到 RGB 分割架构中具有挑战性，因为深度引入了偏离另一种模态的统计数据和特征。有作者 [29] 将深度信息用于将 RGB 图像投影到三维空间中，但是这样会导致显著提高的计算复杂度。有部分作者 [24] [26] [22] [2] [4] 设计了专门定制的卷积，并考虑到了深度信息。然而，这些修改后的卷积往往缺乏优化的实现，因此，速度较慢，不适用于嵌入式硬件上的实时分割。

大多数 RGB-D 分割 [7]、[10]、[9]、[25]、[3]、[15]、[21]、[6] 的方法只是使用两个分支，一个用于 RGB，另一个用于深度数据，并在以后的网络中融合特征表示。这样，每个分支都可以专注于提取特定于形态的特征，比如从 RGB 图像中提取颜色和纹理，以及从深度图像中提取几何的、与光照无关的特征。融合这些特定于形态的特征会导致更强的特征表示。[7] 表明，如果特征在多个阶段被融合，分割性能就会提高，而不是只融合低级或高级特征。

通常，这些特征在每个分辨率阶段都要融合一次，最后一次融合是在两个编码器的末端。对组合特征只使用一个解码器可以减少计算工作量。FuseNet [7] 和 RedNet [10] 将深度特征融合到 RGB 编码器中，这遵循了一种直觉，即在语义上使用互补的深度信息可以进一步增强更丰富的 RGB 特征。SA-Gate [3] 结合了 RGB 和深度特性，并将重新校准的特性融合回两个编码器中。为了使两个编码器相互独立，ACNet [9] 使用了一个额外的虚拟的第三编码器，从两个编码器中获得特定的特征，并处理合并的特征。除了在编码器中融合外，模态特定的特征还可以通过跳过连接来细化公共解码器中的特征，如 RDFNet [15]、SSMA [21] 和 MMAF-Net [6] 中所述，来细化公共解码器中的特征。然而，上述方法都没有关注于嵌入式硬件的高效 RGB-D 分割。

然而，上述方法都没有关注于嵌入式硬件的高效 RGB-D 分割。使用具有 50 层、101 层甚至 152 层的 ResNets 等深度编码器会导致高推理时间，因此，使它们不适合部署到移动机器人上。

## 2.2 有效语义分割

与 RGB-D 分割相比，最近的 RGB 方法 [16] [23] [11] [12] [14] [27] 也降低了计算复杂度，从而实现了实时分割。最有效的分割方法提出了专门定制的网络架构，它减少了操作的数量和参数，以实现更快的推理，同时仍然保持良好的分割性能。ERFNet [16]、LEDNet [23] 或 DABNet [11] 等方法引入了高效的编码器块，用更轻的  $3 \times 3$  卷积替换昂贵的  $3 \times 3$  卷积，如分解、分组或深度可分离卷积。然而，尽管需要更多的操作和内存，SwiftNet [14] 和 BiSeNet [27] 仍然比许多其他方法更快，同时通过简单地使用预先训练的 ResNet18 作为编码器来实现更高的分割性能。这可以推导出在编码器中利用早期和高下采样，一个轻量级解码器，并使用标准的  $3 \times 3$  卷积，目前比分组或深度卷积更有效地实现，并具有很大的表示能力。

根据 SwiftNet 和 BiSeNet，我们的方法还使用了一个基于 resnet 的编码器。然而，为了进一步减少推理时间，我们将所有 ResNet 层中的 ResNet 层中的基本块交换为一个更有效的块。

### 3 本文方法

#### 3.1 本文方法概述

先前的研究中，许多语义分割方法使用编码器-解码器结构，其中编码器提取特征，解码器恢复分割结果。在 RGB-D 语义分割方面，深度图像提供了额外的几何信息。然而，将深度信息与 RGB 信息融合在一起仍存在挑战。另一方面，近期的研究致力于高效的 RGB 语义分割，通过减少计算复杂度实现实时分割。尽管有一些方法已经取得了较好的性能，但仍然缺乏针对嵌入式硬件的高效 RGB-D 分割方法。因此，本研究提出了一种高效的 RGB-D 分割方法，使用基于 Resnet 的编码器，并通过使用更有效的基本块来进一步减少推理时间。此方法能够在嵌入式硬件上实时进行分割，并提供高质量的结果。

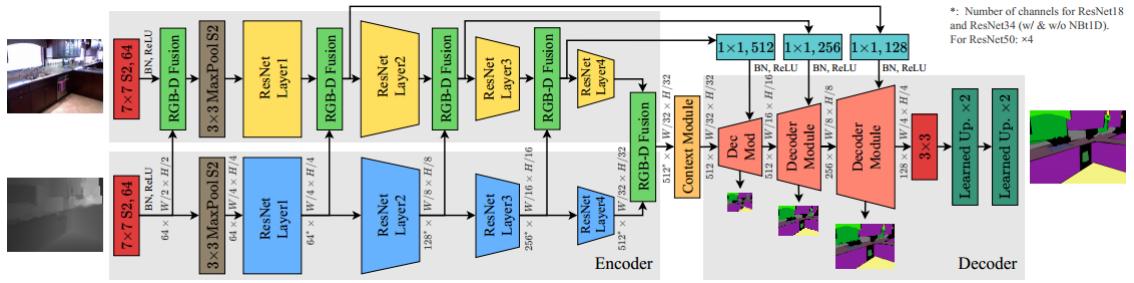


图 1. ESA-Net 架构

本文用于 RGB-D 语义分割的高效场景分析网络 (ESANet) 架构如图 1 所示。它的灵感来自于 SwiftNet [14]，即一个浅编码器与预训练的 ResNet18 骨干和大的下采样。还有一个上下文模块类似于 PSPNet，一个浅解码器与编码器的跳跃式连接，最终上采样为 4 倍。ESANet 对 SwiftNet [28] 没有包含的深度数据使用了额外的编码器。这种深度编码器提取互补的几何信息，并使用注意机制在几个阶段融合到 RGB 编码器中。此外，两种编码器都使用了经过修订的体系结构，支持更快的推断。解码器由多个模块组成，每个模块对生成的特征映射进行 2 倍的上采样，并使用卷积和通过合并编码器的特征来细化特征。最后，解码器将特征映射到类，并将类映射缩放到输入分辨率。为了更快推理速度，整个网络以在 PyTorch 简单组件为主。

#### 3.2 编码器部分

RGB 和深度编码器都使用 ResNet 架构作为骨干。编码器末端的结果特征映射比输入图像小 32 倍。论文为了更快结果使用 ResNet34，并且将 ResNet18 和 ResNet34 每层的基本块替换为空间分解版本。即，每个  $3 \times 3$  卷积被一个  $3 \times 1$  和一个  $1 \times 3$  卷积所取代，其中包含一个 ReLU。所谓的 Non-Bottleneck-1D-Block(NBt1D) 如图 2 所示，最初是在 ERFNet [16] 为另一种网络架构提出的，论文表明该块也可以用于 ResNet，同时减少推理时间和提高分割性能。

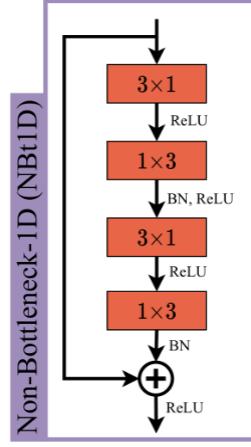


图 2. NBt1D 结构

### 3.3 RGB-D 融合

深度特征在编码器的五个分辨率阶段被融合到 RGB 编码器中。两种模式的特征首先用 Squeeze 和 SE [8] 模块重新加权，然后对元素进行求和，如图 3 所示。使用这种通道注意机制，模型可以根据给定的输入，学习关注哪种模式的哪些特征，抑制哪些特征。实验结果为这种融合机制明显提高了分割。

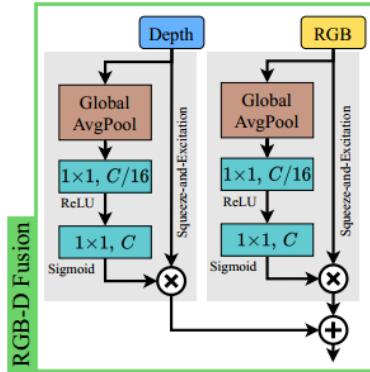


图 3. 深度特征融合到 RGB 编码器

### 3.4 上下文模块

由于 ResNet 的感受野有限，通过使用类似于 PSPNet [28] 中的金字塔池模块中的上下文模块中的几个分支在不同尺度上聚合特征来附加上下文信息，如图 4。论文仔细设计上下文模块，使池大小始终是上下文模块输入分辨率的一个因素，不需要自适应池。实验表明，这个附加的上下文模块提高了分割。

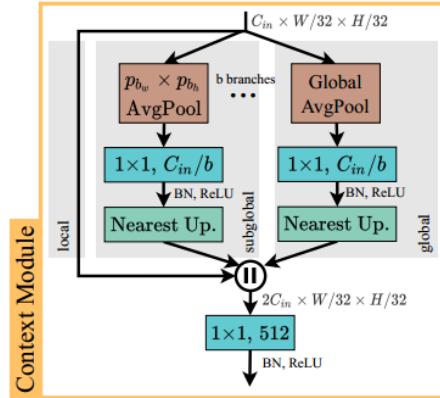


图 4. 上下文模块

### 3.5 解码器

解码器由三个解码器模块组成，如图 5。解码器模块扩展了 SwiftNet [14] 的模块，后者由一个固定数量为 128 通道的  $3 \times 3$  卷积和后续的双线性上采样组成。实验表明，对于室内 RGB-D 分割需要一个更复杂的解码器。因此，在第一个解码器模块中使用 512 个通道，并随着分辨率的增加减少每个  $3 \times 3$  卷积中的通道数量。此外，我们加入了三个额外的 NBt1D 块，以进一步提高细分性能。

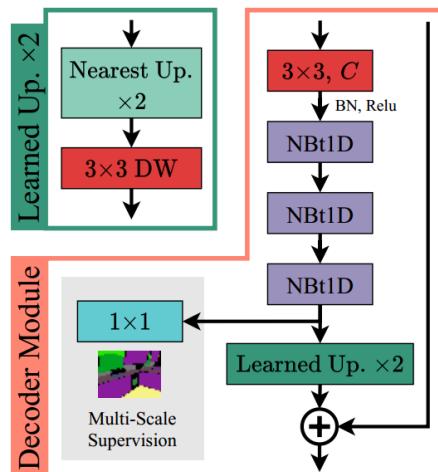


图 5. 解码器

### 3.6 轻量上采样方法

首先使用最近邻上采样来放大分辨率，然后应用  $3 \times 3$  深度卷积来合并相邻特征。通过初始化核来模仿双线性插值，但网络能够在训练中学习如何以更有用的方式组合相邻特征，从而提高了分割性能。此外，为了恢复编码器下采样过程中丢失的细粒度细节，设计了从相同分辨率的编码器到解码器阶段的跳跃式连接。将融合的 RGB-D 编码器特征映射通过  $1 \times 1$  卷积投影到解码器中使用的相同数量的通道，并将它们添加到解码器特征映射中。这种跳跃式连接的融合产生了更详细的语义分割结果。与一些方法相似的是，在解码器中处理特征映射，直到它们比输入图像小 4 倍，并使用  $3 \times 3$  卷积将特征映射转化为各自数据集的类别。为了更

好地监督网络的训练，在每个解码器模块中添加了一个较小尺度的分割，通过  $1 \times 1$  卷积计算，该分割由缩小尺度的地面真实分割监督。

## 4 复现细节

### 4.1 与已有开源代码对比

该论文源码位置为[ESANet 代码](#)，使用了三个数据集 SUNRGB-D [19]、NYUv2 [18] 和 Cityscapes [5]。

代码中经过修改 bug 后可以正常运行。在另外工作中加入了自动混合训练、提前停止、梯度裁剪等尝试，并且在原文基础上添加 AdamW、RMSprop 优化器，以及线性退火策略、周期性学习率尝试，激活函数上加入原文未有的 LeakyReLU、PReLU、ELU 和 SELU 多个激活函数选择。最终结果均接近或者稍微超越原文结果。

### 4.2 实验环境搭建

#### 4.2.1 创建环境

首先将 github 下代码下载保存到本地，网址为[ESANet 代码地址](#)，之后使用 anaconda 新建环境 ESANet，这里现在 python3.8 版本的，因为这个版本比较稳定也比较适应其余大部分其他包，代码如下。

```
1 conda create -n ESANet python=3.8
2 conda activate ESANet
```

#### 4.2.2 数据集处理

本文具有三个数据集，分别是 NYUv2, Cityscapes, SUNRGB-D。

对于 NYUv2 数据集，首先进入 NYUv2 数据集文件夹，按照 README.md 安装依赖文件。同时还需安装 Opencv 的库，否则会出现”No module named ‘cv2’”的错误，安装完成后运行 python 文件。

```
1 cd src/datasets/nyuv2
2 pip install -r ./requirements.txt
3 conda install opencv
4 python prepare_dataset.py ../../..../datasets/nyuv2
```

安装完成后，运行结果如下图 6。

```
(ESANet) PS D:\pythonProject\ESANet\src\datasets\nyuv2> python prepare_dataset.py ../../..../datasets/nyuv2
Traceback (most recent call last):
  File "prepare_dataset.py", line 13, in <module>
    import h5py
  File "C:\ProgramData\anaconda3\envs\ESANet\lib\site-packages\h5py\_init_.py", line 25, in <module>
    from . import _errors
ImportError: DLL load failed while importing _errors: 找不到指定的程序。
```

图 6. NYUv2 出现错误

根据报错指令，问题估计是 h5py 使用 pip 安装版本过低导致，使用如下两行指令先用 pip 卸载原来的 h5py，再次使用 conda 安装尝试，再次运行即可正常下载运行，等待完成即可，如图 7。

```
1 pip uninstall h5py
2 conda install h5py
3 python prepare_dataset.py ../../datasets/nyuv2
```

```
(ESANet) PS D:\pythonProject\ESANet\src\datasets\nyuv2> python prepare_dataset.py ../../datasets/nyuv2
Downloading mat file to: `C:\Users\ADMINI~1\AppData\Local\Temp\nyu_depth_v2_labeled.mat'
nyu_depth_v2_labeled.mat: 2.97GB [25:08, 1.97MB/s]
Loading mat file: `C:\Users\ADMINI~1\AppData\Local\Temp\nyu_depth_v2_labeled.mat'
Processing set: train
100%|██████████| 2.97G/2.97G [00:25<00:00, 1.97MB/s]
Processing set: test
100%|██████████| 2.97G/2.97G [00:25<00:00, 1.97MB/s]
Writing meta files
Removing downloaded mat file: `C:\Users\ADMINI~1\AppData\Local\Temp\nyu_depth_v2_labeled.mat'
```

图 7. NYUV2 完成下载

对于 cityscapes 数据集，同样先进入 cityscapes 文件夹，按照 Readme.md 教程需要安装依赖，原文给出代码是 pip 安装，但是后续为了防止安装包版本过低，统一使用 pip3 安装，代码如下。

```
1 cd ../cityscapes
2 pip3 install -r ./requirements.txt
```

安装完成后，由于 cityscapes 数据集数据量比较大，需要在网站上下载，网址为[cityscapes 数据集地址](#)，选择与 README.md 中对应的数据集，在 README.md 中是如下说明的，如图 8。

2. Download and unzip dataset files:

Use `csDownload` or download the files mentioned below manually from: [Cityscapes Dataset Downloads](#)

```
CITYSCAPES_DOWNLOAD_DIR="/path/where/to/store/cityscapes_downloads"

# using cityscapesScripts
# use "csDownload -l" to list available packages

# labels
csDownload gtFine_trainvaltest.zip -d $CITYSCAPES_DOWNLOAD_DIR      # -> 241MB
# rgb images
csDownload leftImg8bit_trainvaltest.zip -d $CITYSCAPES_DOWNLOAD_DIR    # -> 11GB
# disparity images (only upon request)
csDownload disparity_trainvaltest.zip -d $CITYSCAPES_DOWNLOAD_DIR     # -> 3.5GB
# intrinsic and extrinsic camera parameter to calculate depth
csDownload camera_trainvaltest.zip -d $CITYSCAPES_DOWNLOAD_DIR        # -> 2MB

# unzip files
find $CITYSCAPES_DOWNLOAD_DIR -name '*.zip' -exec unzip -o {} -d $CITYSCAPES_DOWNLOAD_DIR \;
```

图 8. cityscapes 数据集 Readme

因此，我们从网站上选择对应四个文件 `gtFine_trainvaltest.zip`, `leftImg8bit_trainvaltest.zip`, `disparity_trainvaltest.zip`, `camera_trainvaltest.zip`, 解压到 `data` 文件夹，如图 9。

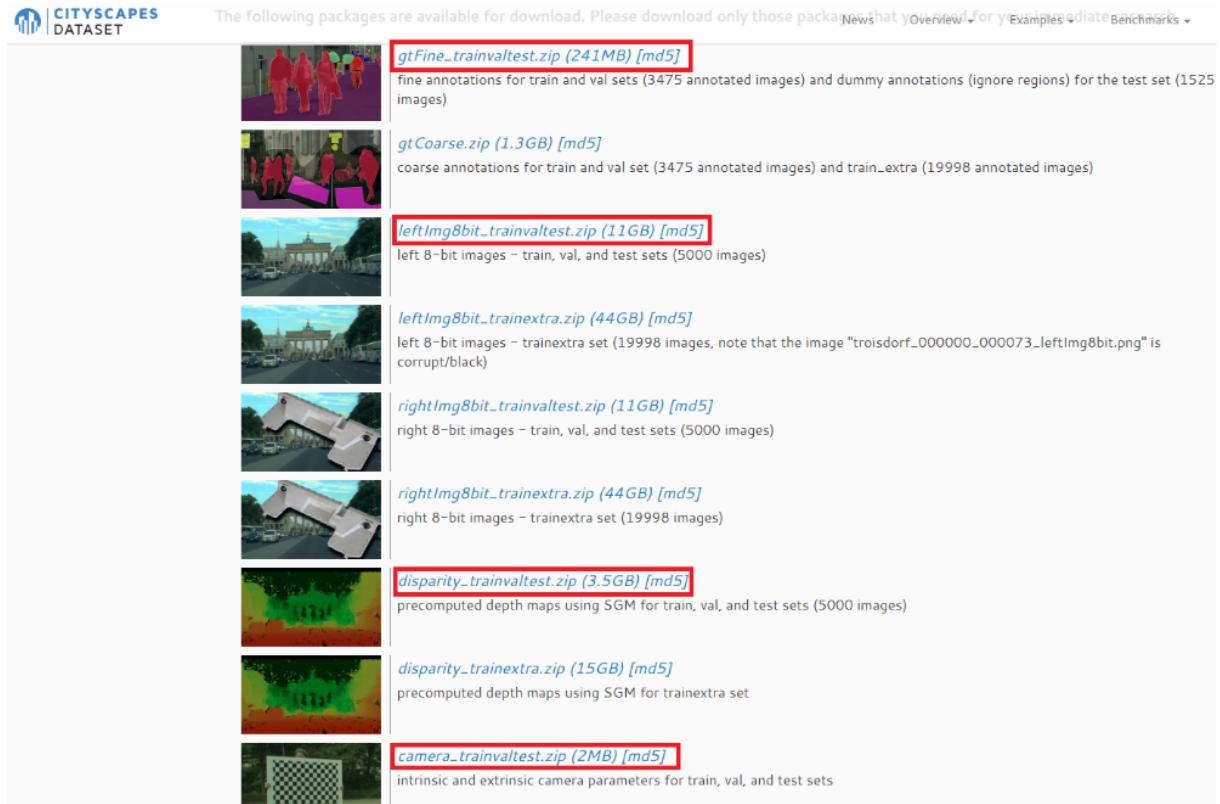


图 9. cityscapes 网站选择

最终提取其中四个文件夹后解压到文件夹 `prepare_data` 是这样的，如图 10。

<< ESANet > src > datasets > cityscapes > prepare_data		
名称	修改日期	类型
camera	2023/11/5 12:58	文件夹
disparity	2023/11/5 13:04	文件夹
gtFine	2023/11/5 13:08	文件夹
leftImg8bit	2023/11/5 13:12	文件夹

图 10. cityscapes 数据集准备

现在运行代码，等待处理完成即可。完成结果如图 11。

```
1 python prepare_dataset.py ../../datasets/cityscapes prepare_data
```

```
(ESANet) PS D:\pythonProject\ESANet\src\datasets\cityscapes> python prepare_dataset.py ../../datasets/cityscapes prepare_data
Copying rgb files
100%
Copying disparity files and creating depth files
100%
Processing label files
100%
Writing meta files
```

图 11. cityscapes 数据集处理完成

完成后，数据集结构如下图12，下面省略部分 txt 文件。

名称		修改日期
test		2023/11/15 15:37
train		2023/11/15 15:38
valid		2023/11/15 15:41
class_colors_1+19.txt		2023/11/15 15:41
class_colors_1+33.txt		2023/11/15 15:41
class_names_1+19.txt		2023/11/15 15:41
class_names_1+33.txt		2023/11/15 15:41

图 12. cityscapes 数据集完成后文件夹

PS：如果出现无法直接使用 conda install 或者 pip install 安装 cityscapesScripts 这个包的时候，大概率是没有进行 pip 换源。下面是一开始进行尝试时候错误，如图 13。

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)'))': /simple/pyquaternion/
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by 'SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)'))': /simple/pyquaternion/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after connection broken by 'SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)'))': /simple/pyquaternion/
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after connection broken by 'SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)'))': /simple/pyquaternion/
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after connection broken by 'SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)'))': /simple/pyquaternion/
WARNING: Could not fetch URL https://pypi.org/simple/pyquaternion/: There was a problem confirming the ssl certificate: HTTPSConnectionPool(host='pypi.org', port=443): Max retries exceeded with url: /simple/pyquaternion/ (Caused by SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)')) - skipping
ERROR: Could not find a version that satisfies the requirement pyquaternion (from versions: none)
ERROR: No matching distribution found for pyquaternion
Could not fetch URL https://pypi.org/simple/pip/: There was a problem confirming the ssl certificate: HTTPSConnectionPool(host='pypi.org', port=443): Max retries exceeded with url: /simple/pip/ (Caused by SSLError(SSLCZeroReturnError(6, 'TLS/SSL connection has been closed (EOF) (_ssl.c:1131)')) - skipping
```

图 13. pip 未换源错误

这里给出对应的换源办法是：创建 C:\Users\Administrator\AppData\Roaming\pip 文件夹，在该文件夹下创建 pip.ini 写入如下内容：

```
1 [global]
2 index-url=http://mirrors.aliyun.com/pypi/simple/
3 [install]
4 trusted-host=mirrors.aliyun.com
```

保存后退出，再次使用 pip install，即可顺利安装，完成换源。

对于 SUNRGB-D 数据集，同样先进入 sunrgbd 文件夹，按照教程安装依赖，这里出现错误，不影响结果，此处无视错误，再使用命令下载数据集。

```
1 cd ../sunrgbd
2 pip3 install -r ./requirements.txt
3 python prepare_dataset.py ../../../../datasets/sunrgbd
```

安装完成时出现错误的结果如图 14。

```
(ESANet) PS D:\pythonProject\ESANet\src\datasets\sunrgbd> pip3 install -r ./requirements.txt
Looking in indexes: http://mirrors.aliyun.com/pypi/simple/
Requirement already satisfied: h5py in c:\programdata\anaconda3\envs\esanet\lib\site-packages (from -r ./requirements.txt (line 1)) (3.9.0)
Requirement already satisfied: numpy in c:\programdata\anaconda3\envs\esanet\lib\site-packages (from -r ./requirements.txt (line 2)) (1.24.3)
Requirement already satisfied: scipy in c:\programdata\anaconda3\envs\esanet\lib\site-packages (from -r ./requirements.txt (line 3)) (1.10.1)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\envs\esanet\lib\site-packages (from -r ./requirements.txt (line 4)) (4.66.1)
ERROR: Could not find a version that satisfies the requirement zipfile (from versions: none)
ERROR: No matching distribution found for zipfile
```

图 14. 安装完成时出现错误

数据集下载完成的结果如图 15。这样即可完成数据集处理部分。

```
(ESANet) PS D:\pythonProject\ESANet\src\datasets\sunrgbd> python prepare_dataset.py ../../../../datasets/sunrgbd
SUNRGBDtoolbox.zip: 570MB [12:20, 770kB/s]
SUNRGBD.zip: 6.89GB [2:01:01, 948kB/s]
Extract images
Extract labels from SUNRGBD toolbox
10335it [02:29, 68.90it/s]
written file train_rgb.txt
written file train_depth.txt
written file train_label.txt
written file test_rgb.txt
written file test_depth.txt
written file test_label.txt
```

图 15. 数据集下载完成

#### 4.2.3 运行环境准备

第一步安装 GPU 版 pytorch，这里选择 11.8 的 pytorch 版本，避免版本过低或者过高不稳定，使用官网给出指令，其中”\”表示换行继续输入内容为。完成结果如图 16。

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

```

pytorch-mutex      pytorch/noarch::pytorch-mutex-1.0-cuda
pyyaml             pkgs/main/linux-64::pyyaml-6.0.1-py38h5eee18b_0
requests           pkgs/main/linux-64::requests-2.31.0-py38h06a4308_0
sympy              pkgs/main/linux-64::sympy-1.11.1-py38h06a4308_0
tbb                pkgs/main/linux-64::tbb-2021.8.0-hdb19cb5_0
torchaudio         pytorch/linux-64::torchaudio-2.1.1-py38_cu118
torchtriton        pytorch/linux-64::torchtriton-2.1.0-py38
torchvision        pytorch/linux-64::torchvision-0.16.1-py38_cu118
typing_extensions pkgs/main/linux-64::typing_extensions-4.7.1-py38h06a4
urllib3            pkgs/main/linux-64::urllib3-1.26.18-py38h06a4308_0
yaml               pkgs/main/linux-64::yaml-0.2.5-h7b6447c_0
zstd               pkgs/main/linux-64::zstd-1.5.5-hc292b87_0

Proceed ([y]/n)? y

Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(ESANet) zikai@amax:/data1/dzk/ESANet$ 

```

图 16. 安装 GPU 版 pytorch

安装完成后，为避免错装为 CPU 版本，可以使用如下测试结果，使用指令进入 python 环境，之后输入”print(torch.\_\_version\_\_)”查看结果，完整代码如下：

```

1 python
2 import torch
3 print(torch.__version__)

```

之后，查看结果，如果如下图 17 即为安装 GPU 版本成功。如果没有后缀 “+cu118”，说明安装的是 CPU 版本的，需要卸载重新安装。

```

>>> import torch
>>> print(torch.__version__)
2.1.1+cu118

```

图 17. GPU 版 pytorch 显示内容

防止包版本过低无法找到，这里不跟随代码中 README.MD 安装，发现还需要安装 pandas, tensorflow, mock 和 pytorch-ignite。

```
1 pip3 install pandas tensorflow-gpu==2.4.1 mock pytorch-ignite
```

如果速度太慢可以使用清华换源网址，代码如下：

```

1 pip3 install pandas tensorflow-gpu==2.4.1 mock pytorch-ignite \
2 -i https://pypi.tuna.tsinghua.edu.cn/simple

```

完成安装后，现在我们从头开始训练模型，发生警告，如图 18。

```

<module>
    from tensorflow.python.framework import sparse_tensor as _sparse_tensor
    File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/tensorflow/python/framework/sparse_tensor.py", line 29, in <module>
        from tensorflow.python.framework import constant_op
        File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/tensorflow/python/framework/constant_op.py", line 29, in <module>
            from tensorflow.python.eager import execute
            File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/tensorflow/python/eager/execute.py", line 27, in <module>
                from tensorflow.python.framework import dtypes
                File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/tensorflow/python/framework/dtypes.py", line 513,
                in <module>
                    np.object,
                    File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/numpy/__init__.py", line 305, in __getattr__
                        raise AttributeError(__former_attrs__[attr])
AttributeError: module 'numpy' has no attribute 'object'.
`np.object` was a deprecated alias for the builtin object. To avoid this error in existing code, use `object` by itself. Doing this will not modify any behavior and is safe.
The aliases was originally deprecated in NumPy 1.20; for more details and guidance see the original release note at:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
(ESANet) zikai@amax:/data1/dzk/ESANet$ █

```

图 18. 开始训练发生警告

查阅资料发现，是 tensorflow 版本中运用 numpy 过低版本，太新 numpy 版本导致无法使用。因此查阅到可以使用的版本对应安装，分别是 tensorflow-gpu=2.4.1, numpy=1.21.1, matplotlib=3.3.4, pandas=0.25.3, tqdm=4.66.1, pycuda。

- 1 pip3 install matplotlib==3.3.4
- 2 pip3 install numpy==1.21.1
- 3 pip3 install pandas==0.25.3
- 4 pip3 install tqdm==4.66.1
- 5 pip3 install pycuda==2023.1

下面列出安装的所有包并且支持正常运行的环境，如图 19。

Name	Version	Name	Version	Name	Version	Name	Version	Name	Version	Name	Version
_libgcc_mutex	0.1	fonttools	4.44.3	fd4_impl_linux-64	2.38	openssl	3.0.12	readline	8.2	tk	8.6.12
_openmp_mutex	5.1	fsspec	2023.4.0	libffi	3.4.4	opt-einsum	3.3.0	requests	2.28.1	torch	2.1.1+cu18
absl-py	0.15.0	gast	0.3.3	libgcc-ng	11.2.0	packaging	23.2	requests-oauthlib	1.3.1	torchaudio	2.1.1+cu18
appdirs	1.4.4	google-auth	2.23.4	libomp	11.2.0	pandas	0.25.3	rsa	4.9	torchvision	0.16.1+cu18
astunparse	1.6.3	google-auth-oauthlib	0.4.6	libstdc++-ng	11.2.0	pillow	9.3.0	scipy	1.10.1	tqdm	4.66.1
ca-certificates	2023.08.22	google-pasta	0.2.0	markdown	3.5.1	pip	23.3	setuptools	68.0.0	triton	2.1.0
cachetools	5.3.2	grpcio	1.32.0	markupsafe	2.1.3	protobuf	3.20.3	six	1.15.0	typing	3.7.4.3
certifi	2022.12.7	h5py	2.10.0	matplotlib	3.3.4	pyasn1	0.5.0	sqlite	3.41.2	typing-extensions	4.8.0
charset-normalizer	2.1.1	humanfriendly	10	mock	4.0.3	pyasn1-modules	0.3.0	sympy	1.12	urllib3	1.26.13
cityscapesscripts	2.2.2	idna	3.4	mpmath	1.3.0	pyparsing	3.1.1	tensorboard	2.11.2	werkzeug	3.0.1
coloredlogs	15.0.1	importlib-metadata	6.8.0	ncurses	6.4	pyquaternion	0.9.9	tensorboard-data-server	0.6.1	wheel	0.41.2
contourpy	1.1.1	importlib-resources	6.1.1	networkx	3	python	3.8.18	tensorboard-plugin-wit	1.8.1	wrapt	1.12.1
cycler	0.12.1	jinja2	3.1.2	numpy	1.19.5	python-dateutil	2.8.2	tensorflow-estimator	2.4.0	xz	5.4.2
filelock	3.9.0	keras-preprocessing	1.1.2	oauthlib	3.2.2	pytorchignite	0.4.13	tensorflow-gpu	2.4.1	zipp	3.17.0
flatbuffers	1.12	kiwisolver	1.4.5	opencc-python	4.8.1.78	pytz	2023.3.post1	termcolor	1.1.0	zlib	1.2.13

图 19. 环境中所有包版本

PS: 如果出现如图 20报错信息，应该是 typing-extensions 版本不对应。

```
(ESANet) zikai@amax:/data1/dzk/ESANet$ python train.py
Traceback (most recent call last):
  File "train.py", line 17, in <module>
    import torch
  File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/torch/__init__.py", line 1429, in <module>
    from torch import optim as optim
  File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/torch/optim/__init__.py", line 8, in <module>
    from .adadelta import Adadelta
  File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/torch/optim/adadelta.py", line 4, in <module>
    from .optimizer import Optimizer, _use_grad_for_differentiable, _default_to_fused_orforeach,
  File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/torch/optim/optimizer.py", line 23, in <module>
    from typing_extensions import ParamSpec, Self, TypeAlias
ImportError: cannot import name 'ParamSpec' from 'typing_extensions' (/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/typing_extensions.py)
```

图 20. typing-extensions 版本不对应

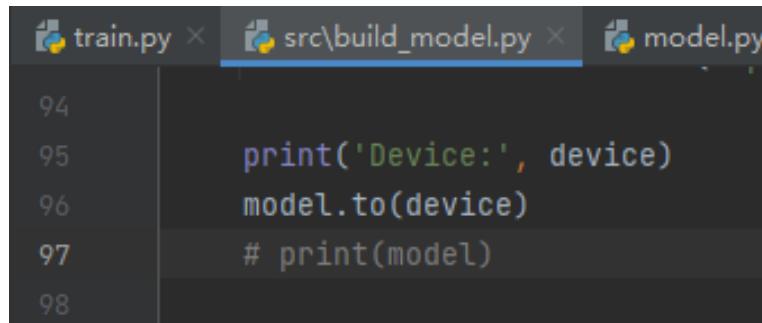
使用 pip3 重新安装 typing\_extensions 可以解决。

```
1 pip3 install typing-extensions==4.8.0
```

### 4.3 代码 bug 修复与使用说明

#### 4.3.1 代码 bug 修复

通读代码发现，在每次运行时，build\_model.py 都会在一开始输出很长的模型结构，不利于调试代码，这里首先屏蔽掉。如图 21。



```
94
95     print('Device:', device)
96     model.to(device)
97     # print(model)
98
```

图 21. 屏蔽输出模型结构

之后阅读文中给出的 README.md 文件，当中说明需要准备预训练文件，下载三份对应权重到 trained\_models 文件夹中，如图 22。

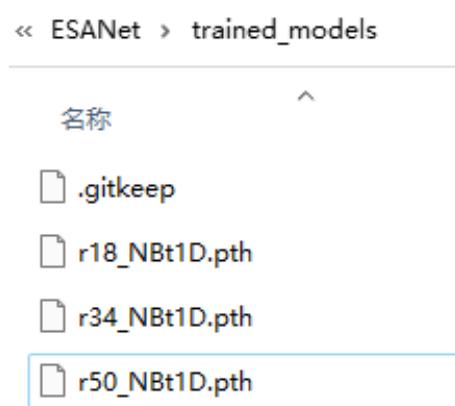


图 22. 准备预训练文件

三份权重网址在“src/models/resnet.py”中，改为对应名字即可。分别为

```
1 18: https://download.pytorch.org/models/resnet18-5c106cde.pth
2 34: https://download.pytorch.org/models/resnet34-333f7ec4.pth
3 50: https://download.pytorch.org/models/resnet50-19c8e357.pth
```

运行代码时候发现有个错误，如图 23。

```
/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/torch/optim/lr_schedule
r.py:1694: UserWarning: To get the last learning rate computed by the scheduler, p
lease use `get_last_lr()`.
  warnings.warn("To get the last learning rate computed by the scheduler, "
Traceback (most recent call last):
  File "train.py", line 616, in <module>
    train_main()
  File "train.py", line 240, in train_main
    ConfusionMatrixTensorflow(n_classes_without_void)
  File "/data1/dzk/ESANet/src/confusion_matrix.py", line 27, in __init__
    self.cm_func = self.build_confusion_matrix_graph()
  File "/data1/dzk/ESANet/src/confusion_matrix.py", line 48, in build_confusion_ma
trix_graph
    self.ph_cm_y_true = tf.compat.v1.placeholder(dtype=self.dtype,
  File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/tensorflow/pyth
on/ops/array_ops.py", line 3176, in placeholder
    raise RuntimeError("tf.placeholder() is not compatible with "
RuntimeError: tf.placeholder() is not compatible with eager execution.
(ESANet) zikai@amax:/data1/dzk/ESANet$
```

图 23. 计算混淆矩阵错误

看到是在 src/confusion\_matrix.py 计算混淆矩阵时候错误，查阅资料发现在 TensorFlow 2.0 中，Eager 模式默认开启，这里将其关闭即可。在文件开头导入包部分加入如下代码

```
1 tf.compat.v1.disable_eager_execution()
```

加入后就可以修改好代码混淆矩阵错误，如图 24。

```

6   import os
7   os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
8
9   import numpy as np
10  import numbers
11  import torch
12  import tensorflow as tf
13  tf.compat.v1.disable_eager_execution()
14
15  from ignite.exceptions import NotComputableError
16  from ignite.metrics import Metric, MetricsLambda

```

图 24. 修改混淆矩阵错误

一次偶然发现，使用 README.md 默认参数 200 轮下跑出结果为 34.3，当被其他原因被迫中断时，想从断点开始继续训练，使用 last\_ckpt 参数希望找到上一次文件继续训练，这时发现出现错误，如图 25。

```

Using /data1/dzk/ESANet/src/datasets/nyuv2/weighting_linear_1+40_test.pickle as class weighting
Using SGD as optimizer
=> no checkpoint found at './results/nyuv2/checkpoints_19_11_2023-19_31_18-754659./results/nyuv2/checkpoints_16
11_2023-14_59_52-638970/ckpt_epoch_200.pth'

```

图 25. 断点重新开始产生错误

这里发现文件路径并不是从这一次的运行文件夹开始寻找，而是从新建的文件夹开始，发现并不太方便，修改为从项目文件夹开始寻找，如图 26。

```

if args.last_ckpt:
    # ckpt_path = os.path.join(ckpt_dir, args.last_ckpt)
    ckpt_path = args.last_ckpt
    epoch_last_ckpt, best_miou, best_miou_epoch = \
        load_ckpt(model, optimizer, ckpt_path, device)
    start_epoch = epoch_last_ckpt + 1

```

图 26. 修改为项目文件夹开始寻找

对于 NYUv2 数据集来说，现在已经可以使用默认参数开始训练，500 轮时间大概是 4.7 个小时，但是对于 Cityscapes 数据集来说，现在还没法开始训练，使用默认参数，会产生报错，如图 27。

```

[ WARN:0@2.453] global loadsolve.cpp:248 findDecoder imread_('./datasets/cityscapes/valid/labels_19/frankf
rt\frankfurt_000000_000294_gtFine_labelIds.png'): can't open/read file: check file path/integrity
Traceback (most recent call last):
  File "train.py", line 645, in <module>
    train_main()
  File "train.py", line 177, in train_main
    pixel_sum_valid_data = valid_loader.dataset.compute_class_weights(
  File "/data1/dzk/ESANet/src/datasets/dataset_base.py", line 169, in compute_class_weights
    label = self.load_label(i)
  File "/data1/dzk/ESANet/src/datasets/cityscapes/pytorch_dataset.py", line 160, in load_label
    return self.load(self._label_dir, self._files['label'][i])
  File "/data1/dzk/ESANet/src/datasets/cityscapes/pytorch_dataset.py", line 140, in _load
    if im.ndim == 3:
AttributeError: 'NoneType' object has no attribute 'ndim'

```

图 27. cityscapes 数据集训练出错

通过逐步溯源代码，可以判断代码是在下面 `src\datasets\cityscapes\pytorch_dataset.py` 中这一段代码中产生的错误，如图 28。

```
129     def _load(self, directory, filename):
130         fp = os.path.join(self._data_dir,
131                           self.split,
132                           directory,
133                           filename)
134         if os.path.splitext(fp)[-1] == '.npy':
135             # depth files as numpy files
136             return np.load(fp)
137         else:
138             # all the other files are pngs
139             im = cv2.imread(fp, cv2.IMREAD_UNCHANGED)
140             if im.ndim == 3:
141                 im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
142
143         return im
144
```

图 28. cityscapes 错误位置溯源

这里先输出 `fp` 查看文件路径，如图 29。

```
./datasets/cityscapes/valid/labels_19/frankfurt/frankfurt_000000_000294_gtFine_labelIds.png
```

图 29. 输出 `fp` 查看文件路径

可以看到，我们文件名已经被解析出错，因此这里加入代码，将反斜杠替换为正斜杠即可。如图 30。

```
1 filename = filename.replace("\\", "/")
```

```
129     def _load(self, directory, filename):
130         # 将反斜杠替换为正斜杠
131         filename = filename.replace("\\", "/")
132         fp = os.path.join(self._data_dir,
133                           self.split,
134                           directory,
135                           filename)
136         if os.path.splitext(fp)[-1] == '.npy':
137             # depth files as numpy files
138             return np.load(fp)
```

图 30. 修改 `fp` 路径中符号

现在对于 Cityscapes 数据集也已经可以开始训练，默认参数写入代码里面，在命令行启动后可以运行，运行 500 轮大概需要 28 小时。

对于 SUNRGB-D，出现与 Cityscapes 一样的错误，文件路径出错，如图 31。

```
(ESANet) zikai@amax:/data1/dzk/ESANet$ python train.py
Compute class weights
Traceback (most recent call last):
  File "train.py", line 673, in <module>
    train_main()
  File "train.py", line 175, in train_main
    class_weighting = train_loader.dataset.compute_class_weights()
  File "/data1/dzk/ESANet/src/datasets/dataset_base.py", line 169, in compute_class_weights
    label = self.load_label(i)
  File "/data1/dzk/ESANet/src/datasets/sunrgbd/pytorch_dataset.py", line 133, in load_label
    label = np.load(os.path.join(self._data_dir,
  File "/home/zikai/.conda/envs/ESANet/lib/python3.8/site-packages/numpy/lib/npyio.py", line 416, in load
    fid = stack.enter_context(open(os_fspath(file), "rb"))
FileNotFoundError: [Errno 2] No such file or directory: './datasets/sunrgbd/SUNRGBD\\kv2/kinect2data/000065_2014-05-16_20-14-38_260595134347_rgbf000121-resize\\label\\label.npy'
```

图 31. SUNRGB-D 数据集训练出错

同样在 src\datasets\sunrgbd\pytorch\_dataset.py 中第 133 行加入替换操作，如图 32。

```
127 def load_label(self, idx):
128     if self.camera is None:
129         label_dir = self.label_dir[self._split]['list']
130     else:
131         label_dir = self.label_dir[self._split]['dict'][self.camera]
132
133     # 将反斜杠替换为正斜杠
134     label_dir[idx] = label_dir[idx].replace("\\", "/")
135     label = np.load(os.path.join(self._data_dir,
136                                 label_dir[idx])).astype(np.uint8)
```

图 32. SUNRGB-D 数据集修改路径

但是还是出现错误，如图 33。说明这样修改并不能从本质上解决问题，需要另寻他路。

```
[ WARN:@0@20.641] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\xtion/sun3ddata/home_han/apartment_han_oct_31_2012_scan1_erika/0003417-000142610580/image/0003417-000142610580.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.642] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\kv1/NYUdata/NYU1300/image/NYU1300.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.642] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\kv1/b3ddodata/img_0212/image/img_0212.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.643] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\xtion/sun3ddata/hotel_florence_jx/florence_hotel_another_room/0001208-000040577128/image/0001208-000040577128.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.643] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\kv2/kinect2data/001977-2014-06-24_16-16-09_260595134347_rgbf000028-resize\image\0000028.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.645] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\realsense/sa/2014_10_21-18_21_20-1311000073/image/0000063.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.646] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\kv1/NYUdata/NYU0162/image/NYU0162.jpg'): can't open/read file: check file path/integrity
[ WARN:@0@20.646] global loadsav.cpp:248 findDecoder imread_('./datasets/sunrgbd/SUNRGBD\xtion/sun3ddata/mit_13_xh_lab2/xh_lab2_1/0002546-000085298220/image/0002546-000085298220.jpg'): can't open/read file: check file path/integrity
```

图 33. SUNRGB-D 路径修改后仍有报错

我一步步调试，发现是源文件中存的文本出现格式错误，如图 34。

```
SUNRGBD\kv2\kinect2data\000002_2014-05-26_14-23-37_260595134347_rgbbf000103-resize\depth_bfx\0000103 8833
SUNRGBD\kv2\kinect2data\000003_2014-05-26_14-24-42_260595134347_rgbbf000040-resize\depth_bfx\0000040.png
SUNRGBD\kv2\kinect2data\000008_2014-05-26_14-30-06_260595134347_rgbbf000060-resize\depth_bfx\0000060.png
SUNRGBD\kv2\kinect2data\000009_2014-05-26_14-32-05_260595134347_rgbbf000034-resize\depth_bfx\0000034.png
SUNRGBD\kv2\kinect2data\000010_2014-05-26_14-32-36_260595134347_rgbbf000020-resize\depth_bfx\0000020.png
SUNRGBD\kv2\kinect2data\000011_2014-05-26_14-33-11_260595134347_rgbbf000020-resize\depth_bfx\0000020.png
SUNRGBD\kv2\kinect2data\000012_2014-05-26_14-35-37_260595134347_rgbbf000180-resize\depth_bfx\0000180.png
```

图 34. SUNRGB-D 源文件格式出错

这样我们只需在读取文本时候修改即可。在 src\datasets\sunrgbd\pytorch\_dataset.py 的 list\_and\_dict\_from\_file 函数中修改为如下代码，如图 35。

```
1 with open(filepath, 'r') as f:
2     raw_file_list = f.read().splitlines()
3 file_list = []
4 for line in raw_file_list:
5     updated_line = line.replace("\\\\", "/")
6     file_list.append(updated_line)
```

```
177     def list_and_dict_from_file(self, filepath):
178         with open(filepath, 'r') as f:
179             raw_file_list = f.read().splitlines()
180             file_list = []
181             for line in raw_file_list:
182                 updated_line = line.replace("\\\\", "/")
183                 file_list.append(updated_line)
184             dictionary = dict()
185             for cam in self.cameras:
186                 dictionary[cam] = [i for i in file_list if cam in i]
187
188             return file_list, dictionary
```

图 35. SUNRGB-D 修改源文件格式

现在 SUNRGB-D 数据集也可以对应开始训练，运行 500 轮大概需要 28 小时。

训练过程中，发现' r18' 的预训练文件格式与' r34' 和' r50' 不太相同，产生了报错。如图 36。

```

File "train.py", line 686, in <module>
    train_main()
File "train.py", line 188, in train_main
    model, device = build_model(args, n_classes=n_classes_without_void)
File "/data1/dzk/ESANet/src/build_model.py", line 48, in build_model
    model = ESANet(
File "/data1/dzk/ESANet/src/models/model.py", line 64, in __init__
    self.encoder_rgb = ResNet18(
File "/data1/dzk/ESANet/src/models/resnet.py", line 396, in ResNet18
    model = load_pretrained_with_different_encoder_block(
File "/data1/dzk/ESANet/src/models/resnet.py", line 493, in load_pretrained_with_different_encoder_block
    for key in checkpoint['state_dict']:
KeyError: 'state_dict'

```

图 36. R18 模型训练出错

溯源时候发现锁定问题源代码如图 37。

```

492             # rename keys and leave out last fully connected layer
493         for key in checkpoint['state_dict']:
494             if 'encoder' in key:
495                 checkpoint['state_dict2'][key.split('encoder.')[ -1]] = \
496                     checkpoint['state_dict'][key]
497             weights = checkpoint['state_dict2']

```

图 37. R18 模型出错代码位置

发现' r34' 预训练格式为图 38，' r18' 预训练格式为图 39。

```

▼ └─checkpoint = {dict: 2} {'state_dict': OrderedDict([('encoder', OrderedDict([('layer1.0.bn1.weight', Parameter: (64, 64, 3, 3)), ('layer1.0.bn1.bias', Parameter: (64,)), ('layer1.0.bn1.running_mean', Tensor: (64,)), ('layer1.0.bn1.running_var', Tensor: (64,))]), ('layer1.0.conv1.weight', Parameter: (64, 3, 7, 7)), ('layer1.0.conv1.bias', Parameter: (64,)), ('layer1.0.conv1.running_mean', Tensor: (64,)), ('layer1.0.conv1.running_var', Tensor: (64,))])}
    > └─'state_dict' = {OrderedDict: 314} OrderedDict([('encoder', OrderedDict([('layer1.0.bn1.weight', Parameter: (64, 64, 3, 3)), ('layer1.0.bn1.bias', Parameter: (64,)), ('layer1.0.bn1.running_mean', Tensor: (64,)), ('layer1.0.bn1.running_var', Tensor: (64,))]), ('layer1.0.conv1.weight', Parameter: (64, 3, 7, 7)), ('layer1.0.conv1.bias', Parameter: (64,)), ('layer1.0.conv1.running_mean', Tensor: (64,)), ('layer1.0.conv1.running_var', Tensor: (64,))])})
    > └─'state_dict2' = {OrderedDict: 0} OrderedDict()
        └─_len_ = {int} 2
    > └─? 受保护的特性
    └─ckpt_path = {str} './trained_models/r34_NBt1D.pth'

```

图 38. R34 预训练格式

```

▼ └─checkpoint = {OrderedDict: 103} OrderedDict([('conv1.weight', Parameter: (64, 3, 7, 7)), ('bn1.running_mean', Tensor: (64,)), ('bn1.running_var', Tensor: (64,)), ('bn1.weight', Parameter: (64,)), ('bn1.bias', Parameter: (64,)), ('layer1.0.bn1.weight', Parameter: (64, 64, 3, 3)), ('layer1.0.bn1.bias', Parameter: (64,)), ('layer1.0.bn1.running_mean', Tensor: (64,)), ('layer1.0.bn1.running_var', Tensor: (64,)), ('layer1.0.conv1.weight', Parameter: (64, 3, 7, 7)), ('layer1.0.conv1.bias', Parameter: (64,)), ('layer1.0.conv1.running_mean', Tensor: (64,)), ('layer1.0.conv1.running_var', Tensor: (64,))])
    > └─'conv1.weight' = {Parameter: (64, 3, 7, 7)} Parameter
    > └─'bn1.running_mean' = {Tensor: (64,)} tensor([ 2.76e-05,  1.0169e-05, ...,  1.0169e-05])
    > └─'bn1.running_var' = {Tensor: (64,)} tensor([ 1.0169e-05,  1.0169e-05, ...,  1.0169e-05])
    > └─'bn1.weight' = {Parameter: (64,)} Parameter containing:
    > └─'bn1.bias' = {Parameter: (64,)} Parameter containing:
    > └─'layer1.0.bn1.weight' = {Parameter: (64, 64, 3, 3)}
    > └─'layer1.0.bn1.bias' = {Parameter: (64,)}
    > └─'layer1.0.bn1.running_mean' = {Tensor: (64,)} tensor([ 2.76e-05,  1.0169e-05, ...,  1.0169e-05])
    > └─'layer1.0.bn1.running_var' = {Tensor: (64,)} tensor([ 1.0169e-05,  1.0169e-05, ...,  1.0169e-05])
    > └─'layer1.0.conv1.weight' = {Parameter: (64, 3, 7, 7)}
    > └─'layer1.0.conv1.bias' = {Parameter: (64,)}
    > └─'layer1.0.conv1.running_mean' = {Tensor: (64,)} tensor([ 2.76e-05,  1.0169e-05, ...,  1.0169e-05])
    > └─'layer1.0.conv1.running_var' = {Tensor: (64,)} tensor([ 1.0169e-05,  1.0169e-05, ...,  1.0169e-05])

```

图 39. R18 预训练格式

可以发现 R18 预训练文件已经把 state\_dict 的所有权重都已经列出来，导致两者格式不统一从而出错，修改代码为如下图 40。这样即可开始训练 resnet18 的实验。

```

492         # rename keys and leave out last fully connected layer
493     if resnet_name == 'r18':
494         for key in checkpoint:
495             checkpoint['state_dict2'][key] = checkpoint[key]
496     else:
497         for key in checkpoint['state_dict']:
498             if 'encoder' in key:
499                 checkpoint['state_dict2'][key.split('encoder.')[ -1]] = \
500                     checkpoint['state_dict'][key]
501
502     weights = checkpoint['state_dict2']

```

图 40. R18 读取预训练文件修改

#### 4.3.2 使用说明

在 ESANet 文件夹下运行 train.py 即可开始运行，默认参数已经写到 train.py 文件中，进入函数会自动进行解析，修改只需修改 set\_config\_nyuv2()、set\_config\_sunrgbd() 以及 set\_config\_city() 参数即可。

1 **python train.py**

### 4.4 创新点

#### 4.4.1 零散修改

在 train.py 中发现在每轮运行时，都会评估一次验证集的 miou，但是前 200 轮时候一般都不太准确，因此，这里加入评估正确率轮数改进，在 200 轮以上才验证比较合理。如图 41。

```

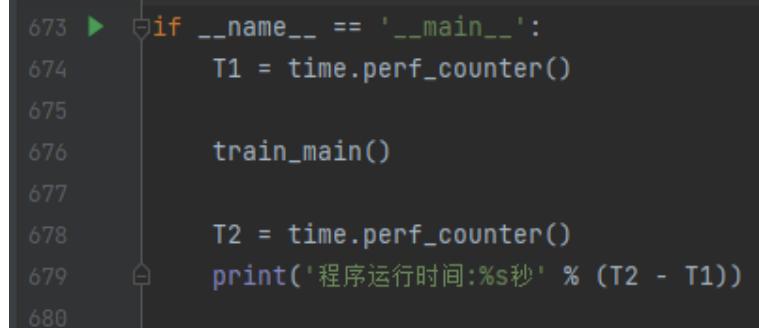
234         # 大于200轮在验证集评估正确率| -----
235     if epoch >= 200:
236         # 代码调用validate()函数进行模型在验证集上的评估。
237         miou, logs = validate(
238             model, valid_loader, device, cameras,
239             confusion_matrices, args.modality, loss_function_valid, logs,
240             ckpt_dir, epoch, loss_function_valid_unweighted,
241             debug_mode=args.debug
242         )
243
244         if args.valid_full_res:
245             miou_full_res, logs = validate(
246                 model, valid_loader_full_res, device, cameras,
247                 confusion_matrices, args.modality, loss_function_valid, logs,
248                 ckpt_dir,

```

图 41. 修改 200 轮以上才验证

加入程序总时间计时器记录一下整个模型对于不同数据集所需训练时间，并且查看是否改进有节省时间。在 train.py 主函数中加入如下代码即可，如图 42。

```
1 T1 = time.perf_counter()
2 train_main()
3 T2 = time.perf_counter()
4 print('程序运行时间:%s秒' % (T2 - T1))
```



```
673 if __name__ == '__main__':
674     T1 = time.perf_counter()
675
676     train_main()
677
678     T2 = time.perf_counter()
679     print('程序运行时间:%s秒' % (T2 - T1))
680
```

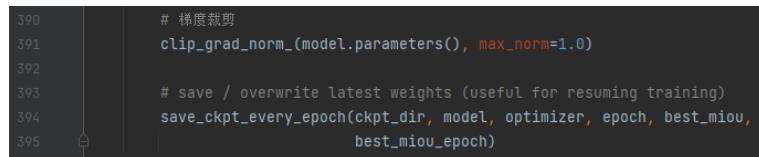
图 42. 查看训练时间

再加入参数在命令行输出，方便查看目前参数，在 train.py 中的 train\_main() 函数在 data preparation 阶段加入 print(args) 即可。

```
1 print(args)
```

梯度裁剪用于控制梯度的大小，加入梯度裁剪避免梯度爆炸或梯度消失的问题。只需在每轮结束时候加入即可。如图

```
1 from torch.nn.utils import clip_grad_norm_
2 clip_grad_norm_(model.parameters(), max_norm=1.0)
```



```
390
391     # 梯度裁剪
392     clip_grad_norm_(model.parameters(), max_norm=1.0)
393
394     # save / overwrite latest weights (useful for resuming training)
395     save_ckpt_every_epoch(ckpt_dir, model, optimizer, epoch, best_miou,
                           best_miou_epoch)
```

图 43. 加入梯度裁剪

但是并没有看到训练结果或者时间有一定程度上升，反而变得训练时长更久，因此此处屏蔽梯度裁剪功能。

#### 4.4.2 自动混合精度

自动混合精度训练可以通过减少内存占用、加速计算、减少显存占用和提高硬件利用率等方式，显著提高深度神经网络训练的效率和速度。在这里加入自动混合精度希望对结果或者训练时长有所改进。

在 train.py 中的 train\_one\_epoch() 函数部分，在开头部分加入导包以及 scaler = GradScaler() 声明，之后在中间加入 with autocast()。原本加入前代码如下图 44。

```

414     # forward pass
415     if modality == 'rgbd':
416         pred_scales = model(image, depth)
417     elif modality == 'rgb':
418         pred_scales = model(image)
419     else:
420         pred_scales = model(depth)
421
422     # 根据模态类型进行前向传播，得到预测的分辨率。
423     # loss computation
424     losses = loss_function_train(pred_scales, target_scales)
425     loss_segmentation = sum(losses)
426
427     total_loss = loss_segmentation
428
429     total_loss.backward()
430     optimizer.step()

```

图 44. 加入自动混合精度前

加入后代码如图 45。需要屏蔽两句话再加入其余三句在 with autocast() 即可。

```

1 # total_loss.backward()
2 # optimizer.step()
3 # 对总损失进行梯度缩放，并进行反向传播和优化器更新。
4 scaler.scale(total_loss).backward()
5 scaler.step(optimizer)
6 scaler.update()

```

```

417     with autocast():
418         # forward pass
419         if modality == 'rgbd':
420             pred_scales = model(image, depth)
421         elif modality == 'rgb':
422             pred_scales = model(image)
423         else:
424             pred_scales = model(depth)
425
426         # 根据模态类型进行前向传播，得到预测的分辨率。
427         # loss computation
428         losses = loss_function_train(pred_scales, t
429         loss_segmentation = sum(losses)
430
431         total_loss = loss_segmentation
432
433         # total_loss.backward()
434         # optimizer.step()
435
436         # 对总损失进行梯度缩放，并进行反向传播和优化器更新。
437         scaler.scale(total_loss).backward()
438         scaler.step(optimizer)
439         scaler.update()

```

图 45. 加入自动混合精度后

这样，在精度不变情况下，NYUv2 数据集训练时间节省 500s，说明我们自动混合精度改进有效。

#### 4.4.3 提前停止

加入 early-stop 参数，控制代码在参数轮数内没有改进，则暂停，默认轮数为 100 轮。首先在 src/args.py 文件中最后加入一行。如图 46。

```
1 self.add_argument('--early_stop', default=100, type=int, \
2 help='Stop early if there is no progress within the number' \
3 'of rounds')
```

```
self.add_argument('--early_stop', default=100, type=int,
                  help='Stop early if there is no progress within the number of rounds')
```

图 46. 加入提前停止参数

之后，在开头中初始化计数器，如图 47。

```
247     else:
248
249
250     start_epoch = 0
251     best_miou = 0
252     best_miou_epoch = 0
253
254     early_stop_counter = 0
```

图 47. 加入提前停止计数器

在判断是否大于最后 miou 值时，加入判断，是归零还是加一。如图 48。

```
348     if miou['all'] > best_miou:
349         best_miou = miou['all']
350         best_miou_epoch = epoch
351         early_stop_counter = 0
352         save_current_checkpoint = True
353     else:
354         early_stop_counter += 1
```

图 48. 判断是加一还是归零

最后，加入判断是否退出即可。如图 49。

```
1 if early_stop_counter >= args.early_stop:
2     print(f'Early stopping at epoch {epoch} as mIoU '
3           f'did not improve for {args.early_stop} consecutive epochs.')
4     break
```

```

366     # Perform early stopping check
367     if early_stop_counter >= args.early_stop:
368         print(f'Early stopping at epoch {epoch} as mIoU '
369               f'did not improve for {args.early_stop} consecutive epochs.')
370         break

```

图 49. 判断是否退出

#### 4.4.4 优化器添加

原文只实现了 SGD 和 Adam，此处加入 AdamW 和 RMSprop 两种优化器，供后人探索更优解。AdamW [13] 是 Adam 优化算法的一个变种，加入了 L2 正则化（权重衰减）。AdamW 的优点是能够同时实现自适应学习率和权重衰减，从而在训练过程中更好地平衡参数更新和模型的正则化。这有助于提高模型的泛化能力和训练稳定性。RMSprop [20] 是一种基于梯度平方的自适应学习率优化算法。RMSprop 的优点是能够自适应地调整学习率，并且在处理具有不同尺度梯度的问题时表现良好。此处加入将有助模型更优解探索。

首先在 src\args.py 中，优化器选择加入两组 AdamW 和 RMSprop 的选择。如图 50。

```

1 self.add_argument('--optimizer', type=str, default='SGD', \
2 choices=['SGD', 'Adam', 'AdamW', 'RMSprop'])

```

```

self.add_argument('--optimizer', type=str, default='SGD',
                  choices=['SGD', 'Adam', 'AdamW', 'RMSprop'])

```

图 50. 加入参数优化器选择

之后在 train.py 中的 get\_optimizer() 加入 AdamW 和 RMSprop 判断。

```

1 elif args.optimizer == 'AdamW':
2     optimizer = torch.optim.AdamW(
3         model.parameters(),
4         lr=args.lr,
5         weight_decay=args.weight_decay,
6         betas=(0.9, 0.999)
7     )
8 elif args.optimizer == 'RMSprop':
9     optimizer = torch.optim.RMSprop(
10        model.parameters(),
11        lr=args.lr,
12        weight_decay=args.weight_decay,
13        alpha=0.9
14    )
15 else:

```

```

16         raise NotImplementedError(
17             'Currently only SGD, Adam, AdamW and RMSprop as '
18             'optimizers are supported. Got {}'.format(args.optimizer))

```

这里参数是经过调整得到与原文结果较为接近参数。修改后代码如下图 51。

```

699     elif args.optimizer == 'Adam':
700         optimizer = torch.optim.Adam(
701             model.parameters(),
702             lr=args.lr,
703             weight_decay=args.weight_decay,
704             betas=(0.9, 0.999)
705         )
706     elif args.optimizer == 'AdamW':
707         optimizer = torch.optim.AdamW(
708             model.parameters(),
709             lr=args.lr,
710             weight_decay=args.weight_decay,
711             betas=(0.9, 0.999)
712         )
713     elif args.optimizer == 'RMSprop':
714         optimizer = torch.optim.RMSprop(
715             model.parameters(),
716             lr=args.lr,
717             weight_decay=args.weight_decay,
718             alpha=0.9
719         )
720     else:
721         raise NotImplementedError(
722             'Currently only SGD, Adam, AdamW and RMSprop as optimizers are '
723             'supported. Got {}'.format(args.optimizer))

```

图 51. 加入 AdamW 和 RMSprop 优化器

现在只需要在参数中对 args.optimizer 更改为 AdamW 和 RMSprop 优化器的选择。

#### 4.4.5 激活函数添加

原文只实现了 ReLU, Swish 和 hSwish，此处加入 LeakyReLU, PReLU, ELU 和 SELU 四种激活函数，供后人探索更优解。LeakyReLU 是一种修正线性单元函数的变体。它在输入小于零时引入一个较小的斜率，以允许负值通过。PReLU 是一个具有可学习参数的修正线性单元函数。与 LeakyReLU 不同，PReLU 的负数部分的斜率是可训练的，而不是固定的。这使得模型可以自适应地学习激活函数的形状和特征。ELU 是一种带指数项的线性单元函数。它在输入小于零时引入一个负指数项，允许负值通过，并且具有平滑的曲线。ELU 在负数区域的值接近于负无穷，这有助于缓解梯度消失问题。SELU 是一种自归一化的激活函数，它在每个神经元的输出上具有稳定的均值和方差。SELU 的设计目标是使得每个层的输入保持相同的均值和方差，从而帮助网络自动调节激活值的范围。这些激活函数设计目标是提供更好的非

线性建模能力，并帮助缓解一些常见的问题，如梯度消失和神经元饱和，加入将有助模型更优解探索。

首先在 src\args.py 中，优化器选择加入 LeakyReLU, PReLU, ELU 和 SELU 四种的选择。如图 52。

```
1 self.add_argument('--activation', type=str, default='relu',
2 choices=['relu', 'swish', 'hswish', 'LeakyReLU', \
3 'PReLU', 'ELU', 'SELU'],
4 help='Which activation function to use in the model')
```

```
# model
self.add_argument('--activation', type=str, default='relu',
                  choices=['relu', 'swish', 'hswish', 'LeakyReLU', 'PReLU', 'ELU', 'SELU'],
                  help='Which activation function to use in the model')
```

图 52. 加入激活函数选择

之后在 src\models\model 加入 LeakyReLU, PReLU, ELU 和 SELU 判断。如图 53。

```
1 from src.models.model_utils import LeakyReLU, PReLU, ELU, SELU
2 elif activation.lower() == 'leakyrelu':
3     self.activation = LeakyReLU()
4 elif activation.lower() == 'prelu':
5     self.activation = PReLU()
6 elif activation.lower() == 'elu':
7     self.activation = ELU()
8 elif activation.lower() == 'selu':
9     self.activation = SELU()
10 else:
11     raise NotImplementedError(
12         'Only relu, swish, hswish, leakyrelu, prelu, elu and selu '
13         'as activation function are supported so far. Got {}'
14         .format(activation))
```

```

47     if activation.lower() == 'relu':
48         self.activation = nn.ReLU(inplace=True)
49     elif activation.lower() in ['swish', 'silu']:
50         self.activation = Swish()
51     elif activation.lower() == 'hswish':
52         self.activation = Hswish()
53     elif activation.lower() == 'leakyrelu':
54         self.activation = LeakyReLU()
55     elif activation.lower() == 'prelu':
56         self.activation = PReLU()
57     elif activation.lower() == 'elu':
58         self.activation = ELU()
59     elif activation.lower() == 'selu':
60         self.activation = SELU()
61     else:
62         raise NotImplementedError(
63             'Only relu, swish, hswish, leakyrelu, prelu, elu and selu '
64             'as activation function are supported so far. Got {}'.format(activation))

```

图 53. 加入激活函数到 model 中

现在进入 src\models\model\_utils 完成 LeakyReLU, PReLU, ELU 和 SELU 四个激活函数的编写。按如下代码加入末尾即可。

```

1 class LeakyReLU(nn.Module):
2     def __init__(self, negative_slope=0.01, inplace=True):
3         super(LeakyReLU, self).__init__()
4         self.negative_slope = negative_slope
5         self.inplace = inplace
6
7     def forward(self, x):
8         return nn.functional.leaky_relu(x,
9             negative_slope=self.negative_slope, inplace=self.inplace)
10
11 class PReLU(nn.Module):
12     def __init__(self, num_parameters=1, init=0.25):
13         super(PReLU, self).__init__()
14         self.num_parameters = num_parameters
15         self.init = init
16         self.weight = \
17             nn.Parameter(torch.Tensor(num_parameters).fill_(init))
18
19     def forward(self, x):
20         return nn.functional.prelu(x, self.weight)
21
22 class ELU(nn.Module):
23     def __init__(self, alpha=1.0, inplace=True):
24         super(ELU, self).__init__()

```

```

25         self.alpha = alpha
26         self.inplace = inplace
27
28     def forward(self, x):
29         return nn.functional.elu(x,
30             alpha=self.alpha, inplace=self.inplace)
31
32 class SELU(nn.Module):
33     def __init__(self, inplace=True):
34         super(SELU, self).__init__()
35         self.inplace = inplace
36
37     def forward(self, x):
38         return nn.functional.selu(x, inplace=self.inplace)

```

这样就完成四种激活函数加入选择。

#### 4.4.6 学习率调度策略添加

原文只实现了 OneCycleLR，此处加入 ExponentialLR 和 CyclicLR 两种学习率调度策略，供后人探索更优解。ExponentialLR 是指学习率按照指数函数下降。在每个 epoch 之后，学习率都会乘以一个小于 1 的因子。CyclicLR 是指在一定范围内逐渐增加学习率，然后再逐渐减小学习率。这种周期性的学习率调整可以帮助模型跳出局部最优点，探索更广阔的参数空间。

首先在 src\args.py 中，加入学习率调度策略参数'lr\_scheduler'。如图 54。

```

1 self.add_argument('--lr_scheduler', default='OneCycleLR',
2 choices=['OneCycleLR', 'ExponentialLR', 'CyclicLR'],
3 help='Learning rate scheduling strategy')

```

```

189 self.add_argument('--lr_scheduler', default='OneCycleLR',
190 choices=['OneCycleLR', 'ExponentialLR', 'CyclicLR'],
191 help='Learning rate scheduling strategy')

```

图 54. 加入学习率调度策略选择

现在在 train.py 主函数中加入判断，如图 55。

```

1 if args.lr_scheduler == 'ExponentialLR':
2     lr_scheduler = torch.optim.lr_scheduler. \
3         ExponentialLR(optimizer, gamma=0.95)
4 elif args.lr_scheduler == 'CyclicLR':
5     lr_scheduler = torch.optim.lr_scheduler. \
6         CyclicLR(optimizer, base_lr=0.001,
7         max_lr=0.1, step_size_up=50, mode='triangular2')

```

```

8     else:
9         lr_scheduler = OneCycleLR(
10            optimizer,
11            max_lr=[i['lr'] for i in optimizer.param_groups],
12            total_steps=args.epochs,
13            div_factor=25,
14            pct_start=0.1,
15            anneal_strategy='cos',
16            final_div_factor=1e4
17        )

```

```

238     if args.lr_scheduler == 'ExponentialLR':
239         lr_scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.95)
240     elif args.lr_scheduler == 'CyclicLR':
241         lr_scheduler = torch.optim.lr_scheduler.CyclicLR(optimizer,
242             base_lr=0.001, max_lr=0.1, step_size_up=50, mode='triangular2')
243     else:
244         lr_scheduler = OneCycleLR(
245            optimizer,
246            max_lr=[i['lr'] for i in optimizer.param_groups],
247            total_steps=args.epochs,
248            div_factor=25,
249            pct_start=0.1,
250            anneal_strategy='cos',
251            final_div_factor=1e4
252        )

```

图 55. 加入学习率调度策略到主函数

现在只需在参数选择'ExponentialLR' 或者'CyclicLR' 即可使用这两种学习率调度策略。注意到 OneCycleLR 有参数 anneal\_strategy 退火策略，将该值修改为'linear' 可以选择线性退火策略，后续实验也进行产生，这里不再赘述修改策略。

## 5 实验结果分析

### 5.1 数据集说明

本文在两个常用的 RGB-D 室内数据集 SUNRGB-D [19] 和 NYUv2 [18] 上评估了其方法。为了证明方法也适用于其他领域的应用，还在城市景观数据集 Cityscapes [5] 上展示了结果，这是最广泛用于语义分割的户外数据集。

NYUv2 和 SUNRGB-D: NYUv2 包含 1,449 张室内 RGB-D 图像，其中 795 张用于训练，654 张用于测试，并且使用了常见的 40 类标签设置。SUNRGB-D 有 37 个类，包括 10,335 张室内 RGB-D 图像，包括 NYUv2 的所有图像。有 5285 张训练图像和 5050 张测试图像。本文消融研究是基于 NYUv2，因为它更小，因此，可以导致更快的训练。然而，根据 [1] 的说法，在一个子集上的训练就足以得到一个可靠的模型选择。对于这两个数据集，本文使用了  $640 \times 480$  的网络输入分辨率。

为了证明本文方法也适用于其他领域，如户外环境，进一步提出了对城市景观数据集的评估。首先关注  $1024 \times 512$  的较小分辨率，因为它通常用于高效分割。此外，由于大多数方法只依赖于 RGB 作为输入，这里也比较本文方法的单模态 RGB 版本。

## 5.2 复现原文结果

首先在这里给出原文对于三个数据集的结果。对于室内数据集 NYUv2 和 SUNRGB-D，原文表示，FPS 由 NVIDIA Jetson AGX Xavier 给出，这里后续不对 FPS 进行复现，如图 56。

Method	Backbone	NYUv2	SUN-RGB-D	FPS
FuseNet [9]	$2 \times$ VGG16	-	37.29	†
RedNet [10]	$2 \times$ R34	-	46.8	26.0
SSMA [24]	$2 \times$ mod. R50	-	44.43	12.4
MMAF-Net [25]	$2 \times$ R50	-	45.5	N/A
RedNet [10]	$2 \times$ R50	-	47.8	22.1
RDFNet [23]	$2 \times$ R50	47.7*	-	7.2
ACNet [11]	$3 \times$ R50	48.3	48.1	16.5
SA-Gate [22]	$2 \times$ R50	50.4	49.4*	11.9
SGNet [19]	R101	49.0	47.1	N/A ▽
Idempotent [21]	$2 \times$ R101	49.9	47.6	N/A ▽
2.5D Conv [16]	R101	48.5	48.2	N/A ▽
MMAF-Net [25]	$2 \times$ R152	44.8	47.0	N/A ▽
RDFNet [23]	$2 \times$ R152	50.1*	47.7*	5.8
ESANet-R18	$2 \times$ R18	47.32	46.24	34.7
ESANet-R18-NBt1D	$2 \times$ R18 NBt1D	48.17	46.85	36.3
ESANet-R34	$2 \times$ R34	48.81	47.08	27.5
<b>ESANet-R34-NBt1D</b>	$2 \times$ R34 NBt1D	50.30	48.17	29.7
ESANet-R50	$2 \times$ R50	50.53	48.31	22.6
ESANet (pre. SceneNet)	$2 \times$ R34 NBt1D	51.58	48.04	29.7

TABLE I: Mean intersection over union of our ESANet compared to state-of-the-art methods on NYUv2 and SUNRGB-D test set ordered by SUNRGB-D performance and backbone complexity. FPS is reported for NVIDIA Jetson AGX Xavier (Jetpack 4.4, TensorRT 7.1, Float16). Legend: R: ResNet, \*: additional test-time augmentation, i.e., flipping or multi-scale (not timed), N/A: no implementation available, †: includes operations, which are not supported by TensorRT, and ▽: expected to be slower due to complex backbone.

图 56. NYUv2 和 SUNRGB-D 原文结果

对于城市景观数据集 Cityscapes，原文表示，FPS 由 NVIDIA Jetson AGX Xavier 给出，而 test 测试结果由测试服务器给出，因此这里后续不对 FPS 和 test 数据集进行复现，如图 57。

Method	1024×512			2048×1024		
	Val	Test <sup>◦</sup>	FPS	Val	Test <sup>◦</sup>	FPS
RGB	ERFNet [26]	-	69.7	49.9	-	-
	LEDNet [27]	-	70.6*	38.5	-	-
	ESPNetv2 [32]	66.4	66.2	47.4	-	-
	SwiftNet [30]	70.2	-	64.5	75.4	75.5
	BiSeNet [31]	-	-	-	74.8	74.7
	PSPNet [33]	-	-	-	-	20.8
	DeepLabv3 [37]	-	-	-	79.3	81.3*
RGB-D	ESANet-R18-NBt1D	71.48	-	37.2	77.95	-
	<b>ESANet-R34-NBt1D</b>	72.70	72.87	32.3	78.47	77.56
	ESANet-R50	73.88	-	24.9	79.23	-
	SSMA [24]	-	-	-	82.19*	82.31*
	SA-Gate [22]	-	-	-	81.7	82.8
	LDFNet [44]	68.48	71.3	25.3	-	-
	ESANet-R18-NBt1D	74.65	-	28.9	79.25	-
	<b>ESANet-R34-NBt1D</b>	75.22	75.65	23.4	80.09	78.42
	ESANet-R50	75.66	-	16.9	79.97	-
						4.0

TABLE II: Mean intersection over union of our ESANet on Cityscapes for both common input resolutions compared to state-of-the-art methods. FPS is reported for NVIDIA Jetson AGX Xavier (Jetpack 4.4, TensorRT 7.1, Float16). Legend: ◦: test server result, \*: trained with additional coarse data.

图 57. Cityscapes 原文结果

现在，给出我们本地运行的结果。首先是我们本地默认参数下运行的 NYUv2 数据集。按照 Readme.md 下运行结果，可以看出，大部分与原文均很接近，只有 1 个点内差距，只有 R18-NBt1D 差距较大。如图 58。

Method	lr	weight	height	result	epoch	训练时间
R18	0.01	640	480	47.07 ↓0.25	474	15065s
R18-NBt1D	0.01	640	480	39.96 ↓8.21	492	15588s
R34	0.01	640	480	48.20 ↓0.61	474	15739s
R34-NBt1D	0.01	640	480	49.38 ↓0.92	211	17516s
R50	0.01	640	480	50.27 ↓0.26	421	18058s

图 58. 默认参数下 NYUV2 结果

现在是 SUNRGB-D 数据集，按照 Readme.md 下运行结果，可以看出，均与原文也很接近，也只有 R18-NBt1D 差距较大。如图 59。

Method	lr	weight	height	result	epoch	训练时间
R18	0.01	640	480	45.60 ↓0.64	345	101520s
R18-NBt1D	0.01	640	480	43.59 ↓3.26	416	102309s
R34	0.01	640	480	46.80 ↓0.28	378	98142s
R34-NBt1D	0.01	640	480	47.65 ↓0.52	394	107965s
R50	0.01	640	480	48.16 ↓0.15	494	111489s

图 59. 默认参数下 SUNRGB-D 结果

最后是 Cityscapes 数据集，按照 Readme.md 下运行结果，可以看出，RGB 部分结果可能会比原文要好，这可能是原文并没有注重 RGB 调参导致的，而 RGB-D 均与原文也很接近，也只有 R18-NBt1D 差距较大。如图 60。

	Method	1024*512			2048*1024			训练时间
		Val	epoch	FPS	Val	epoch	FPS	
RGB	ESANet-R18-NBt1D	67 ↓4.48	497	-	73.9 ↓4.05	464	-	117268
	ESANet-R34-NBt1D	72.59 ↓0.11	446	-	78.94 ↑0.47	450	-	117505
	ESANet-R50	74.16 ↑0.28	418	-	79.3 ↑0.07	417	-	118053
RGB-D	ESANet-R18-NBt1D	69.9 ↓4.75	482	-	76.03 ↓3.22	463	-	117485
	ESANet-R34-NBt1D	74.22 ↓1.00	441	-	79.18 ↓0.91	413	-	117160
	ESANet-R50	75.31 ↓0.35	431	-	79.61 ↓0.37	492	-	117562

图 60. 默认参数下 Cityscapes 结果

### 5.3 加入创新部分测试结果

这里一开始对 SUNRGB-D 尝试调了学习率和长宽，但是效果并不好，反而下降了，如图 61。

Method	lr	weight	height	result	epoch	训练时间
R34-NBt1D	0.0001	512	1024	40.61 ↓7.56	425	146160s
R34-NBt1D	0.01	512	1024	46.12 ↓0.96	327	135940s

图 61. 修改 SUNRGB-D 后结果

由于 Cityscapes 和 SUNRGB-D 训练时间过长，后续不再对其进行调参。根据 [1] 的说法，在一个子集上的训练就足以得到一个可靠的模型选择，因此后续只需在 NYUv2 加入结果寻找改进即可判断是否有效。

首先尝试 0.0001 学习率上，效果会不会变好，如图 62。实际情况是效果均有不同跨度退步。

Method	lr	weight	height	result	epoch	训练时间
R18	0.0001	640	480	28.68	468	15471s
R18-NBt1D	0.0001	640	480	28.6	468	14766s
R34-NBt1D	0.0001	640	480	39.46	374	16621s
R50	0.0001	640	480	40.95	476	17133s

图 62. 修改学习率 lr=0.0001 的结果

尝试修改分辨率从 640\*480 到 1024\*512，结果发现仍是会不同跨度退步，并且训练时间更长。如图 63。

Method	lr	weight	height	result	epoch	训练时间
R34-NBt1D	0.0001	1024	512	35.88	368	23926s
R34-NBt1D	0.0001	640	480	39.81	398	17114s

图 63. 修改分辨率为 1024\*512 的结果

加入自动混合精度计算，结果发现会具有一定精度损失，但是训练时间更短，所需内存更少，可以考虑加入混合精度计算。如图 64。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.0001	640	480	39.81	398	17114s	
R34-NBt1D	0.0001	640	480	39.46	374	16621s	加入自动混合精度训练

图 64. 加入自动混合精度的结果

加入提前停止后，考虑到 R18、R18-NBt1D 和 R34 三者模型在 450 轮以后才找到最优结果，有可能在 500 轮还欠拟合，从 500 轮开始继续对三个模型继续训练 500 轮，但是效果并没有优化，在 599 轮提前结束。

加入我们的几个优化器一起对比结果，按照经验，调整对应的学习率，得到结果如图 65。可以看到 AdamW 在精度上并不是差距太大，但是训练时间大大缩小，如果后续对这方面有要求，可以考虑改用 AdamW。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.01	640	480	49.38	211	17516s	optimizer=SGD
R34-NBt1D	0.0001	640	480	49.26	406	18451s	optimizer=Adam
R34-NBt1D	0.0001	640	480	49.12	326	14601s	optimizer=AdamW
R34-NBt1D	0.0001	640	480	46.24	417	18594s	optimizer=RMSprop

图 65. 选择不同优化器的结果

猜测分辨率从 1024\*512 到 640\*480，效果更好，如果变得更小会不会越小越好，因此将分辨率降为 224\*224，查看结果变化，结果发现并不是越小越好，如图 66。

Method	lr	weight	height	result	epoch	训练时间
R34-NBt1D	0.01	640	480	49.38	211	17516s
R34-NBt1D	0.01	224	224	25.28	478	13416s

图 66. 选择更小分辨率的结果

修改原文的余弦退火策略为线性退火策略，查看是否有改进，结果发现原文的更佳，如图 67。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.01	640	480	49.38	211	17516s	anneal_strategy=cos
R34-NBt1D	0.01	640	480	48.77	244	10364s	anneal_strategy=linear

图 67. 选择线性退火策略的结果

修改原文的学习率调度策略 OneCycleLR，查看是否有改进，结果发现原文的更佳，如图 68。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.01	640	480	49.38	211	17516s	OneCycleLR
R34-NBt1D	0.01	640	480	45.56	230	9218s	ExponentialLR
R34-NBt1D	0.01	640	480	48.12	317	13257s	CyclicLR

图 68. 选择不同的学习率调度的结果

加入梯度裁剪，查看是否有改进，结果发现并无改进，但是梯度裁剪可以防止过拟合这一优点，仍然可以考虑后续加入。如图 69。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.01	640	480	49.38	211	17516s	
R34-NBt1D	0.01	640	480	49.31	429	16044s	梯度裁剪

图 69. 加入梯度裁剪的结果

尝试多种激活函数，查看是否有改进，结果发现并无改进，但是不同的激活函数后面调整网络结果也可以考虑后续加入优化。如图 70。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.01	640	480	49.38	211	17516s	ReLU
R34-NBt1D	0.01	640	480	49.08	224	9212s	LeakyReLU
R34-NBt1D	0.01	640	480	38.43	484	17163s	PReLU
R34-NBt1D	0.01	640	480	42.98	425	15848s	ELU
R34-NBt1D	0.01	640	480	39.33	404	15288s	SELU

图 70. 修改不同激活函数后的结果

尝试使用两个 GPU，观察是否训练过程加快，或者精度提升，但是结果并没有，但是使用的内存减少了，如果后续需要在 ESANet 基础上训练更大的网络，可以考虑加入。如图 71。

Method	lr	weight	height	result	epoch	训练时间	改进措施
R34-NBt1D	0.01	640	480	49.38	211	17516s	GPU=1
R34-NBt1D	0.01	640	480	48.12	317	22705s	GPU=2

图 71. 修改为多 GPU 的结果

现在尝试 batch\_size 和 worker，观察精度是否有所提升。精度并没有任何改进，如图 72。

Method	lr	weight	height	result	epoch	训练时间	batch_size	worker
R34-NBt1D	0.01	640	480	49.38	211	17516s	8	8
R34-NBt1D	0.01	640	480	49.03	387	15802s	16	8
R34-NBt1D	0.01	640	480	48.4	217	9345s	32	8
R34-NBt1D	0.01	640	480	49.33	382	16365s	8	16

图 72. 修改为 batch\_size 和 worker 的结果

考虑到 R18-NBt1D 与原文差距较大，并且都在 450 轮以上才得到结果，考虑调整学习率，观察结果，仍没有改进，如图 73。

Method	lr	weight	height	result	epoch	训练时间
R18-NBt1D	0.01	640	480	39.96	492	15588s
R18-NBt1D	0.005	640	480	38.82	389	13839s
R18-NBt1D	0.05	640	480	35.22	569	15586s

图 73. 修改 R18-NBt1D 学习率的结果

## 6 总结与展望

文档对 ESANet 代码进行复现，虽然代码并不能在一开始就能运行，但是在最终调整运行后，结果与论文接近。本文在原文基础上，另外加入了自动混合训练、提前停止、梯度裁剪等尝试，并且在原文基础上添加 AdamW、RMSprop 优化器，以及线性退火策略、周期性学习率尝试，激活函数上加入原文未有的 LeakyReLU、PReLU、ELU 和 SELU 多个激活函数选择。虽然结果上并没有超越原文精度，但是提供了很多新功能给后人探索。实现不足是对于 R18-NBt1D 结果，没有办法做到与原文相近，未来可以考虑优化该结果。

基于本文提出的方法和结果，未来可以考虑网络架构的进一步优化，作者已经在几个室内数据集上实现了最先进的性能，但可能还有进一步优化网络架构的空间，以提高分割的准确性和速度。也可以考虑与其他机器人感知任务的集成，语义分割只是移动机器人需要执行的众多感知任务之一。可以观察这种方法与其他任务（如对象检测、跟踪和映射）的集成情况如何。

## 参考文献

- [1] Jorg Bornschein, Francesco Visin, and Simon Osindero. Small data, big decisions: Model selection in the small-data regime. In *International conference on machine learning*, pages 1035–1044. PMLR, 2020.
- [2] Lin-Zhuo Chen, Zheng Lin, Ziqin Wang, Yong-Liang Yang, and Ming-Ming Cheng. Spatial information guided convolution for real-time rgbd semantic segmentation. *IEEE Transactions on Image Processing*, 30:2313–2324, 2021.
- [3] Xiaokang Chen, Kwan-Yee Lin, Jingbo Wang, Wayne Wu, Chen Qian, Hongsheng Li, and Gang Zeng. Bi-directional cross-modality feature propagation with separation-and-aggregation gate for rgb-d semantic segmentation. In *European Conference on Computer Vision*, pages 561–577. Springer, 2020.
- [4] Yunlu Chen, Thomas Mensink, and Efstratios Gavves. 3d neighborhood convolution: Learning depth-aware features for rgb-d and rgb semantic segmentation. In *2019 International Conference on 3D Vision (3DV)*, pages 173–182. IEEE, 2019.
- [5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.
- [6] F Fooladgar and S Kasaei. Multi-modal attention-based fusion model for semantic segmentation of rgb-depth images. arxiv 2019. *arXiv preprint arXiv:1912.11691*.
- [7] Caner Hazirbas, Lingni Ma, Csaba Domokos, and Daniel Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Computer Vision–*

- [8] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [9] Xinxin Hu, Kailun Yang, Lei Fei, and Kaiwei Wang. Acnet: Attention based network to exploit complementary features for rgbd semantic segmentation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1440–1444. IEEE, 2019.
- [10] J Jiang, L Zheng, F Luo, and Z Zhang. Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation. arxiv 2018. *arXiv preprint arXiv:1806.01054*.
- [11] G Li, I Yun, J Kim, and J Kim. Dabnet: Depth-wise asymmetric bottleneck for real-time semantic segmentation. arxiv 2019. *arXiv preprint arXiv:1907.11357*.
- [12] Shao-Yuan Lo, Hsueh-Ming Hang, Sheng-Wei Chan, and Jing-Jhih Lin. Efficient dense modules of asymmetric convolution for real-time semantic segmentation. In *Proceedings of the ACM multimedia Asia*, pages 1–6. 2019.
- [13] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. 2018.
- [14] Marin Orsic, Ivan Kreso, Petra Bevandic, and Sinisa Segvic. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12607–12616, 2019.
- [15] Seong-Jin Park, Ki-Sang Hong, and Seungyong Lee. Rdfnet: Rgb-d multi-level residual feature fusion for indoor semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 4980–4989, 2017.
- [16] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2017.
- [17] Daniel Seichter, Mona Köhler, Benjamin Lewandowski, Tim Wengefeld, and Horst-Michael Gross. Efficient rgb-d semantic segmentation for indoor scene analysis. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 13525–13531. IEEE, 2021.
- [18] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7–13, 2012, Proceedings, Part V* 12, pages 746–760. Springer, 2012.

- [19] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.
- [20] Tijmen Tieleman and Geoffrey Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn*, 17, 2012.
- [21] Abhinav Valada, Rohit Mohan, and Wolfram Burgard. Self-supervised model adaptation for multimodal semantic segmentation. *International Journal of Computer Vision*, 128(5):1239–1285, 2020.
- [22] Weiyue Wang and Ulrich Neumann. Depth-aware cnn for rgb-d segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 135–150, 2018.
- [23] Yu Wang, Quan Zhou, Jia Liu, Jian Xiong, Guangwei Gao, Xiaofu Wu, and Longin Jan Latecki. Lednet: A lightweight encoder-decoder network for real-time semantic segmentation. In *2019 IEEE international conference on image processing (ICIP)*, pages 1860–1864. IEEE, 2019.
- [24] Yajie Xing, Jingbo Wang, Xiaokang Chen, and Gang Zeng. 2.5 d convolution for rgb-d semantic segmentation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1410–1414. IEEE, 2019.
- [25] Yajie Xing, Jingbo Wang, Xiaokang Chen, and Gang Zeng. Coupling two-stream rgb-d semantic segmentation network by idempotent mappings. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1850–1854. IEEE, 2019.
- [26] Yajie Xing, Jingbo Wang, and Gang Zeng. Malleable 2.5 d convolution: Learning receptive fields along the depth-axis for rgb-d scene parsing. In *European Conference on Computer Vision*, pages 555–571. Springer, 2020.
- [27] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 325–341, 2018.
- [28] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.
- [29] Yiran Zhong, Yuchao Dai, and Hongdong Li. 3d geometry-aware semantic labeling of outdoor street scenes. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2343–2349. IEEE, 2018.