基于 SegNet 的街景分割实验指导书

一、 实验目的

1. 掌握深度学习在计算机视觉领域的应用，熟悉深度学习基础知识，包括卷积神经网络和图像分割技术。
2. 通过实践，了解 SegNet 模型的基本原理，掌握模型训练、验证和测试的流程，以及如何评估模型在街景分割任务上的性能。

二、 实验要求

1. 利用 Python 语言和深度学习框架（本实验指导书以 Pytorch 为例）构造简单的街景分割模型，以实现街景分割任务。
2. 提供评估指标的数值，包括像素准确率，平均像素准确率，平均交并比等，本实验对指标数值不做要求。（参考文献 https://arxiv.org/pdf/1511.00561）
3. 如果选择做此实验作业，按规定时间在课程网站提交实验报告、代码以及 PPT。

三、 实验原理

1. 模型结构

SegNet 是一种用于图像分割的深度卷积神经网络。它通过对输入图像进行像素级别的分类，将图像分割为不同的类别。SegNet 模型主要由编码器（Encoder）和解码器（Decoder）两部分组成，编码器负责提取图像特征，解码器负责将特征映射回原始图像尺寸并进行分类。 如图 1 所示。

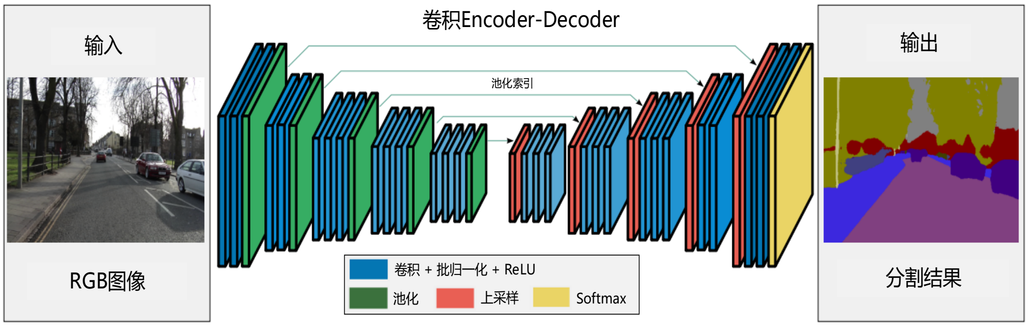


图 1 SegNet 网络结构

2. 模型输入

SegNet 的输入是一张待分割的图像，通常是彩色图像。输入图像的尺寸可以根据具体任务和数据集而定，但通常会经过预处理，如缩放、裁剪和归一化，以满足模型的输入要求。

3. 模型输出

SegNet 的输出是对输入图像的像素级别的分类结果，即对每个像素点进行分类，将图像分割为不同的类别。输出通常是一个与输入图像尺寸相同的矩阵，每个像素值表示该像素所属的类别。

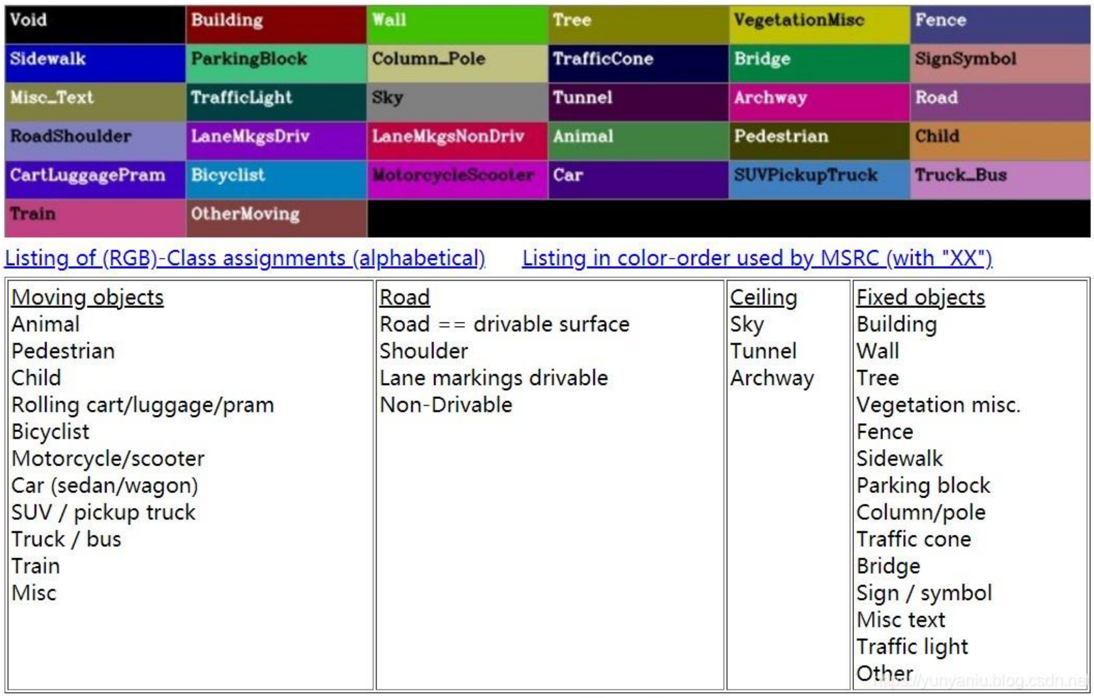
具体地，SegNet 的解码器输出的是一个与输入图像相同大小的矩阵，其中每个像素对应一个类别。这个矩阵可以看作是对输入图像的分割结果，每个像素值表示该像素所属的类别，如道路、建筑物、汽车等。

四、 实验所用工具以及数据集

本实验基于 SegNet 进行街景分割任务。使用的数据集是 Cambridgedriving Labeled Video Database (CamVid)。数据集下载地址：http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

CamVid 是一个常用的用于语义分割的数据集，特别是在自动驾驶和计算机视觉领域。该数据集包含来自驾驶视频的图像和相应的像素级标签，用于将图像

中的每个像素分类为不同的类别，如道路、行人、汽车、建筑物等 32 个不同的类别，标签使用颜色编码，每种颜色代表一个类别。该数据集包含数百个来自驾驶场景的图像，分辨率为 960x720 像素。这些图像涵盖了不同的天气条件、场景和路面情况。每个图像都有相应的像素级标签，用于指示每个像素的类别，如道路、行人、汽车等。



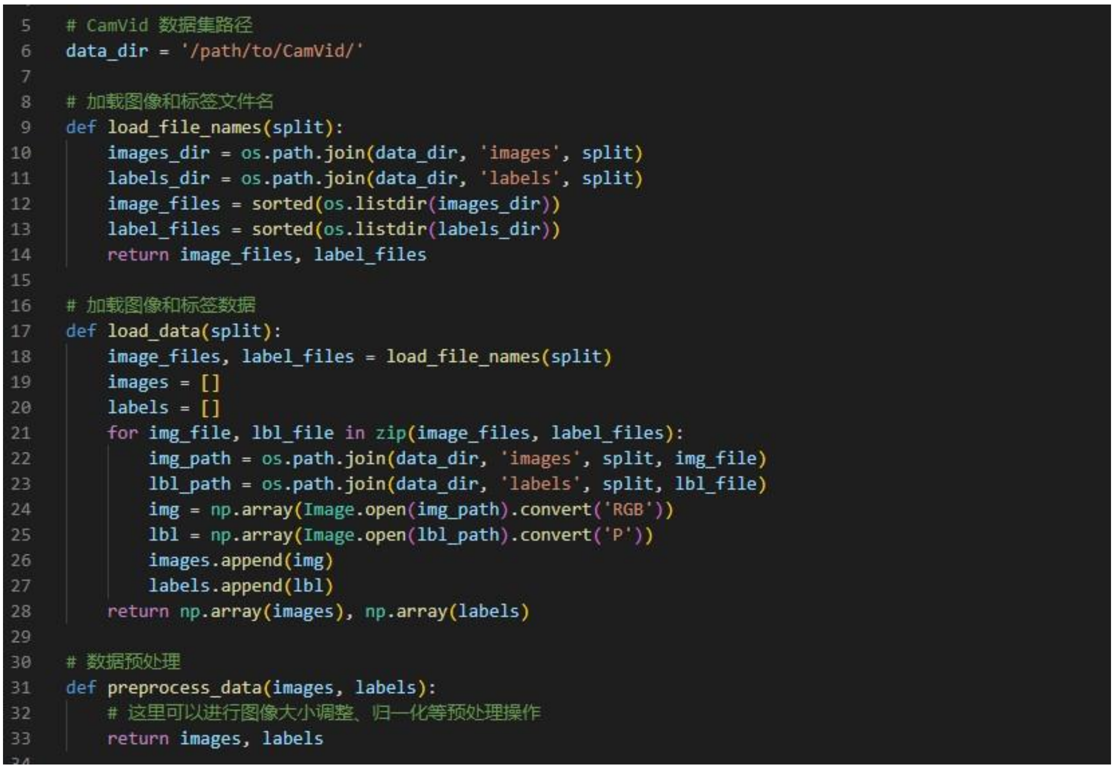
Void	Building	Wall	Tree	VegetationMisc	Fence
Sidewalk	ParkingBlock	Column_Pole	TrafficCone	Bridge	SignSymbol
Misc_Text	TrafficLight	Sky	Tunnel	Archway	Road
RoadShoulder	LaneMkgsDriv	LaneMkgsNonDriv	Animal	Pedestrian	Child
CartLuggagePram	Bicyclist	MotorcycleScooter	Car	SUVPickupTruck	Truck_Bus
Train	OtherMoving				

Listing of (RGB)-Class assignments (alphabetical)   Listing in color-order used by MSRC (with "XX")

Moving objects	Road	Ceiling	Fixed objects
Animal Pedestrian Child Rolling cart/luggage/pram Bicyclist Motorcycle/scooter Car (sedan/wagon) SUV / pickup truck Truck / bus Train Misc	Road == drivable surface Shoulder Lane markings drivable Non-Drivable	Sky Tunnel Archway	Building Wall Tree Vegetation misc. Fence Sidewalk Parking block Column/pole Traffic cone Bridge Sign / symbol Misc text Traffic light Other

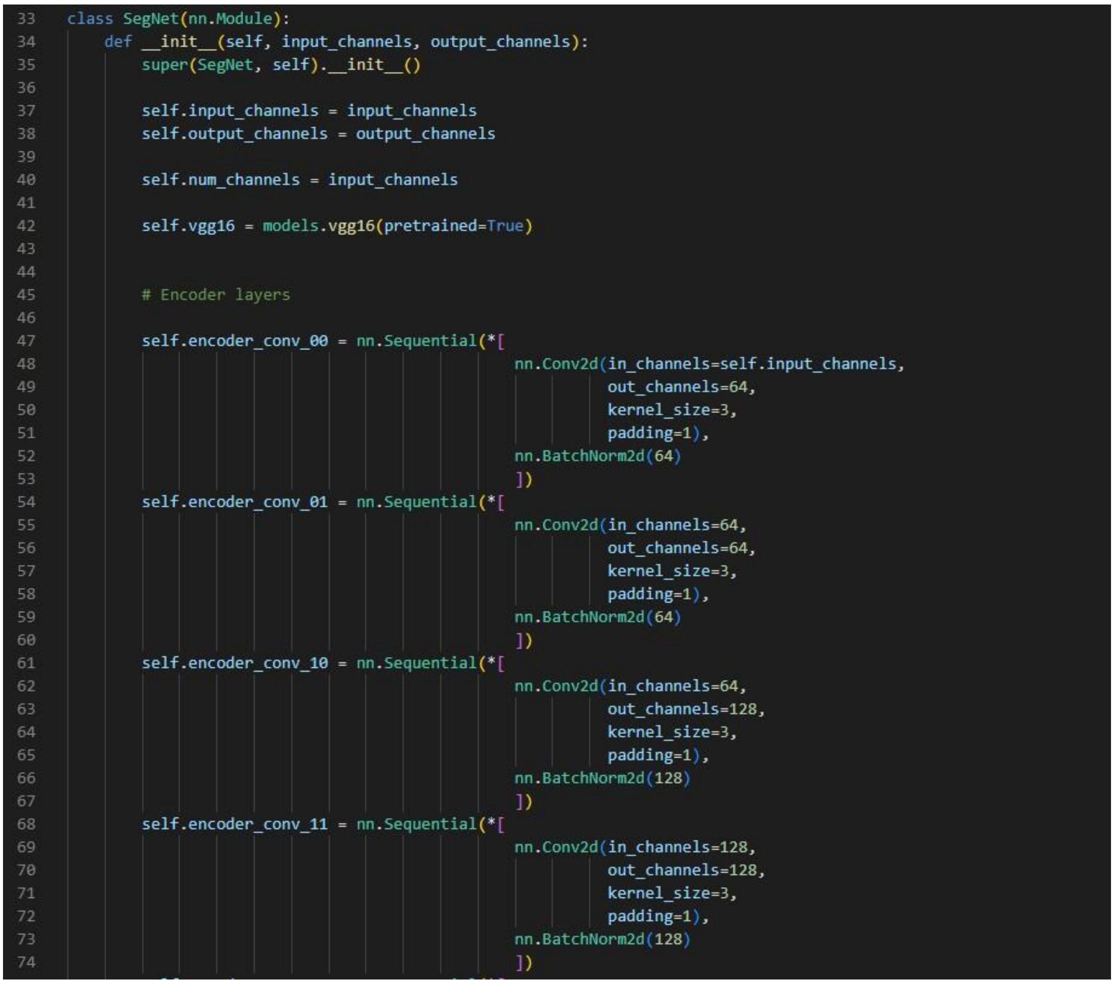
五、 实验步骤和方法

1. 数据加载和处理

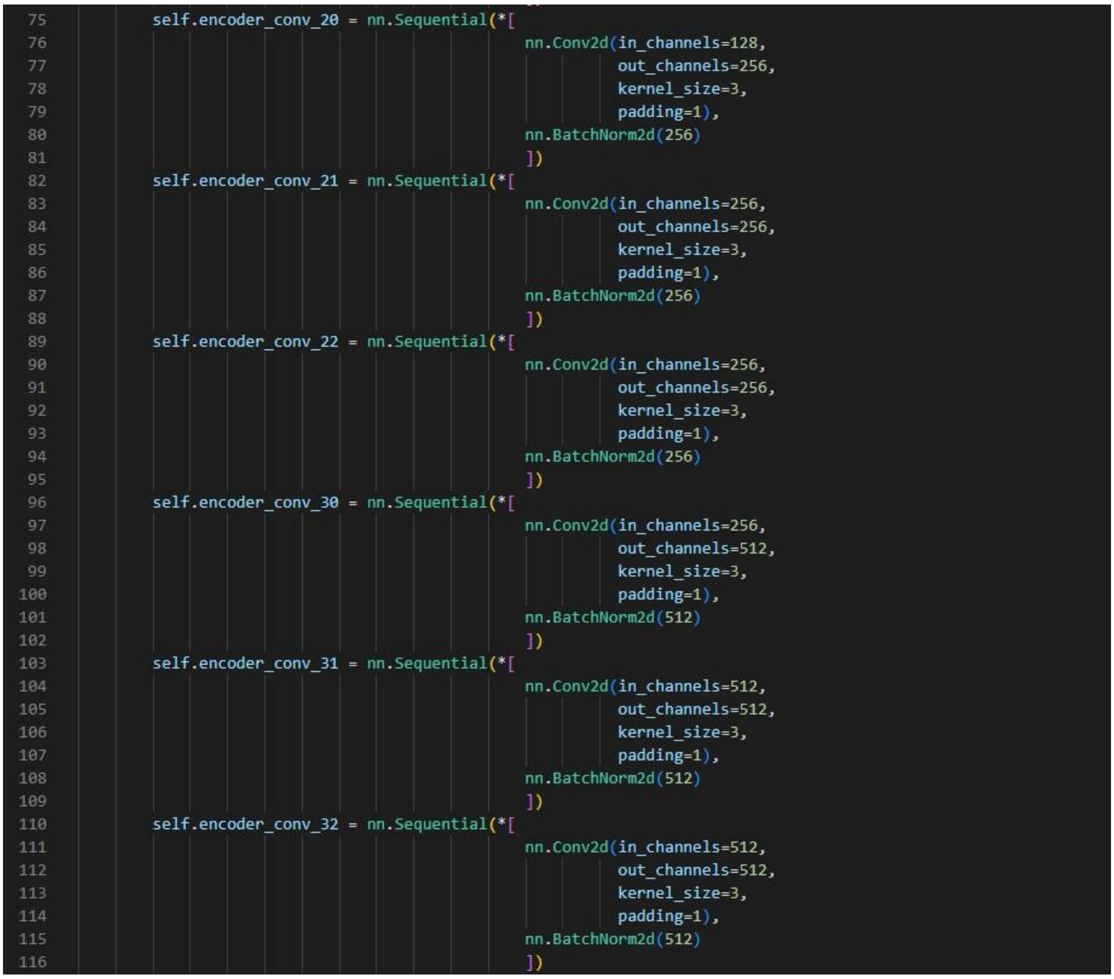


```python
# CamVid 数据集路径
data_dir = '/path/to/CamVid/'

# 加载图像和标签文件名
def load_file_names(split):
    images_dir = os.path.join(data_dir, 'images', split)
    labels_dir = os.path.join(data_dir, 'labels', split)
    image_files = sorted(os.listdir(images_dir))
    label_files = sorted(os.listdir(labels_dir))
    return image_files, label_files

# 加载图像和标签数据
def load_data(split):
    image_files, label_files = load_file_names(split)
    images = []
    labels = []
    for img_file, lbl_file in zip(image_files, label_files):
        img_path = os.path.join(data_dir, 'images', split, img_file)
        lbl_path = os.path.join(data_dir, 'labels', split, lbl_file)
        img = np.array(Image.open(img_path).convert('RGB'))
        lbl = np.array(Image.open(lbl_path).convert('P'))
        images.append(img)
        labels.append(lbl)
    return np.array(images), np.array(labels)

# 数据预处理
def preprocess_data(images, labels):
    # 这里可以进行图像大小调整、归一化等预处理操作
    return images, labels
```

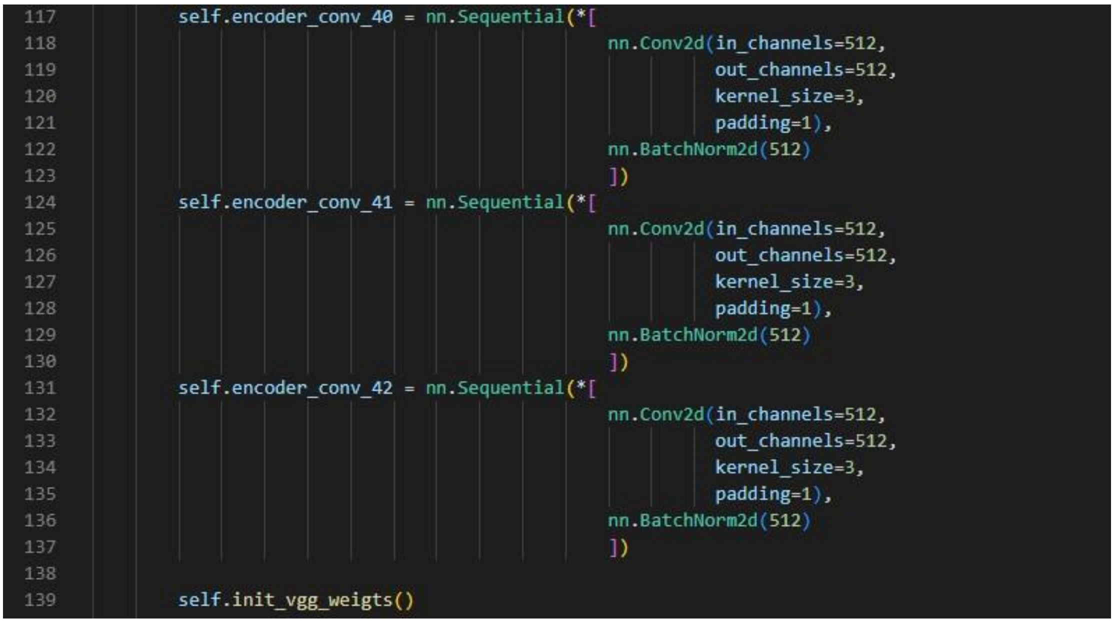
2. 模型构建



```python
class SegNet(nn.Module):
    def __init__(self, input_channels, output_channels):
        super(SegNet, self).__init__()

        self.input_channels = input_channels
        self.output_channels = output_channels

        self.num_channels = input_channels

        self.vgg16 = models.vgg16(pretrained=True)


        # Encoder layers

        self.encoder_conv_00 = nn.Sequential(*[
                                                nn.Conv2d(in_channels=self.input_channels,
                                                          out_channels=64,
                                                          kernel_size=3,
                                                          padding=1),
                                                nn.BatchNorm2d(64)
                                                ])
        self.encoder_conv_01 = nn.Sequential(*[
                                                nn.Conv2d(in_channels=64,
                                                          out_channels=64,
                                                          kernel_size=3,
                                                          padding=1),
                                                nn.BatchNorm2d(64)
                                                ])
        self.encoder_conv_10 = nn.Sequential(*[
                                                nn.Conv2d(in_channels=64,
                                                          out_channels=128,
                                                          kernel_size=3,
                                                          padding=1),
                                                nn.BatchNorm2d(128)
                                                ])
        self.encoder_conv_11 = nn.Sequential(*[
                                                nn.Conv2d(in_channels=128,
                                                          out_channels=128,
                                                          kernel_size=3,
                                                          padding=1),
                                                nn.BatchNorm2d(128)
                                                ])
```

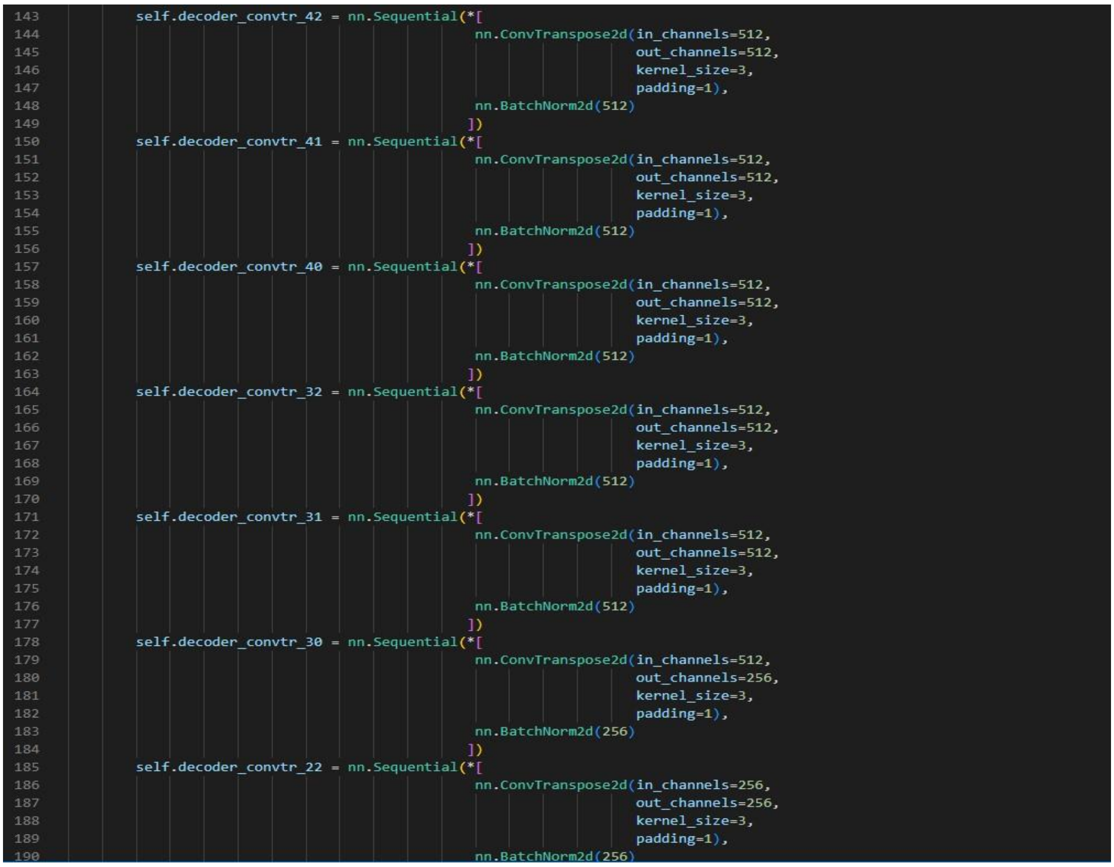


```
        self.encoder_conv_20 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=128,
                                                  out_channels=256,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(256)
                                        ])
        self.encoder_conv_21 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=256,
                                                  out_channels=256,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(256)
                                        ])
        self.encoder_conv_22 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=256,
                                                  out_channels=256,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(256)
                                        ])
        self.encoder_conv_30 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=256,
                                                  out_channels=512,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(512)
                                        ])
        self.encoder_conv_31 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=512,
                                                  out_channels=512,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(512)
                                        ])
        self.encoder_conv_32 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=512,
                                                  out_channels=512,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(512)
                                        ])
```

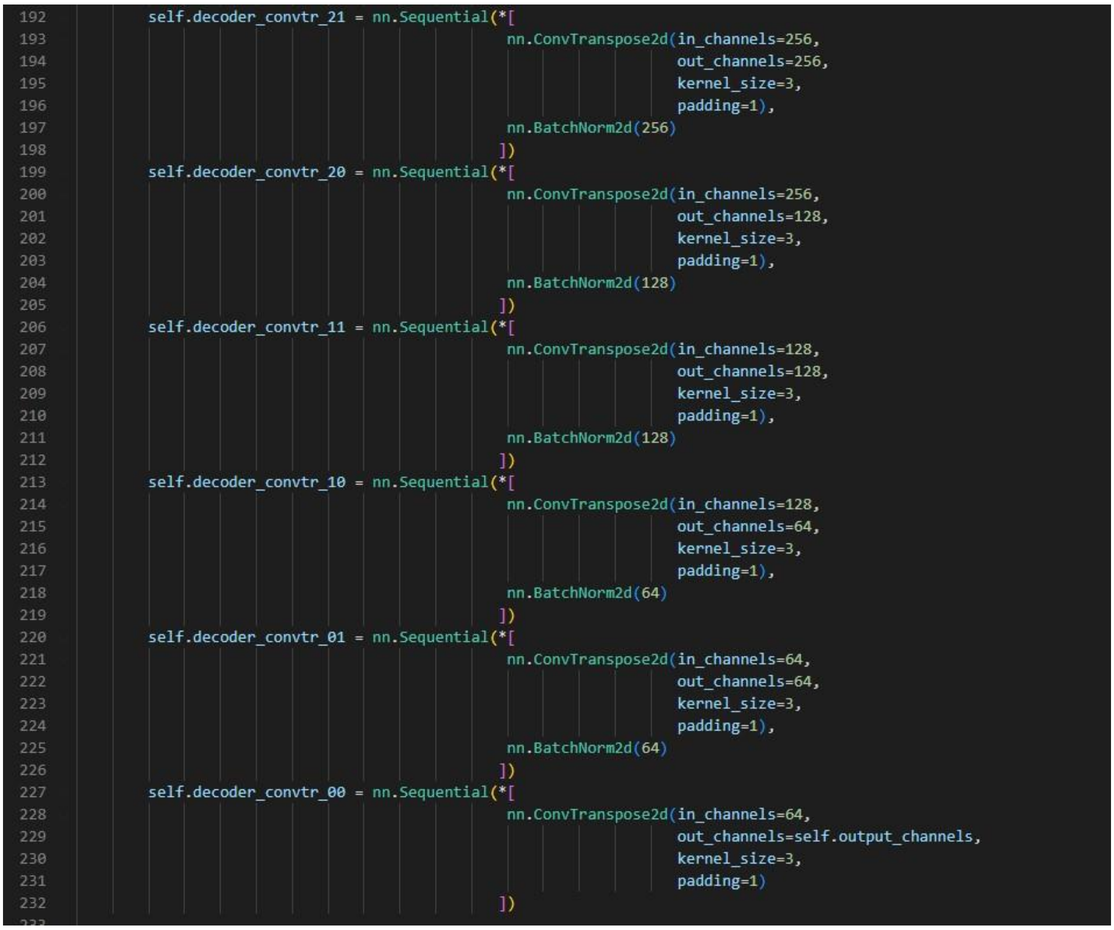


```
        self.encoder_conv_40 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=512,
                                                  out_channels=512,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(512)
                                        ])
        self.encoder_conv_41 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=512,
                                                  out_channels=512,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(512)
                                        ])
        self.encoder_conv_42 = nn.Sequential(*[
                                        nn.Conv2d(in_channels=512,
                                                  out_channels=512,
                                                  kernel_size=3,
                                                  padding=1),
                                        nn.BatchNorm2d(512)
                                        ])

        self.init_vgg_weigts()
```

上述为 encoder 定义，接下来是 decoder

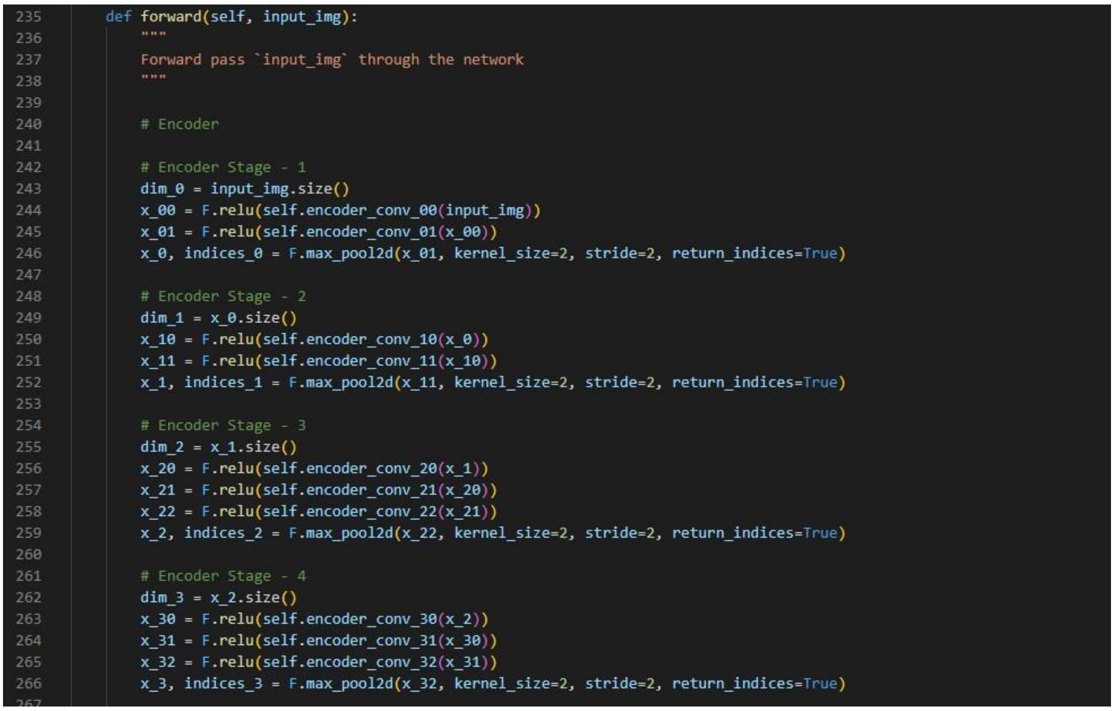


```
        self.decoder_convtr_42 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=512,
                                                                   out_channels=512,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(512)
                                               ])
        self.decoder_convtr_41 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=512,
                                                                   out_channels=512,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(512)
                                               ])
        self.decoder_convtr_40 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=512,
                                                                   out_channels=512,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(512)
                                               ])
        self.decoder_convtr_32 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=512,
                                                                   out_channels=512,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(512)
                                               ])
        self.decoder_convtr_31 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=512,
                                                                   out_channels=512,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(512)
                                               ])
        self.decoder_convtr_30 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=512,
                                                                   out_channels=256,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(256)
                                               ])
        self.decoder_convtr_22 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=256,
                                                                   out_channels=256,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(256)
```

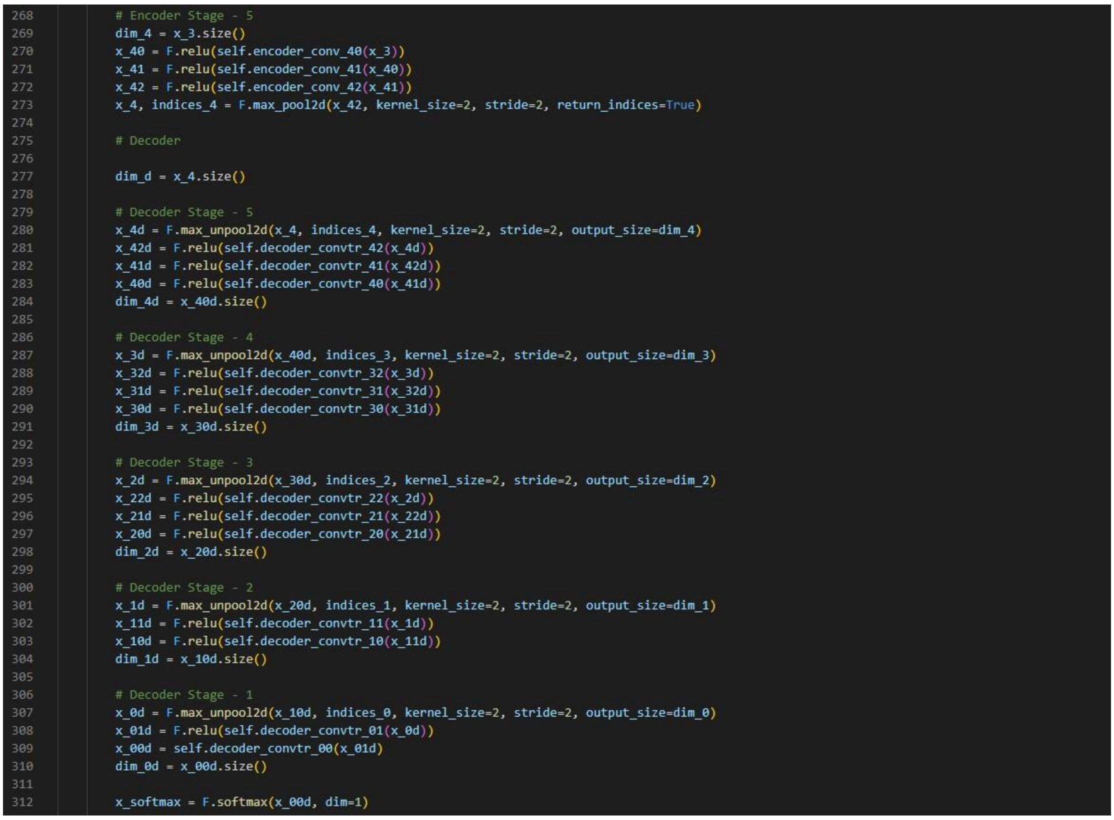


```
        self.decoder_convtr_21 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=256,
                                                                   out_channels=256,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(256)
                                               ])
        self.decoder_convtr_20 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=256,
                                                                   out_channels=128,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(128)
                                               ])
        self.decoder_convtr_11 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=128,
                                                                   out_channels=128,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(128)
                                               ])
        self.decoder_convtr_10 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=128,
                                                                   out_channels=64,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(64)
                                               ])
        self.decoder_convtr_01 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=64,
                                                                   out_channels=64,
                                                                   kernel_size=3,
                                                                   padding=1),
                                                nn.BatchNorm2d(64)
                                               ])
        self.decoder_convtr_00 = nn.Sequential(*[
                                                nn.ConvTranspose2d(in_channels=64,
                                                                   out_channels=self.output_channels,
                                                                   kernel_size=3,
                                                                   padding=1)
                                               ])
```

接下来是 forward 计算。

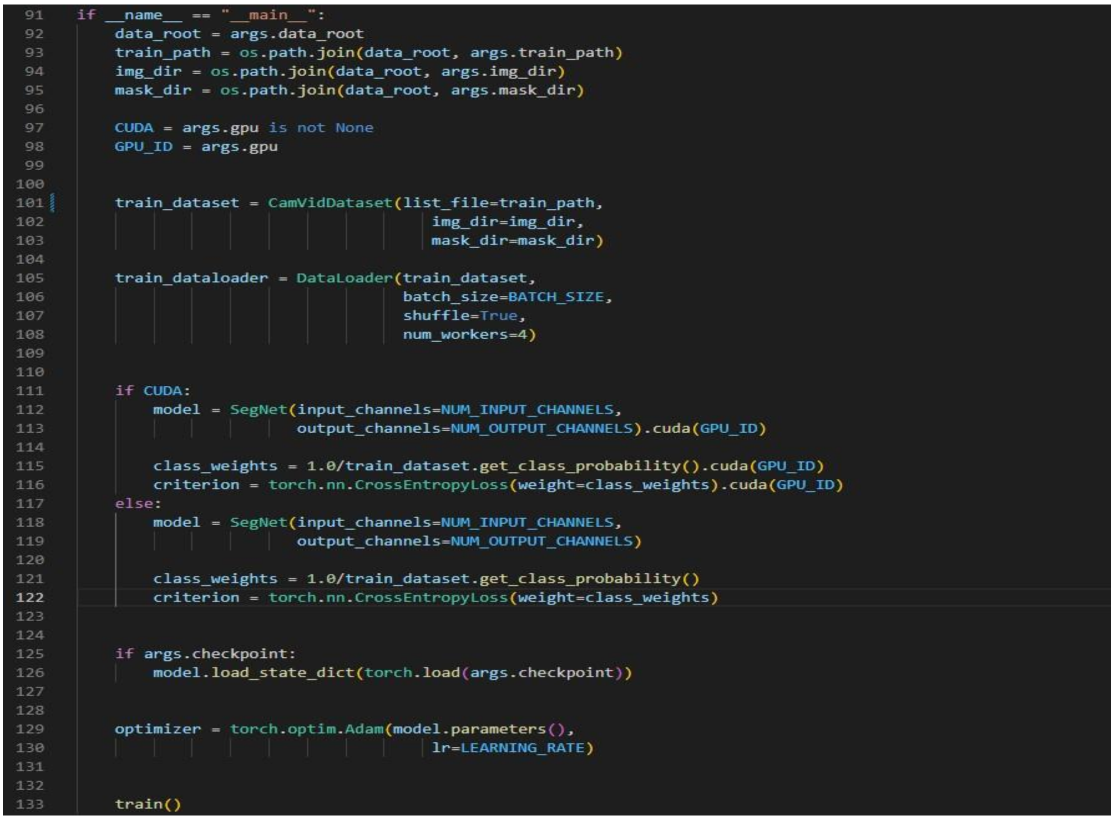


```python
    def forward(self, input_img):
        """
        Forward pass `input_img` through the network
        """

        # Encoder

        # Encoder Stage - 1
        dim_0 = input_img.size()
        x_00 = F.relu(self.encoder_conv_00(input_img))
        x_01 = F.relu(self.encoder_conv_01(x_00))
        x_0, indices_0 = F.max_pool2d(x_01, kernel_size=2, stride=2, return_indices=True)

        # Encoder Stage - 2
        dim_1 = x_0.size()
        x_10 = F.relu(self.encoder_conv_10(x_0))
        x_11 = F.relu(self.encoder_conv_11(x_10))
        x_1, indices_1 = F.max_pool2d(x_11, kernel_size=2, stride=2, return_indices=True)

        # Encoder Stage - 3
        dim_2 = x_1.size()
        x_20 = F.relu(self.encoder_conv_20(x_1))
        x_21 = F.relu(self.encoder_conv_21(x_20))
        x_22 = F.relu(self.encoder_conv_22(x_21))
        x_2, indices_2 = F.max_pool2d(x_22, kernel_size=2, stride=2, return_indices=True)

        # Encoder Stage - 4
        dim_3 = x_2.size()
        x_30 = F.relu(self.encoder_conv_30(x_2))
        x_31 = F.relu(self.encoder_conv_31(x_30))
        x_32 = F.relu(self.encoder_conv_32(x_31))
        x_3, indices_3 = F.max_pool2d(x_32, kernel_size=2, stride=2, return_indices=True)
```

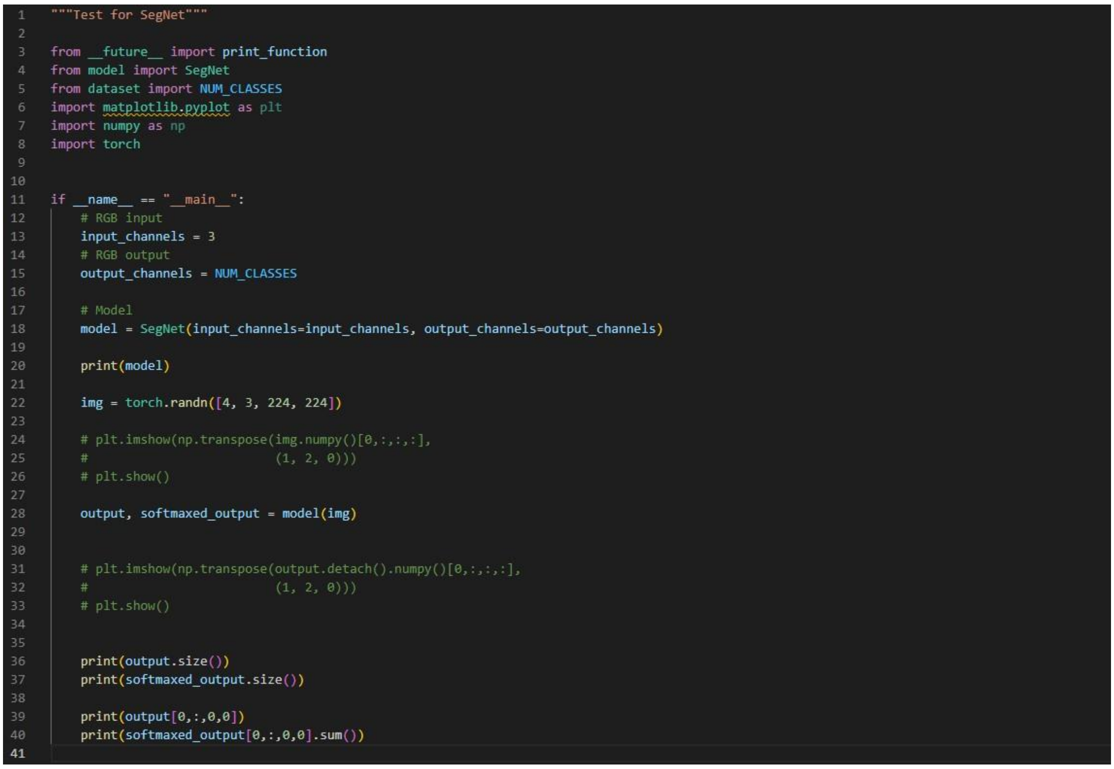


```python
        # Encoder Stage - 5
        dim_4 = x_3.size()
        x_40 = F.relu(self.encoder_conv_40(x_3))
        x_41 = F.relu(self.encoder_conv_41(x_40))
        x_42 = F.relu(self.encoder_conv_42(x_41))
        x_4, indices_4 = F.max_pool2d(x_42, kernel_size=2, stride=2, return_indices=True)

        # Decoder

        dim_d = x_4.size()

        # Decoder Stage - 5
        x_4d = F.max_unpool2d(x_4, indices_4, kernel_size=2, stride=2, output_size=dim_4)
        x_42d = F.relu(self.decoder_convtr_42(x_4d))
        x_41d = F.relu(self.decoder_convtr_41(x_42d))
        x_40d = F.relu(self.decoder_convtr_40(x_41d))
        dim_4d = x_40d.size()

        # Decoder Stage - 4
        x_3d = F.max_unpool2d(x_40d, indices_3, kernel_size=2, stride=2, output_size=dim_3)
        x_32d = F.relu(self.decoder_convtr_32(x_3d))
        x_31d = F.relu(self.decoder_convtr_31(x_32d))
        x_30d = F.relu(self.decoder_convtr_30(x_31d))
        dim_3d = x_30d.size()

        # Decoder Stage - 3
        x_2d = F.max_unpool2d(x_30d, indices_2, kernel_size=2, stride=2, output_size=dim_2)
        x_22d = F.relu(self.decoder_convtr_22(x_2d))
        x_21d = F.relu(self.decoder_convtr_21(x_22d))
        x_20d = F.relu(self.decoder_convtr_20(x_21d))
        dim_2d = x_20d.size()

        # Decoder Stage - 2
        x_1d = F.max_unpool2d(x_20d, indices_1, kernel_size=2, stride=2, output_size=dim_1)
        x_11d = F.relu(self.decoder_convtr_11(x_1d))
        x_10d = F.relu(self.decoder_convtr_10(x_11d))
        dim_1d = x_10d.size()

        # Decoder Stage - 1
        x_0d = F.max_unpool2d(x_10d, indices_0, kernel_size=2, stride=2, output_size=dim_0)
        x_01d = F.relu(self.decoder_convtr_01(x_0d))
        x_00d = self.decoder_convtr_00(x_01d)
        dim_0d = x_00d.size()

        x_softmax = F.softmax(x_00d, dim=1)
```

3. 训练和测试



```python
if __name__ == "__main__":
    data_root = args.data_root
    train_path = os.path.join(data_root, args.train_path)
    img_dir = os.path.join(data_root, args.img_dir)
    mask_dir = os.path.join(data_root, args.mask_dir)

    CUDA = args.gpu is not None
    GPU_ID = args.gpu


    train_dataset = CamVidDataset(list_file=train_path,
                                     img_dir=img_dir,
                                     mask_dir=mask_dir)

    train_dataloader = DataLoader(train_dataset,
                                  batch_size=BATCH_SIZE,
                                  shuffle=True,
                                  num_workers=4)


    if CUDA:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,
                       output_channels=NUM_OUTPUT_CHANNELS).cuda(GPU_ID)

        class_weights = 1.0/train_dataset.get_class_probability().cuda(GPU_ID)
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda(GPU_ID)
    else:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,
                       output_channels=NUM_OUTPUT_CHANNELS)

        class_weights = 1.0/train_dataset.get_class_probability()
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights)


    if args.checkpoint:
        model.load_state_dict(torch.load(args.checkpoint))


    optimizer = torch.optim.Adam(model.parameters(),
                                     lr=LEARNING_RATE)


    train()
```



```python
"""Test for SegNet"""

from __future__ import print_function
from model import SegNet
from dataset import NUM_CLASSES
import matplotlib.pyplot as plt
import numpy as np
import torch


if __name__ == "__main__":
    # RGB input
    input_channels = 3
    # RGB output
    output_channels = NUM_CLASSES

    # Model
    model = SegNet(input_channels=input_channels, output_channels=output_channels)

    print(model)

    img = torch.randn([4, 3, 224, 224])

    # plt.imshow(np.transpose(img.numpy()[0,:,:,:],
    #                          (1, 2, 0)))
    # plt.show()

    output, softmaxed_output = model(img)


    # plt.imshow(np.transpose(output.detach().numpy()[0,:,:,:],
    #                          (1, 2, 0)))
    # plt.show()


    print(output.size())
    print(softmaxed_output.size())

    print(output[0,:,0,0])
    print(softmaxed_output[0,:,0,0].sum())

```