

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。

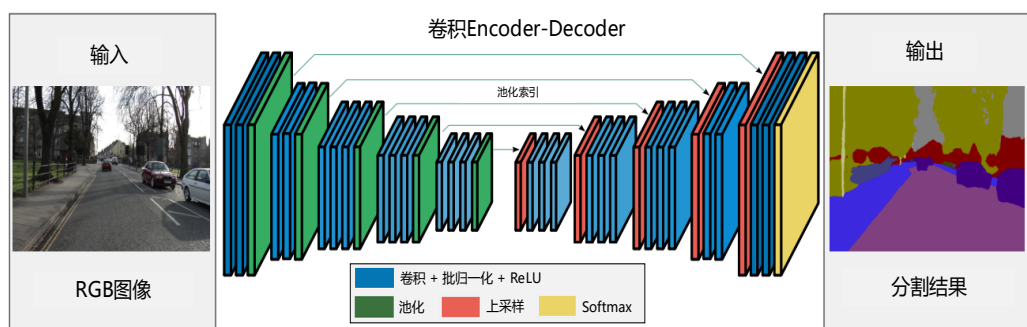


图 1 SegNet 网络结构

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 \(with "XX"\)](#)

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

五、 实验步骤和方法

1. 数据加载和处理

```

5  # CamVid 数据集路径
6  data_dir = '/path/to/CamVid/'
7
8  # 加载图像和标签文件名
9  def load_file_names(split):
10     images_dir = os.path.join(data_dir, 'images', split)
11     labels_dir = os.path.join(data_dir, 'labels', split)
12     image_files = sorted(os.listdir(images_dir))
13     label_files = sorted(os.listdir(labels_dir))
14     return image_files, label_files
15
16 # 加载图像和标签数据
17 def load_data(split):
18     image_files, label_files = load_file_names(split)
19     images = []
20     labels = []
21     for img_file, lbl_file in zip(image_files, label_files):
22         img_path = os.path.join(data_dir, 'images', split, img_file)
23         lbl_path = os.path.join(data_dir, 'labels', split, lbl_file)
24         img = np.array(Image.open(img_path).convert('RGB'))
25         lbl = np.array(Image.open(lbl_path).convert('P'))
26         images.append(img)
27         labels.append(lbl)
28     return np.array(images), np.array(labels)
29
30 # 数据预处理
31 def preprocess_data(images, labels):
32     # 这里可以进行图像大小调整、归一化等预处理操作
33     return images, labels
34

```

2. 模型构建

```

33 class SegNet(nn.Module):
34     def __init__(self, input_channels, output_channels):
35         super(SegNet, self).__init__()
36
37         self.input_channels = input_channels
38         self.output_channels = output_channels
39
40         self.num_channels = input_channels
41
42         self.vgg16 = models.vgg16(pretrained=True)
43
44
45         # Encoder layers
46
47         self.encoder_conv_00 = nn.Sequential(*[
48             nn.Conv2d(in_channels=self.input_channels,
49                       out_channels=64,
50                       kernel_size=3,
51                       padding=1),
52             nn.BatchNorm2d(64)
53         ])
54         self.encoder_conv_01 = nn.Sequential(*[
55             nn.Conv2d(in_channels=64,
56                       out_channels=64,
57                       kernel_size=3,
58                       padding=1),
59             nn.BatchNorm2d(64)
60         ])
61         self.encoder_conv_10 = nn.Sequential(*[
62             nn.Conv2d(in_channels=64,
63                       out_channels=128,
64                       kernel_size=3,
65                       padding=1),
66             nn.BatchNorm2d(128)
67         ])
68         self.encoder_conv_11 = nn.Sequential(*[
69             nn.Conv2d(in_channels=128,
70                       out_channels=128,
71                       kernel_size=3,
72                       padding=1),
73             nn.BatchNorm2d(128)
74         ])

```

```

75         self.encoder_conv_20 = nn.Sequential(*[
76             nn.Conv2d(in_channels=128,
77                       out_channels=256,
78                       kernel_size=3,
79                       padding=1),
80             nn.BatchNorm2d(256)
81         ])
82         self.encoder_conv_21 = nn.Sequential(*[
83             nn.Conv2d(in_channels=256,
84                       out_channels=256,
85                       kernel_size=3,
86                       padding=1),
87             nn.BatchNorm2d(256)
88         ])
89         self.encoder_conv_22 = nn.Sequential(*[
90             nn.Conv2d(in_channels=256,
91                       out_channels=256,
92                       kernel_size=3,
93                       padding=1),
94             nn.BatchNorm2d(256)
95         ])
96         self.encoder_conv_30 = nn.Sequential(*[
97             nn.Conv2d(in_channels=256,
98                       out_channels=512,
99                       kernel_size=3,
100                      padding=1),
101             nn.BatchNorm2d(512)
102         ])
103         self.encoder_conv_31 = nn.Sequential(*[
104             nn.Conv2d(in_channels=512,
105                       out_channels=512,
106                       kernel_size=3,
107                       padding=1),
108             nn.BatchNorm2d(512)
109         ])
110         self.encoder_conv_32 = nn.Sequential(*[
111             nn.Conv2d(in_channels=512,
112                       out_channels=512,
113                       kernel_size=3,
114                       padding=1),
115             nn.BatchNorm2d(512)
116         ])
117         self.encoder_conv_40 = nn.Sequential(*[
118             nn.Conv2d(in_channels=512,
119                       out_channels=512,
120                       kernel_size=3,
121                       padding=1),
122             nn.BatchNorm2d(512)
123         ])
124         self.encoder_conv_41 = nn.Sequential(*[
125             nn.Conv2d(in_channels=512,
126                       out_channels=512,
127                       kernel_size=3,
128                       padding=1),
129             nn.BatchNorm2d(512)
130         ])
131         self.encoder_conv_42 = nn.Sequential(*[
132             nn.Conv2d(in_channels=512,
133                       out_channels=512,
134                       kernel_size=3,
135                       padding=1),
136             nn.BatchNorm2d(512)
137         ])
138
139         self.init_vgg_weights()

```

上述为 encoder 定义，接下来是 decoder

接下来是 forward 计算。

```
235 def forward(self, input_img):
236     """
237     Forward pass `input_img` through the network
238     """
239
240     # Encoder
241
242     # Encoder Stage - 1
243     dim_0 = input_img.size()
244     x_00 = F.relu(self.encoder_conv_00(input_img))
245     x_01 = F.relu(self.encoder_conv_01(x_00))
246     x_0, indices_0 = F.max_pool2d(x_01, kernel_size=2, stride=2, return_indices=True)
247
248     # Encoder Stage - 2
249     dim_1 = x_0.size()
250     x_10 = F.relu(self.encoder_conv_10(x_0))
251     x_11 = F.relu(self.encoder_conv_11(x_10))
252     x_1, indices_1 = F.max_pool2d(x_11, kernel_size=2, stride=2, return_indices=True)
253
254     # Encoder Stage - 3
255     dim_2 = x_1.size()
256     x_20 = F.relu(self.encoder_conv_20(x_1))
257     x_21 = F.relu(self.encoder_conv_21(x_20))
258     x_22 = F.relu(self.encoder_conv_22(x_21))
259     x_2, indices_2 = F.max_pool2d(x_22, kernel_size=2, stride=2, return_indices=True)
260
261     # Encoder Stage - 4
262     dim_3 = x_2.size()
263     x_30 = F.relu(self.encoder_conv_30(x_2))
264     x_31 = F.relu(self.encoder_conv_31(x_30))
265     x_32 = F.relu(self.encoder_conv_32(x_31))
266     x_3, indices_3 = F.max_pool2d(x_32, kernel_size=2, stride=2, return_indices=True)
267
268     # Encoder Stage - 5
269     dim_4 = x_3.size()
270     x_40 = F.relu(self.encoder_conv_40(x_3))
271     x_41 = F.relu(self.encoder_conv_41(x_40))
272     x_42 = F.relu(self.encoder_conv_42(x_41))
273     x_4, indices_4 = F.max_pool2d(x_42, kernel_size=2, stride=2, return_indices=True)
274
275     # Decoder
276
277     dim_d = x_4.size()
278
279     # Decoder Stage - 5
280     x_4d = F.max_unpool2d(x_4, indices_4, kernel_size=2, stride=2, output_size=dim_4)
281     x_42d = F.relu(self.decoder_convtr_42(x_4d))
282     x_41d = F.relu(self.decoder_convtr_41(x_42d))
283     x_40d = F.relu(self.decoder_convtr_40(x_41d))
284     dim_4d = x_40d.size()
285
286     # Decoder Stage - 4
287     x_3d = F.max_unpool2d(x_40d, indices_3, kernel_size=2, stride=2, output_size=dim_3)
288     x_32d = F.relu(self.decoder_convtr_32(x_3d))
289     x_31d = F.relu(self.decoder_convtr_31(x_32d))
290     x_30d = F.relu(self.decoder_convtr_30(x_31d))
291     dim_3d = x_30d.size()
292
293     # Decoder Stage - 3
294     x_2d = F.max_unpool2d(x_30d, indices_2, kernel_size=2, stride=2, output_size=dim_2)
295     x_22d = F.relu(self.decoder_convtr_22(x_2d))
296     x_21d = F.relu(self.decoder_convtr_21(x_22d))
297     x_20d = F.relu(self.decoder_convtr_20(x_21d))
298     dim_2d = x_20d.size()
299
300     # Decoder Stage - 2
301     x_1d = F.max_unpool2d(x_20d, indices_1, kernel_size=2, stride=2, output_size=dim_1)
302     x_11d = F.relu(self.decoder_convtr_11(x_1d))
303     x_10d = F.relu(self.decoder_convtr_10(x_11d))
304     dim_1d = x_10d.size()
305
306     # Decoder Stage - 1
307     x_0d = F.max_unpool2d(x_10d, indices_0, kernel_size=2, stride=2, output_size=dim_0)
308     x_01d = F.relu(self.decoder_convtr_01(x_0d))
309     x_00d = self.decoder_convtr_00(x_01d)
310     dim_0d = x_00d.size()
311
312     x_softmax = F.softmax(x_00d, dim=1)
```

3. 训练和测试


```

91 if __name__ == "__main__":
92     data_root = args.data_root
93     train_path = os.path.join(data_root, args.train_path)
94     img_dir = os.path.join(data_root, args.img_dir)
95     mask_dir = os.path.join(data_root, args.mask_dir)
96
97     CUDA = args.gpu is not None
98     GPU_ID = args.gpu
99
100
101     train_dataset = CamVidDataset(list_file=train_path,
102                                  img_dir=img_dir,
103                                  mask_dir=mask_dir)
104
105     train_dataloader = DataLoader(train_dataset,
106                                  batch_size=BATCH_SIZE,
107                                  shuffle=True,
108                                  num_workers=4)
109
110
111     if CUDA:
112         model = SegNet(input_channels=NUM_INPUT_CHANNELS,
113                        output_channels=NUM_OUTPUT_CHANNELS).cuda(GPU_ID)
114
115         class_weights = 1.0/train_dataset.get_class_probability().cuda(GPU_ID)
116         criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda(GPU_ID)
117     else:
118         model = SegNet(input_channels=NUM_INPUT_CHANNELS,
119                        output_channels=NUM_OUTPUT_CHANNELS)
120
121         class_weights = 1.0/train_dataset.get_class_probability()
122         criterion = torch.nn.CrossEntropyLoss(weight=class_weights)
123
124
125     if args.checkpoint:
126         model.load_state_dict(torch.load(args.checkpoint))
127
128
129     optimizer = torch.optim.Adam(model.parameters(),
130                                  lr=LEARNING_RATE)
131
132
133     train()

```

```

1 """Test for SegNet"""
2
3 from __future__ import print_function
4 from model import SegNet
5 from dataset import NUM_CLASSES
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import torch
9
10
11 if __name__ == "__main__":
12     # RGB input
13     input_channels = 3
14     # RGB output
15     output_channels = NUM_CLASSES
16
17     # Model
18     model = SegNet(input_channels=input_channels, output_channels=output_channels)
19
20     print(model)
21
22     img = torch.randn([4, 3, 224, 224])
23
24     # plt.imshow(np.transpose(img.numpy()[0,:,:,:],
25                             # (1, 2, 0)))
26     # plt.show()
27
28     output, softmaxed_output = model(img)
29
30
31     # plt.imshow(np.transpose(output.detach().numpy()[0,:,:,:],
32                             # (1, 2, 0)))
33     # plt.show()
34
35
36     print(output.size())
37     print(softmaxed_output.size())
38
39     print(output[0,:,:0])
40     print(softmaxed_output[0,:,:0].sum())
41

```