


# 基于DeepLabV3的语义分割模型

 更新于 2020-09-08 10:00:34

运行

 8  3

包含数据集2个		收起
green.jpg	354.33 K	
sheep.jpg	378.58 K	

语义分割是计算机视觉领域的关键问题之一，在像素级别上的分类，属于同一类的像素都要被归为一类。越来越多的应用，如自动驾驶汽车、人机交互、虚拟现实等，通过语义分割，从图像中获取信息。随着深度学习的普及，卷积神经网络等结构被应用于语义分割问题，显著提高了精度和计算效率。

本案例使用以Resnet101为主干网络的DeepLabV3模型，对图像进行语义分割，并讲解语义分割的流程和思想。



## 目录

- 1. [数据集简介](#)
- 2. [DeepLabV3模型介绍](#)
- 3. [数据预处理](#)
- 4. [语义分割](#)
  - 4.1 [识别物体类别](#)
  - 4.2 [语义分割结果可视化](#)

## [5. 前景分割](#)

## [6. 总结](#)

# 1 数据集简介

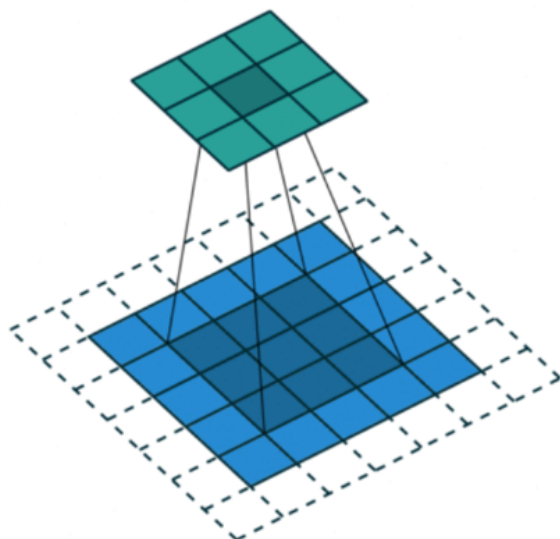
本案例采用的DeepLabV3模型，已经在COCO Train 2017数据集中的一个子集上进行了训练。该子集包含20个可以检测的类别，与Pascal VOC数据集中的20个类别一致。如自行车、鸟、船、狗、沙发等类别。

数据集中的图像包含不同角度、不同形态的各类别物体。数据集中已经标注出20类物体，并注明其所属的类别，便于进行语义分割。

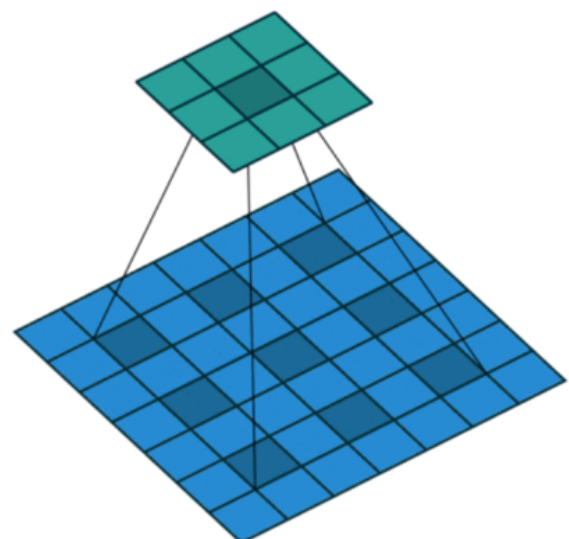
# 2 DeepLabV3模型介绍

DeepLabV3模型是利用深度学习网络进行语义分割的模型。其核心是ASPP网络（Atrous Spatial Pyramid Pooling）。ASPP网络是带有空洞卷积的空间金字塔结构。在进行语义分割时，被检测物体往往尺度不相同。解决这一问题的一种方法是，在进行语义分割时，先通过卷积神经网络检测出物体，再对物体的大小进行缩放，最后融合。但这种方式的计算量较大。ASPP网络既能够得到多尺度的物体的信息，计算量也相对较小。

首先介绍空洞卷积的思想。如下图所示，相对于普通的卷积核，空洞卷积核通过对输入等间隔采样，可以在不增加计算量的情况下，增大卷积核的感受野。



Standard Convolution with a 3 x 3 kernel (and padding)



Dilated Convolution with a 3 x 3 kernel and dilation rate 2

空间金字塔网络（SPP-Net）和空洞卷积构成了ASPP网络结构，SPP网络结构的目的是为了获得不同尺度的物体的特征。

因为卷积神经网络中的全连接层，要求训练和测试时输入的数据维度一致，一种方式即之前所述的改变图像大小，但是计算量大，且会丢失图像信息。SPP网络可以输入不同大小的图像，并将其特征转换为同一大小输入进全连接层。

转换的方式如下图，不同尺度的数据传入卷积层，得到不同尺度的特征图。将特征图同时传入池化核不相同的三种池化层，融合三个池化层的输出数据，作为特征，传入全连接层。通过这一方式，不同尺度的图像，传入全连接层的特征图大小一致。

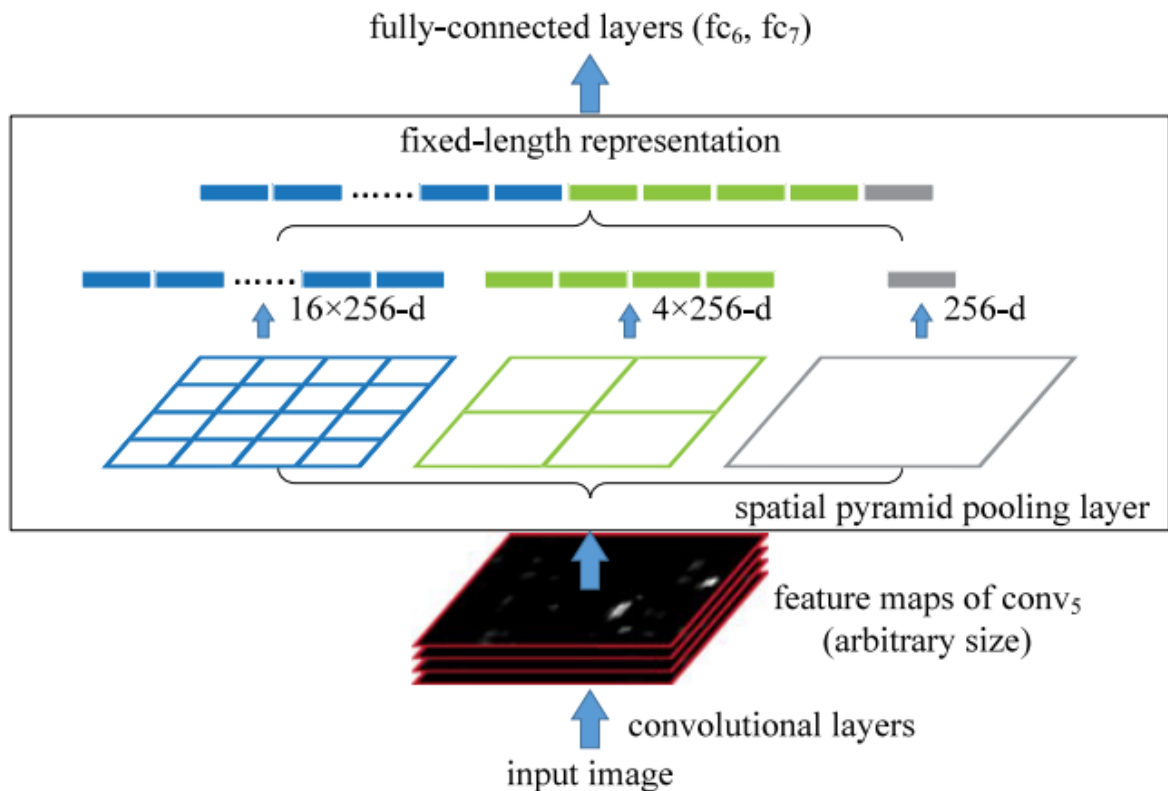


Fig. 3. A network structure with a *spatial pyramid pooling layer*. Here 256 is the filter number of the conv<sub>5</sub> layer, and conv<sub>5</sub> is the last convolutional layer.

```
# 导入需要的库
from torchvision import models
import torchvision.transforms as T
from PIL import Image
import matplotlib.pyplot as plt
import torch
import cv2
```

```
import numpy as np
```

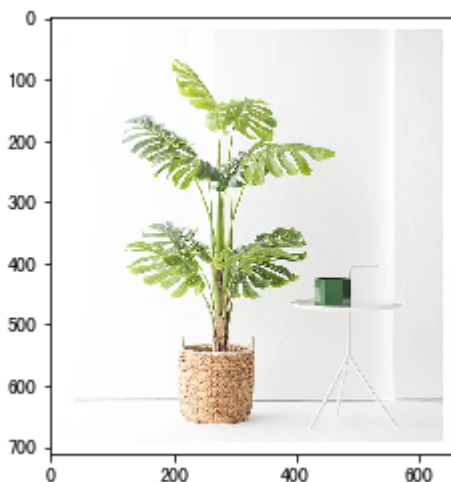
接下来，通过设定 `pretrained=True` 和调用 `eval()` 函数，加载预训练模型进行语义分割测试。

```
# 读取DeepLabV3模型
dlab = models.segmentation.deeplabv3_resnet101(pretrained=True).eval()
```

### 3 数据预处理

首先读取数据，我们加载一幅包含盆栽的图片，盆栽是模型可识别的20个类别之一。

```
# 读取RGB图像数据
img = Image.open('./input/green.jpg').convert('RGB')
plt.imshow(img)
plt.show()
```



在使用DeepLabV3模型得到图像的语义分割图前，我们需要对输入图像进行数据预处理，使得图像转换为模型需要的输入数据的格式。

模型需要输入的图像格式为：三通道图像；至少为 $224 \times 224$ 像素；并选择大型数据库ImageNet中图像的平均值和标准差为参数，对输入图像进行归一化操作。

归一化不改变图像的信息，将像素值的取值范围从 `[0, 255]` 转变为 `[0, 1]`。归一化操作可以加快训练网络的收敛性；因为神经网络是以样本出现概率来进行训练和预测的，归一化到 `[0, 1]` 范围可以统一样本的统计概率分布。

```
mean = [0.485, 0.456, 0.406]
std = [0.222, 0.224, 0.225]
```

```
std = [0.229, 0.224, 0.225]

trf = T.Compose([T.Resize(256),      # 将图像变为256×256大小
                 T.CenterCrop(224), # 将图像从中心剪裁为224×224大小
                 T.ToTensor(),       # 数据转化为Tensor格式
                 T.Normalize(mean = [0.485, 0.456, 0.406],
                             std = [0.229, 0.224, 0.225]))
                 # 对图像进行归一化处理
```

此外，因为模型是分批次进行训练和预测数据。因此输入数据的格式为：

$$[N_i \times C_i \times H_i \times W_i]$$

- $N_i$ : 批次的数量 (batch size)
- $C_i$ : 图像的通道数，本案例使用RGB图像，通道数为3
- $H_i$ : 图像的高
- $W_i$ : 图像的宽

我们需要通过 `unsqueeze` 函数，将单张图像的三维数据，转化为四维数据，即添加一个维度的数据，代表1个批次。

```
# 转化为四维数据
inp = trf(img).unsqueeze(0)
```

## 4 语义分割

### 4.1 识别物体类别

接下来，我们可以使用DeepLabV3对图像进行语义分割。模型的输出数据的维度为

$$[N_o \times C_o \times H_o \times W_o]$$

- $N_o$ : 批次的数量 (batch size)，与 $N_i$ 相同
- $C_o$ : 需要识别的物体的类别和背景类，本案例中为21
- $H_o$ : 图像的高，与 $H_i$ 相同
- $W_o$ : 图像的宽，与 $W_i$ 相同

```
out = dlab(inp)['out']
print("输出数据的维度", out.shape)
```

输出数据的维度 `torch.Size([1, 21, 224, 224])`

输出数据 `out` 储存着图像中每一个通道的像素值属于21类物体（20类物体及1个背景类）之一的概率。我们使用 `argmax` 函数，找到概率值最大的类别，判定该像素最可能属于该类物体。以此判断该图像中包含哪一类物体。

```
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()
print("输出图像的大小", om.shape)
print("图像中包含的类别", np.unique(om))
```

输出图像的大小 (224, 224)

图像中包含的类别 [ 0 16]

`np.unique(om)` 值为 `[0, 16]`，代表图像包含背景类（用数字 0 表示），和第16类物体，即“盆栽”。

## 4.2 语义分割结果可视化

接下来我们将语义分割的结果，用图像表示出来，更加直观地判断分割的效果。

具体的操作方式是：因为语义分割是像素级的分割，模型已经判断出每个像素属于哪一类物体。通过对属于同一类别的像素，赋予同一种颜色，语义分割的结果可以通过图像进行展示。

```
# 储存不同类别对应的颜色
label_colors = np.array([(0, 0, 0), # 0=background
                        (128, 0, 0), (0, 128, 0), (128, 128, 0), (0, 0, 128), (128, 0, 128),
                        # 6=bus, 7=car, 8=cat, 9=chair, 10=cow
                        (0, 128, 128), (128, 128, 128), (64, 0, 0), (192, 0, 0), (64, 128, 0),
                        # 11=dining table, 12=dog, 13=horse, 14=motorbike, 15=person
                        (192, 128, 0), (64, 0, 128), (192, 0, 128), (64, 128, 128), (192, 128, 128),
                        # 16=potted plant, 17=sheep, 18=sofa, 19=train, 20=tv/monitor
                        (0, 64, 0), (128, 64, 0), (0, 192, 0), (128, 192, 0), (0, 64, 128)])

# 属于同一类别的像素, 具有相同的颜色
def decode_segmap(image, label_colors, nc=21):
    # 三张灰度图像, 储存三通道的像素值
    r = np.zeros_like(image).astype(np.uint8)
    g = np.zeros_like(image).astype(np.uint8)
    b = np.zeros_like(image).astype(np.uint8)

    for l in range(0, nc):
        idx = image == l
        r[idx] = label_colors[l, 0]
        g[idx] = label_colors[l, 1]
        b[idx] = label_colors[l, 2]
```



```
# 将三张灰度图像组成RGB图像
rgb = np.stack([r, g, b], axis=2)
return rgb
```

画出图像如下图，我们观察到绿色部分，对应的是图像中的“盆栽”，黑色部分为背景类。

```
rgb = decode_segmap(om, label_colors)
plt.imshow(rgb)
plt.axis('off')
plt.show()
```



## 5 前景分割

按照上述进行语义分割的思想，我们也可以对图像进行前景分割。前景分割指，计算机从一幅图片中判断出哪个是前景物体，哪个是背景物体，并从中分割出感兴趣的前景关键物体。

```
img = Image.open('./input/sheep.jpg').convert('RGB')
plt.imshow(img)
plt.axis('off')
plt.show()
```



首先对图像进行语义分割，操作步骤与上述一致，输出包含语义分割结果的变量 `rgb` 。其中图像类别为 `[0, 17]` ，说明图像包含背景类、第17类物体“羊”。

```
# 转化为四维数据
inp = trf(img).unsqueeze(0)
out = dlab(inp)['out']
print("输出数据的维度", out.shape)
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()
print("输出图像的大小", om.shape)
print("图像中包含的类别", np.unique(om))
rgb = decode_segmap(om, label_colors)
```

输出数据的维度 `torch.Size([1, 21, 224, 224])`  
 输出图像的大小 `(224, 224)`  
 图像中包含的类别 `[ 0 17]`

之后，我们利用语义分割得到的数据，对图像进行前景分割。首先复制原图像，将其储存在 `foreground` 变量中，并将图像大小转换为，与语义分割输出数据一致，即图像大小为  $224 \times 224$ 。

```
# 读取数据
foreground=cv2.imread("./input/sheep.jpg")
# 转化为RGB图像格式
foreground=cv2.cvtColor(foreground, cv2.COLOR_BGR2RGB)
# 修改图像大小
r = np.zeros_like(om).astype(np.uint8)
foreground=cv2.resize(foreground, (r.shape[1], r.shape[0]))
```

现在我们定义背景图，储存在变量 `background` 中，初始情况下定义为与语义分割输出数据大小一致、像素值都为 `255` 的白色图像。

```
background=255*np.ones_like(rgb).astype(np.uint8)
# 将数值转换为float格式, 便于进行运算
foreground=foreground.astype(float)
background=background.astype(float)
```

使用 `cv2.threshold` 函数找到图像的前景图。因为在语义分割函数中，我们设定代表背景类（即应该识别出的背景图）的颜色为黑色，像素值为 `0` ，故任何值高于 `0` 的像素点都会被认为是前景，根据语义分割的输出结果，该前景图为羊的图像。

输出结果为 `[th, alpha]` ，其中 `th` 代表判断阈值，这里应为 `0` ，用以判断像素点是

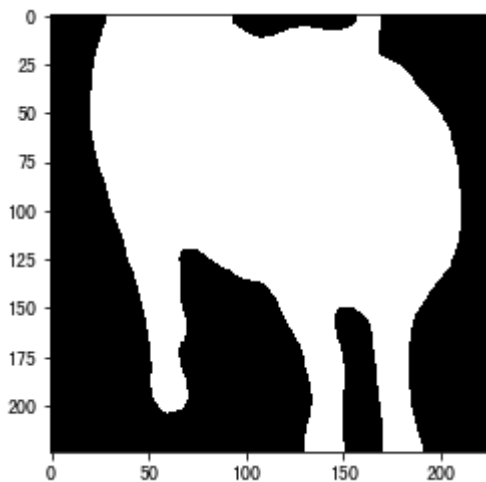


否属于前景。`alpha` 为二进制掩膜，可以看到白色部分为我们需要的前景图，黑色部分为背景图，即背景类。

```
gray = cv2.cvtColor(rgb, cv2.COLOR_BGR2GRAY)
th, alpha = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)
alpha = cv2.cvtColor(alpha, cv2.COLOR_GRAY2RGB)
print("阈值th:", th)
plt.imshow(alpha)
```

阈值th: 0.0

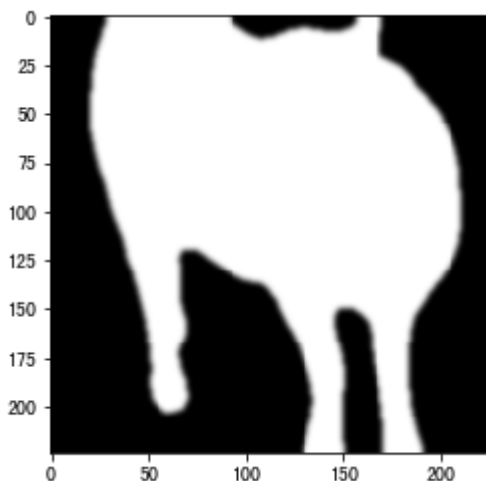
<matplotlib.image.AxesImage at 0x7f2373555b00>



接下来使用 `cv2.GaussianBlur` 函数，对掩膜进行高斯模糊，让图像的边缘更加平滑。使输出的前景图的边缘不会过于锐利。

```
alpha = cv2.GaussianBlur(alpha, (7, 7), 0)
plt.imshow(alpha)
```

<matplotlib.image.AxesImage at 0x7f237046b4e0>



取/白，我们使用一起的地板  $\alpha$  为两个图像的乘积得到背景图。将  $\alpha$  中的像素值

除以 255，则所有的像素值在  $[0, 1]$  之间，且前景图部分的值为 1，背景图部分的值为 0。因此，将原图像和  $\alpha$  相乘，可以保留前景图部分。将白色图像  $\text{background}$  和  $1-\alpha$  相乘，可以得到白色的背景图。

```
alpha=alpha.astype(float)/255
foreground=cv2.multiply(1.0-(alpha==0), foreground)
background=cv2.multiply(1.0-alpha, background)
outImage=cv2.add(foreground, background)
```

```
plt.imshow(outImage/255)
plt.axis('off')
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ( $[0..1]$  for floats or  $[0..255]$  for integers).



## 6 总结

本案例介绍了如何使用以Resnet101为主干网络的DeepLabV3模型对图像进行语义分割。并使用语义分割的输出结果，对图像进行前景分割。观察输出结果，证明模型识别需要分割物体的准确率较高。

© 2018 CookData 京ICP备1705652	竞赛	关于	产品	服务	帮助	联系
3号-1 (http://www.beian.miit.gov.cn/)	平台	我们 (/	介绍 (/	条款 (/	中心 (/	我们 (/
	(http://	foote	foote	foote	foote	foote
	cookd	r?sub	r?sub	r?sub	r?sub	r?sub
	ata.c	=0)	=0)	=1)	=2)	=3)
	n/com					
	petitio					
	n/)					