# 8.PyTorch深度体验

## 8.1.图像分类预测

模型如何完成图像分类？



将图像转换为tensor --> 模型 --> 输出向量 --> 取向量的最大值作为预测结果

代码基本步骤：  
 1. 获取数据与模型  
 2. 数据变换，如RGB → 4D-Tensor  
 3. 前向传播  
 4. 输出保存预测结果

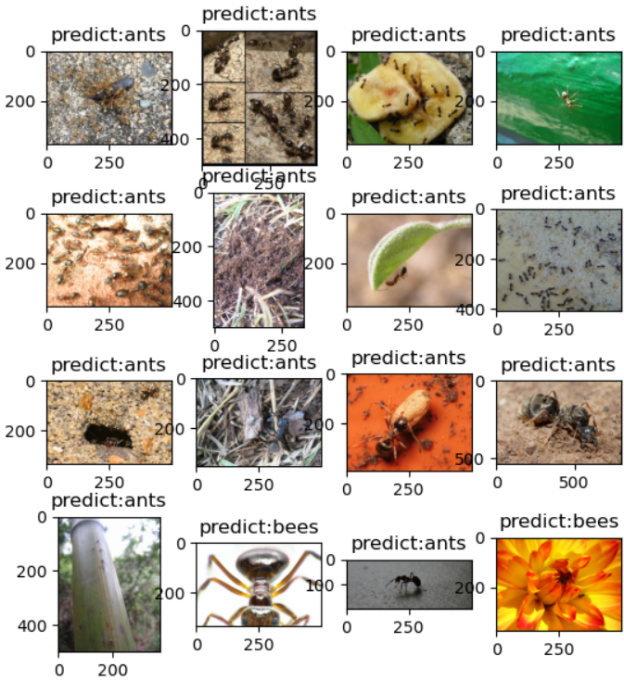
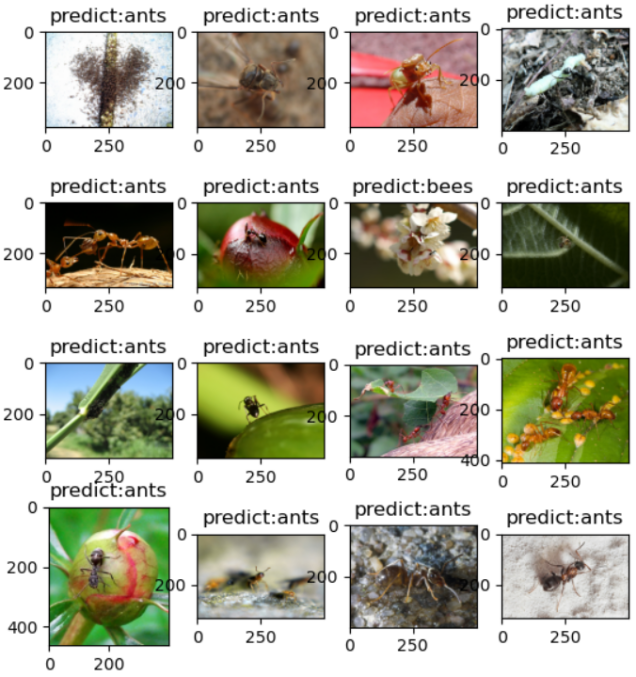
注意事项：  
 1. 确保 model处于eval状态而非training  
 2. 设置torch.no\_grad()，减少内存消耗  
 3. 数据预处理需保持一致， RGB or BGR

代码实现：

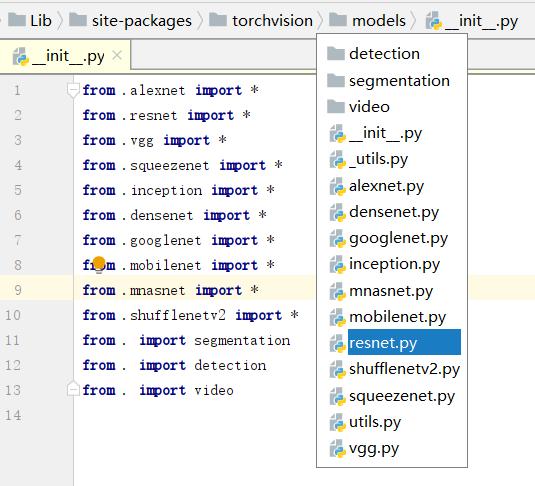
以模型微调中的分类模型为例，进行预测

*# -\*- coding: utf-8 -\*-***import** os  
**import** time  
**import** torch.nn **as** nn  
**import** torch  
**import** torchvision.transforms **as** transforms  
**from** PIL **import** Image  
**from** matplotlib **import** pyplot **as** plt  
**import** torchvision.models **as** models  
BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
device = torch.device(**"cuda" if** torch.cuda.is\_available() **else "cpu"**)  
*# device = torch.device("cpu")  
  
# 配置可视化开关*vis = **True***# vis = False  
# 设置可视化时每行有几张图片*vis\_row = 4  
  
norm\_mean = [0.485, 0.456, 0.406]  
norm\_std = [0.229, 0.224, 0.225]  
  
*# 对数据预处理，注意和模型的预处理方式保持一致*inference\_transform = transforms.Compose([  
 transforms.Resize(256),  
 transforms.CenterCrop(224),  
 transforms.ToTensor(),  
 transforms.Normalize(norm\_mean, norm\_std),  
])  
  
*# 标签的分类*classes = [**"ants"**, **"bees"**]  
  
*# 将图片进行预处理得到张量，即数据转换为模型读取的形式***def** img\_transform(img\_rgb, transform=**None**):  
  
 **if** transform **is None**:  
 **raise** ValueError(**"找不到transform！必须有transform对img进行处理"**)  
  
 img\_t = transform(img\_rgb)  
 **return** img\_t  
  
*# 获取文件夹下format格式的文件名***def** get\_img\_name(img\_dir, format=**"jpg"**):  
  
 file\_names = os.listdir(img\_dir)  
 img\_names = list(filter(**lambda** x: x.endswith(format), file\_names))  
  
 **if** len(img\_names) < 1:  
 **raise** ValueError(**"{}下找不到{}格式数据"**.format(img\_dir, format))  
 **return** img\_names  
  
  
**def** get\_model(m\_path, vis\_model=**False**):  
  
 *# 创建resnet18模型* resnet18 = models.resnet18()  
 *# 获取全连接层的输入参数* num\_ftrs = resnet18.fc.in\_features  
 *# 自定义全连接层，设置输入为2，即二分类* resnet18.fc = nn.Linear(num\_ftrs, 2)  
  
 *# 根据模型路径加载模型，为检查点* checkpoint = torch.load(m\_path)  
 *# 将参数加载到模型中* resnet18.load\_state\_dict(checkpoint[**'model\_state\_dict'**])  
  
 *# 打印模型的信息，如每一层的类型、shape 和 参数量等* **if** vis\_model:  
 *# 需要导入torchsummary包，安装：pip install torchsummary* **from** torchsummary **import** summary  
 summary(resnet18, input\_size=(3, 224, 224), device=**"cpu"**)  
  
 **return** resnet18  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *# 指定数据路径* img\_dir = os.path.join(**".."**, **".."**, **"data/hymenoptera\_data/val/ants"**)  
 *# 指定模型路径，为蚂蚁蜜蜂之前保存的检查点* model\_path = **"./checkpoint\_24\_epoch.pkl"** *# 用来统计总预测时间* time\_total = 0  
 *# 定义两个list，分别用来存放图片和对应的预测类型* img\_list, img\_pred = list(), list()  
  
 *# 1. data  
 # 获取指定路径下所有文件名* img\_names = get\_img\_name(img\_dir)  
 *# 获取文件的数量* num\_img = len(img\_names)  
  
 *# 2. model  
 # 获取模型* resnet18 = get\_model(model\_path, **True**)  
 *# 将模型放到指定设备上* resnet18.to(device)  
 *# 将模型设置为测试模式* resnet18.eval()  
  
 *# 下面的所有运算无需保存梯度* **with** torch.no\_grad():  
 *# 遍历文件名* **for** idx, img\_name **in** enumerate(img\_names):  
  
 *# 拼接每个文件的全路径* path\_img = os.path.join(img\_dir, img\_name)  
  
 *# step 1/4 : path --> img 根据路径读取rgb图片* img\_rgb = Image.open(path\_img).convert(**'RGB'**)  
  
 *# step 2/4 : img --> tensor 将rgb图像转换为张量* img\_tensor = img\_transform(img\_rgb, inference\_transform)  
 *# 3d --> 4d* img\_tensor.unsqueeze\_(0)  
 *# 将数据放到指定设备上* img\_tensor = img\_tensor.to(device)  
  
 *# step 3/4 : tensor --> vector  
 # 统计运行时间* time\_tic = time.time()  
 *# 将数据放到模型上得到输出结果* outputs = resnet18(img\_tensor)  
 time\_toc = time.time()  
  
 *# step 4/4 : visualization  
 # torch.max分别返回最大值和对应的索引，这里需要用到索引* \_, pred\_int = torch.max(outputs.data, 1)  
 *# 获取索引对应的分类名称* pred\_str = classes[int(pred\_int)]  
  
 **if** vis:  
 *# 将图片和预测结果添加到对应的list中* img\_list.append(img\_rgb)  
 img\_pred.append(pred\_str)  
  
 *# 当行列数都达到设置的vis\_row或者到达最后一张图片时* **if** (idx+1) % (vis\_row\*vis\_row) == 0 **or** num\_img == idx+1:  
 *# 遍历每张图片* **for** i **in** range(len(img\_list)):  
 *# 设置vis\_row行vis\_row列的子图，并在子图中展示对应下标的图片* plt.subplot(vis\_row, vis\_row, i+1).imshow(img\_list[i])  
 *# 设置图片标题为预测标签* plt.title(**"predict:{}"**.format(img\_pred[i]))  
 plt.show()  
 plt.close()  
 *# 显示完图片后，清空图片list和预测类型的list* img\_list, img\_pred = list(), list()  
  
 *# 计算每次预测时间并累加到总时间上* time\_s = time\_toc-time\_tic  
 time\_total += time\_s  
  
 *# 打印第几张图/总图片数，图片名称，当前图片的预测消耗时间* print(**'{:d}/{:d}: {} {:.3f}s '**.format(idx + 1, num\_img, img\_name, time\_s))  
  
 *# 最后打印当前使用的设备、总耗时、每张图片平均耗时* print(**"\ndevice:{} total time:{:.1f}s mean:{:.3f}s"**.  
 format(device, time\_total, time\_total/num\_img))  
 *# 如果GPU可用，打印当前GPU的型号* **if** torch.cuda.is\_available():  
 print(**"GPU name:{}"**.format(torch.cuda.get\_device\_name()))

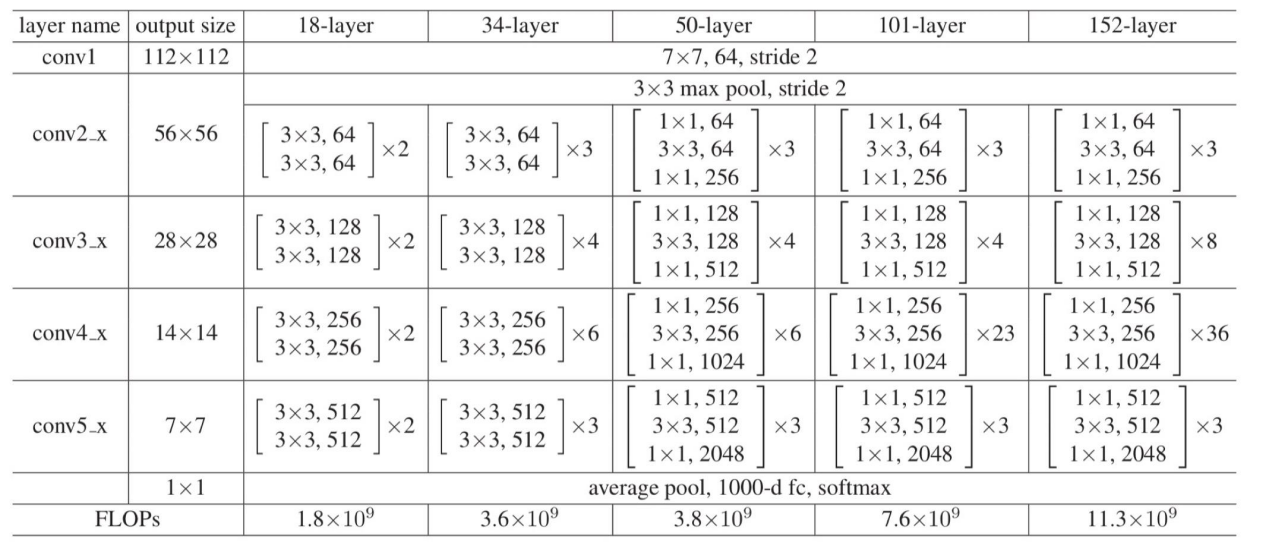
|  |
| --- |
| Layer (type) Output Shape Param #  ================================================================  Conv2d-1 [-1, 64, 112, 112] 9,408  BatchNorm2d-2 [-1, 64, 112, 112] 128  ReLU-3 [-1, 64, 112, 112] 0  MaxPool2d-4 [-1, 64, 56, 56] 0  Conv2d-5 [-1, 64, 56, 56] 36,864  ...  BasicBlock-66 [-1, 512, 7, 7] 0  AdaptiveAvgPool2d-67 [-1, 512, 1, 1] 0  Linear-68 [-1, 2] 1,026  ================================================================  Total params: 11,177,538  Trainable params: 11,177,538  Non-trainable params: 0  ----------------------------------------------------------------  Input size (MB): 0.57  Forward/backward pass size (MB): 62.79  Params size (MB): 42.64  Estimated Total Size (MB): 106.00  ----------------------------------------------------------------  1/70: 10308379\_1b6c72e180.jpg 1.834s  2/70: 1053149811\_f62a3410d3.jpg 0.010s  ...  70/70: Hormiga.jpg 0.005s  device:cuda total time:2.4s mean:0.034s  GPU name:GeForce RTX 2060 |



在图像分类中，PyTorch提供了很多经典的模型，例如：

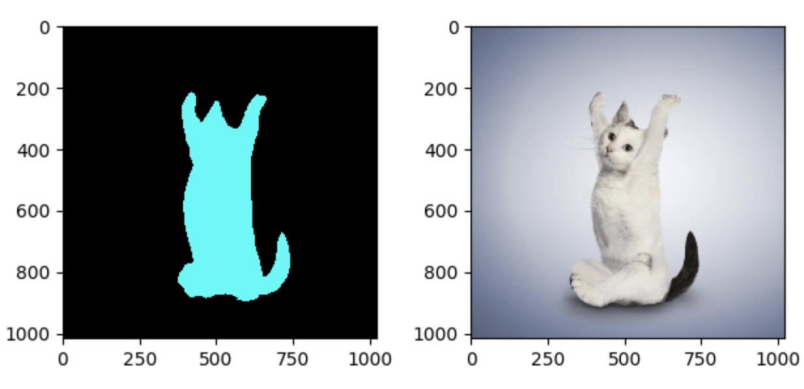


其中，我们使用的为resnet模型，其结构如下



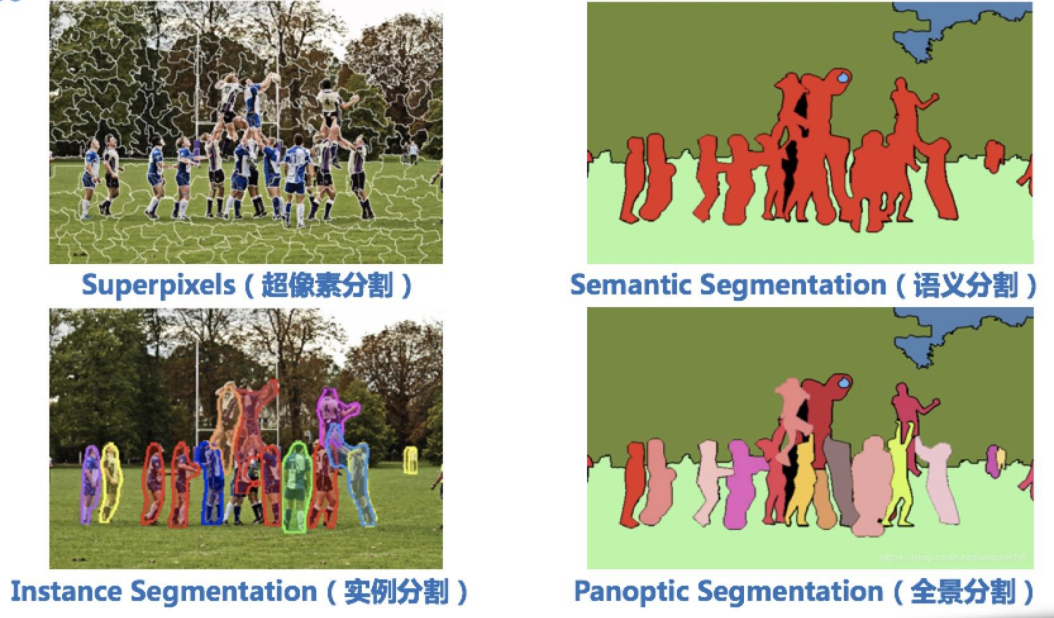
## 8.2.图像分割

图像分割：将图像每一个像素分类



图像分割分类：

1.超像素分割：少量超像素代替大量像素，常用于图像预处理  
 2. 语义分割：逐像素分类，无法区分个体  
 3. 实例分割：对个体目标进行分割，像素级目标检测  
 4. 全景分割：语义分割结合实例分割



模型如何完成图像分割？



模型

torch.hub介绍：

为了调用各种经典机器学习模型，今后你不必重复造轮子了。

Facebook推出PyTorch Hub，一个包含计算机视觉、自然语言处理领域的诸多经典模型的聚合中心，让你调用起来更方便。

有多方便？

图灵奖得主Yann LeCun强烈推荐，无论是ResNet、BERT、GPT、VGG、PGAN还是MobileNet等经典模型，只需输入一行代码，就能实现一键调用。

PyTorch Hub的使用简单到不能再简单，不需要下载模型，只用了一个torch.hub.load()就完成了对图像分类模型AlexNet的调用。

示例：

import torch

model = torch.hub.load('pytorch/vision', 'alexnet', pretrained=True)

model.eval()

PyTorch Hub允许用户对已发布的模型执行以下操作：

1、查询可用的模型;  
 2、加载模型;  
 3、查询模型中可用的方法。

1、查询可用的模型  
 用户可以使用torch.hub.list()这个API列出repo中所有可用的入口点。比如你想知道PyTorch Hub中有哪些可用的计算机视觉模型：

>>> torch.hub.list('pytorch/vision')

>>>

['alexnet',

'deeplabv3\_resnet101',

'densenet121',

...

'vgg16',

'vgg16\_bn',

'vgg19',

'vgg19\_bn']

2、加载模型

在上一步中能看到所有可用的计算机视觉模型，如果想调用其中的一个，也不必安装，只需一句话就能加载模型。

model = torch.hub.load('pytorch/vision', 'deeplabv3\_resnet101', pretrained=True)

至于如何获得此模型的详细帮助信息，可以使用下面的API：

print(torch.hub.help('pytorch/vision', 'deeplabv3\_resnet101'))

3、查看模型可用方法

从PyTorch Hub加载模型后，你可以用dir(model)查看模型的所有可用方法。以bertForMaskedLM模型为例：

>>> dir(model)

>>>

['forward'

...

'to'

'state\_dict',

]

如果你对forward方法感兴趣，使用help(model.forward) 了解运行运行该方法所需的参数。

>>> help(model.forward)

>>>

Help on method forward in module pytorch\_pretrained\_bert.modeling:

forward(input\_ids, token\_type\_ids=None, attention\_mask=None, masked\_lm\_labels=None)

...

具体hub的更多操作参见https://pytorch.org/hub

接下来，我们需要使用deeplabv3\_resnet101模型，当执行：

model = torch.hub.load('pytorch/vision', 'deeplabv3\_resnet101', pretrained=True)

代码时，会下载从GitHub存储库中加载具有预训练权重的模型，默认分支为master，上面代码第一个参数为github项目名，第二个参数为模型名称，第三个参数为是否使用与训练好的模型权重。

如果不想在线下载，也可以将提前下载好的模型及参数放入对应的目录，如：

pytorch\_vision\_master放入C:\Users\用户名\.cache\torch\hub目录下，

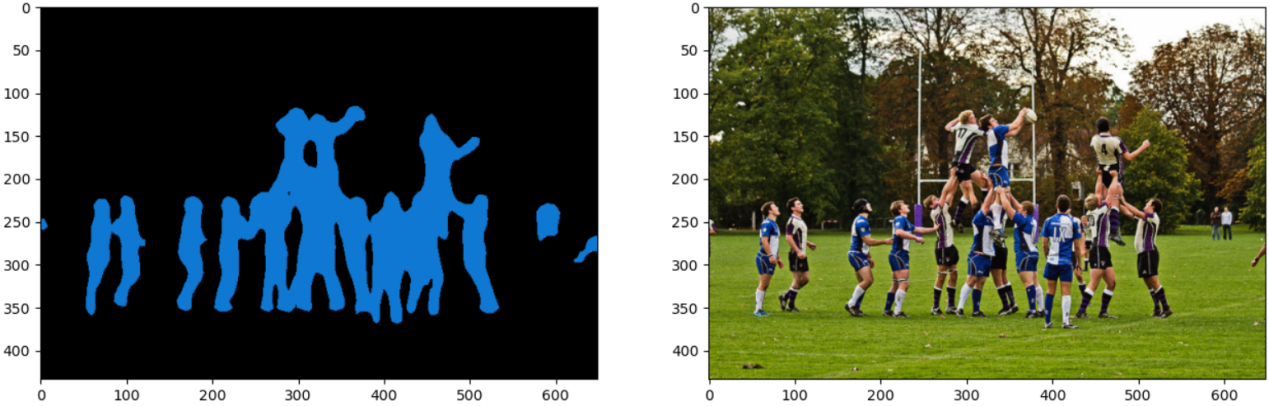
resnet101-5d3b4d8f.pth和deeplabv3\_resnet101\_coco-586e9e4e.pth放入

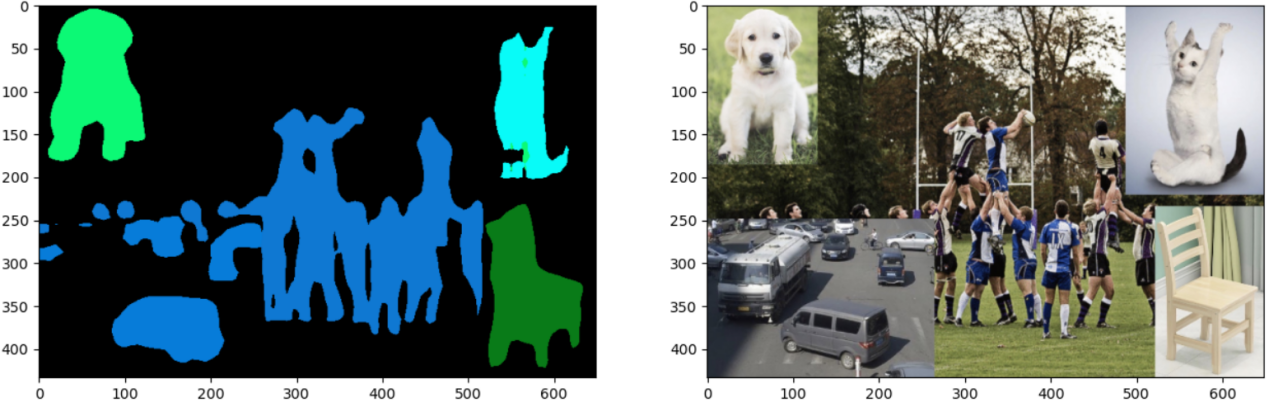
C:\Users\用户名\.cache\torch\checkpoints目录下

代码实现：

*# -\*- coding: utf-8 -\*-***import** os  
**import** time  
**import** torch  
**import** torchvision.transforms **as** transforms  
**from** PIL **import** Image  
**from** matplotlib **import** pyplot **as** plt  
  
BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
device = torch.device(**"cuda" if** torch.cuda.is\_available() **else "cpu"**)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
  
 *# 指定需要分割的图片路径* path\_img = os.path.join(BASE\_DIR, **"demo\_img1.png"**)  
 *# path\_img = os.path.join(BASE\_DIR, "demo\_img2.png")  
 # path\_img = os.path.join(BASE\_DIR, "demo\_img3.png")  
  
 # 图片预处理* preprocess = transforms.Compose([  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  
 ])  
  
 *# 1. 加载图片* input\_image = Image.open(path\_img).convert(**"RGB"**)  
 *# 加载模型，使用torch.hub工具进行模型的加载和使用* model = torch.hub.load(**'pytorch/vision'**, **'deeplabv3\_resnet101'**, pretrained=**True**)  
 *# 模型设置为测试模式* model.eval()  
  
 *# 2. 图片预处理* input\_tensor = preprocess(input\_image)  
 *# 扩展维度* input\_bchw = input\_tensor.unsqueeze(0)  
  
 *# 3. 指定设备* **if** torch.cuda.is\_available():  
 input\_bchw = input\_bchw.to(device)  
 model.to(device)  
  
 *# 4. 开始测试* **with** torch.no\_grad():  
 tic = time.time()  
 *# 打印输入的shape* print(**"input img tensor shape:{}"**.format(input\_bchw.shape))  
 *# 输出为字典，这里获取字典的'out'对应的value* output\_4d = model(input\_bchw)[**'out'**]  
 *# 输出的维度为batch\_size，这里取出第0个* output = output\_4d[0]  
 *# 打印测试时间* print(**"pass: {:.3f}s use: {}"**.format(time.time() - tic, device))  
 *# 打印输出shape，输出的第一个维度为类别，deeplabv3\_resnet101支持21个类别* print(**"output img tensor shape:{}"**.format(output.shape))  
 *# 获取每个像素的类别* output\_predictions = output.argmax(0)  
  
 *# 5. 调整各类别颜色* palette = torch.tensor([2 \*\* 25 - 1, 2 \*\* 15 - 1, 2 \*\* 21 - 1])  
 *# 调整21个类别的三基色* colors = torch.as\_tensor([i **for** i **in** range(21)])[:, **None**] \* palette  
 colors = (colors % 255).numpy().astype(**"uint8"**)  
  
 *# 对各类别上色  
 # fromarray：将array转换为image，先将输出转换为byte，然后指定为cpu，再转换为numpy* r = Image.fromarray(output\_predictions.byte().cpu().numpy()).resize(input\_image.size)  
 *# 为图像增加调色板* r.putpalette(colors)  
 plt.subplot(121).imshow(r)  
 plt.subplot(122).imshow(input\_image)  
 plt.show()  
  
 *# 21个类别，其中第一个类别为背景* classes = [**'\_\_background\_\_'**,  
 **'aeroplane'**, **'bicycle'**, **'bird'**, **'boat'**,  
 **'bottle'**, **'bus'**, **'car'**, **'cat'**, **'chair'**,  
 **'cow'**, **'diningtable'**, **'dog'**, **'horse'**,  
 **'motorbike'**, **'person'**, **'pottedplant'**,  
 **'sheep'**, **'sofa'**, **'train'**, **'tvmonitor'**]

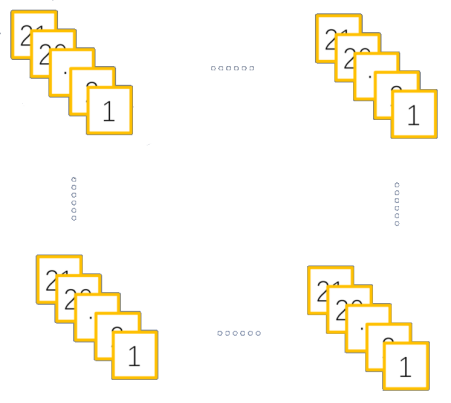
|  |
| --- |
| input img tensor shape:torch.Size([1, 3, 433, 649])  pass: 1.511s use: cuda  output img tensor shape:torch.Size([21, 433, 649]) |



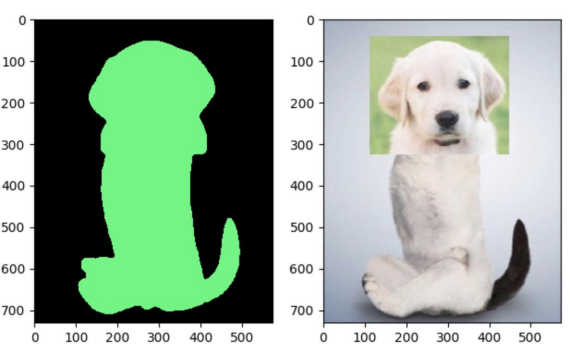
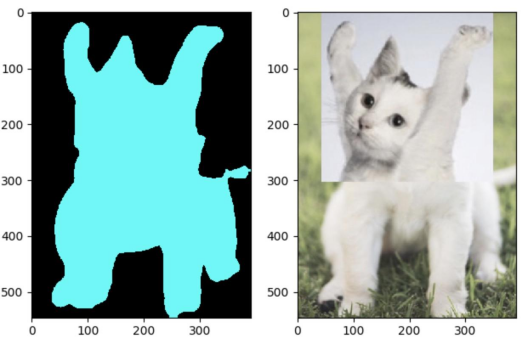


模型如何完成图像分割？  
 答：图像分割由模型与人类配合完成  
 模型：将数据映射到特征  
 人类：定义特征的物理意义，解决实际问题

图像分割输出特征图：



图像分割的思考：



如上图，左边是将猫的头与狗的身体拼接，右边是将狗的头与猫的身体拼接，最后在预测时，左边都预测为了猫，右侧都预测为了狗，由此可见，当预测时，是以头部为主要类别的预测，然后与头部相连的部分都预测为同一类。

## 8.3.目标检测

目标检测：判断图像中目标的位置

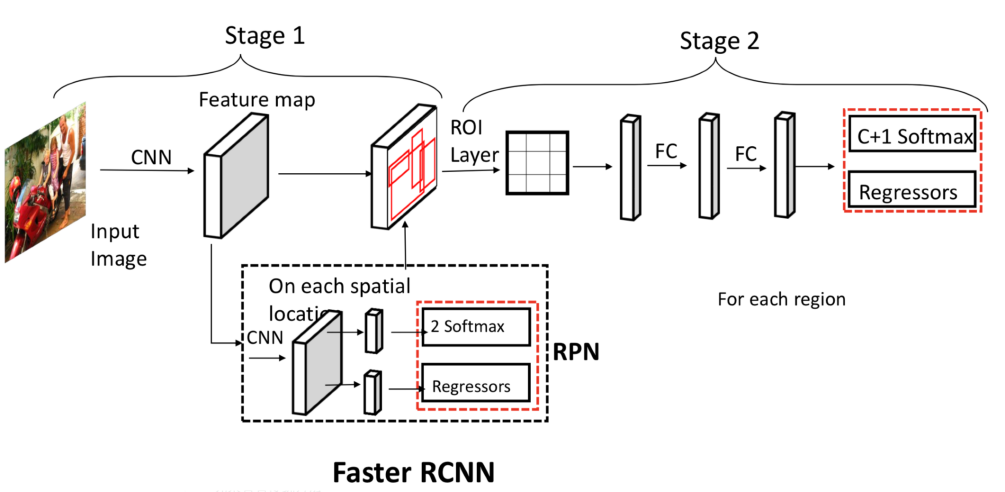
目标检测两要素  
 1. 分类：分类向量[p0, …, pn]  
 2. 回归：回归边界框[x1, y1, x2, y2]



模型如何完成目标检测

将3D张量映射到两个张量  
 1. 分类张量： shape为 [N, c+1]  
 2. 边界框张量： shape为 [N, 4]

边界框数量N如何确定？



代码实现：

*# -\*- coding: utf-8 -\*-***import** os  
**import** time  
**import** torch.nn **as** nn  
**import** torch  
**import** numpy **as** np  
**import** torchvision.transforms **as** transforms  
**import** torchvision  
**from** PIL **import** Image  
**from** matplotlib **import** pyplot **as** plt  
  
BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
device = torch.device(**"cuda" if** torch.cuda.is\_available() **else "cpu"**)  
  
*# coco数据集支持的类别，共91 种*COCO\_INSTANCE\_CATEGORY\_NAMES = [  
 **'\_\_background\_\_'**, **'person'**, **'bicycle'**, **'car'**, **'motorcycle'**, **'airplane'**, **'bus'**,  
 **'train'**, **'truck'**, **'boat'**, **'traffic light'**, **'fire hydrant'**, **'N/A'**, **'stop sign'**,  
 **'parking meter'**, **'bench'**, **'bird'**, **'cat'**, **'dog'**, **'horse'**, **'sheep'**, **'cow'**,  
 **'elephant'**, **'bear'**, **'zebra'**, **'giraffe'**, **'N/A'**, **'backpack'**, **'umbrella'**, **'N/A'**, **'N/A'**,  
 **'handbag'**, **'tie'**, **'suitcase'**, **'frisbee'**, **'skis'**, **'snowboard'**, **'sports ball'**,  
 **'kite'**, **'baseball bat'**, **'baseball glove'**, **'skateboard'**, **'surfboard'**, **'tennis racket'**,  
 **'bottle'**, **'N/A'**, **'wine glass'**, **'cup'**, **'fork'**, **'knife'**, **'spoon'**, **'bowl'**,  
 **'banana'**, **'apple'**, **'sandwich'**, **'orange'**, **'broccoli'**, **'carrot'**, **'hot dog'**, **'pizza'**,  
 **'donut'**, **'cake'**, **'chair'**, **'couch'**, **'potted plant'**, **'bed'**, **'N/A'**, **'dining table'**,  
 **'N/A'**, **'N/A'**, **'toilet'**, **'N/A'**, **'tv'**, **'laptop'**, **'mouse'**, **'remote'**, **'keyboard'**, **'cell phone'**,  
 **'microwave'**, **'oven'**, **'toaster'**, **'sink'**, **'refrigerator'**, **'N/A'**, **'book'**,  
 **'clock'**, **'vase'**, **'scissors'**, **'teddy bear'**, **'hair drier'**, **'toothbrush'**]  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
  
 *# path\_img = os.path.join(BASE\_DIR, "demo\_img1.png")* path\_img = os.path.join(BASE\_DIR, **"demo\_img2.png"**)  
  
 *# config* preprocess = transforms.Compose([  
 transforms.ToTensor(),  
 ])  
  
 *# 1. 加载数据与模型* input\_image = Image.open(path\_img).convert(**"RGB"**)  
 model = torchvision.models.detection.fasterrcnn\_resnet50\_fpn(pretrained=**True**)  
 model.eval()  
  
 *# 2. 图片预处理* img\_chw = preprocess(input\_image)  
  
 *# 3. 转换到指定设备* **if** torch.cuda.is\_available():  
 img\_chw = img\_chw.to(**'cuda'**)  
 model.to(**'cuda'**)  
  
 *# 4. 开始测试，输入不再是4d 张量，而是3d 组成的list* input\_list = [img\_chw]  
 **with** torch.no\_grad():  
 tic = time.time()  
 print(**"input img tensor shape:{}"**.format(input\_list[0].shape))  
 *# 同样，输出也是list，与输入一一对应* output\_list = model(input\_list)  
 *# 获取第一个输出，是一个字典，包含边界框、输出的score、输出的labels* output\_dict = output\_list[0]  
 print(**"pass: {:.3f}s"**.format(time.time() - tic))  
 *# 打印输出的字典的key、value* **for** k, v **in** output\_dict.items():  
 print(**"key:{}, value:{}"**.format(k, v))  
  
 *# 5. 可视化，分别获取输出的目标位置、得分、label* out\_boxes = output\_dict[**"boxes"**].cpu()  
 out\_scores = output\_dict[**"scores"**].cpu()  
 out\_labels = output\_dict[**"labels"**].cpu()  
  
 fig, ax = plt.subplots(figsize=(12, 12))  
 ax.imshow(input\_image, aspect=**'equal'**)  
  
 num\_boxes = out\_boxes.shape[0]  
 *# 显示的最大框数量* max\_vis = 40  
 *# 设置分类的概率，即只绘制大于这个概率的目标* thres = 0.5  
  
 **for** idx **in** range(0, min(num\_boxes, max\_vis)):  
  
 score = out\_scores[idx].numpy()  
 bbox = out\_boxes[idx].numpy()  
 class\_name = COCO\_INSTANCE\_CATEGORY\_NAMES[out\_labels[idx]]  
  
 **if** score < thres:  
 **continue** *# plt.Rectangle添加矩形框* ax.add\_patch(plt.Rectangle((bbox[0], bbox[1]), bbox[2] - bbox[0], bbox[3] - bbox[1], fill=**False**,  
 edgecolor=**'red'**, linewidth=3.5))  
 ax.text(bbox[0], bbox[1] - 2, **'{:s} {:.3f}'**.format(class\_name, score), bbox=dict(facecolor=**'blue'**, alpha=0.5),  
 fontsize=14, color=**'white'**)  
 plt.show()  
 plt.close()

|  |
| --- |
| input img tensor shape:torch.Size([3, 624, 1270])  pass: 2.186s  key:boxes, value:tensor([[2.1437e+01, 4.0840e+02, 5.6342e+01, 5.3993e+02],  [2.7507e+02, 4.1659e+02, 3.1846e+02, 5.2799e+02],  ...  [1.0730e+03, 4.9895e+02, 1.1603e+03, 6.1907e+02],  [3.9530e+02, 4.2951e+02, 4.2193e+02, 4.6254e+02]], device='cuda:0')  key:labels, value:tensor([ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  ...  1, 1, 1, 1, 31, 1, 1, 1, 1, 31], device='cuda:0')  key:scores, value:tensor([0.9860, 0.9852, 0.9780, 0.9779, 0.9774, 0.9739, 0.9500, 0.9464, 0.9456,  0.9088, 0.8751, 0.8721, 0.8549, 0.8533, 0.8455, 0.8064, 0.8006, 0.7799,  ...  0.2353, 0.2311, 0.2245, 0.2233, 0.2224, 0.2205, 0.2173, 0.2157, 0.2141,  0.2109], device='cuda:0') |

