学生实验报告

学号	1120192933	学院	计算机学院
姓名	李桐	专业	人工智能

商品识别模型训练

1 实验目的

（1）理解 pytorch 在机器学习中的应用。
（2）掌握使用 pytorch 搭建神经网络的方法。

2 实验原理

（1）pytorch 模型构建方法
（2）triple loss、circle loss 和 ID loss 原理
（3）Radam 原理
（4）WarmupLR 原理

3 实验条件与环境

要求	名称	版本要求	备注
编程语言	python	3.6 以上	
开发环境	dsw	无要求	
第三方工具包/库/插件	opencv-python	4.5 以上	
第三方工具包/库/插件	tqdm	4.32	
第三方工具包/	pytorch	1.0 以上	

库/插件			
第三方工具包/库/插件	mmdetection	1.2	
其他工具	无	无要求	
硬件环境	台式机、笔记本均可	无要求	

4 实验步骤及操作

步骤序号	1
步骤名称	构建数据集
步骤描述	使用 pytorch 的 dataloader 构建数据集。
代码及讲解	见下方

```
self.path='./data/i_train.json'
#path = './data/i_train.json'
with open(self.path, 'r') as f:
    self.x = json.load(f)
#数据处理
if kind=='train':
    self.transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])])  # 归一化
else:
    self.transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])])  # 归一化
```

在初始化的位置读入之前数据处理的 json 文件，后续读入图片以及标签。

```
def __getitem__(self, i):

    img=Image.open(self.x['images'][i]['file_name']).convert('RGB')
    label = self.x['annotations'][i]['category_id']

    img = self.transform(img) #图片处理

    label=np.array(label)
    label=torch.from_numpy(label)#转tensor，变成onehot需要用tensor
    label=label.to(torch.int64)
    label = torch.nn.functional.one_hot(label, 23).float()
    return img, label
```

getitem 函数。这部分传入序号 index，需要输出对应的图片以及标签。

(1)图片：根据图片路径列表以及 index，获得相应的图片路径。利用 PIL 库读取，同时利用 transforms 进行图片处理，转为 tensor 形式。

(2)标签：根据标签列表以及 index，获得相应的标签。同时转为 one-hot 形式，以供训练。

步骤序号	2
步骤名称	构建检测模型
步骤描述	使用 imagenet 作为预训练模型，构建 resnet 模型。
代码及讲解	ResNet 应该算是老牌、经典的模型了，模型的设置也基本是固

	定的，各个参数都有约定俗成的命名。我会把 ResNet 分为两个系列，ResNet18 与 ResNet34 是同一个系列，模块的构成相似，而 ResNet50、ResNet101、ResNet152 是另外一个系列，比简单的 ResNet18、34 的层数更多，而且新增了瓶颈结构，通道数会先增加后减少。
	`class ResNet(nn.Module):`
	`  def __init__(self, block, num_blocks, num_classes=3):`
	`  super(ResNet, self).__init__()`
	`  self.in_planes = 64`
	`  # 网络输入部分`
	`  self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)`
	`  self.bn1 = nn.BatchNorm2d(64)`
	`  self.relu = nn.ReLU(inplace=True)`
	`  self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)`
	`  # 中间卷积部分`
	`  self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)`
	`  self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)`
	`  self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)`
	`  self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)`
	`  # 平均池化和全连接层`
	`  self.avgpool = nn.AvgPool2d(7, stride=1)`
	`  self.linear = nn.Linear(512 * block.expansion, num_classes)`
	`  for m in self.modules():`
	`  if isinstance(m, nn.Conv2d):`
	预训练的 ResNet50 其实 pytorch 直接就提供了，可以这样调用:
	`import torchvision`
	`model=torchvision.models.resnet50(pretrained=True)`
	`model.fc=nn.Linear(model.fc.in_features,23)`
	`model=model.to(device)`

步骤序号	3
步骤名称	构建损失函数
步骤描述	构建损失函数。
代码及讲解	`criterion = nn.MSELoss().to(device)` 直接调用 torch.nn 的损失函数，一般用 MSE 或者交叉熵损失函数，分类一般其实用交叉熵，这里我只用了 MSE。

步骤序号	4
步骤名称	构建优化器
步骤描述	使用 RAdam+lookahead 或者 Ranger 优化器。

代码及讲解	`class Ranger(Optimizer):`
	`  def __init__(self, params, lr=1e-3,  # lr`
	`  alpha=0.5, k=6, N_sma_threshhold=5,  # Ranger options`
	`  betas=(.95, 0.999), eps=1e-5, weight_decay=1e-5,  # Adam options`
	`  use_gc=True, gc_conv_only=False, gc_loc=True`
	`  ):`
	`  # parameter checks`
	`  if not 0.0 <= alpha <= 1.0:`
	`  raise ValueError(f'Invalid slow update rate: {alpha}')`
	`  if not 1 <= k:`
	`  raise ValueError(f'Invalid lookahead steps: {k}')`
	`  if not lr > 0:`
	`  raise ValueError(f'Invalid Learning Rate: {lr}')`
	`  if not eps > 0:`
	`  raise ValueError(f'Invalid eps: {eps}')`
	`  # prep defaults and init torch.optim base`
	`  defaults = dict(lr=lr, alpha=alpha, k=k, step_counter=0, betas=betas,`
	`  N_sma_threshhold=N_sma_threshhold, eps=eps, weight_decay=weight_decay)`
	`  super().__init__(params, defaults)`
	`  # adjustable threshold`
	`  self.N_sma_threshhold = N_sma_threshhold`
	`  # look ahead params`
	`  self.alpha = alpha`
	`  self.k = k`
	RAdam 可以说是优化者在培训开始时建立的最佳基础。RAdam 利用动态整流器根据方差调整 Adam 的自适应动量，并有效地提供自动预热，根据当前数据集定制，以确保扎实的训练开始。
	LookAhead 受到深度神经网络损失表面理解的最新进展的启发，

	并在整个训练期间提供了稳健和稳定探索的突破。“减少了对广泛超参数调整的需求”，同时实现“以最小的计算开销实现不同深度学习任务的更快收敛”。
	因此，两者都在深度学习优化的不同方面提供了突破，并且这种组合具有高度协同性，可能为您的深度学习结果提供最佳的两种改进。因此，对更加稳定和强大的优化方法的追求仍在继续，通过结合两项最新突破（RAdam + LookAhead），Ranger 的整合有望为深度学习提供又一步。
	这里根据 Ranger 原论文构建了优化器，实现了 RAdam+lookahead 的功能。与 torch 提供的优化器的实例化、使用方法相同。
	`def __setstate__(self, state):`
	`print("set state called")`
	`super(Ranger, self).__setstate__(state)`
	`def step(self, closure=None):`
	`loss = None`
	`for group in self.param_groups:`
	`for p in group['params']:`
	`if p.grad is None:`
	`continue`
	`grad = p.grad.data.float()`
	`if grad.is_sparse:`
	`raise RuntimeError(`
	`'Ranger optimizer does not support sparse gradients')`

步骤序号	5
步骤名称	选择学习率调整策略
步骤描述	进行学习率的调整。

代码及讲解	`if epoch%30==0:`
	`for p in optimizer.param_groups:`
	`p['lr'] *= 0.8`
	这里我设置的是手动调整的方法。直接每 30 轮衰减一次。

步骤序号	6
步骤名称	商品检测模型训练
步骤描述	对上一节课程构建神经网络进行训练。
代码及讲解	python mmdetection/tools/train.py 3-1.py 直接利用提供的 train 文件，并且利用上一节写的 config 进行训练就行了。

步骤序号	7
步骤名称	商品识别模型训练
步骤描述	对本节课程构建的神经网络进行训练。
代码及讲解	首先，将模型转移至设备（cuda 或者 cpu）上，根据参数确定优化器和损失函数。若有已经训练的模型，则加载训练模型，并且自适应设备。 之后逐轮开始训练。在设定的 epoch 轮次内，开始循环，加载 data_loader 中的数据，图片及标签迁移到设备上。 图片输入到模型中，生成预测标签 pre_labels。预测标签以及真是标签是一个 23 维的向量，按照之前定义的 loss 函数，计算损失值，反向传播、更新参数。

之后的就是一些辅助操作了，包括输出每一个 epoch 的第一个 batch 的 loss、手动释放内存、利用 validation 函数进行验证、保存准确率最高的训练模型。这里我设定的是每一轮训练后都需要进行验证，并且进行学习率的指数衰减。

进行训练前，需要定义训练集、测试集，定义模型，输入到 train 函数中即可开始训练。

```
for epoch in range(100):

    model.train()  # 训练模式：允许使用批样本归一化
    # 循环外可以自行添加必要内容
    for index, data in enumerate(train_loader, 0):
        images, true_labels = data
        images=images.to(device)
        true_labels=true_labels.to(device)

        optimizer.zero_grad()
        # 前向传播
        pre_labels=model(images)
        # 计算损失
        loss = criterion(pre_labels, true_labels)
        #print(pre_labels.shape, true_labels.shape)
        # 后向传播
        loss.backward()
        # 更新模型
        optimizer.step()

        if index==0:
            print('Train - Epoch %d, Batch: %d, Loss: %f' % (epoch, index, loss.data.item()))

    # 手动释放内存
    del images, true_labels, pre_labels
    x=validation(data_loader=val_loader, model=model)
    print('acc:',x)
    test.append(x)
    if x>x_max:
        x_max=x
```

5 实验结果及分析讨论

（1）最终结果的具体结果（文字说明）

成功训练，训练结果很棒。

上一节的训练最终结果:bbox_mAP: 0.8890, bbox_mAP_50: 1.0000, bbox_mAP_75: 0.9950, bbox_mAP_s: -1.0000, bbox_mAP_m: -1.0000, bbox_mAP_l: 0.8890, bbox_mAP_copypaste: 0.889 1.000 0.995 -1.000 -1.000 0.889

本节识别的训练结果最后可以达到 100%的准确率。

（2）最终结果界面截图（界面截图）

上一节的截图:

```
2022-05-24 01:30:57,504 - mmdet - INFO -
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.889
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.995
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.889
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.911
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.911
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.911
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.911

2022-05-24 01:30:57,505 - mmdet - INFO - Exp name: 3-1.py
2022-05-24 01:30:57,506 - mmdet - INFO - Epoch(val) [12][51]    bbox_mAP: 0.8890, bbox_mAP_50: 1.0000, bbox_mAP_75: 0.9950, bbox_mAP_s: -1.0000, bbo
x_mAP_m: -1.0000, bbox_mAP_l: 0.8890, bbox_mAP_copypaste: 0.889 1.000 0.995 -1.000 -1.000 0.889
(PyTorch-1.8) [ma-user work]$
```

本节的截图:

```
Train - Epoch 16, Batch: 0, Loss: 0.003743
验证集数据总量： 51 预测正确的数量： 51
当前模型在验证集上的准确率为： 1.0
acc: 1.0
Train - Epoch 17, Batch: 0, Loss: 0.002562
验证集数据总量： 51 预测正确的数量： 51
当前模型在验证集上的准确率为： 1.0
acc: 1.0
Train - Epoch 18, Batch: 0, Loss: 0.003060
验证集数据总量： 51 预测正确的数量： 51
当前模型在验证集上的准确率为： 1.0
acc: 1.0
Train - Epoch 19, Batch: 0, Loss: 0.002480
验证集数据总量： 51 预测正确的数量： 51
当前模型在验证集上的准确率为： 1.0
acc: 1.0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0392156862745098, 0.1568627450980392, 0.47058823529411764, 0.7254901960784313, 0.8431372549019608, 0.8823529411764706, 0
.9607843137254902, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0392156862745098, 0.1568627450980392, 0.47058823529411764, 0.7254901960784313, 0.8431372549019608, 0.8823529411764706, 0
.9607843137254902, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

（3）最终结果的说明（注意事项或提醒）

本节课的 6、7 都是训练，所以出现了两个结果，上面已经清晰的表示出来了。不过其实我在上一节实验的时候就已经对上一节的进行了训练了……本节课的实验其实是运行了 100 次，但是忘记截图了，又补充跑了一遍 20epoch 的截了个图（就是上面的图），因为后面发现 15epcoh 左右就已经很好了，所以就选择了 20 个 epoch。对了，这里用的数据集也是延续了上一节课的 demo 数据集。直接把训练集复制一遍作为测试集，训练集上进行数据增强。

（4）最终结果的解读与讨论

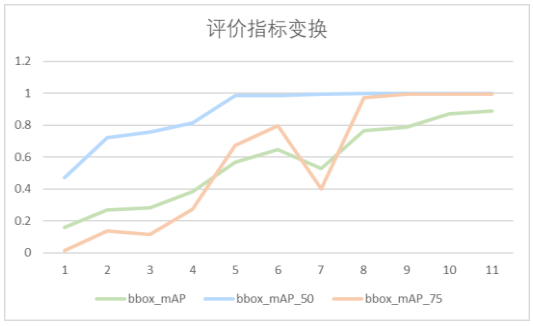
上一节课的：

我其实在开始训练之前一直觉得可能效果会比较差呢。没想到啊没想到！居然肉眼可见的效果变好了。可以简单了解一下什么是 mAP：

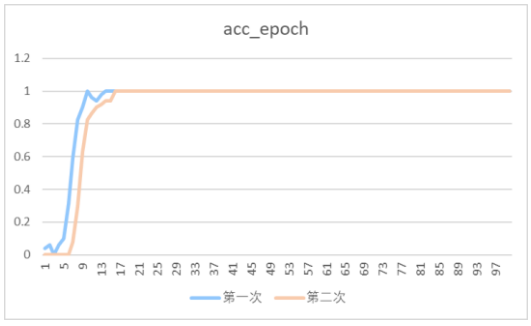
对于目标检测而言，每一个类别都可计算出其 Precision 和 Recall，每个类别都可以得到一条 P-R 曲线，曲线下的面积就是 AP 的值。

假设存在 M 张图片，对于其中一张图片而言，其具有 N 个检测目标，其具有 K 个检测类别，使用检测器得到了 S 个 Bounding Box(BB)，每个 BB 里包含 BB 所在的位置以及 K 个类的得分 C。利用 BB 所在的位置可以得到与其对应的 GroundTruth 的 IOU 值。mAP 即 mean Average Precision，即各类 AP 的平均值。

简单理解就是评价检测好坏的一个指标。才运行了 12 个 epoch，mAP 效果逐渐上升！我认为是预训练模型的功劳！



上一节的



本一节的

本节课的：

本节课的实验是识别的实验，我其实跑了好多次的。每次实验的趋势其实都是差不多，前几个 epoch 效果比较差，可能只有零点几，但是大概 15epoch 左右就已经可以完美的达到 100%的准确率了。我感觉确实就是预训练模型的好处了，前面的特征提取已经训练好了，只需要微调最后的全连接层就可以了。

6 收获与体会

总体上理解 pytorch 在机器学习中的应用，掌握使用 pytorch 搭建神经网络的方法掌握了 pytorch 模型构建方法、triple loss、circle loss 和 ID loss 原理、Radam 原理、WarmupLR 原理。

以前对 Radam 和 LookAhead 了解得比较少：

RAdam 可以说是优化者在培训开始时建立的最佳基础。RAdam 利用动态整流器根据方差调整 Adam 的自适应动量，并有效地提供自动预热，根据当前数据集定制，以确保扎实的训练开始。

LookAhead 受到深度神经网络损失表面理解的最新进展的启发，并在整个训练期间提供了稳健和稳定探索的突破。“减少了对广泛超参数调整的需求”，同时实现“以最小的计算开销实现不同深度学习任务的更快收敛”。

因此，两者都在深度学习优化的不同方面提供了突破，并且这种组合具有高度协同性，可能为您的深度学习结果提供最佳的两种改进。因此，对更加稳定和强大的优化方法的追求仍在继续，通过结合两项最新突破（RAdam + LookAhead），Ranger 的整合有望为深度学习提供又一步。

7 备注及其他

无。