Table of Contents:

Compose类

> `torchvision`是*pytorch*的一个图形库，它服务于*PyTorch*深度学习框架的，主要用来构建计算机视觉模型。`torchvision.transforms`主要是用于常见的一些图形变换。以下是`torchvision`的构成：

> 1. `torchvision.datasets`：一些加载数据的函数及常用的数据集接口；
> 2. `torchvision.models`：包含常用的模型结构（含预训练模型），例如*AlexNet*、*VGG*、*ResNet*等；
> 3. `torchvision.transforms`：常用的图片变换，例如裁剪、旋转等；
> 4. `torchvision.utils`：其他的一些有用的方法。

这个类的主要作用是串联多个图片变换的操作，构造参数是一个列表，包括了一系列要进行的操作，例如，我们的项目中有如下代码：

```
def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
data = PennFudanDataset('..\\data\\PennFudanPed', get_transform(train= True))
self.transforms = transforms
print(self.transforms)
```

通过`get_transform`函数得到的`Compose`对象，包含一个操作：将图像转换成Tensor类型，如果是训练集，对图像依概率进行翻转操作。

DataLoader类

初始化DataLoader类的参数

> *dataset*：加载的数据集*(Dataset*对象*)* *batch_size : batch size*，一次训练使用的样本数 *shuffle:*：是否将数据打乱 *sampler*： 样本抽样 *num_workers*：使用多进程加载的进程数，*0*代表不使用多进程 *collate_fn*：如何将多个样本数据拼接成一个*batch*，一般使用默认的拼接方式即可 *pin_memory*：是否将数据保存在*pin*

*memory*区，*pin memory*中的数据转到*GPU*会快一些 *drop_last*：*dataset*中的数据个数可能不是*batch_size*的整数倍，*drop_last*为*True*会将多出来不足一个*batch*的数据丢弃

训练模型

定义data_loader

```
dataset = PennFudanDataset('..\\data\\PennFudanPed', get_transform(train=True))
indices = torch.randperm(len(dataset)).tolist()
'''
[61, 135, 2, 15, 89, 105, 137, 22, 47, 107, 56, 106, 155, 81, 95, 41, 9, 78, 167, 103, 88,
77, 14, 138, 99, 72, 32, 146, 164, 20, 162, 24, 1, 79, 16, 123, 143, 55, 6, 33, 38, 132,
168, 98, 85, 30, 71, 151, 93, 86, 161, 43, 113, 39, 42, 112, 124, 74, 18, 116, 110, 70,
104, 4, 62, 21, 50, 117, 65, 133, 91, 3, 69, 10, 114, 80, 96, 84, 156, 23, 66, 119, 59,
166, 12, 17, 127, 7, 51, 11, 109, 63, 64, 67, 121, 31, 148, 27, 160, 60, 102, 140, 141, 68,
126, 58, 131, 34, 46, 134, 118, 139, 73, 150, 157, 149, 100, 49, 111, 75, 26, 94, 154, 163,
152, 92, 115, 28, 90, 25, 82, 57, 130, 37, 153, 5, 159, 165, 125, 87, 145, 129, 0, 83, 169,
45, 52, 147, 144, 97, 158, 108, 35, 44, 36, 120, 13, 8, 101, 76, 142, 53, 128, 48, 29, 54,
122, 40, 136, 19]
'''
dataset = torch.utils.data.Subset(dataset, indices[:10])
data_loader = torch.utils.data.DataLoader(
        dataset, batch_size=2, shuffle=True, num_workers=4,
        collate_fn=utils.collate_fn)
```

由于 `gpu` 内存不够，只能使用 `cpu` 进行训练，因此只选取少量数据看看能不能够进行训练。

训练用的data_loader，每次获取数据时会得到两个对象， `img` ，和 `target` 。 `img` 已经变成内容为0-1， `target` 中包含待检测目标的识别框四个角的坐标。

模型根据 `img` 得到输出 `out` ， `out` 中包含信息 `boxes` 等，即模型得到的目标检测框，再跟目标输出 `target` 中的对应信息进行比较，计算误差，从而进行梯度下降的操作。

定义优化器

```
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
```

使用优化器对模型中的参数进行更新，构造优化器对象需要指定模型中需要更新的参数，除了上述代码中的方式，官网还提示可以用另外一种方式指定更新特定层的参数：

```
optim.SGD([
                {'params': model.base.parameters()},
                {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

这意味着 `model.base` 的参数将使用默认的学习率 `1e-2` ， `model.classifier` 的参数将使用 `1e-3` 的学习率，并且动量 `0.9` 会用于所有参数。

正向传播

> *The behavior of the model changes depending if it is in training or evaluation mode.*

模型的输出会根据模型进行改变，训练时输入两个参数img和target。模型根据img得到输出，并于目标输出中的信息进行比较，计算误差。

训练模式下，模型输出的就是误差信息。

```
loss_dict = model(images, targets)
```

反向传播

```
optimizer.zero_grad()
losses.backward()
optimizer.step()
```

更新参数前，要首先将梯度初始为0，以下面的手动梯度下降为例：

> “

```
weights = [0] * n
alpha = 0.0001
max_Iter = 50000
for i in range(max_Iter):
    loss = 0
    d_weights = [0] * n
    for k in range(m):
        h = dot(input[k], weights)
        d_weights = [d_weights[j] + (label[k] - h) * input[k][j] for j in range(n)]
        loss += (label[k] - h) * (label[k] - h) / 2
    d_weights = [d_weights[k]/m for k in range(n)]
    weights = [weights[k] + alpha * d_weights[k] for k in range(n)]
    if i%10000 == 0:
        print "Iteration %d loss: %f"%(i, loss/m)
        print(weights)
```

`optimizer.zero_grad()` 对应 `d_weights = [0] * n`，因为一个batch的loss关于weight的导数是所有sample的loss关于weight的导数的累加和。

`loss.backward()` 对应 `d_weights = [d_weights[j] + (label[k] - h) * input[k][j] for j in range(n)]`。即反向传播求梯度。

`optimizer.step()` 对应 `weights = [weights[k] + alpha * d_weights[k] for k in range(n)]`，即更新所有参数。

加载微调后的模型进行预测

每个 `epoch` 后保存模型参数，以后加载模型后，载入保存的参数。进行目标检测。

```
for epoch in range(num_epochs):    # train for one epoch, printing every 10 iterations
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
    # update the learning rate
    lr_scheduler.step()
    # 保存参数
    utils.save_on_master({
        'model': model.state_dict()
    },
        '..\\models\\model_{}.pth'.format(epoch))
```

```
    # evaluate on the test dataset
    evaluate(model, data_loader_test, device=device)
```

```
model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=False,
num_classes=num_classes)model.to(device)model.eval()save =
torch.load('..\\models\\model_0.pth')model.load_state_dict(save['model'])
```

由于显卡内存不够，使用cpu进行训练，训练次数不多，检测效果一般。

总结

今天基本完成了 `PyTorch` 上第一个图像方面的教程。

阅读了几篇文章，https://zhuanlan.zhihu.com/p/37998710 https://zhuanlan.zhihu.com/p/31426458

文章中对Mask R-CNN和Fast RCNN进行了介绍。

虽然对模型进行了训练，并且加载参数进行了检测。但模型结构并不是我们定义的，只是加载了这个已经定义并预训练的模型进行了一些操作。对于很多内容还没有清晰的认识。接下来不再学习其他的教程，而是从小的demo入手，尝试亲手构建模型。并且理论方面的知识还要加深。

存在的问题

1、不了解 `Mask R-CNN` 的结构。

2、不了解每种梯度下降算法的主要特点，以及实现方式。

3、不了解学习率更新方面的知识。

后续任务

1、找一些教程，要包含模型的实现，不用太复杂。

2、找 `Mask R-CNN` 的源代码，或一些比较详细的介绍进行阅读，了解其结构，以及相关原理。

3、学习梯度下降算法，并结合 `PyTorch` 。

4、学习学习率调整算法，并结合 `PyTorch` 。