深度学习实战-图像识别篇

预训练模型图像分类

POET

2024 年 2 月 23 日

上一章我们从网络上爬取图片，制作了自己的图像数据集，并按照七三的比例将其分为训练集和测试集。这一章，我们学习使用预训练好的模型对图像中的物体进行分类。

1 载入预训练图像分类模型

在 torchvision.model 中定义了许多模型用于计算机视觉的任务中：

- 图像分类
- 语义分割
- 目标检测
- 实例分割
- 人物关键点检测
- 视频分类

下面将给出部分预训练模型的例子：

```
import torchvision.models as models

resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
shufflenet = models.shufflenet_v2_x1_0()
mobilenet = models.mobilenet_v2()
resnext50_32x4d = models.resnext50_32x4d()
wide_resnet50_2 = models.wide_resnet50_2()
mnasnet = models.mnasnet1_0()
```

当 pretrained 参数为 true 时，就导入了训练的模型。

2 准备数据集

这一章使用上一章抓取的图片进行训练，这个数据集只包括四种不同的猫。

2.1 加载数据集

加载数据集代码如下所示：

```
from torchvision import datasets, transforms

# 在训练集上：扩充、归一化
# 在验证集上：归一化
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456,
            0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456,
            0.406], [0.229, 0.224, 0.225])
    ]),
}
#数据集存放路径
data_dir = 'data/hymenoptera_data'
```

```
21      #使用torchvision.datasets.ImageFolder类快速封装
            数据集
22      #此处使用了lambda语法
23      image_datasets = {x: datasets.ImageFolder(os.
            path.join(data_dir, x),data_transforms[x])
24                        for x in ['train', 'val']}
25      dataloaders = {x: torch.utils.data.DataLoader(
            image_datasets[x], batch_size=4,shuffle=True,
             num_workers=4)
26                     for x in ['train', 'val']}
27      #读取数据集的数目
28      dataset_sizes = {x: len(image_datasets[x]) for x
            in ['train', 'val']}
29      #读取数据集中的图像种类
30      class_names = image_datasets['train'].classes
```

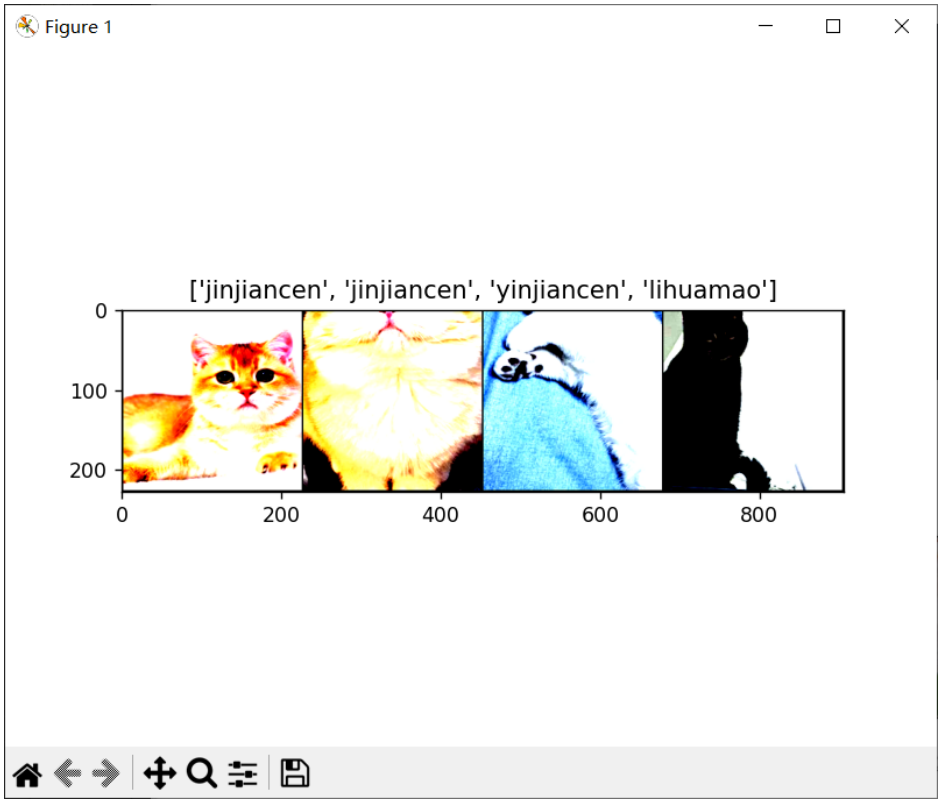
2.2 使用 matplotliob 可视化数据集

由于一个 batch 中的图像是保存在 tensor 中，一张维度是 [H,W,C]、值范围是 [0,255] 的图片需要经过 ToTensor 转换成维度是 [C,H,W]、值范围是 [0,1] 的 Tensor，在经过 Normalize 完成归一化。而使用 matplotlib 显示的图像需要对图像进行反向操作才能正常显示。

```
1       import matplotlib.pyplot as plt
2
3       def imshow(inp, title=None):
4           # 可视化一组 Tensor 的图片
5           inp = inp.numpy().transpose((1, 2, 0)) #转换
                图片维度
6           mean = np.array([0.485, 0.456, 0.406])
7           std = np.array([0.229, 0.224, 0.225])
8           inp = std * inp + mean #反向操作
9           inp = np.clip(inp, 0, 1)
```

```
10          plt.imshow(inp)
11          if title is not None:
12              plt.title(title)
13          plt.pause(0.001) # 暂停一会儿，为了将图片显
                示出来
14      # 获取一批训练数据
15      inputs, classes = next(iter(dataloaders['train'
            ]))
16      # 批量制作网格
17      out = torchvision.utils.make_grid(inputs)
18      imshow(out, title=[class_names[x] for x in
            classes])
```

运行结果如下图所示:



3 模型训练函数

虽然 ResNet 模型有训练好的参数，但是我们还是要进行训练。下面的代码进行了五次训练，并展示出每个 epoch 的准确度。这个函数包括加载模型，正向传播，计算损失函数，利用优化器进行学习优化的过程。

```
def train_model(model, criterion, optimizer,
    scheduler, num_epochs=25):
"""训练模型，并返回在验证集上的最佳模型和准确率
Args:
- model(nn.Module): 要训练的模型
- criterion: 损失函数
- optimizer(optim.Optimizer): 优化器
- scheduler: 学习率调度器
- num_epochs(int): 最大 epoch 数
Return:
- model(nn.Module): 最佳模型
- best_acc(float): 最佳准确率
"""
since = time.time()

best_model_wts = copy.deepcopy(model.state_dict
    ())
best_acc = 0.0

for epoch in range(num_epochs):
    print(f'Epoch␣{epoch}/{num_epochs␣-␣1}')
    print('-' * 10)

    # 训练集和验证集交替进行前向传播
    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()  # 设置为训练模式，可
```

```
                以更新网络参数
26          else:
27              model.eval()    # 设置为预估模式，不
                    可更新网络参数
28
29          running_loss = 0.0
30          running_corrects = 0
31
32          # 遍历数据集
33          for inputs, labels in dataloaders[phase
                ]:
34              inputs = inputs.to(device)
35              labels = labels.to(device)
36
37              # 清空梯度，避免累加了上一次的梯度
38              optimizer.zero_grad()
39
40              with torch.set_grad_enabled(phase ==
                    'train'):
41                  # 正向传播
42                  outputs = model(inputs)
43                  _, preds = torch.max(outputs, 1)
44                  loss = criterion(outputs, labels
                        )
45
46                  # 反向传播且仅在训练阶段进行优化
47                  if phase == 'train':
48                      loss.backward() # 反向传播
49                      optimizer.step()
50
51              # 统计loss、准确率
52              running_loss += loss.item() * inputs
                    .size(0)
```

```
53                  running_corrects += torch.sum(preds
                        == labels.data)
54              if phase == 'train':
55                  scheduler.step()
56
57              epoch_loss = running_loss /
                    dataset_sizes[phase]
58              epoch_acc = running_corrects.double() /
                    dataset_sizes[phase]
59
60              print(f'{phase} Loss: {epoch_loss:.4f}
                    Acc: {epoch_acc:.4f}')
61
62              # 发现了更优的模型，记录起来
63              if phase == 'val' and epoch_acc >
                    best_acc:
64                  best_acc = epoch_acc
65                  best_model_wts = copy.deepcopy(model
                        .state_dict())
66
67          print()
68
69      time_elapsed = time.time() - since
70      print(f'Training complete in {time_elapsed //
            60:.0f}m {time_elapsed % 60:.0f}s')
71      print(f'Best val Acc: {best_acc:4f}')
72
73      # 加载训练的最好的模型
74      model.load_state_dict(best_model_wts)
75      return model
```

4 使用 torchvision 微调模型

因为 ResNet 模型的全连接层输出为 1000 层，而我们的数据集输出应该仅为 4 层，所以要修改模型的 fc，也就是全连接层，代码如下：

4.1 不固定模型参数进行训练

注意，这种方法意味着使模型每一步都基于预训练的参数进行更新，一般我们需要将除了修改的输出层之外的层进行冻结，不更新参数，只基于前面层数提取出的特征进行训练。

```
model = models.resnet18(pretrained=True) # 加载预训练模型
num_ftrs = model.fc.in_features # 获取低级特征维度
model.fc = nn.Linear(num_ftrs, 2) # 替换新的输出层
model = model.to(device)
# 交叉熵作为损失函数
criterion = nn.CrossEntropyLoss()
# 所有参数都参加训练
optimizer_ft = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
# 每过 7 个 epoch 将学习率变为原来的 0.1
scheduler = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
model_conv = train_model(model_conv, criterion, optimizer_conv, exp_lr_scheduler, num_epochs=5)
```

4.2 固定模型参数

微调预训练模型，需要修改模型的内部结构，使其符合具体任务。模型所用框架不一样，在将其他框架编写的模型迁移到 PyTorch 中时，无法使它们兼容。此时可以采取 Pipeline 形式将预训练模型的参数固定，或者说将前一个模型的输出保存下来，将该输出作为 PyTorch 模型的输入。

采取这种思路，我们可以将模型除了输出层之外的所有层看成一个特征提取器。在训练模型时，这些层的权重不参与训练，不可优化。在 PyTorch 中将权重设置为不可训练，只需将 requires-grad 设置为 False 即可。例如，下述代码可以将 ResNet18 的所有层设置为不可训练。

```
model_conv = torchvision.models.resnet18(pretrained=True) # 加载预训练模型
for param in model_conv.parameters(): # 锁定模型所有参数
    param.requires_grad = False

num_ftrs = model_conv.fc.in_features  # 获取低级特征维度
model_conv.fc = nn.Linear(num_ftrs, 2) # 替换新的输出层

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# 只有最后一层全连接层fc，参加训练
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# 每过 7 个 epoch 将学习率变为原来的 0.1
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

```
    model_conv = train_model(model_conv, criterion,
        optimizer_conv,
                             exp_lr_scheduler,
                                num_epochs=5)
    visualize_model(model_conv)

    plt.ioff()
    plt.show()
```

结果如下图：

```
Epoch 0/4
----------
train Loss: 0.6532 Acc: 0.6516
val Loss: 0.3686 Acc: 0.8562

Epoch 1/4
----------
train Loss: 0.6901 Acc: 0.7254
val Loss: 0.3106 Acc: 0.9020

Epoch 2/4
----------
train Loss: 0.4054 Acc: 0.8484
val Loss: 0.7362 Acc: 0.7843

Epoch 3/4
----------
train Loss: 0.5336 Acc: 0.7500
val Loss: 0.2671 Acc: 0.8889

Epoch 4/4
----------
train Loss: 0.4952 Acc: 0.7951
val Loss: 0.2063 Acc: 0.9085

Training complete in 1m 0s
Best val Acc: 0.908497  CSDN @Xyzz1223
```

5 观察模型预测结果

接下来将模型切换到 eval 模式，也就是对测试集进行预测，并可视化结果：

```
    def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(
            dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2,
                    images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {
                    class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
visualize_model(model_conv)

plt.ioff()
plt.show()
```

预测结果如下图：

可以看出预测准确度很高。