实验 6、实验 7 迁移学习

一、　实验内容

实验内容：使用PyTorch 的数据集套件从本地加载数据的方法，迁移训练好的大型神经网络模型到自己模型中的方法，迁移学习与普通深度学习方法的效果区别，两种迁移学习方法的区别

实验环境：Python-3.11;PyTorch-cu121;CUDA-12.1;VS code/JupyterNotebook

二、　实验过程和结果

1　从图片文件中加载训练数据

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
import torch.nn.functional as F
import numpy as np
import torchvision
from torchvision import datasets,models,transforms
import matplotlib.pyplot as plt
import time
import copy
import os
%matplotlib inline
```

```
data_dir = 'transfer-data'
image_size = 224
```

```
train_dataset=datasets.ImageFolder(os.path.join(data_dir,'train'),
                                   transforms.Compose([
                                       transforms.RandomResizedCrop(image_size),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
                                   ])
                                   )
```

```
val_dataset = datasets.ImageFolder(os.path.join(data_dir,'val'),
                                   transforms.Compose([
                                       transforms.Resize(256),
                                       transforms.CenterCrop(image_size),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
                                   ])
                                   )
```

```
train_loader = torch.utils.data.DataLoader(train_dataset,batch_size=4,shuffle=True,num_workers=4)
val_loader = torch.utils.data.DataLoader(val_dataset,batch_size=4,shuffle=True,num_workers=4)

num_classes = len(train_dataset.classes)
num_classes
```

```
2
```

以上代码引入了必要的库，设置了数据集路径，运用 datasets 中的 ImageFolder 加载训练和验证数据集，针对图像的三个颜色通道通过设置均值和标准差对图片进行标准化，并且对图像进行了随机翻转，这样的操作可对图像进行增强便于机器进行特征识别，图像随机翻转使训练的模型鲁棒性更高。

```
use_cuda = torch.cuda.is_available()

dtype = torch.cuda.FloatTensor if use_cuda else torch.FloatTensor
itype = torch.cuda.FloatTensor if use_cuda else torch.LongTensor
```

该段代码对是否使用 cuda 变量定义，针对使用 cuda 情况改变数据类型

```
def imshow(inp, title= None):
    inp = inp.numpy().transpose((1,2,0))

    mean = np.array([0.485,0.456,0.406])
    std = np.array([0.229,0.224,0.225])
    inp = std*inp + mean
    inp = np.clip(inp,0,1)

    plt.imshow(inp)
    plt.axis('off')
    if title is not None:
        plt.title(title)
    plt.pause(0.001)
```

```
images,labels = next(iter(train_loader))
out = torchvision.utils.make_grid(images)
imshow(out,title=[train_dataset.classes[x] for x in labels])
```

该段代码定义了 imshow 函数，用于输出数据集图像。数据集中图像大小为 3*224*224 而 plt 中 imshow 函数要求图片为 224*224*3，且其颜色取值如果为整数应在[0,255]范围中，浮点数应在[0,1]中，代码首先对图片的维度次序进行了转换，再通过去标准化将图片颜色取值设定到规定范围中，通过 plt.imshow 函数绘制图片。

2　训练一个普通的卷积神经网络

```
depth = [4,8]

class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet,self).__init__()
        self.conv1 = nn.Conv2d(3,4,5,padding = 2)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(depth[0],depth[1],5,padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1],512)
        self.fc2 = nn.Linear(512,num_classes)

    def forward(self,x):
        x= F.relu(self.conv1(x))
        x= self.pool(x)
        x= F.relu(self.conv2(x))
        x= self.pool(x)

        x= x.view(-1,image_size // 4 * image_size // 4 * depth[1] )
        x = F.relu(self.fc1(x))
        x = F.dropout(x,training=self.training)
        x = self.fc2(x)
        x = F.log_softmax(x,dim = 1)
        return x
```

```
def rightness(predictions,labels):
    pred = torch.max(predictions.data,1)[1]
    rights = pred.eq(labels.data.view_as(pred)).sum()

    return rights,len(labels)
```

```
net = ConvNet()

net = net.cuda() if use_cuda else net
criterion  = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(),lr = 0.0001,momentum=0.9)
```

```
def train_model(data,target):
    net.train()
    output = net(data)
    loss = criterion(output,target)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    right = rightness(output,target)
    loss = loss.cpu() if use_cuda else loss
    return right,loss

def evaluation_model():
    net.eval()
    vals = []
    for data,target in val_loader:
        data,target = Variable(data,requires_grad=True),Variable(target)
        if use_cuda:
            data,target = data.cuda(),target.cuda()
        output = net(data)
        val =rightness(output,target)
        vals.append(val)
    return vals

def rightness(output,target):
    preds = output.data.max(dim = 1,keepdim = True)[1]
    return preds.eq(target.data.view_as(preds)).cpu().sum(),len(target)
```

上面代码参照实验四通过类定义的方式构建了一个卷积神经网络，定义函数 rightness 输出预测正确个数和总数，train_model 对模型进行一次训练返回正确率和损失值，evaluation_model 使用校验集对模型有效性进行验证。

```
record = []

num_epochs = 20
net.train(True)
best_model = net
best_r = 0.0
for epoch in range(num_epochs):
    train_rights=[]
    train_losses=[]

    for batch_idx,(data,target)in enumerate(train_loader):
        data,target=Variable(data),Variable(target)
        if use_cuda:
            data,target =data.cuda(),target.cuda()
        output=net(data)
        loss=criterion(output,target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        right=rightness(output,target)
        train_rights.append(right)
        loss=loss.cpu() if use_cuda else loss
        train_losses.append(loss.data.numpy())

    train_r=(sum([tup[0] for tup in train_rights]),
             sum([tup[1] for tup in train_rights]))
    net.eval()
    test_loss=0
    correct=0
    vals=[]

    for data,target in val_loader:
        if use_cuda:
            data,target=data.cuda(), target.cuda()

        data,target=Variable(data,requires_grad=True),Variable(target)
        output=net(data)
        val=rightness(output,target)
        vals.append(val)

    val_r=(sum([tup[0] for tup in vals]),sum([tup[1] for tup in vals]))
    val_ratio=1.0*val_r[0].numpy()/val_r[1]
    if val_ratio>best_r:
        best_r=val_ratio
        best_model=copy.deepcopy(net)

    print('训练周期: {}\tLoss:{:.6f}\t训练正确率:{:.2f}%,校验正确率:{:.2f}%'.format(
        epoch,np.mean(train_losses),
        100.*train_r[0].numpy()/train_r[1],100.*val_r[0].numpy()/val_r[1]))
    record.append([np.mean(train_losses),1.*train_r[0].data.numpy()/train_r[1],
                   1.*val_r[0].data.numpy()/val_r[1]])
```

这段代码正式开始训练神经网络，与上一实验相同，每轮分成若干撮训练，每撮计算正确率和损失值追加入列表中，每轮进行一次校验集验证，验证结果追加入列表，每轮计算一次平均正确率和损失值输出并保存到 record 中用于绘制训练效果图。其中每轮训练结束进行一次判断验证正确率和最优的正确率，将 20 轮训练中正确率最高的模型赋值给 best_modeel。

```
训练周期：0     Loss:0.691106   训练正确率:52.05%,校验正确率:45.75%
训练周期：1     Loss:0.685786   训练正确率:52.05%,校验正确率:47.06%
训练周期：2     Loss:0.689645   训练正确率:45.08%,校验正确率:52.29%
训练周期：3     Loss:0.680060   训练正确率:52.05%,校验正确率:50.33%
训练周期：4     Loss:0.675034   训练正确率:56.97%,校验正确率:51.63%
训练周期：5     Loss:0.670734   训练正确率:56.97%,校验正确率:56.21%
训练周期：6     Loss:0.671191   训练正确率:56.56%,校验正确率:54.25%
训练周期：7     Loss:0.664275   训练正确率:60.66%,校验正确率:56.21%
训练周期：8     Loss:0.669922   训练正确率:56.97%,校验正确率:56.86%
训练周期：9     Loss:0.658172   训练正确率:57.38%,校验正确率:56.21%
训练周期：10    Loss:0.651171   训练正确率:61.89%,校验正确率:58.82%
训练周期：11    Loss:0.646295   训练正确率:61.48%,校验正确率:59.48%
训练周期：12    Loss:0.635348   训练正确率:60.25%,校验正确率:57.52%
训练周期：13    Loss:0.637562   训练正确率:59.84%,校验正确率:57.52%
训练周期：14    Loss:0.624826   训练正确率:62.30%,校验正确率:59.48%
训练周期：15    Loss:0.639523   训练正确率:60.25%,校验正确率:60.13%
训练周期：16    Loss:0.610565   训练正确率:62.30%,校验正确率:59.48%
训练周期：17    Loss:0.632788   训练正确率:57.38%,校验正确率:62.75%
训练周期：18    Loss:0.603081   训练正确率:64.34%,校验正确率:62.75%
训练周期：19    Loss:0.600452   训练正确率:64.75%,校验正确率:63.40%
```

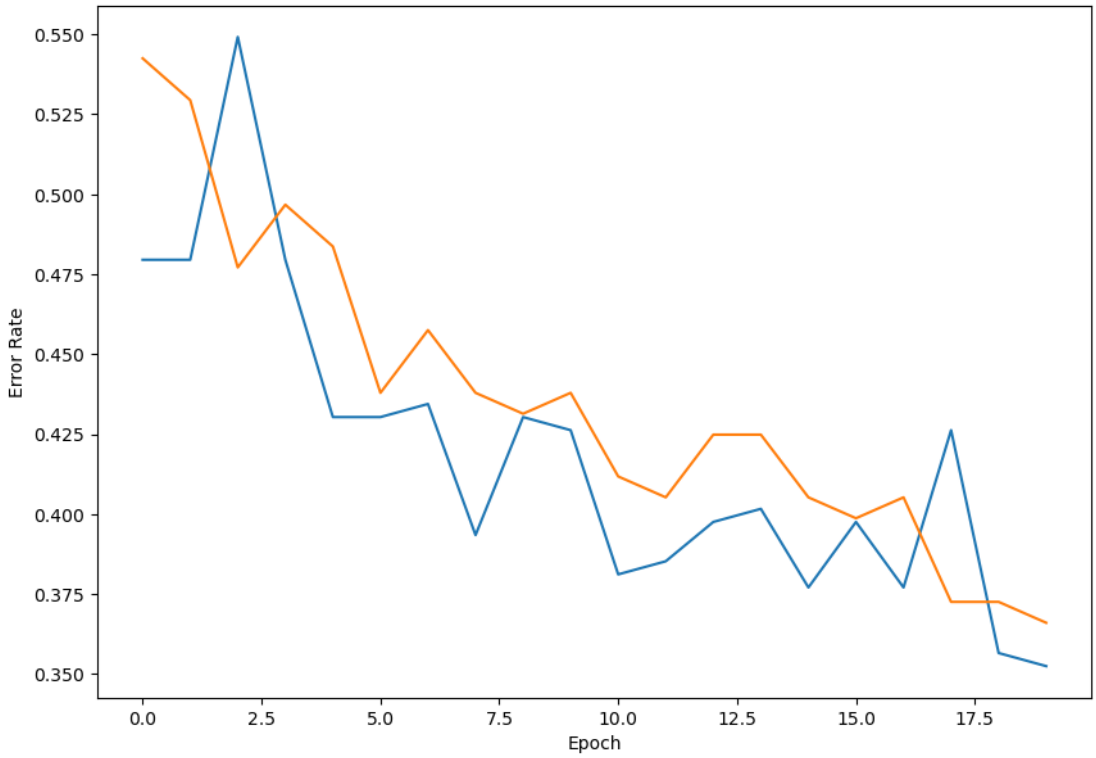
```
net.eval()
test_loss =0
correct=0
vals=[]
for data,target in val_loader:
    data,target =Variable(data,requires_grad=True),Variable(target)
    if use_cuda:
        data,target=data.cuda(),target.cuda()
    output=net(data)#将特征数据入网路，得到分类的输出
    val=rightness(output,target)#获得正确样本数以及总样本数
    vals.append(val)
rights =(sum([tup[0] for tup in vals]),sum([tup[1] for tup in vals]))
right_rate =1.0*rights[0].data.numpy()/rights[1]
right_rate
```

```
0.6339869281045751
```

```
x=[x[0] for x in record]
y=[1-x[1] for x in record]
z=[1-x[2] for x in record]
plt.figure(figsize=(10,7))
plt.plot(y)
plt.plot(z)
plt.xlabel('Epoch')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```

训练过程中采用了 dropout，在 eval 模式下禁用 dropout 再次使用验证集对模型正确率进行测试，输出正确率为 63.398%。对训练过程中 record 中保存的每轮训练的训练正确率和校验正确率通过绘图输出。



3　加载已经训练好的 ResNet 进行迁移学习

```
# torch.utils.models_zoo.load_url('http//labfile.oss.aliyuncs.com/course/1073/resnet18-5c106cde.pth')
net=models.resnet18(pretrained=True)
net=net.cuda() if use_cuda else net
net
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
...
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

① 预训练模式

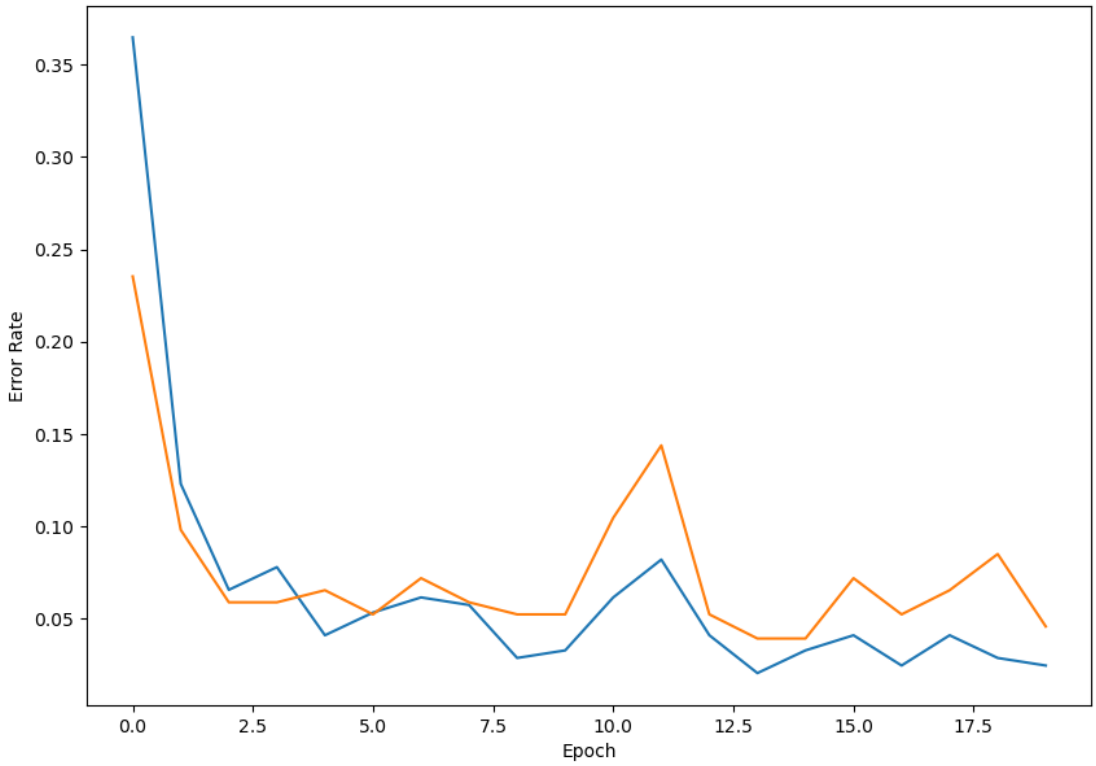
```
    num_ftrs=net.fc.in_features
    net.fc=nn.Linear(num_ftrs,2)
    net.fc=net.fc.cuda() if use_cuda else net.fc
    criterion=nn.CrossEntropyLoss()
    optimizer=optim.SGD(net.parameters(),lr=0.0001,momentum=0.9)
```

```
record=[]

num_epochs=20
net.train(True)
best_model=net
best_r=0.0
for epoch in range(num_epochs):
    train_rights=[]
    train_losses=[]
    for batch_idx,(data,target) in enumerate(train_loader):
        data,target=Variable(data),Variable(target)
        if use_cuda:
            data,target=data.cuda(),target.cuda()
        output=net(data)
        loss=criterion(output,target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        right=rightness(output,target)
        train_rights.append(right)
        loss=loss.cpu() if use_cuda else loss
        train_losses.append(loss.data.numpy())

    train_r=(sum([tup[0] for tup in train_rights]),
             sum([tup[1] for tup in train_rights]))
    net.eval()
    test_loss=0
    correct=0
    vals=[]

    for data,target in val_loader:
        if use_cuda:
            data,target=data.cuda(),target.cuda()
        data,target=Variable(data,requires_grad=True),Variable(target)
        output=net(data)
        val=rightness(output,target)
        vals.append(val)

    val_r=(sum([tup[0] for tup in vals]),sum([tup[1] for tup in vals]))
    val_ratio=1.0*val_r[0].numpy()/val_r[1]

    if val_ratio>best_r:
        best_r=val_ratio
        best_model=copy.deepcopy(net)

    print('训练周期:{}\tLoss:{:.6f}\t训练正确率:{:.2f}%,校验正确率:{:.2f}%'.format(
        epoch,np.mean(train_losses),
        100.*train_r[0].numpy()/train_r[1],100.*val_r[0].numpy()/val_r[1]))
```

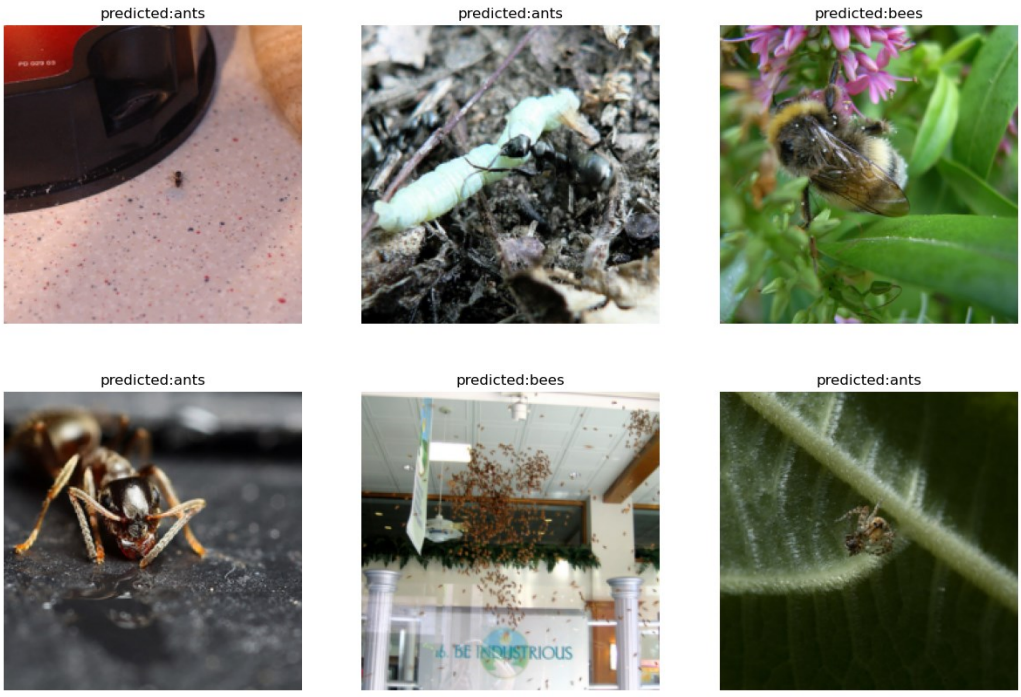
```
训练周期:0      Loss:0.624581   训练正确率:63.52%,校验正确率:76.47%
训练周期:1      Loss:0.351795   训练正确率:87.70%,校验正确率:90.20%
训练周期:2      Loss:0.175692   训练正确率:93.44%,校验正确率:94.12%
训练周期:3      Loss:0.178283   训练正确率:92.21%,校验正确率:94.12%
训练周期:4      Loss:0.129555   训练正确率:95.90%,校验正确率:93.46%
训练周期:5      Loss:0.136132   训练正确率:94.67%,校验正确率:94.77%
训练周期:6      Loss:0.144967   训练正确率:93.85%,校验正确率:92.81%
训练周期:7      Loss:0.143914   训练正确率:94.26%,校验正确率:94.12%
训练周期:8      Loss:0.093558   训练正确率:97.13%,校验正确率:94.77%
训练周期:9      Loss:0.075583   训练正确率:96.72%,校验正确率:94.77%
训练周期:10     Loss:0.110709   训练正确率:93.85%,校验正确率:89.54%
训练周期:11     Loss:0.170555   训练正确率:91.80%,校验正确率:85.62%
训练周期:12     Loss:0.107263   训练正确率:95.90%,校验正确率:94.77%
训练周期:13     Loss:0.069880   训练正确率:97.95%,校验正确率:96.08%
训练周期:14     Loss:0.072832   训练正确率:96.72%,校验正确率:96.08%
训练周期:15     Loss:0.098742   训练正确率:95.90%,校验正确率:92.81%
训练周期:16     Loss:0.070915   训练正确率:97.54%,校验正确率:94.77%
训练周期:17     Loss:0.102823   训练正确率:95.90%,校验正确率:93.46%
训练周期:18     Loss:0.089968   训练正确率:97.13%,校验正确率:91.50%
训练周期:19     Loss:0.062817   训练正确率:97.54%,校验正确率:95.42%
```

```
x=[x[0] for x in record]
y=[1-x[1] for x in record]
z=[1-x[2] for x in record]
#plt.plot(x)
plt.figure(figsize=(10,7))
plt.plot(y)
plt.plot(z)
plt.xlabel('Epoch')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



```
def visualize_model(model,num_images=6):
    images_so_far = 0
    fig = plt.figure(figsize=(15,10))
    for i, data in enumerate(val_loader):
        inputs,labels = data
        inputs,labels = Variable(inputs),Variable(labels)
        if use_cuda:
            inputs,labels = inputs.cuda(),labels.cuda()
        outputs = model(inputs)
        _, preds=torch.max(outputs.data,1)
        preds = preds.cpu().numpy() if use_cuda else preds.numpy()

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

        for j in range(inputs.size()[0]):
            images_so_far += 1
            plt.subplot(2, num_images//2,images_so_far)
            plt.axis('off')
            plt.title('predicted:{}'.format(val_dataset.classes[preds[j]]))
            inp=data[0][j].numpy().transpose((1,2,0))
            mean = np.array([0.485,0.456,0.406])
            std = np.array([0.229,0.224,0.225])
            inp = std*inp + mean
            inp = np.clip(inp,0,1)
            plt.imshow(inp)

            if images_so_far ==num_images:
                return

visualize_model(net)
plt.ioff()
plt.show()
```



第三部分代码对已经训练好的残差网络进行迁移学习。通过调用输出ResNet18的结构，其中最后的全连接层为隐层512输出层1000的前馈层，而我们只需要辨别蜜蜂和蚂蚁，因此将最后的前馈全连接层修改为512隐层和2输出神经元。通过将resnet18首先实例化，再将变量的fc层直接修改，就完成了迁移嫁接过程。

第一次尝试采用预训练模式，将预训练好的 ResNet 参数直接迁移作为新网络训练的起点，便于新的网络训练中可以更快的收敛，参数达到局部最优，这种方式灵活性和适应性强。训练的过程和其他网络无异。

同样输出每轮训练中的损失值和训练校验正确率，可见通过迁移学习得到的模型正确率远高于自搭建的卷积神经网络，训练过程中正确率的收敛速度也更快。

最后定义了一个可视化模型函数 visualize_model，其实际作用是从验证集中抽取若干数据送入模型中，以子图形式显示图像和模型对它的预测值。指导书代码出现了一些错误，图像不能以子图形式显示，我将前一部分imshow的代码稍作修改替换掉了子图绘制部分的函数，保证图像能以正常的色彩值输出。

第二次代码如下，采用了固定值模式，就是训练过程中不改变迁移部分的参数，只改变更改大小的前馈全连接层参数的值，当使用反向传播算法的时候，误差反传过程会在迁移模块中停止，从而不改变迁移模块中的权重数值。采用这种方式，可保留被迁移部分的知识不被破坏。实现的方法很简单，将模型实例化后将参数全部设定为 requires_grad=False，这样自动微分和反向传播就不会改变参数值，再将前馈全连接层惊醒替换，送入训练的轮回中训练。

虽然固定值模式不如预训练模式灵活，通过训练结果比较可以看出固定值模式训练的正确率变化更加平缓，但预训练模式下的正确率能够达到更高的数值。

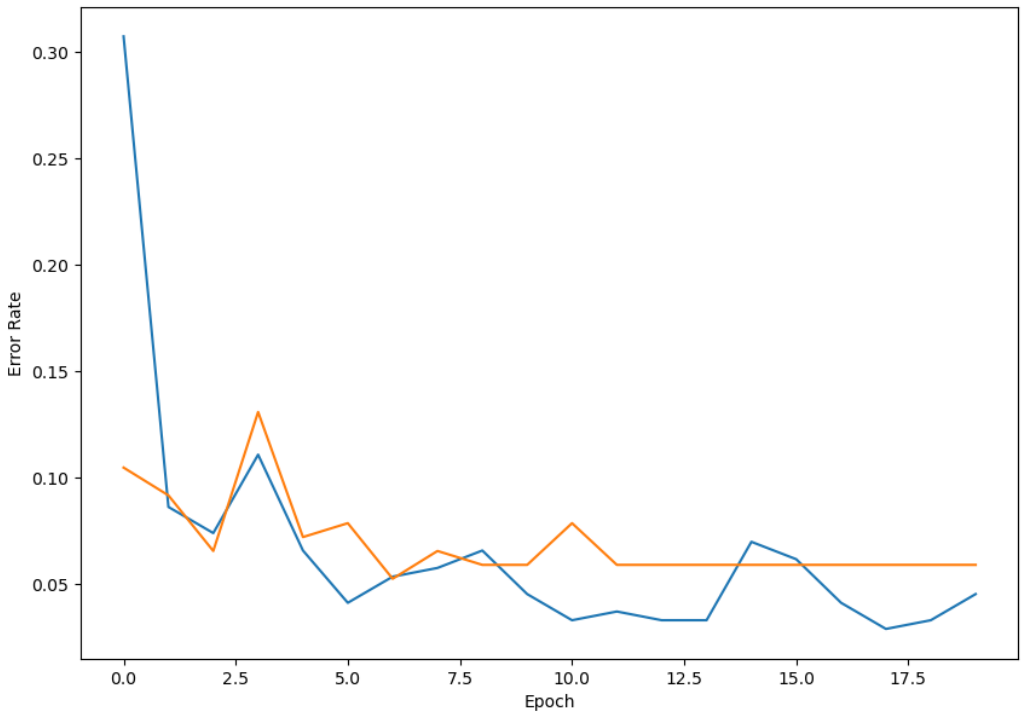
② 固定值模式

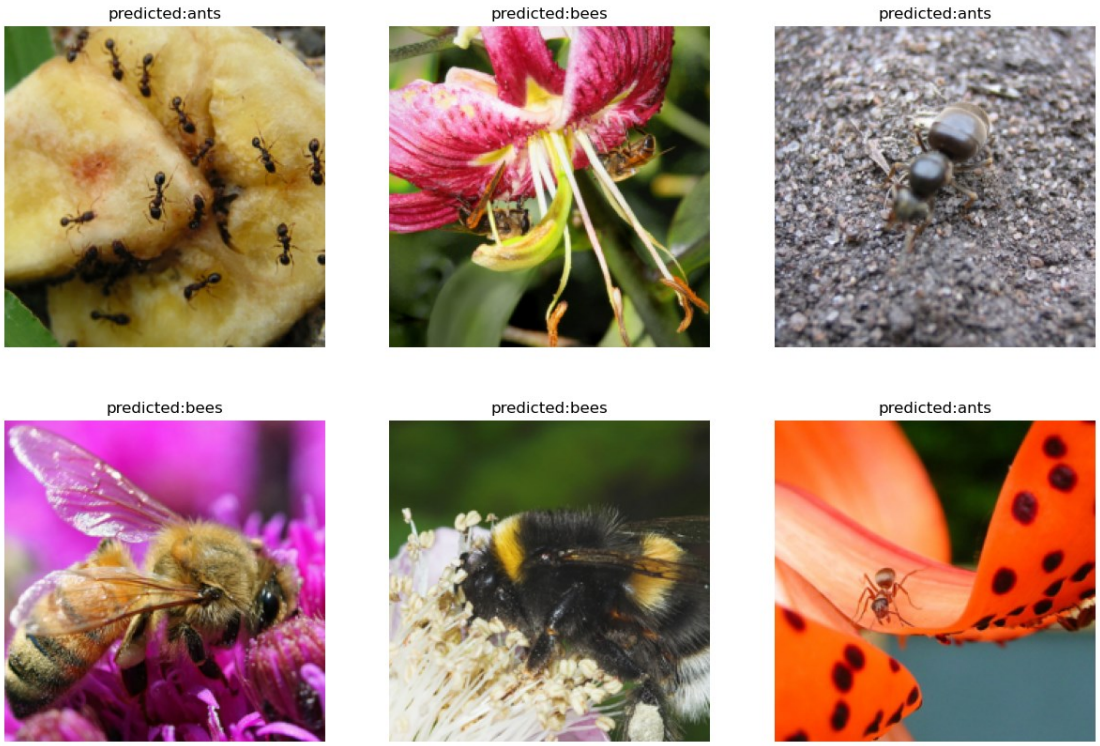
```
net = torchvision.models.resnet18(pretrained=True)
net=net.cuda() if use_cuda else net
for param in net.parameters():
    param.requires_grad=False

num_ftrs=net.fc.in_features
net.fc=nn.Linear(num_ftrs,2)
net.fc=net.fc.cuda() if use_cuda else net.fc

criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(net.fc.parameters(),lr=0.001,momentum=0.9)
```

```
record=[]

num_epochs=20
net.train(True)
best_model=net
best_r=0.0
for epoch in range(num_epochs):
    train_rights=[]
    train_losses=[]
    for batch_idx,(data,target) in enumerate(train_loader):
        data,target=Variable(data),Variable(target)
        if use_cuda:
            data,target=data.cuda(),target.cuda()
        output=net(data)
        loss=criterion(output,target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        right=rightness(output,target)
        train_rights.append(right)
        loss=loss.cpu() if use_cuda else loss
        train_losses.append(loss.data.numpy())

    train_r=(sum([tup[0] for tup in train_rights]),
             sum([tup[1] for tup in train_rights]))
    net.eval()
    test_loss=0
    correct=0
    vals=[]

    for data,target in val_loader:
        if use_cuda:
            data,target=data.cuda(),target.cuda()
        data,target =Variable(data,requires_grad=True),Variable(target)
        output=net(data)
        val=rightness(output,target)
        vals.append(val)

    val_r=(sum([tup[0] for tup in vals]),sum([tup[1] for tup in vals]))
    val_ratio=1.0*val_r[0].numpy()/val_r[1]

    if val_ratio>best_r:
        best_r=val_ratio
        best_model=copy.deepcopy(net)

    print('训练周期:{}\tLoss:{:.6f}\t训练正确率:{:.2f}%,校验正确率:{:.2f}%'.format(
        epoch,np.mean(train_losses),
        100.*train_r[0].numpy()/train_r[1],100.*val_r[0].numpy()/val_r[1]
    ))
```

```
训练周期:0      Loss:0.520400   训练正确率:69.26%,校验正确率:89.54%
训练周期:1      Loss:0.214119   训练正确率:91.39%,校验正确率:90.85%
训练周期:2      Loss:0.183247   训练正确率:92.62%,校验正确率:93.46%
训练周期:3      Loss:0.226830   训练正确率:88.93%,校验正确率:86.93%
训练周期:4      Loss:0.140807   训练正确率:93.44%,校验正确率:92.81%
训练周期:5      Loss:0.118054   训练正确率:95.90%,校验正确率:92.16%
训练周期:6      Loss:0.138595   训练正确率:94.67%,校验正确率:94.77%
训练周期:7      Loss:0.127718   训练正确率:94.26%,校验正确率:93.46%
训练周期:8      Loss:0.114810   训练正确率:93.44%,校验正确率:94.12%
训练周期:9      Loss:0.121461   训练正确率:95.49%,校验正确率:94.12%
训练周期:10     Loss:0.106182   训练正确率:96.72%,校验正确率:92.16%
训练周期:11     Loss:0.080661   训练正确率:96.31%,校验正确率:94.12%
训练周期:12     Loss:0.096118   训练正确率:96.72%,校验正确率:94.12%
训练周期:13     Loss:0.073769   训练正确率:96.72%,校验正确率:94.12%
训练周期:14     Loss:0.138894   训练正确率:93.03%,校验正确率:94.12%
训练周期:15     Loss:0.149256   训练正确率:93.85%,校验正确率:94.12%
训练周期:16     Loss:0.097592   训练正确率:95.90%,校验正确率:94.12%
训练周期:17     Loss:0.080443   训练正确率:97.13%,校验正确率:94.12%
训练周期:18     Loss:0.107887   训练正确率:96.72%,校验正确率:94.12%
训练周期:19     Loss:0.097883   训练正确率:95.49%,校验正确率:94.12%
```

```python
x=[x[0] for x in record]
y=[1-x[1] for x in record]
z=[1-x[2] for x in record]
plt.figure(figsize=(10,7))
plt.plot(y)
plt.plot(z)
plt.xlabel('Epoch')
plt.ylabel('Error Rate')

visualize_model(best_model)
plt.ioff()
plt.show()
```





三、　心得体会

本次实验练习了如何进行迁移学习，实验大致围绕一个主要问题分为三部分，针对蚂蚁蜜蜂分类问题采用卷积神经网络、迁移学习预训练的残差神经网络和固定值的残差神经网络进行分类。

迁移学习主要有预训练和固定值两种迁移方式。预训练模型是将网络的结构和参数迁移到新的任务中，参数的初始化取值设定为源网络的数值，在新的网络中，所有的参数都会被重新训练。固定值模型是将源网络的结构和参数迁移到新的任务中后，新网络中参数的取值设定为原网络的数值，不参与反向传播和训练。而在实际应用过程中，预训练方式往往能取得更好的迁移效果。

实验中迁移过程十分简便，将预训练的模型实例化后直接更改模型某一层的属性值。迁移学习通过利用已训练好的模型，构建用于解决特定新问题的方式大大减少了个人训练模型的时间和资源开销。