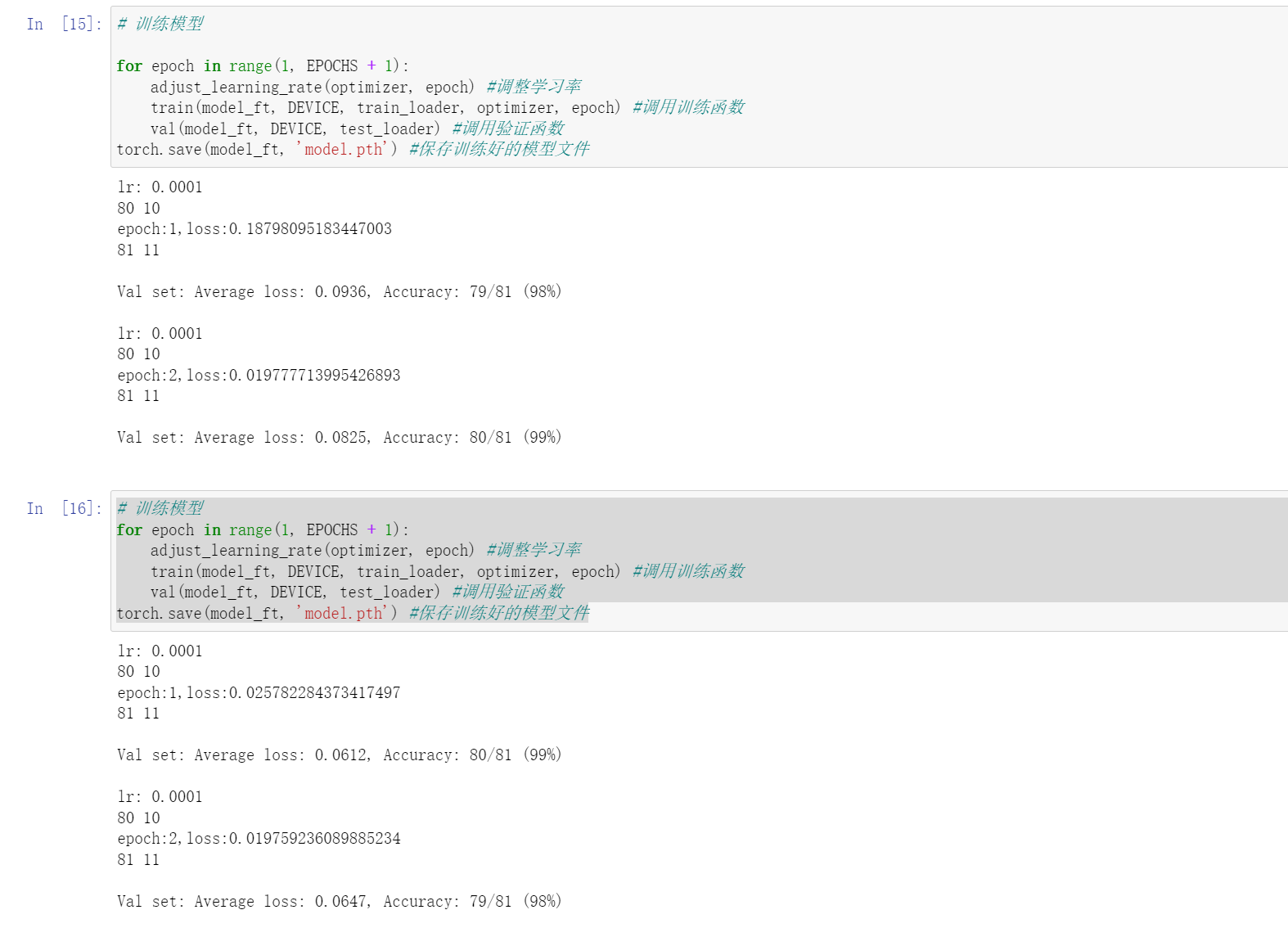
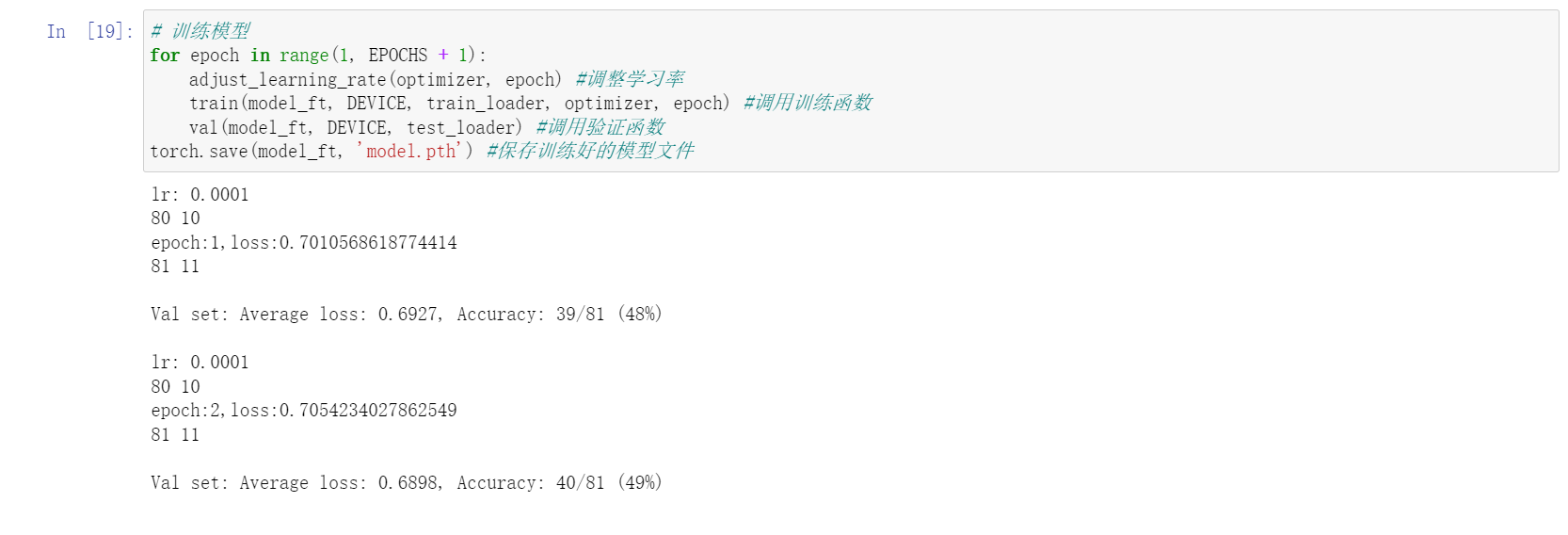
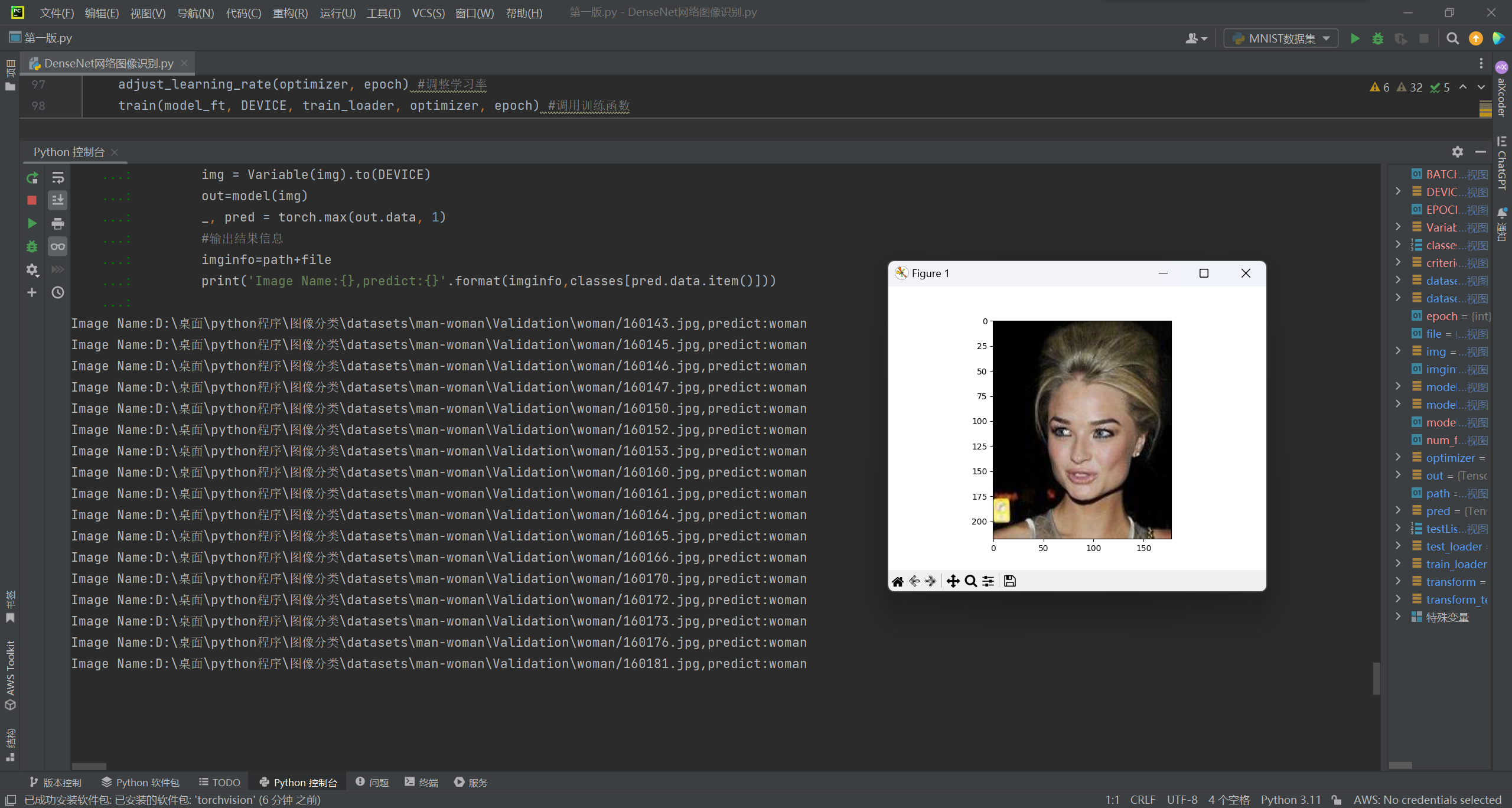
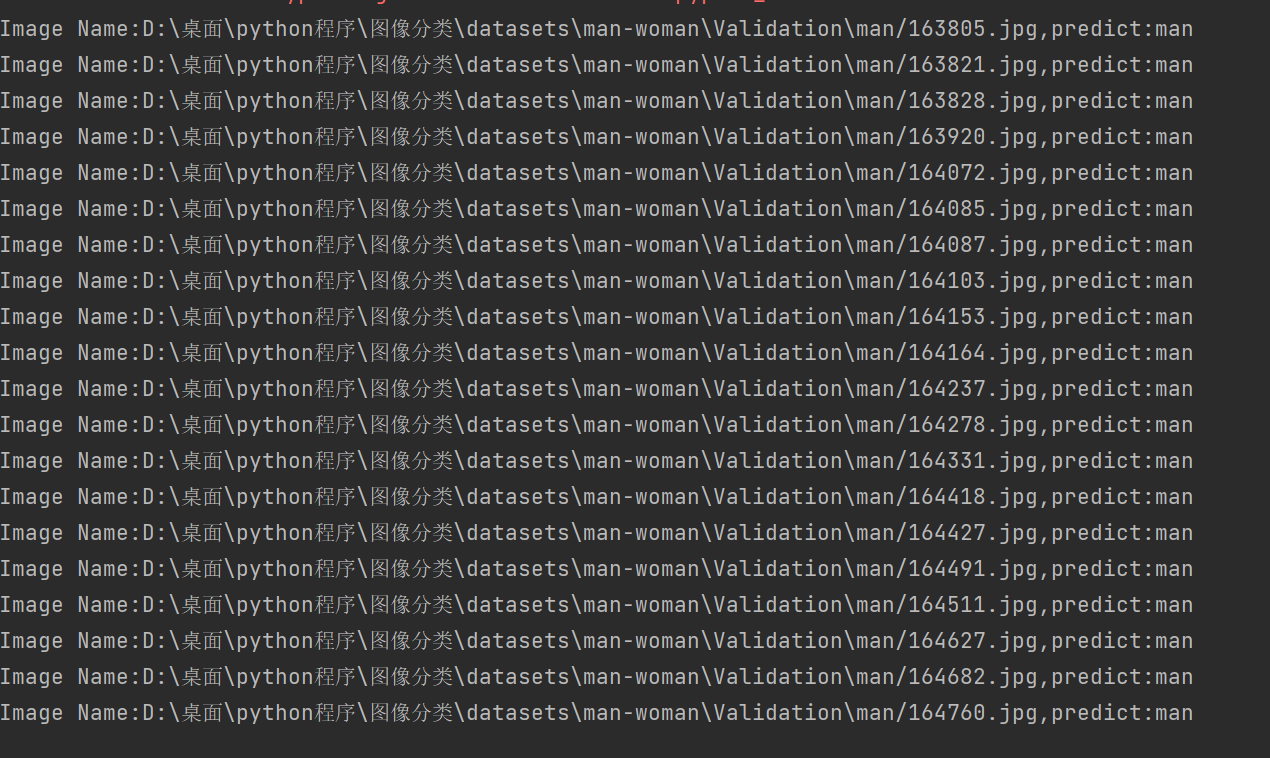
使用DenseNet模型结果：





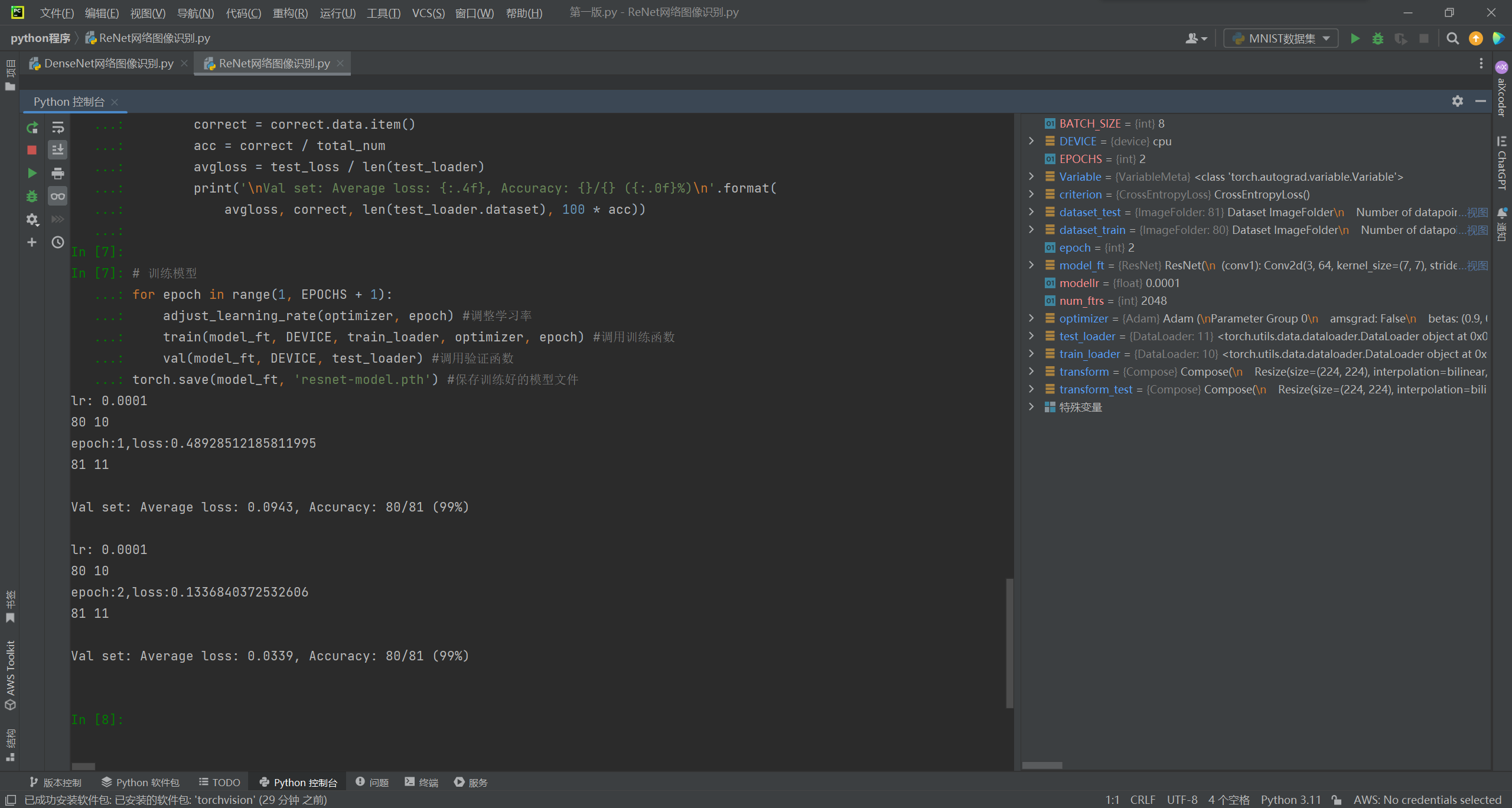


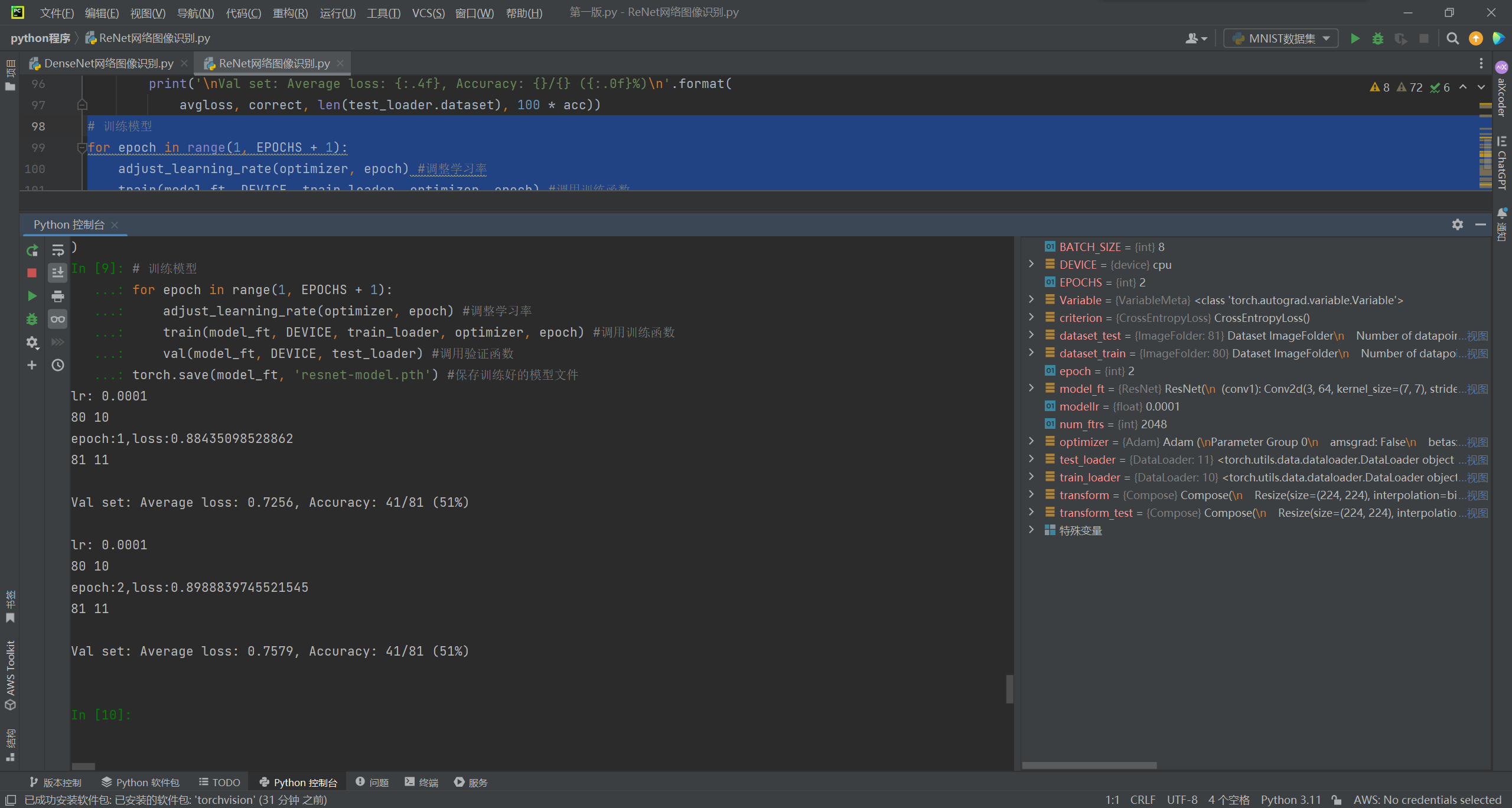


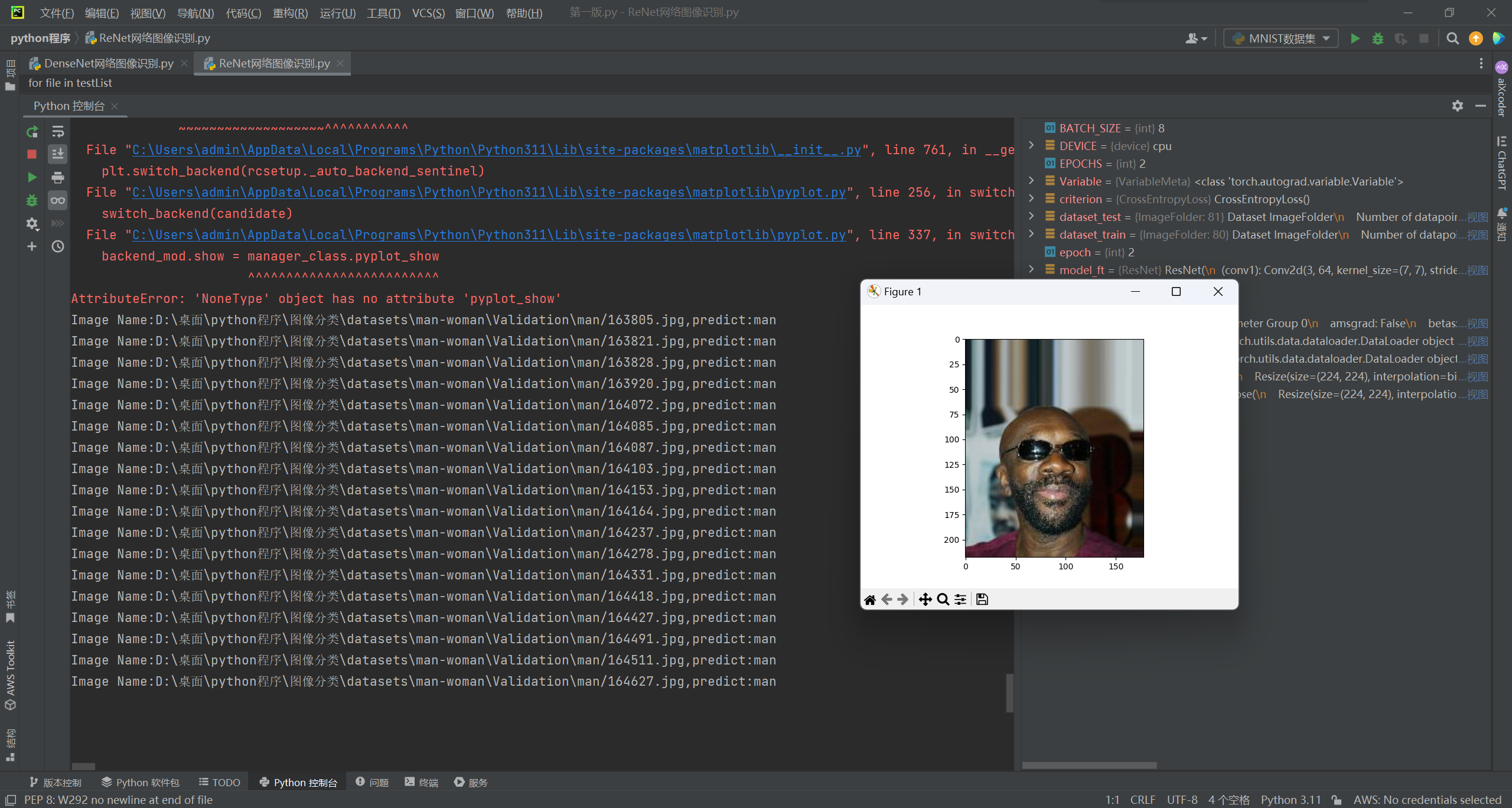
代码：

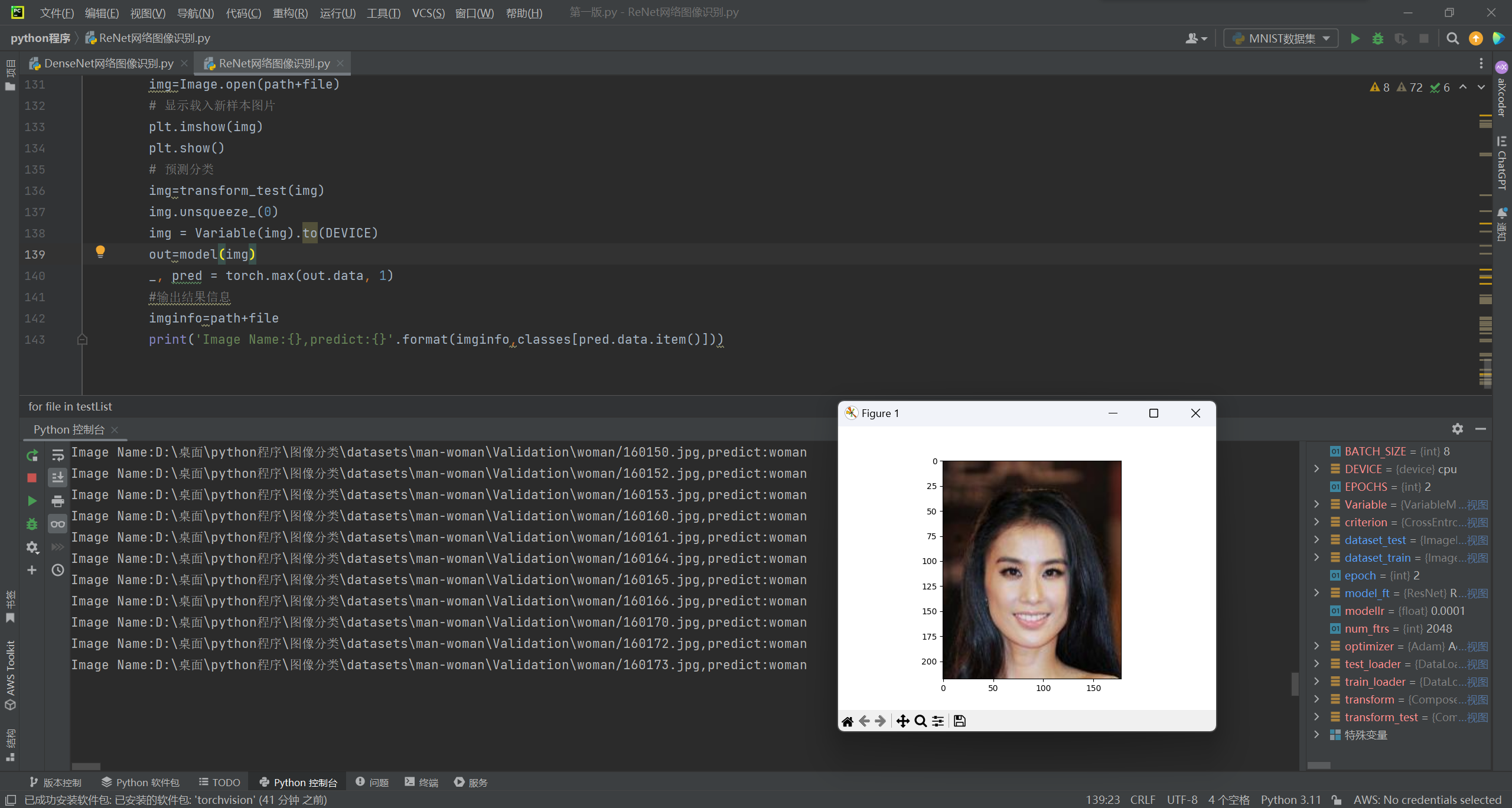
import torch.optim as optim  
import torch  
import torch.nn as nn  
import torch.nn.parallel  
import torch.utils.data  
import torch.utils.data.distributed  
import torchvision.transforms as transforms  
import torchvision.datasets as datasets  
from torch.autograd import Variable  
from torchvision.models import densenet121  
  
modellr = 1e-4 #模型学习率  
BATCH\_SIZE = 8 #训练批尺寸  
EPOCHS = 2 # 训练轮数  
DEVICE = torch.device('cpu') #直接上CPU  
  
# 数据预处理  
transform = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  
])  
transform\_test = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  
])  
dataset\_train = datasets.ImageFolder("D:\桌面\python程序\图像分类\datasets\man-woman\Train", transform) #如果你的数据存储路径与示例不同，需根据实际位置修改路径  
dataset\_test = datasets.ImageFolder("D:\桌面\python程序\图像分类\datasets\man-woman\Test",transform\_test) #如果你的数据存储路径与示例不同，需根据实际位置修改路径  
# 打印训练集图片信息，包括路径及文件名、分类标签值  
print(dataset\_train.imgs)  
# 导入图片数据  
train\_loader = torch.utils.data.DataLoader(dataset\_train, batch\_size=BATCH\_SIZE, shuffle=True)  
test\_loader = torch.utils.data.DataLoader(dataset\_test, batch\_size=BATCH\_SIZE, shuffle=False)  
  
# 选用交叉熵损失作为loss函数  
criterion = nn.CrossEntropyLoss()  
# 模型采用densenet121结构  
model\_ft = densenet121(pretrained=True) #载入并使用预训练模型参数来提升效果，以便快速得到收敛好的模型  
num\_ftrs = model\_ft.classifier.in\_features  
# 更改最后一层的全连接，将类别设置为2  
model\_ft.classifier = nn.Linear(num\_ftrs, 2)  
model\_ft.to(DEVICE)  
# 选择Adam优化器  
optimizer = optim.Adam(model\_ft.parameters(), lr=modellr)  
# 实现学习率动态调整  
def adjust\_learning\_rate(optimizer, epoch):  
 modellrnew = modellr \* (0.1 \*\* (epoch // 10))  
 print("lr:", modellrnew)  
 for param\_group in optimizer.param\_groups:  
 param\_group['lr'] = modellrnew  
  
# 定义训练函数  
def train(model, device, train\_loader, optimizer, epoch):  
 model.train()  
 sum\_loss = 0  
 total\_num = len(train\_loader.dataset)  
 print(total\_num, len(train\_loader))  
 for batch\_idx, (data, target) in enumerate(train\_loader):  
 data, target = Variable(data).to(device), Variable(target).to(device)  
 output = model(data)  
 loss = criterion(output, target)  
 optimizer.zero\_grad()  
 loss.backward() # 反向传播  
 optimizer.step()  
 print\_loss = loss.data.item()  
 sum\_loss += print\_loss  
 if (batch\_idx + 1) % 50 == 0:  
 print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(  
 epoch, (batch\_idx + 1) \* len(data), len(train\_loader.dataset),  
 100. \* (batch\_idx + 1) / len(train\_loader), loss.item()))  
 ave\_loss = sum\_loss / len(train\_loader)  
 print('epoch:{},loss:{}'.format(epoch, ave\_loss))  
# 定义验证函数  
def val(model, device, test\_loader):  
 model.eval()  
 test\_loss = 0  
 correct = 0  
 total\_num = len(test\_loader.dataset)  
 print(total\_num, len(test\_loader))  
 with torch.no\_grad():  
 for data, target in test\_loader:  
 data, target = Variable(data).to(device), Variable(target).to(device)  
 output = model(data)  
 loss = criterion(output, target)  
 \_, pred = torch.max(output.data, 1)  
 correct += torch.sum(pred == target)  
 print\_loss = loss.data.item()  
 test\_loss += print\_loss  
 correct = correct.data.item()  
 acc = correct / total\_num  
 avgloss = test\_loss / len(test\_loader)  
 print('\nVal set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(  
 avgloss, correct, len(test\_loader.dataset), 100 \* acc))  
# 训练模型  
for epoch in range(1, EPOCHS + 1):  
 adjust\_learning\_rate(optimizer, epoch) #调整学习率  
 train(model\_ft, DEVICE, train\_loader, optimizer, epoch) #调用训练函数  
 val(model\_ft, DEVICE, test\_loader) #调用验证函数  
torch.save(model\_ft, 'model.pth') #保存训练好的模型文件  
  
  
# 测试自定义DenseNet模型的性能  
# 如果测试代码不作为单独的python程序编写执行，则可省略该部分导入工作  
import os  
import torch.utils.data.distributed  
import torchvision.transforms as transforms  
from PIL import Image  
import matplotlib.pyplot as plt  
from torch.autograd import Variable  
  
classes=('man','woman')  
transform\_test = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  
])  
DEVICE = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  
model = torch.load("model.pth")  
model.eval()  
model.to(DEVICE)  
path="D:\桌面\python程序\图像分类\datasets\man-woman\Validation\man/"  
testList=os.listdir(path)  
for file in testList:  
 img=Image.open(path+file)  
 # 显示载入新样本图片  
 plt.imshow(img)  
 plt.show()  
 # 预测分类  
 img=transform\_test(img)  
 img.unsqueeze\_(0)  
 img = Variable(img).to(DEVICE)  
 out=model(img)  
 \_, pred = torch.max(out.data, 1)  
 #输出结果信息  
 imginfo=path+file  
 print('Image Name:{},predict:{}'.format(imginfo,classes[pred.data.item()]))

使用ResNet模型得出的结果：









代码：

import torch.optim as optim  
import torch  
import torch.nn as nn  
import torch.nn.parallel  
import torch.utils.data  
import torch.utils.data.distributed  
import torchvision.transforms as transforms  
import torchvision.datasets as datasets  
from torch.autograd import Variable  
from torchvision.models import resnet50  
  
modellr = 1e-4 #模型学习率  
BATCH\_SIZE = 8 #训练批尺寸  
EPOCHS = 2 # 训练轮数  
DEVICE = torch.device('cpu') #直接上CPU  
  
# 数据预处理  
transform = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  
])  
transform\_test = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  
])  
dataset\_train = datasets.ImageFolder("D:\桌面\python程序\图像分类\datasets\man-woman\Train", transform) #如果你的数据存储路径与示例不同，需根据实际位置修改路径  
dataset\_test = datasets.ImageFolder("D:\桌面\python程序\图像分类\datasets\man-woman\Test",transform\_test) #如果你的数据存储路径与示例不同，需根据实际位置修改路径  
# 打印训练集图片信息，包括路径及文件名、分类标签值  
print(dataset\_train.imgs)  
# 导入图片数据  
train\_loader = torch.utils.data.DataLoader(dataset\_train, batch\_size=BATCH\_SIZE, shuffle=True)  
test\_loader = torch.utils.data.DataLoader(dataset\_test, batch\_size=BATCH\_SIZE, shuffle=False)  
  
  
criterion = nn.CrossEntropyLoss()  
# ResNet50神经网络模型的设置代码  
model\_ft = resnet50(pretrained=True) # 载入预训练模型参数  
# 获取模型的特征数量（对于ResNet50, 它的全连接层的输入特征数）  
num\_ftrs = model\_ft.fc.in\_features  
# 更改最后一层的全连接层，将输出类别设置为2  
model\_ft.fc = nn.Linear(num\_ftrs, 2)  
# 将模型移动到指定的设备  
model\_ft.to(DEVICE)  
# 选择Adam优化器  
optimizer = optim.Adam(model\_ft.parameters(), lr=modellr)  
# 实现学习率动态调整  
def adjust\_learning\_rate(optimizer, epoch):  
 modellrnew = modellr \* (0.1 \*\* (epoch // 10))  
 print("lr:", modellrnew)  
 for param\_group in optimizer.param\_groups:  
 param\_group['lr'] = modellrnew  
  
  
# 定义训练函数  
def train(model, device, train\_loader, optimizer, epoch):  
 model.train()  
 sum\_loss = 0  
 total\_num = len(train\_loader.dataset)  
 print(total\_num, len(train\_loader))  
 for batch\_idx, (data, target) in enumerate(train\_loader):  
 data, target = Variable(data).to(device), Variable(target).to(device)  
 output = model(data)  
 loss = criterion(output, target)  
 optimizer.zero\_grad()  
 loss.backward() # 反向传播  
 optimizer.step()  
 print\_loss = loss.data.item()  
 sum\_loss += print\_loss  
 if (batch\_idx + 1) % 50 == 0:  
 print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(  
 epoch, (batch\_idx + 1) \* len(data), len(train\_loader.dataset),  
 100. \* (batch\_idx + 1) / len(train\_loader), loss.item()))  
 ave\_loss = sum\_loss / len(train\_loader)  
 print('epoch:{},loss:{}'.format(epoch, ave\_loss))  
# 定义验证函数  
def val(model, device, test\_loader):  
 model.eval()  
 test\_loss = 0  
 correct = 0  
 total\_num = len(test\_loader.dataset)  
 print(total\_num, len(test\_loader))  
 with torch.no\_grad():  
 for data, target in test\_loader:  
 data, target = Variable(data).to(device), Variable(target).to(device)  
 output = model(data)  
 loss = criterion(output, target)  
 \_, pred = torch.max(output.data, 1)  
 correct += torch.sum(pred == target)  
 print\_loss = loss.data.item()  
 test\_loss += print\_loss  
 correct = correct.data.item()  
 acc = correct / total\_num  
 avgloss = test\_loss / len(test\_loader)  
 print('\nVal set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(  
 avgloss, correct, len(test\_loader.dataset), 100 \* acc))  
# 训练模型  
for epoch in range(1, EPOCHS + 1):  
 adjust\_learning\_rate(optimizer, epoch) #调整学习率  
 train(model\_ft, DEVICE, train\_loader, optimizer, epoch) #调用训练函数  
 val(model\_ft, DEVICE, test\_loader) #调用验证函数  
torch.save(model\_ft, 'resnet-model.pth') #保存训练好的模型文件  
  
  
  
# 如果测试代码不作为单独的python程序编写执行，则可省略该部分导入工作  
import os  
import torch.utils.data.distributed  
import torchvision.transforms as transforms  
from PIL import Image  
import matplotlib.pyplot as plt  
from torch.autograd import Variable  
  
classes=('man','woman')  
transform\_test = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  
])  
DEVICE = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  
model = torch.load("resnet-model.pth")  
model.eval()  
model.to(DEVICE)  
path="D:\桌面\python程序\图像分类\datasets\man-woman\Validation\man/"  
testList=os.listdir(path)  
for file in testList:  
 img=Image.open(path+file)  
 # 显示载入新样本图片  
 plt.imshow(img)  
 plt.show()  
 # 预测分类  
 img=transform\_test(img)  
 img.unsqueeze\_(0)  
 img = Variable(img).to(DEVICE)  
 out=model(img)  
 \_, pred = torch.max(out.data, 1)  
 #输出结果信息  
 imginfo=path+file  
 print('Image Name:{},predict:{}'.format(imginfo,classes[pred.data.item()]))