**中国矿业大学计算机学院**

**2020 级本科生实验报告**

课程名称 软件工程实践

报告时间 2022-11-29

学生姓名 李建业

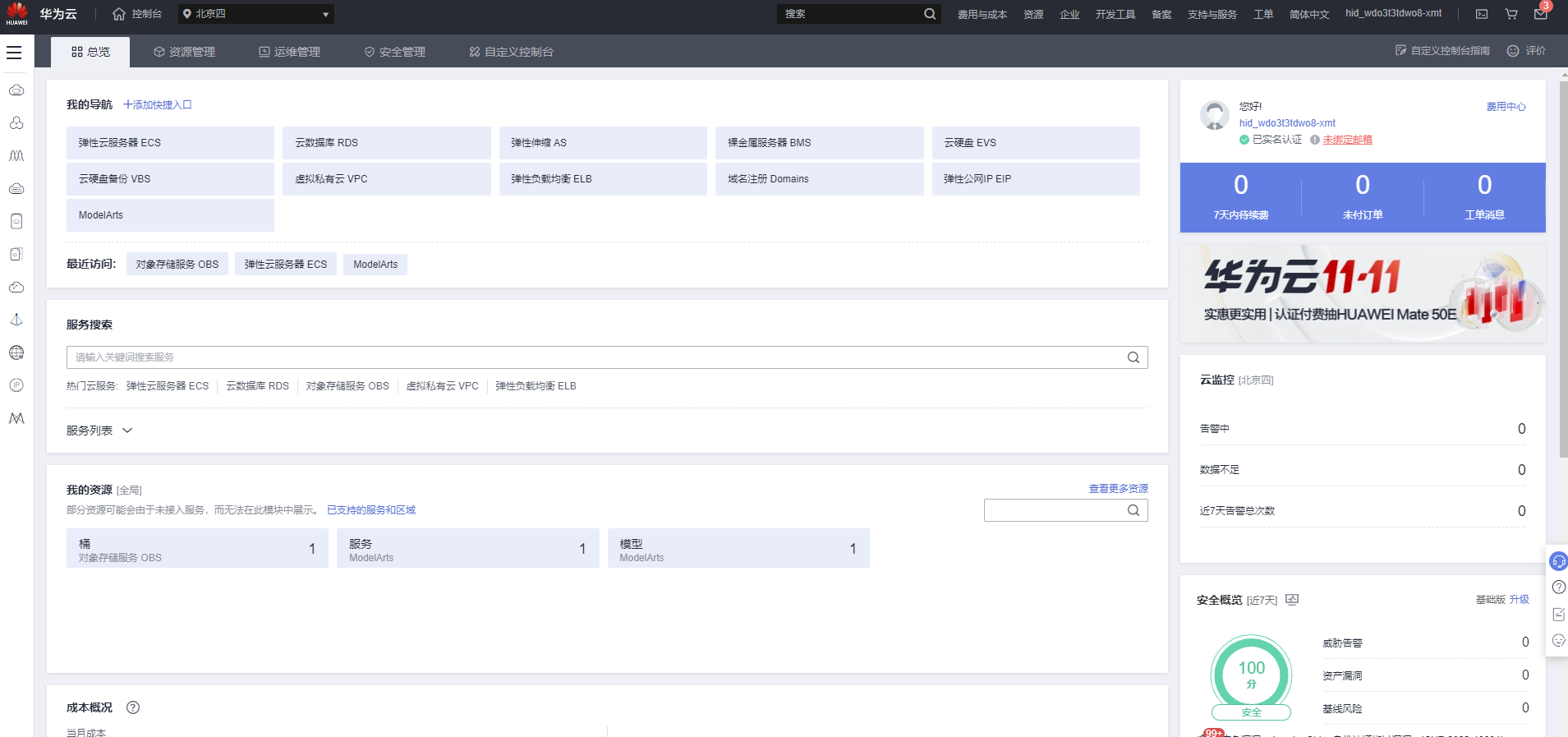
学 号 12203743

专 业 计算机科学与技术

任课教师 刘迎春

实验步骤

1. 登录成功之后进入控制台



1. 找到对象存储服务OBS（可以在服务搜索里输入obs），选择左侧桶列表，之后创建桶

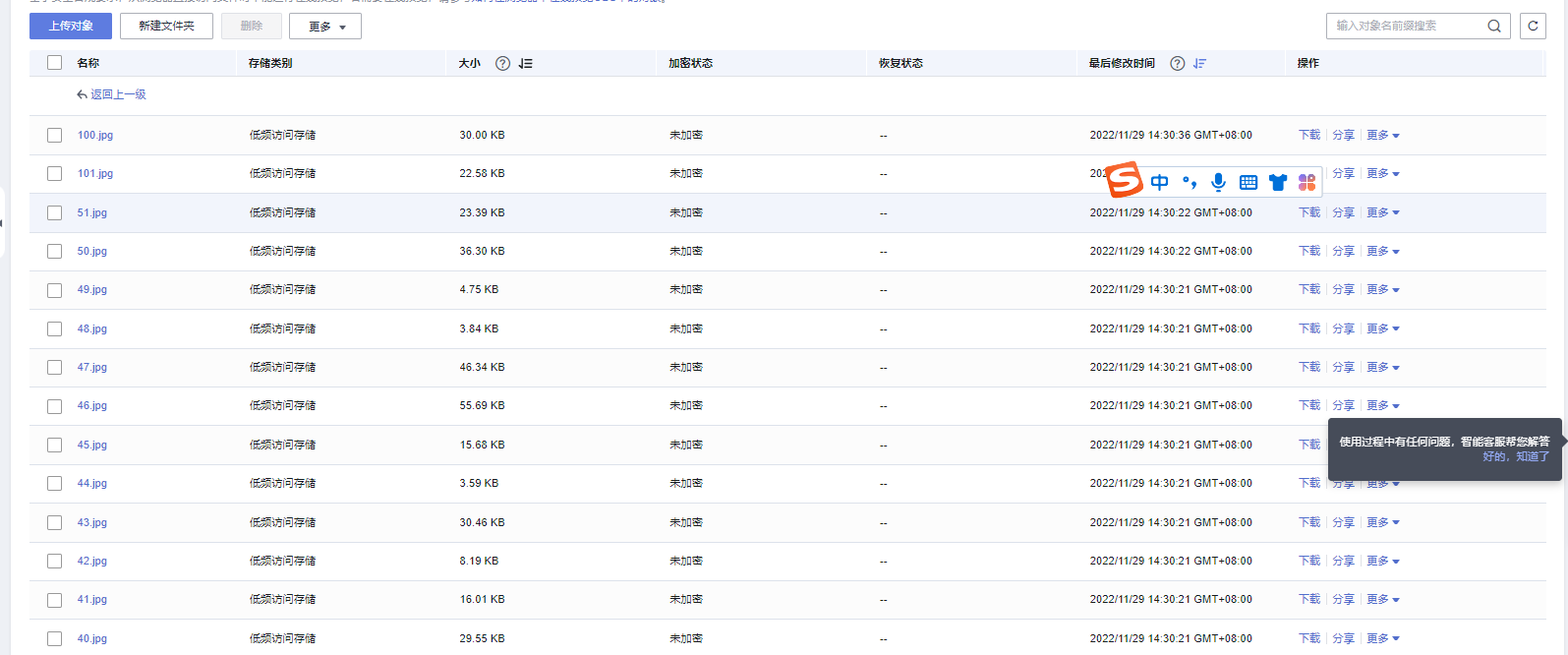


1. 此为创建桶的页面，如果出现另外一个冲突的操作当前正作用在这个资源上，请重试。A conflicting conditional operation is currently in progress against this resource. Please try again. 的错误，则为桶名称重复，请改正。



1. 创建成功后左键单击桶名称进入桶，新建文件夹input，output

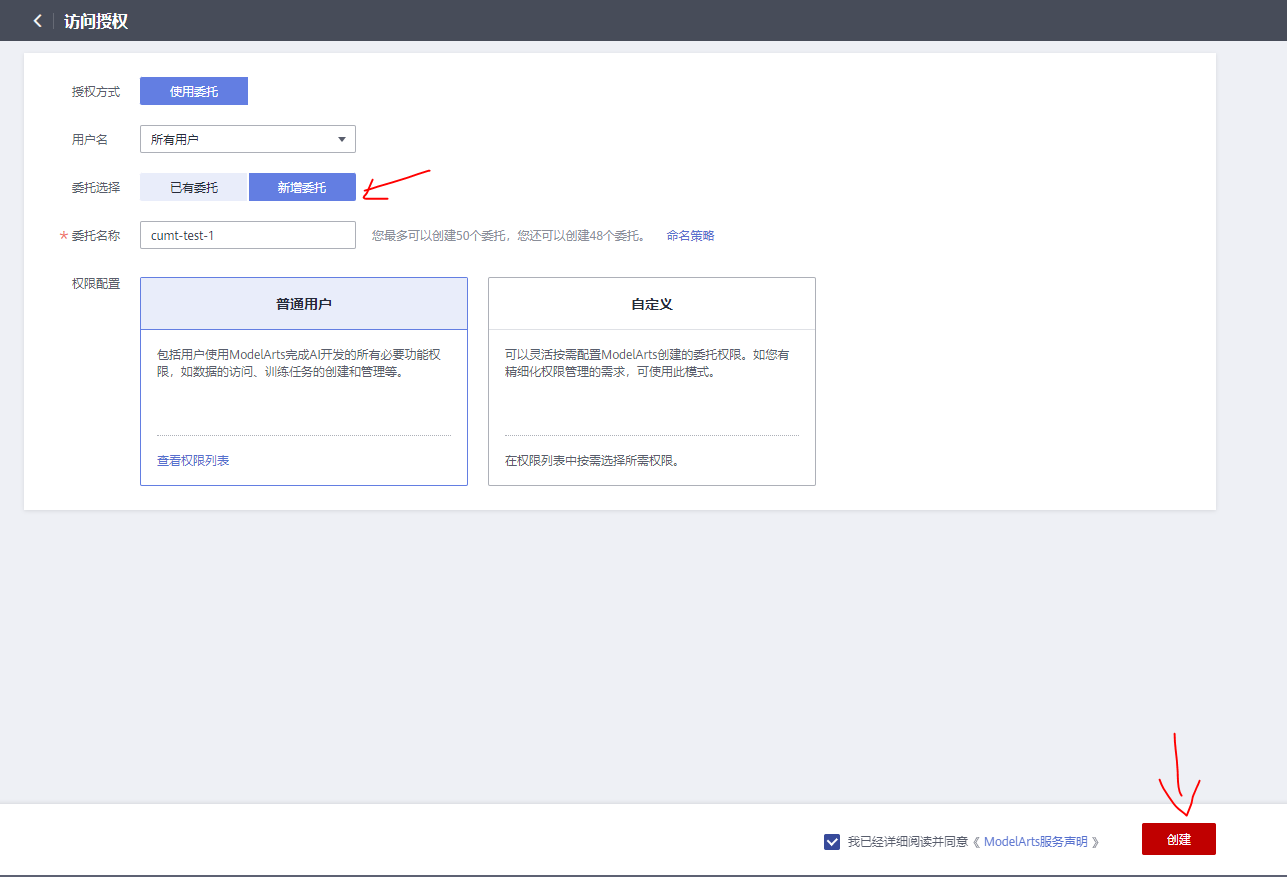




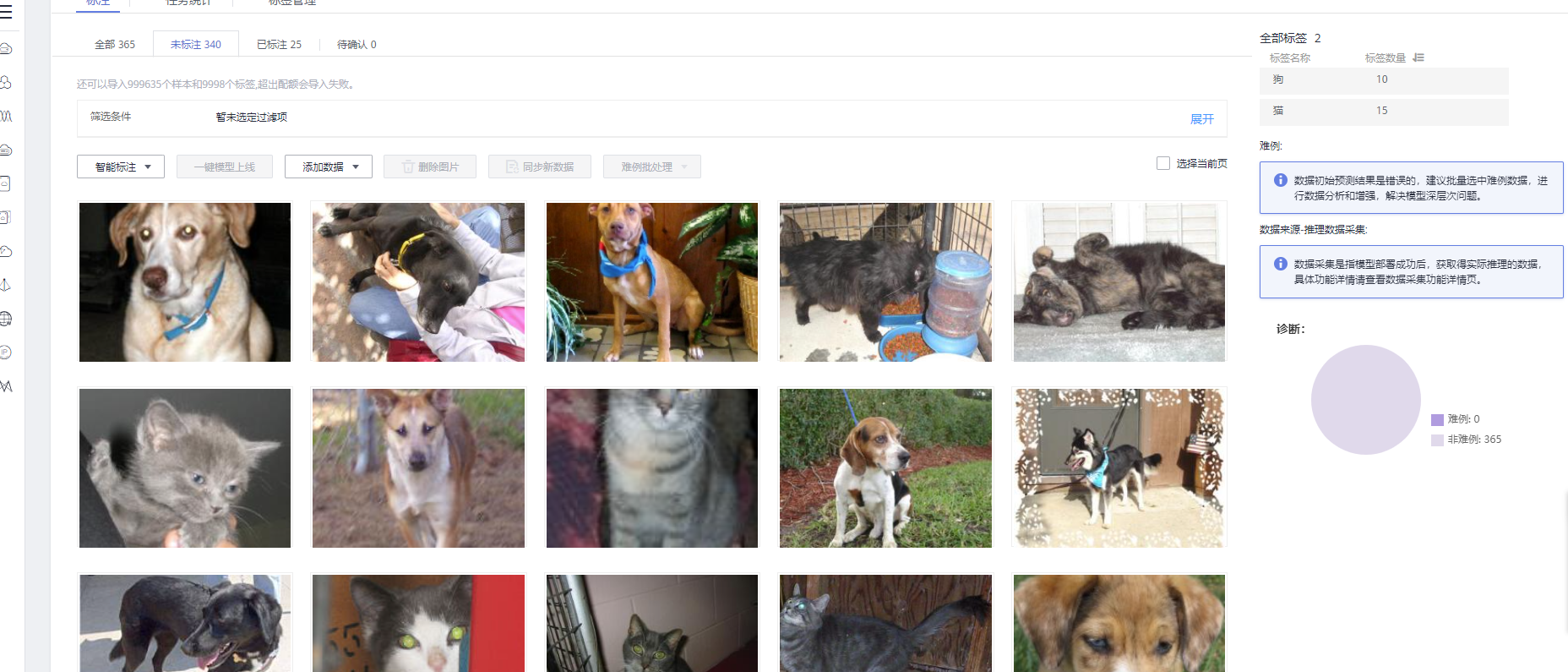
进入input后上传对象，把要分类的图片上传至input，本例子中使用400张左右的猫狗图片

* 1. 回到服务列表，找出ModelArts ，在左侧菜单中选择数据管理，点击数据集，并点击链接创建一个新的委托

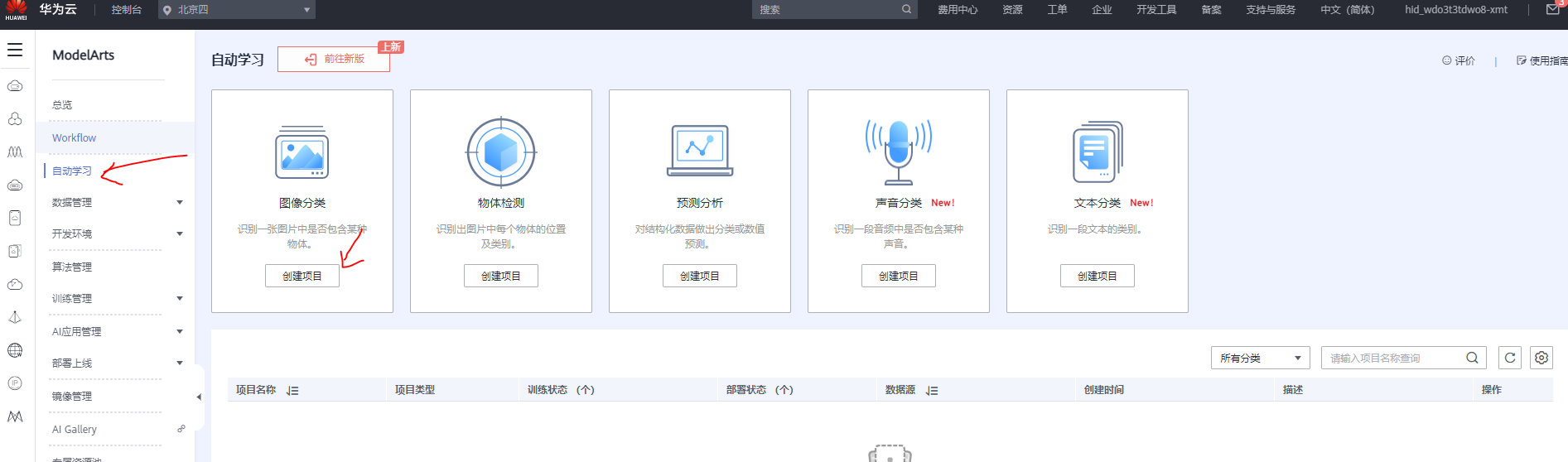




* 1. 选择数据管理中的数据标注，手动为图片添加猫，狗标签

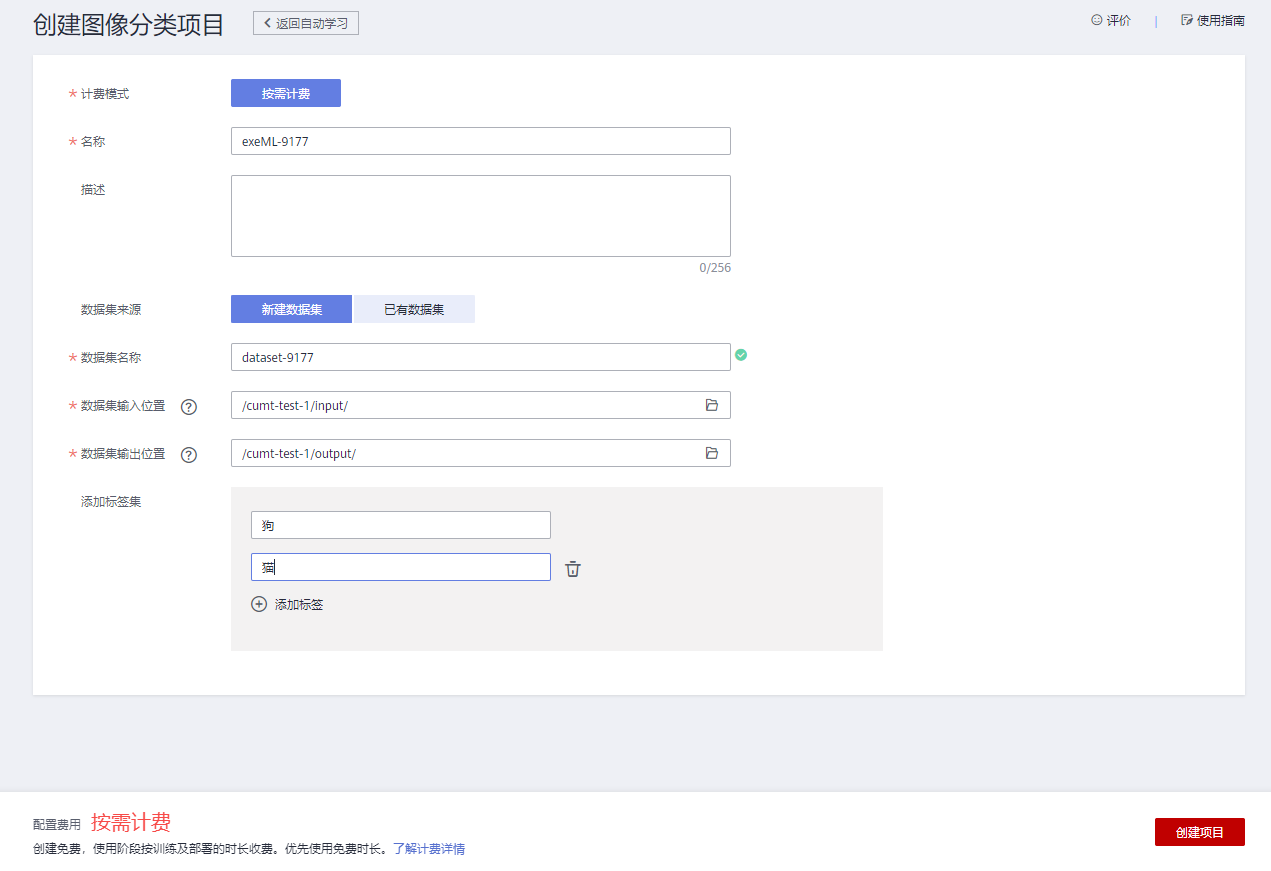


* 1. 之后点击左侧自动学习，创建一个图像分类项目

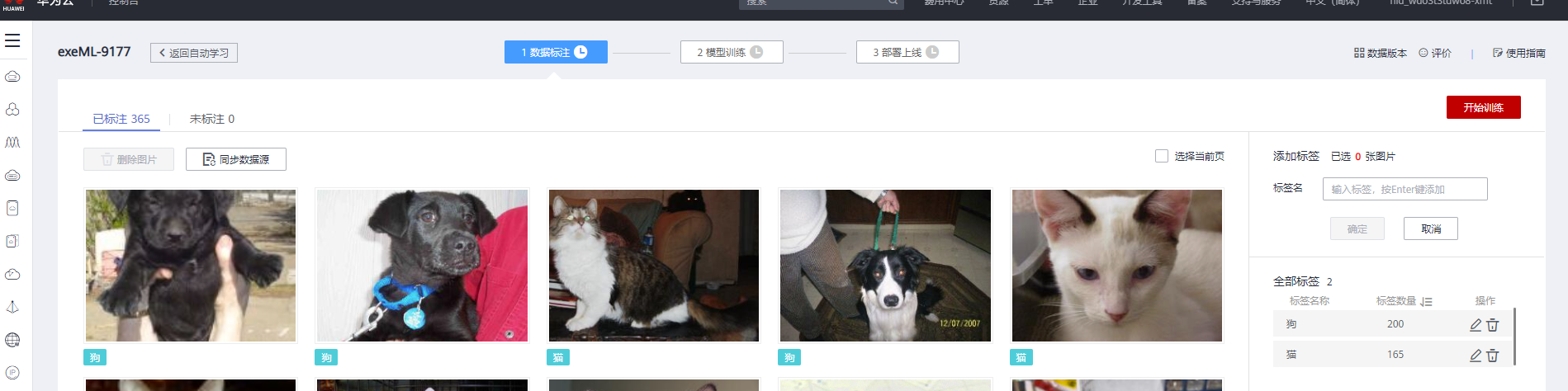


* 1. 名称任意，数据集名称任意，数据集输入为input输出为output

并添加标签：狗，猫

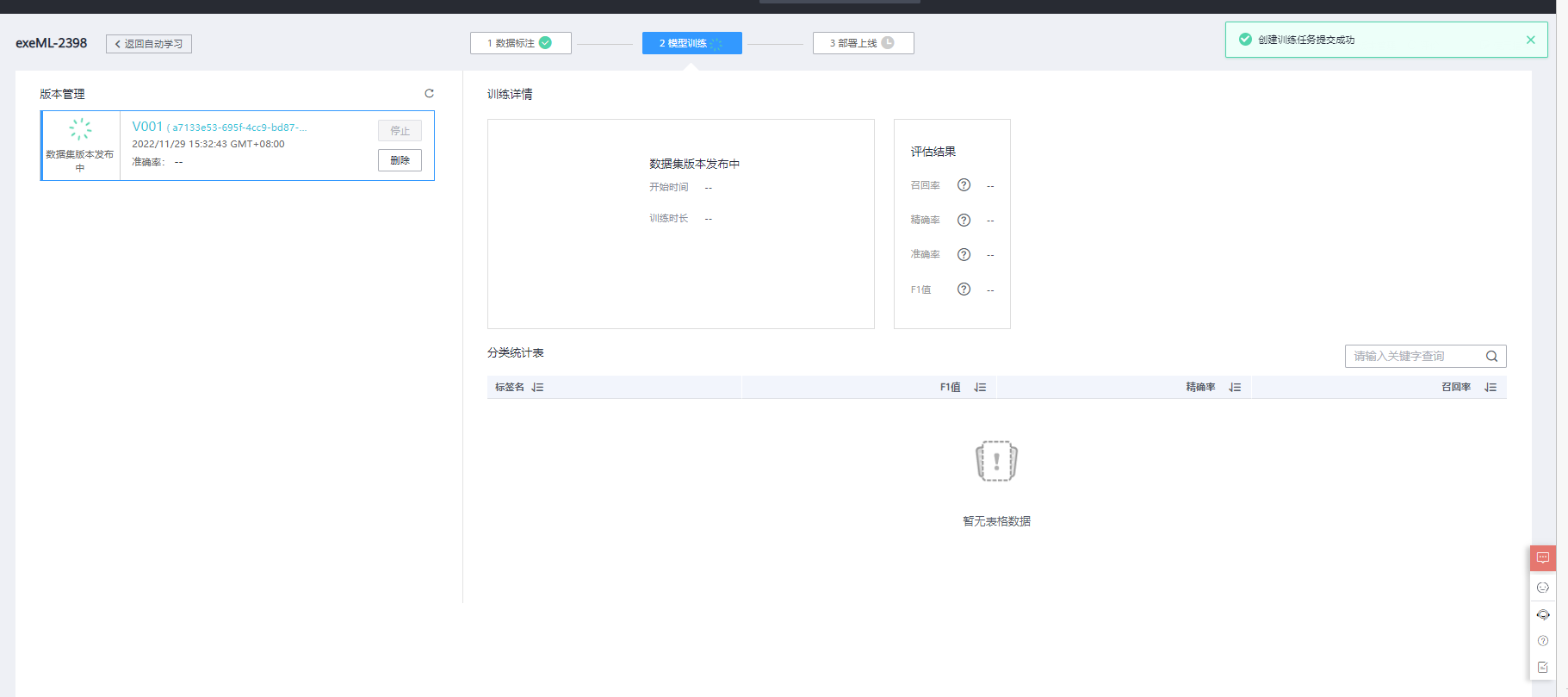


* 1. 进入项目点击开始训练即可使用预存的图像分类方法进行训练

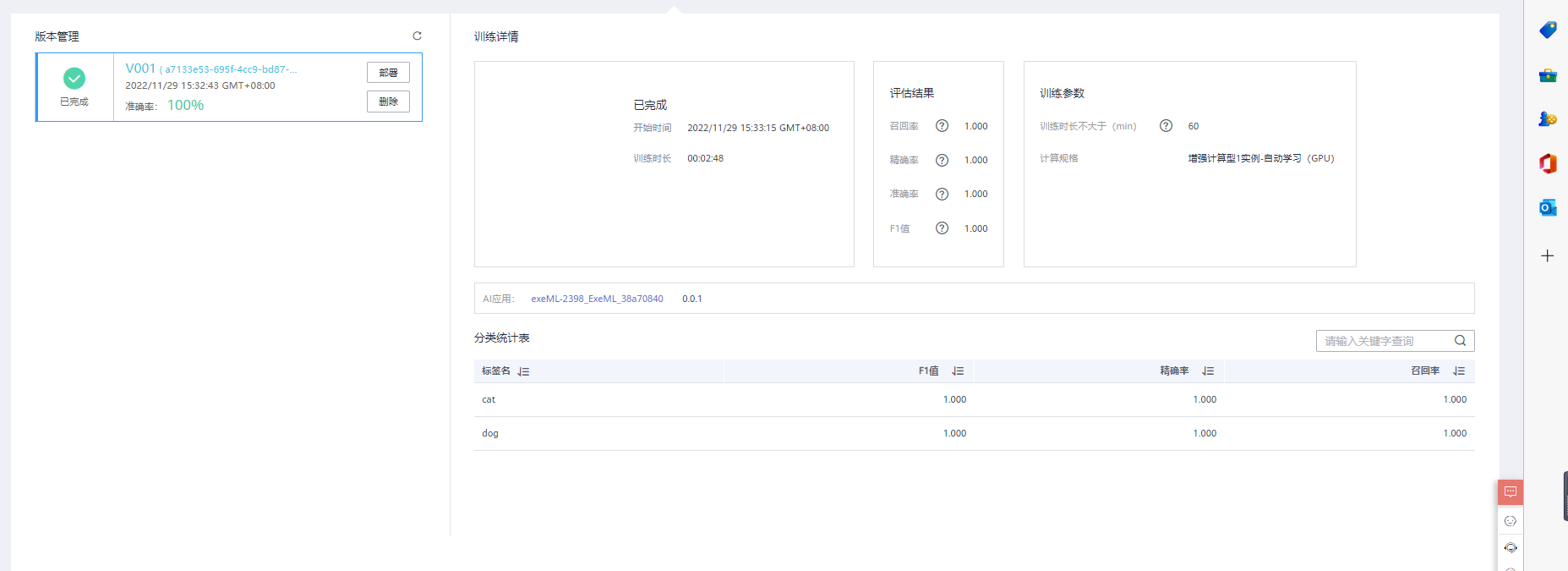




* 1. 训练中。。。



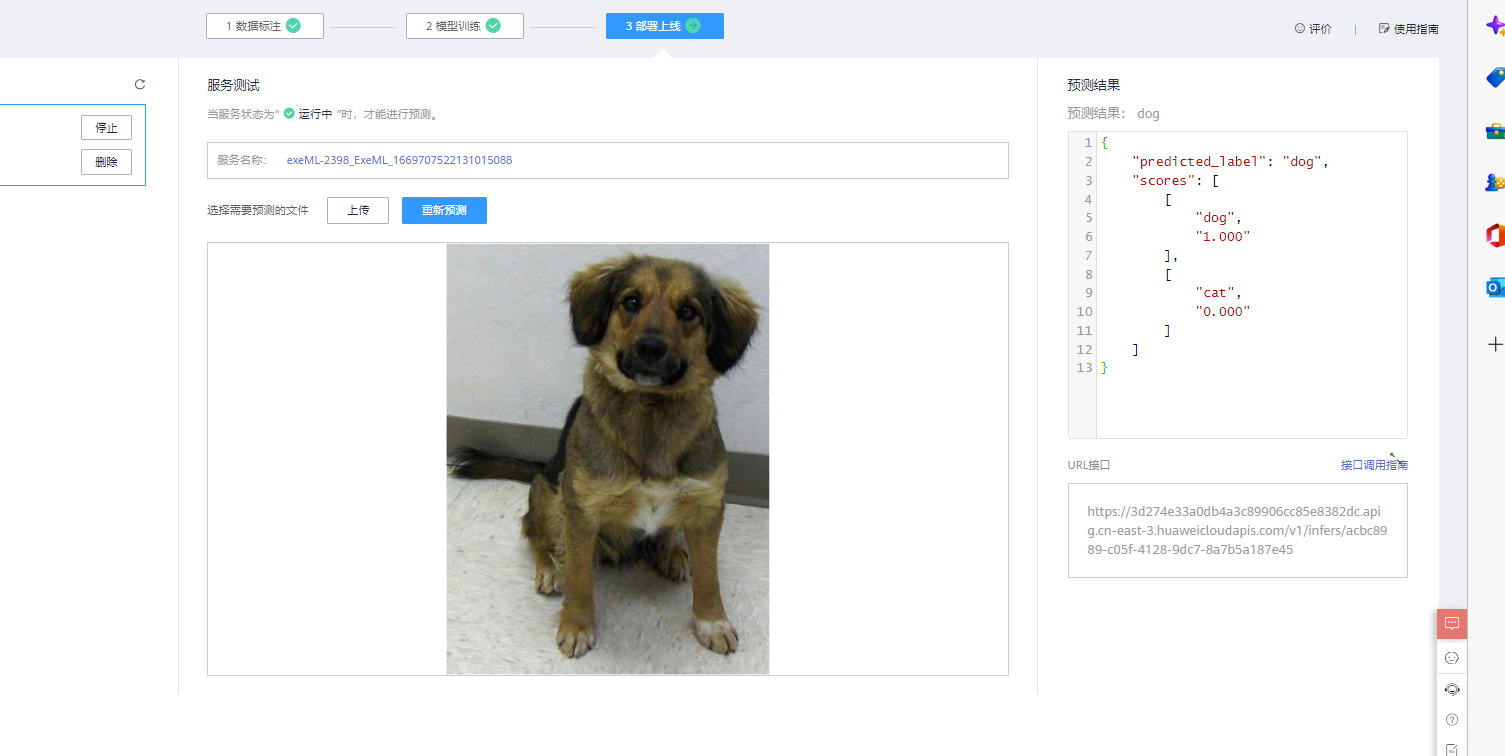
训练结束



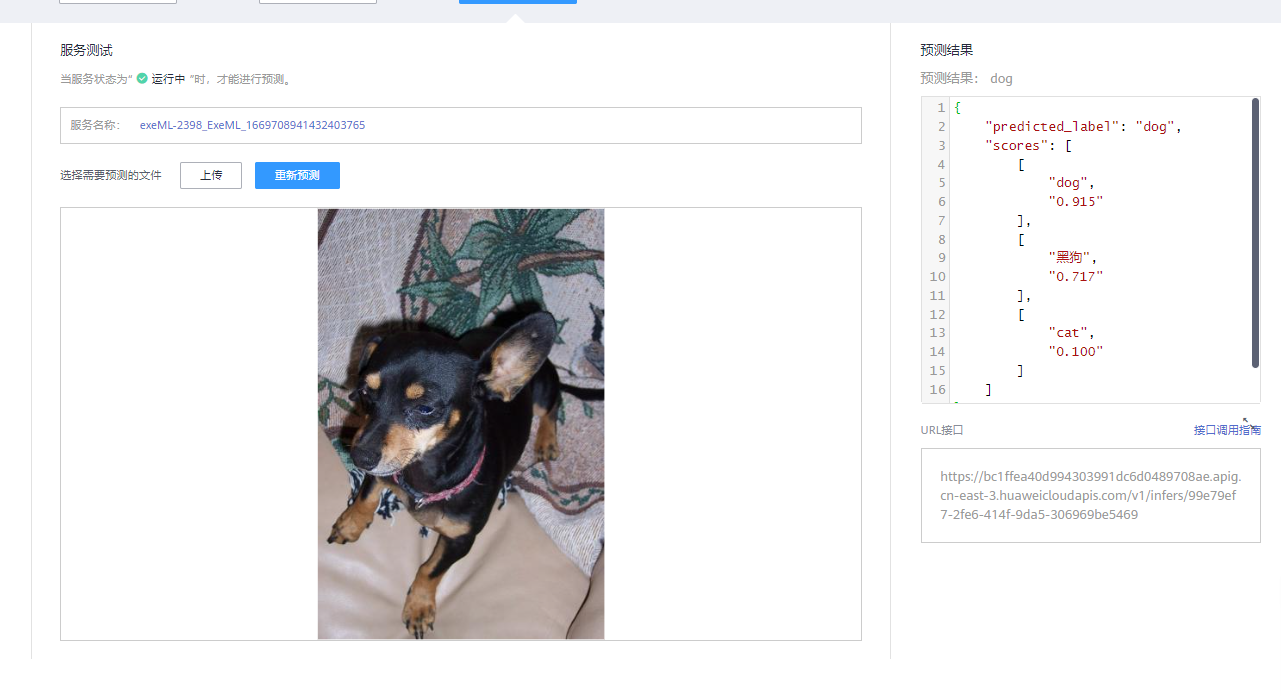
* 1. 点击部署，即可查询训练结果以及测试



* 1. 可以通过上传新的图片来测试模型结果

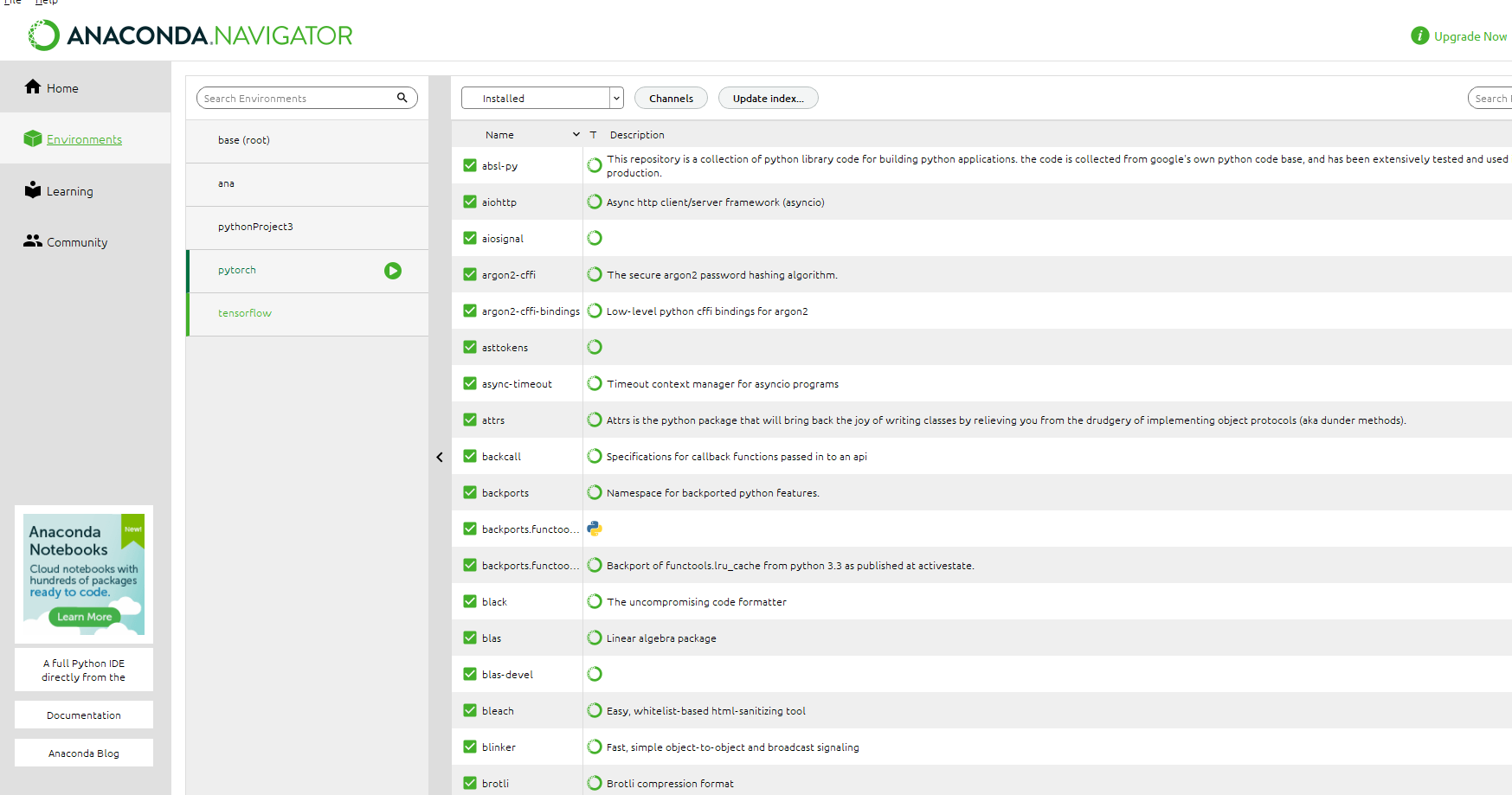


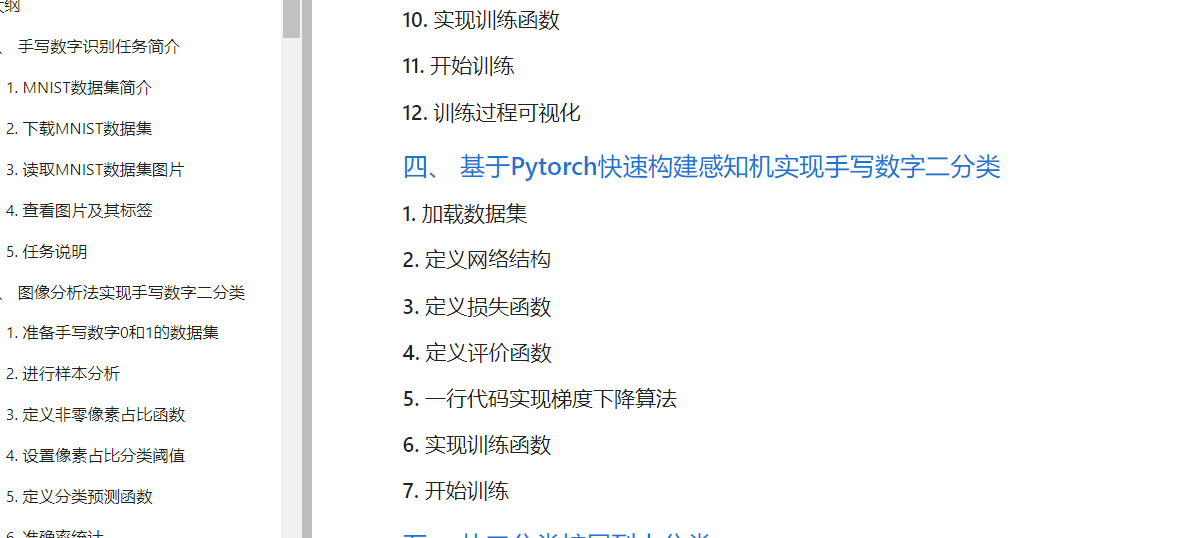
当我们新增标签为黑狗时，发现识别情况也较为可观。



下面是通过我本人近期通过华为云指导学习的深度学习实践知识。下面展示第四章的内容。

本地已具有pytorch环境（采用anaconda进行管理）





**初始化一个张量**

torch.tensor([[1., -1.], [1., -1.]])  
torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))

硬性翻译：参考网站[https://pytorch.org/docs/stable/tensors.html#initializing-and-basic-operations](https://pytorch.org/docs/stable/tensors.html" \l "initializing-and-basic-operations)

**专业名词解释**

**1.参数与超参数**

参数：模型f(x,θ）中θ为模型参数，可以通过优化算法进行学习

超参数：用来定义模型结构或优化策略。

**2.batch\_size**

每次处理的数据数量。

**3.epoch轮次**

把一个数据集，循环运行多少轮。

**4.transforms变换**

主要讲图片转化为tensor，旋转图片，以及正则化。

**5.nomalize正则化**

模型出现过拟合现象时，降低模型复杂度。

**6.卷积层**

由卷积核构建，卷积核简称为卷积，也称为滤波器，卷积的大小可以在实际需要时自定义其长和宽

**7.池化层**

对图片进行压缩(降采样)的一种方法，如max pooling average pooling 等。

**8.激活层**

激活函数的作用就是，在所有的隐藏层之间添加一个激活函数，这样的输出就是一个非线性函数了，因而神经网络的表达能力更加强大

**9.损失函数**

在深度学习中，损失反应模型最后预测结果与实际真值之间的差距，可以用来分析训练过程的好坏，模型是否收敛等，例如均等损失，交叉熵损失等。

**手写字体的识别流程**

1. 定义超参数
2. 构建transforms，主要对图像做变换
3. 下载、加载数据集MNIST
4. 构建网络模型法
5. 定义训练方法
6. 定义测试方法
7. 开始训练模型，输出预测结果

**实战**

**代码**

#加载必要的库  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
from torchvision import datasets,transforms  
​  
#定义超参数  
BATCH\_SIZE=16#每批处理的数据  
DEVICE=torch.device("cuda"if torch.cuda.is\_available()else"cpu")#用cpu还是gpu  
EPOCHS=10#训练数据集的轮次  
​  
#构建pipline 对图像做处理  
pipeline =transforms.Compose([  
    transforms.ToTensor(),#将图片转换成tensor  
    transforms.Normalize((0.1307,),(0.3081,))#正则化 降低模型复杂度  
 ])  
​  
#下载、加载数据集  
from torch.utils.data import DataLoader  
​  
#下载数据集  
train\_set=datasets.MNIST("data",train=True,download=True,transform=pipeline)  
test\_set=datasets.MNIST("data",train=False,download=True,transform=pipeline)  
#加载数据  
train\_loader=DataLoader(train\_set,batch\_size=BATCH\_SIZE,shuffle=True)#shuffle是打乱的意思  
test\_loader=DataLoader(train\_set,batch\_size=BATCH\_SIZE,shuffle=True)  
​  
#构建网络模型  
class Digit(nn.Module):#继承Moudle类  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.convl=nn.Conv2d(1,10,5)#二维卷积 1：灰度图片的通道 10：输出通道 5：卷积核kernl  
        self.conv2=nn.Conv2d(10,20,3)#10:输入通道20 ：输出通道 3：卷积核大小  
        #全连接层 线性层  
        self.fcl=nn.Linear(20\*10\*10,500)##20\*10\*10输入通道 500输出通道  
        self.fc2=nn.Linear(500,10)#500输入通道 10输出通道  
    def forward(self,x):  
        input\_size=x.size(0)#batch\_size  
        x=self.convl(x)  #输入：batch\*1\*28\*28，输出：batch\*10\*24\*24（28-5+1）  
        x=F.relu(x)#保持shape不变 激活层  
        x=F.max\_pool2d(x,2,2)#池化层 对图片进行压缩   输入：batch\*10\*24\*24  
        #输出 batch\*10\*12\*12  
        x=self.conv2(x)#输入：batch\*10\*12\*12 输出：batch\*20\*（12-3+1）-（12-3+1）  
        x=F.relu(x)  
        #拉伸  
        x=x.view(input\_size,-1)  #-1 自动计算维度20\*10\*10=2000  
        #进入全连接层  
        x=self.fcl(x)#输入：batch\*2000 输出batch\*500  
        x=F.relu(x)  
        x=self.fc2(x)#输入batch\*500 输出：batch\*10  
        output=F.log\_softmax(x,dim=1)#计算分类，每个数字的概率值  
        return output  
​  
#定义优化器  
model=Digit().to(DEVICE)  
optimizer = optim.Adam(model.parameters())  
#定义训练方法  
def train\_model(model,device,train\_loader,optimizer,epoch):  
    model.train()  
    for batch\_index,(data,target) in enumerate(train\_loader):  
        data,target=data.to(device),target.to(device)#部署到device上去  
         #梯度初始化为0  
        optimizer.zero\_grad()  
        #训练后的结果  
        output=model(data)  
        #计算损失  
        loss=F.cross\_entropy(output,target)#交叉熵损失函数  
        #找到概率值最大的下标  
        pred=output.max(1,keepdim=True)  
        #反向传播  
        loss.backward()  
        #参数优化  
        optimizer.step()  
        if batch\_index%3000==0:  
            print("Train Epoch : {} \t Loss:{:.6f}".format(epoch,loss.item()))  
​  
#定义测试方法  
def test\_model(model,device,test\_loader):  
    #模拟验证  
    model.eval()  
    #正确率  
    correct=0.0  
    #测试损失  
    test\_loss=0.0  
    with torch.no\_grad():#不会计算梯度，也不会进行反向传播  
        for data, target in test\_loader:  
            #部署到device上  
            data,target=data.to(device),target.to(device)  
            #测试数据  
            output=model(data)  
            #测试损失  
            test\_loss+=F.cross\_entropy(output,target).item()  
            #找到概率值最大的索引  
            pred=output.max(1,keepdim=True)[1]#值 索引  
            #另外两种方法 pred=torch.max(output,dim=1)  
            #pred=output.argmax(dim=1)  
            #累计正确的数目  
            correct+=pred.eq(target.view\_as(pred)).sum().item()  
        test\_loss/=len(test\_loader.dataset)  
        print("Test-Average loss:{:.4f},Accuracy:{:.3f}\n".format(test\_loss,100.0\*correct/len(test\_loader.dataset)))  
​  
#调用方法  
for epoch in range(1,EPOCHS+1):  
    train\_model(model,DEVICE,train\_loader,optimizer,epoch)  
    test\_model(model,DEVICE,test\_loader)

**运行结果**

C:\ana\envs\pytorch\python.exe D:/pythonProject1/main.pyTrain Epoch : 1 Loss:2.325215Train Epoch : 1 Loss:0.001763Test-Average loss:0.0023,Accuracy:98.838

Train Epoch : 2 Loss:0.003396Train Epoch : 2 Loss:0.000558Test-Average loss:0.0015,Accuracy:99.257

Train Epoch : 3 Loss:0.038988Train Epoch : 3 Loss:0.009089Test-Average loss:0.0010,Accuracy:99.525

Train Epoch : 4 Loss:0.013535Train Epoch : 4 Loss:0.000193Test-Average loss:0.0006,Accuracy:99.665

Train Epoch : 5 Loss:0.000430Train Epoch : 5 Loss:0.065546Test-Average loss:0.0009,Accuracy:99.478

Train Epoch : 6 Loss:0.000847Train Epoch : 6 Loss:0.000007Test-Average loss:0.0004,Accuracy:99.787

Train Epoch : 7 Loss:0.001314Train Epoch : 7 Loss:0.000021Test-Average loss:0.0004,Accuracy:99.785

Train Epoch : 8 Loss:0.000016Train Epoch : 8 Loss:0.000010Test-Average loss:0.0003,Accuracy:99.853

Train Epoch : 9 Loss:0.000063Train Epoch : 9 Loss:0.000112Test-Average loss:0.0004,Accuracy:99.847

Train Epoch : 10 Loss:0.000394Train Epoch : 10 Loss:0.003856Test-Average loss:0.0004,Accuracy:99.817

精准度在99.817以上

实验体会：

通过本次实验本人增长了深度学习的相关知识，熟悉了华为云深度学习平台的使用，并通过自己的实践完成了手写字识别的案例。了解到深度学习网络其中的奥秘。