ModelArts

快速入门（国际站）

文档版本 01

发布日期 2024-09-18

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 ModelArts 入门指引

本文旨在帮助您了解ModelArts的基本使用流程以及相关的常见问题，帮助您快速上手ModelArts服务。

面向不同AI基础的开发者，本文档提供了相应的入门教程，帮助用户更快速地了解ModelArts的功能，您可以根据经验选择相应的教程。

根据经验选择您的使用方式

- 如果您是业务开发者，没有AI开发经验，您可以使用ModelArts的自动学习功能，进行零基础构建AI模型，详细介绍请参见**使用ModelArts Standard自动学习实现物体检测**。
- 如果您是AI工程师，熟悉代码编写和调测，您可以使用ModelArts提供的在线代码开发环境，编写训练代码进行AI模型的开发，详细介绍请参见**使用线上环境Notebook构建模型**。
- 如果您有自己的算法，想改造适配后迁移到ModelArts平台上进行训练和推理，您可以参考**使用自定义算法构建模型（手写数字识别）**。

2 使用 ModelArts Standard 自定义算法实现手写数字识别

本文为用户提供如何将本地的自定义算法通过简单的代码适配，实现在ModelArts上进行模型训练与部署的全流程指导。

场景描述

本案例用于指导用户使用PyTorch1.8实现手写数字图像识别，示例采用的数据集为MNIST官方数据集。

通过学习本案例，您可以了解如何在ModelArts平台上训练作业、部署推理模型并预测的完整流程。

操作流程

开始使用如下样例前，请务必按**准备工作**指导完成必要操作。

1. **步骤一：准备训练数据**：下载MNIST数据集。
2. **步骤二：准备训练文件和推理文件**：编写训练与推理代码。
3. **步骤三：创建OBS桶并上传文件**：创建OBS桶和文件夹，并将数据集和训练脚本，推理脚本，推理配置文件上传到OBS中。
4. **步骤四：创建训练作业**：进行模型训练。
5. **步骤五：推理部署**：训练结束后，将生成的模型导入ModelArts用于创建AI应用，并将AI应用部署为在线服务。
6. **步骤六：预测结果**：上传一张手写数字图片，发起预测请求获取预测结果。
7. **后续操作：清除资源**：运行完成后，停止服务并删除OBS中的数据，避免不必要的扣费。

准备工作

- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

b. 在弹出的“添加授权”窗口中，选择：

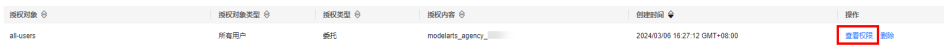
- **授权对象类型：所有用户**
- **委托选择：新增委托**
- **权限配置：普通用户**

选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 2-1 配置委托访问授权

c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

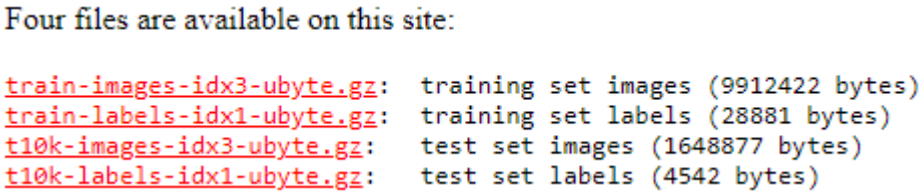
图 2-2 查看委托配置信息



步骤一：准备训练数据

本案例使用的数据是MNIST数据集，您可以在浏览器中搜索“MNIST数据集”下载如**图2-3**所示的4个文件。

图 2-3 MNIST 数据集



- “train-images-idx3-ubyte.gz”：训练集的压缩包文件，共包含60000个样本。
- “train-labels-idx1-ubyte.gz”：训练集标签的压缩包文件，共包含60000个样本的类别标签。
- “t10k-images-idx3-ubyte.gz”：验证集的压缩包文件，共包含10000个样本。

- “t10k-labels-idx1-ubyte.gz”：验证集标签的压缩包文件，共包含10000个样本的类别标签。

说明

以上数据样例仅供参考。

步骤二：准备训练文件和推理文件

针对此案例，ModelArts提供了需使用的训练脚本、推理脚本和推理配置文件。请参考如下文件内容。

说明

粘贴“.py”文件代码时，请直接新建“.py”文件，否则会可能出现“SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence”报错。

在本地电脑中创建训练脚本“train.py”，内容如下：

```
# base on https://github.com/pytorch/examples/blob/main/mnist/main.py

from __future__ import print_function

import os
import gzip
import codecs
import argparse
from typing import IO, Union

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

import shutil


# 定义网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
# 模型训练，设置模型为训练模式，加载训练数据，计算损失函数，执行梯度下降
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


# 模型验证，设置模型为验证模式，加载验证数据，计算损失函数和准确率
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


# 以下为pytorch mnist
# https://github.com/pytorch/vision/blob/v0.9.0/torchvision/datasets/mnist.py
def get_int(b: bytes) -> int:
    return int(codecs.encode(b, 'hex'), 16)


def open_maybe_compressed_file(path: Union[str, IO]) -> Union[IO, gzip.GzipFile]:
    """Return a file object that possibly decompresses 'path' on the fly.
       Decompression occurs when argument `path` is a string and ends with '.gz' or '.xz'.
    """
    if not isinstance(path, torch._six.string_classes):
        return path
    if path.endswith('.gz'):
        return gzip.open(path, 'rb')
    if path.endswith('.xz'):
        return lzma.open(path, 'rb')
    return open(path, 'rb')


SN3_PASCALVINCENT_TYPEMAP = {
    8: (torch.uint8, np.uint8, np.uint8),
    9: (torch.int8, np.int8, np.int8),
    11: (torch.int16, np.dtype('>i2'), 'i2'),
    12: (torch.int32, np.dtype('>i4'), 'i4'),
    13: (torch.float32, np.dtype('>f4'), 'f4'),
    14: (torch.float64, np.dtype('>f8'), 'f8')
}


def read_sn3_pascalvincent_tensor(path: Union[str, IO], strict: bool = True) -> torch.Tensor:
```

```
    """Read a SN3 file in "Pascal Vincent" format (Lush file 'libidx/idx-io.lsh').
       Argument may be a filename, compressed filename, or file object.
    """
    # read
    with open_maybe_compressed_file(path) as f:
        data = f.read()
    # parse
    magic = get_int(data[0:4])
    nd = magic % 256
    ty = magic // 256
    assert 1 <= nd <= 3
    assert 8 <= ty <= 14
    m = SN3_PASCALVINCENT_TYPEMAP[ty]
    s = [get_int(data[4 * (i + 1): 4 * (i + 2)]) for i in range(nd)]
    parsed = np.frombuffer(data, dtype=m[1], offset=(4 * (nd + 1)))
    assert parsed.shape[0] == np.prod(s) or not strict
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)


def read_label_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 1)
    return x.long()


def read_image_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 3)
    return x


def extract_archive(from_path, to_path):
    to_path = os.path.join(to_path, os.path.splitext(os.path.basename(from_path))[0])
    with open(to_path, "wb") as out_f, gzip.GzipFile(from_path) as zip_f:
        out_f.write(zip_f.read())
# --- 以上为pytorch mnist
# --- end


# raw mnist 数据处理
def convert_raw_mnist_dataset_to_pytorch_mnist_dataset(data_url):
    """
    raw

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz

    processed

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz
        MNIST/raw
            train-images-idx3-ubyte
            train-labels-idx1-ubyte
            t10k-images-idx3-ubyte
            t10k-labels-idx1-ubyte
        MNIST/processed
            training.pt
            test.pt
```

```
    """
    resources = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz"
    ]

    pytorch_mnist_dataset = os.path.join(data_url, 'MNIST')

    raw_folder = os.path.join(pytorch_mnist_dataset, 'raw')
    processed_folder = os.path.join(pytorch_mnist_dataset, 'processed')

    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    print('Processing...')

    for f in resources:
        extract_archive(os.path.join(data_url, f), raw_folder)

    training_set = (
        read_image_file(os.path.join(raw_folder, 'train-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 'train-labels-idx1-ubyte'))
    )
    test_set = (
        read_image_file(os.path.join(raw_folder, 't10k-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 't10k-labels-idx1-ubyte'))
    )
    with open(os.path.join(processed_folder, 'training.pt'), 'wb') as f:
        torch.save(training_set, f)
    with open(os.path.join(processed_folder, 'test.pt'), 'wb') as f:
        torch.save(test_set, f)

    print('Done!')


def main():
    # 定义可以接收的训练作业运行参数
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')

    parser.add_argument('--data_url', type=str, default=False,
                        help='mnist dataset path')
    parser.add_argument('--train_url', type=str, default=False,
                        help='mnist model path')

    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True,
                        help='For Saving the current Model')
    args = parser.parse_args()

    use_cuda = not args.no_cuda and torch.cuda.is_available()
```

```
    torch.manual_seed(args.seed)

    # 设置使用 GPU 还是 CPU 来运行算法
    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    # 定义数据预处理方法
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    # 将 raw mnist 数据集转换为 pytorch mnist 数据集
    convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

    # 分别创建训练和验证数据集
    dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                       transform=transform)
    dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                       transform=transform)

    # 分别构建训练和验证数据迭代器
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # 初始化神经网络模型并复制模型到计算设备上
    model = Net().to(device)
    # 定义训练优化器和学习率策略，用于梯度下降计算
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    # 训练神经网络，每一轮进行一次验证
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # 保存模型与适配 ModelArts 推理模型包规范
    if args.save_model:

        # 在 train_url 训练参数对应的路径内创建 model 目录
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # 按 ModelArts 推理模型包规范，保存模型到 model 目录内
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # 复制推理代码与配置文件到 model 目录内
        the_path_of_current_file = os.path.dirname(__file__)
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/customize_service.py'),
os.path.join(model_path, 'customize_service.py'))
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/config.json'), os.path.join(model_path,
'config.json'))

if __name__ == '__main__':
    main()
```

在本地电脑中创建推理脚本“customize_service.py”，内容如下：

```
import os
import log
```

```
import json

import torch.nn.functional as F
import torch.nn as nn
import torch
import torchvision.transforms as transforms

import numpy as np
from PIL import Image

from model_service.pytorch_model_service import PTServingBaseService

logger = log.getLogger(__name__)

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 模型推理服务
class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)

        # 调用自定义函数加载模型
        self.model = Mnist(model_path)

        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # 接收request数据，并转换为模型可以接受的输入格式
    def _preprocess(self, data):
        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    # 将推理的结果进行后处理，得到预期的输出格式，该结果就是最终的返回值
    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    # 对于输入数据进行前向推理，得到推理结果
    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)
```

```
        return result

# 定义网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()

    # 加载模型
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))

    # CPU 或者 GPU 映射
    model.to(device)

    # 声明为推理模式
    model.eval()

    return model
```

在本地电脑中推理配置文件“config.json”，内容如下：

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```

步骤三：创建 OBS 桶并上传文件

将上一步中的数据和代码文件、推理代码文件与推理配置文件，从本地上传到OBS桶中。在ModelArts上运行训练作业时，需要从OBS桶中读取数据和代码文件。

1. 登录OBS管理控制台，按照如下示例创建OBS桶和文件夹。

   创建的OBS桶所在区域和后续使用ModelArts必须在同一个区域Region，否则会导致训练时找不到OBS桶。具体操作可参见**查看OBS桶与ModelArts是否在同一区域**。

创建OBS桶时，桶的存储类别请勿选择“归档存储”，归档存储的OBS桶会导致模型训练失败。

OBS桶路径和文件夹命名要求如下：

- {OBS桶}：OBS对象桶，用户可以自定义名称，例如：test-modelarts-xx
- {OBS文件夹}：OBS文件夹，自定义名称，此处举例为pytorch
- - mnist-data：OBS文件夹，用于存放训练数据集，可以自定义名称，此处举例为mnist-data
- - mnist-code：OBS文件夹，用于存放训练脚本train.py，可以自定义名称，此处举例为mnist-code
- - infer：OBS文件夹，用于存放推理脚本customize_service.py和配置文件config.json
- - mnist-output ： OBS文件夹，用于存放训练输出模型，可以自定义名称，此处举例为mnist-output

2. 上传**Step1 准备训练数据**中下载的MNIST数据集压缩包文件到OBS的“mnist-data”文件夹中。

⚠ 注意

- 上传数据到OBS中时，请不要加密，否则会导致训练失败。
- 文件无需解压，直接上传压缩包至OBS中即可。

3. 上传训练脚本“train.py”到“mnist-code”文件夹中。
4. 上传推理脚本“customize_service.py”和推理配置文件“config.json”到“mnist-code”的“infer”文件中。

步骤四：创建训练作业

1. 登录ModelArts管理控制台，选择和OBS桶相同的区域。
2. 在“权限管理”中检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
3. 在左侧导航栏选择“模型训练 > 训练作业”进入训练作业页面，单击“创建训练作业”。
4. 填写创建训练作业相关信息。
   - “创建方式”：选择“自定义算法”。
   - “启动方式”：选择“预置框架”，下拉框中选择PyTorch，pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64。
   - “代码目录”：选择已创建的OBS代码目录路径，例如“/test-modelarts-xx/pytorch/mnist-code/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “启动文件”：选择代码目录下上传的训练脚本“train.py”。
   - “输入”：单击“增加训练输入”，设置训练输入的“参数名称”为“data_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-data/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “输出”：单击“增加训练输出”，设置训练输出的“参数名称”为“train_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-

xx/pytorch/mnist-output/”（test-modelarts-xx需替换为您的OBS桶名称）。预下载至本地目录选择“不下载”。

- “资源类型”：选择GPU单卡的规格。如果有免费GPU规格，可以选择免费规格进行训练。
- 其他参数保持默认即可。

说明

本样例代码为单机单卡场景，选择GPU多卡规格会导致训练失败。

如果在设置训练输入和输出选择OBS路径时，找不到已创建的OBS桶？请查看创建的桶和ModelArts服务是否在同一区域，详细操作请参考**查看OBS桶与ModelArts是否在同一个区域**。

5. 单击“提交”，确认训练作业的参数信息，确认无误后单击“确定”。

   页面自动返回“训练作业”列表页，当训练作业状态变为“已完成”时，即完成了模型训练过程。

   说明

   本案例的训练作业预计运行十分钟。

   如果训练作业状态一直在等待中状态，表示当前所选的资源池规格资源紧张，作业需要进行排队，请耐心等待。请参考**训练作业一直在等待中（排队）？**。

6. 单击训练作业名称，进入作业详情界面查看训练作业日志信息，观察日志是否有明显的Error信息，如果有则表示训练失败，请根据日志提示定位原因并解决。
7. 在训练详情页左下方单击训练输出路径，如**图2-4**所示，跳转到OBS目录，查看是否存在model文件夹，且model文件夹中是否有生成训练模型。如果未生成model文件夹或者训练模型，可能是训练输入数据不完整导致，请检查训练数据上传是否完整，并重新训练。

图 2-4 训练输出路径

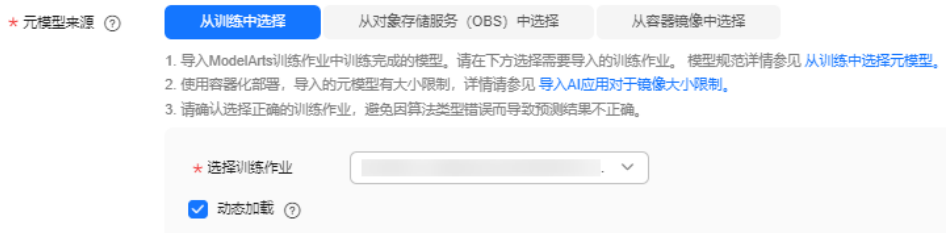
步骤五：推理部署

模型训练完成后，可以创建AI应用，将AI应用部署为在线服务。

1. 在ModelArts管理控制台，单击左侧导航栏中的“AI应用”，进入“自定义应用”页面，单击“创建应用”。
2. 在“创建应用”页面，填写相关参数，然后单击“立即创建”。

   在“元模型来源”中，选择“从训练中选择”页签，选择**步骤四：创建训练作业**中完成的训练作业，勾选“动态加载”。AI引擎的值是系统自动写入的，无需设置。

   图 2-5 设置元模型来源



3. 在AI应用列表页面，当AI应用状态变为“正常”时，表示AI应用创建成功。单击AI应用操作列的“部署”，弹出“版本列表”，单击操作列“部署>在线服务”，将AI应用部署为在线服务。

   图 2-6 部署在线服务

4. 在“部署”页面，参考下图填写参数，然后根据界面提示完成在线服务创建。本案例适用于CPU规格，节点规格需选择CPU。如果有免费CPU规格，可选择免费规格进行部署（每名用户限部署一个免费的在线服务，如果您已经部署了一个免费在线服务，需要先将其删除才能部署新的免费在线服务）。

   图 2-7 部署模型

   完成服务部署后，返回在线服务页面列表页，等待服务部署完成，当服务状态显示为“运行中”，表示服务已部署成功。

步骤六：预测结果

1. 在“在线服务”页面，单击在线服务名称，进入服务详情页面。

2. 单击“预测”页签，请求类型选择“multipart/form-data”，请求参数填写“image”，单击“上传”按钮上传示例图片，然后单击“预测”。

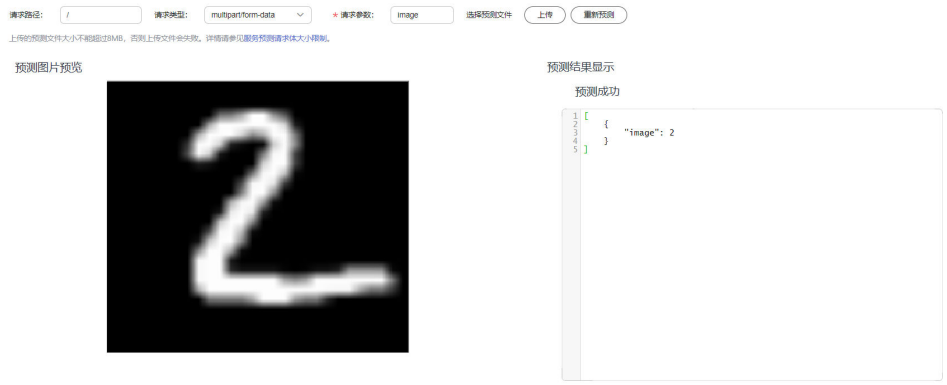
   预测完成后，预测结果显示区域将展示预测结果，根据预测结果内容，可识别出此图片的数字是“2”。

   说明

   本案例中使用的MNIST是比较简单的用做demo的数据集，配套算法也是比较简单的用于教学的神经网络算法。这样的数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练集中的图片相似（黑底白字）才可能预测准确。

图 2-8 示例图片

图 2-9 预测结果展示



后续操作：清除资源

如果不再需要使用此模型及在线服务，建议清除相关资源，避免产生不必要的费用。

- 在“在线服务”页面，“停止”或“删除”刚创建的在线服务。
- 进入OBS，删除本示例使用的OBS桶、文件夹和文件夹中的文件。

3 ModelArts 入门实践

本章节列举了一些常用的实践案例，方便您快速了解并使用ModelArts完成AI开发。

表 3-1 常用最佳实践

实践		描述	适用人群
模型训练	**从 0 制作自定义镜像并用于训练（PyTorch+CPU/GPU）**	本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Pytorch，训练使用的资源是CPU或GPU。	面向熟悉代码编写和调测的AI工程师
推理部署	**从0-1制作自定义镜像并创建AI应用**	针对ModelArts不支持的AI引擎，您可以构建自定义镜像，并将镜像导入ModelArts，创建为AI应用。本文详细介绍如何使用自定义镜像完成AI应用的创建，并部署成在线服务。	