人工智能-实验4

计科210X 甘晴void 202108010XXX

【实验工程】完成的实验已上传github

https://github.com/wolfvoid/ResNet-mushroom-classifier

一、实验目的

- 了解深度学习的基本原理
- 能够使用深度学习开源工具
- 应用深度学习算法求解实际问题

二、实验要求

- 解释深度学习原理
- 采用深度学习框架完成课程综合实验，并对实验结果进行分析
- 回答思考题。

三、 实验内容

选题：基于ResNet的毒蘑菇识别分类问题

3.1 深度学习原理

深度学习是一种机器学习方法，它试图通过模拟人类大脑的工作方式来解决复杂的问题。其核心原理是构建一系列由多层神经网络组成的模型，通过这些网络层逐层提取和学习数据的特征，从而实现对数据的高效建模和预测。

以下是深度学习的基本原理：

1. 神经网络结构：
    - 深度学习模型通常由多个神经网络层组成，每一层都包含多个神经元。这些层可以分为输入层、隐藏层和输出层。
    - 输入层接收原始数据，隐藏层逐层提取数据的特征，输出层生成模型的预测结果。
2. 前向传播：
    - 前向传播是指输入数据通过神经网络从输入层流向输出层的过程。
    - 在前向传播过程中，每一层的神经元将接收上一层的输出，并将其加权求和后通过激活函数输出到下一层。
3. 反向传播：

- 反向传播是深度学习模型通过计算损失函数的梯度，并利用梯度下降法来更新模型参数的过程。
- 通过比较模型的预测结果和真实标签，可以计算出模型预测的误差，然后利用链式法则逐层计算每一层的梯度，从而更新模型参数以最小化误差。

4. 损失函数：

- 损失函数用于衡量模型预测结果与真实标签之间的差异。
- 常见的损失函数包括均方误差（MSE）【回归任务】、交叉熵损失【分类任务】等。

5. 优化算法：

- 优化算法用于更新模型参数以最小化损失函数。
- 常见的优化算法包括随机梯度下降（SGD）、Adam、RMSProp等。

通过不断地调整模型的参数，深度学习模型可以自动学习数据中的复杂模式和特征，从而实现对数据的高效建模和预测。深度学习在图像识别、自然语言处理、语音识别等领域取得了巨大成功，并在许多应用中取得了 state-of-the-art 的性能。

一个比较完整的深度学习流程大致是这样的：

1. 数据准备阶段：

- 收集数据：收集与问题相关的数据集，确保数据集的质量和多样性。
- 数据清洗：处理数据中的噪声、缺失值和异常值。
- 数据标准化：对数据进行标准化或归一化处理，以便模型更好地学习。

2. 模型选择阶段：

- 确定模型类型：根据问题的性质选择适当的深度学习模型，如卷积神经网络（CNN）、循环神经网络（RNN）或者变换器（Transformer）等。
- 模型架构设计：确定模型的层数、每一层的神经元数量以及激活函数等。

3. 训练阶段：

- 划分数据集：将数据集划分为训练集、验证集和测试集。
- 初始化模型参数：随机初始化模型的权重和偏置。
- 前向传播：将训练数据输入到模型中，并计算模型的预测值。
- 计算损失：使用损失函数计算模型预测值与真实值之间的差异。
- 反向传播：根据损失函数的梯度，使用反向传播算法更新模型的参数。
- 优化算法：选择合适的优化算法，如随机梯度下降（SGD）、Adam等，调整模型参数以最小化损失函数。
- 调参：通过调整模型的超参数（如学习率、批量大小等），优化模型的性能。

4. 模型评估阶段：

- 使用验证集评估模型的性能，调整模型的超参数和架构，直到获得满意的性能。
- 使用测试集对最终模型进行评估，评估模型在未见过的数据上的泛化能力。

5. 模型部署阶段：
    - 将训练好的模型部署到生产环境中，以便对新数据进行预测。
    - 监控模型性能，并定期更新模型以适应新的数据和需求。
6. 模型优化阶段：
    - 定期对模型进行优化和调整，以提高模型的性能和稳健性。
    - 可以通过增加数据、调整模型架构、改进算法等方式来优化模型。

后面会讲到，本次实验只包含这个流程的一小部分。

★3.2 ResNet原理

下面ResNet提出的原论文。李沐曾经说过，假设你在使用卷积神经网络，有一半的可能性就是在使用 ResNet 或它的变种。ResNet 论文被引用数量突破了 10 万+。可见ResNet的重要地位。

甚至在之后的2017年的transformer中，也可以见到残差网络的声影，可见其效果与影响的重要性。

《Deep Residual Learning for Image Recognition》

残差网络是为了解决深度神经网络（DNN）隐藏层过多时的网络退化问题而提出。

什么是“退化”？

我们知道，对浅层网络逐渐叠加layers，模型在训练集和测试集上的性能会变好，因为模型复杂度更高了，表达能力更强了，可以对潜在的映射关系拟合得更好。而“退化”指的是，给网络叠加更多的层后，性能却快速下降的情况。

训练集上的性能下降，可以排除过拟合，**BN**（**Batch Normalization**）层的引入也基本解决了**plain net**的梯度消失和梯度爆炸问题。如果不是过拟合以及梯度消失导致的，那原因是什么？

按道理，给网络叠加更多层，浅层网络的解空间是包含在深层网络的解空间中的，深层网络的解空间至少存在不差于浅层网络的解，因为只需将增加的层变成恒等映射，其他层的权重原封不动copy浅层网络，就可以获得与浅层网络同样的性能。更好的解明明存在，为什么找不到？找到的反而是更差的解？

显然，这是个优化问题，反映出结构相似的模型，其优化难度是不一样的，且难度的增长并不是线性的，越深的模型越难以优化。

有两种解决思路，一种是调整求解方法，比如更好的初始化、更好的梯度下降算法等；另一种是调整模型结构，让模型更易于优化。ResNet的设计者采用了后一种方法。

ResNet的作者从后者入手，探求更好的模型结构。将堆叠的几层layer称之为一个block，对于某个block，其可以拟合的函数为$F(x)$，如果期望的潜在映射为$H(x)$，**与其让$F(x)$直接学习潜在的映射，不如去学习残差$H(x)-x$，即$F(x):=H(x)-x$，这样原本的前向路径上就变成了$F(x)+x$，用$F(x)+x$来拟合$H(x)$。**作者认为这样可能更易于优化，因为**相比于让$F(x)$学习成恒等映射，让$F(x)$学习成0要更加容易——后者通过L2正则就可以轻松实现。**这样，对于冗余的block，只需$F(x)\to 0$就可以得到恒等映射，性能不减。

> Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a **residual mapping**. Formally, denoting the desired underlying mapping as $H(x)$, we let the stacked nonlinear layers fit another mapping of $F(x):=H(x)-x$. The original mapping is recast into $F(x)+x$. We **hypothesize** that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, **if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.**
>
> —— from Deep Residual Learning for Image Recognition

下面的问题就变成了$F(x)+x$该怎么设计了。

下面这个结构就是残差网络的核心部件

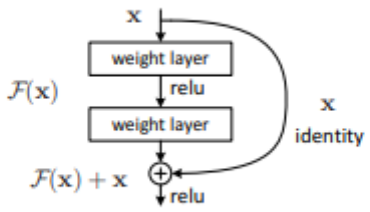
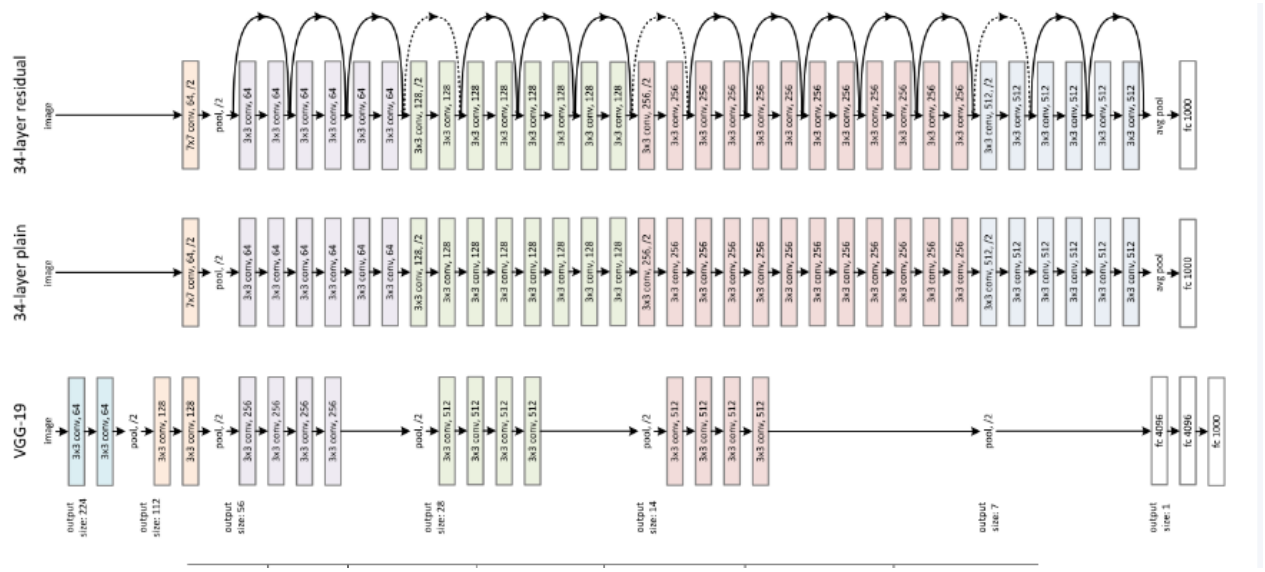


Figure 2. Residual learning: a building block.

这个部件在ResNet中被反复复用，如下（原论文的图片转置）。

这是ResNet18和ResNet50的网络结构对比。其基础部件都是多个`3*3`卷积层和`1*1`卷积层，以及开始的一个`7*7`的卷积层，区别在于层数不一样。



这里从数学原理推导了反向传播中ResNet的一个较好的性质。

令$h(x_l)$为shortcut路径上的变换，f为addition之后的变换，原Residual Block中$f = ReLU$，**当h和f均为恒等映射时，可以得到任意两层x_L和x_l之间的关系，此时信息可以在x_l和x_L间无损直达，如下前向传播中的x_l以及反向传播中的1。**

$$\begin{aligned}\mathbf{y}_l &= h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)\\ \mathbf{x}_{l+1} &= f(\mathbf{y}_l)\\ \mathbf{x}_{l+1} &= \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)\\ \mathbf{x}_L &= \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\end{aligned}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

反向传播中的这个1具有一个很好的性质，**任意两层间的反向传播，这一项都是1，可以有效地避免梯度消失和梯度爆炸。**如果h和f不是恒等映射，就会让这一项变得复杂，若是令其为一个大于或小于1的scale因子，反向传播连乘后就可能导致梯度爆炸或消失，层数越多越明显，这也是ResNet比highway network性能好的原因。**需要注意的是，BN层解决了plain net的梯度消失和爆炸，这里的1可以避免short cut 路径上的梯度消失和爆炸。**

shortcut路径将反向传播由连乘形式变为加法形式，让网络最终的损失在反向传播时可以无损直达每一个block，也意味着每个block的权重更新都部分地直接作用在最终的损失上。看上面前向传播的公式，可以看到某种**ensemble**形式，信息虽然可以在任意两层之间直达，但这种直达其实是隐含的，对某个block而言，它只能看到加法的结果，而不知道加法中每个加数是多数，**从信息通路上讲尚不彻底——由此也诞生了DenseNet**。

对于残差路径的改进，作者进行了不同的对比实验，最终得到了**将BN和ReLU统一放在weight前的full pre-activation结构**。

简单来说，通过引入残差连接，ResNet使得梯度可以更加顺利地从网络的后层传播到前层，避免了梯度消失和梯度爆炸的问题，从而实现了训练非常深的神经网络。

非常神奇，就一个小小的改动，就能提升很多。有时候数学就是这么神奇。

3.3 本次实验代码结构概览

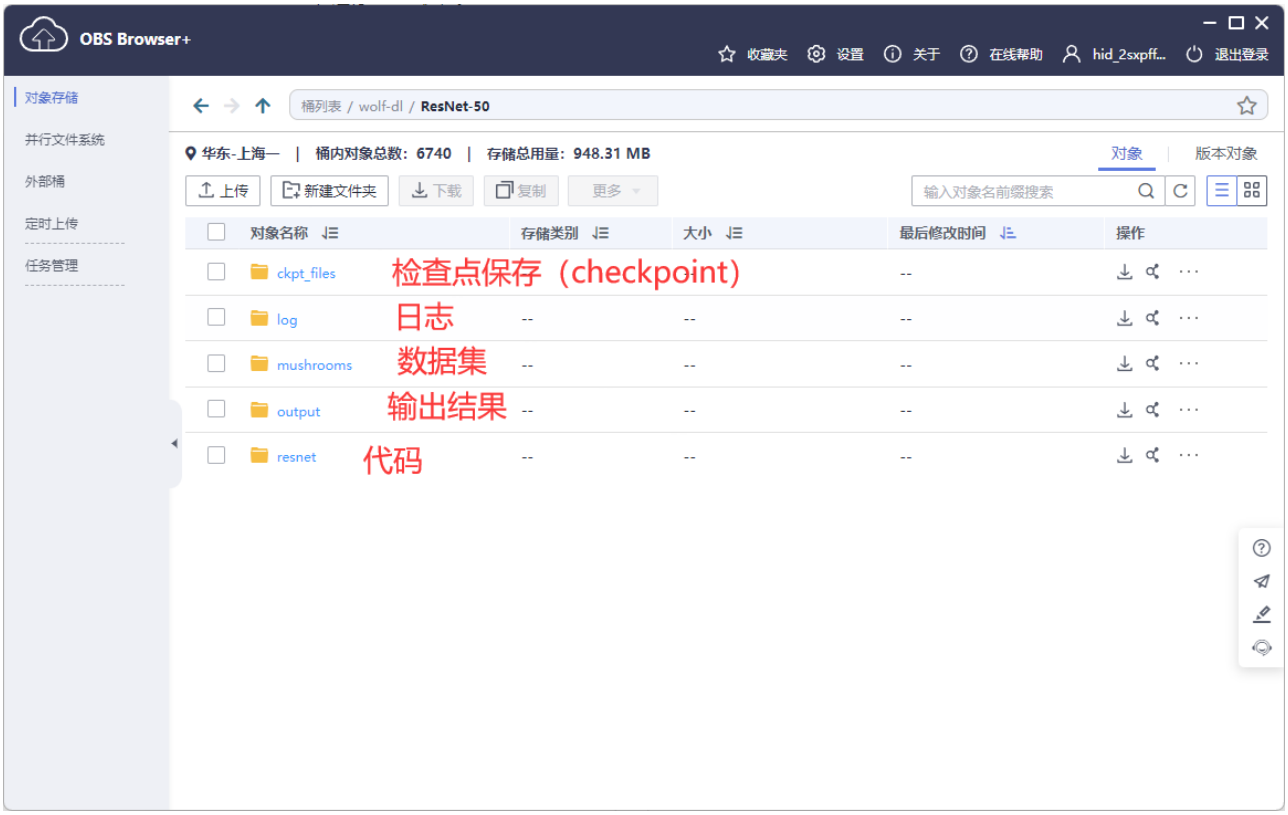
本次实验的代码是现成的，数据集也是现成的。

实际上我能做的只有使用训练数据集和训练代码，训练并保存模型参数权重。然后使用推理代码，对需要预测的数据进行推理得到最终的结果。这上面的一般完整深度学习流程相比已经减少太多了。

我个人理解实验主要是想让我们大致走一遍深度学习流程的主要部分，主要包括训练，推理。另外就是对云平台以及一些框架的使用，怎么让任务跑在云平台上等。

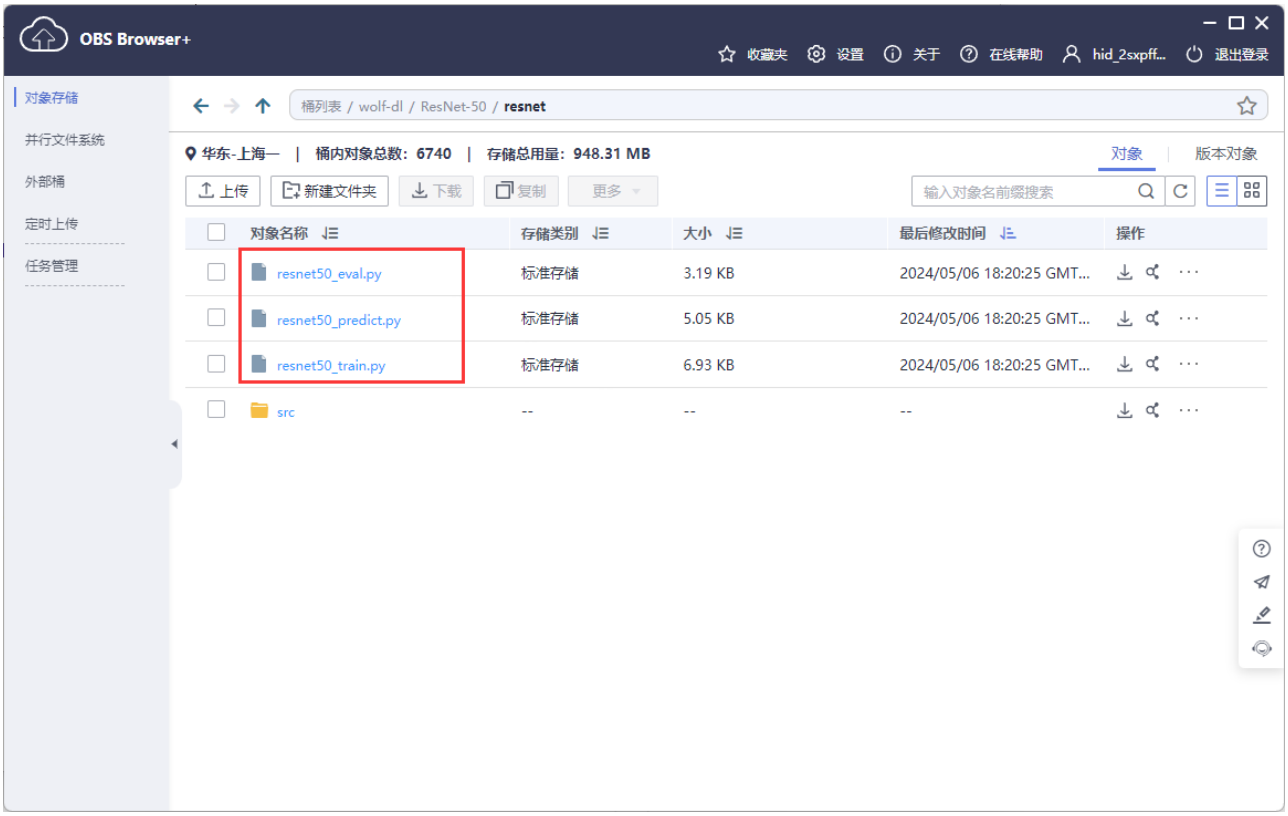
我选择的任务是毒蘑菇的分类，代码使用的是ResNet框架。该框架最早于2015年在《Deep Residual Learning for Image Recognition》中提出，其核心思想是通过跳跃连接将输入直接传递到输出，从而允许网络学习残差（residual）映射，而不是直接学习原始映射。这种残差学习的方式使得网络能够更容易地学习到恒等映射（identity mapping），从而解决了深度网络训练中的梯度消失和梯度爆炸问题。

下面是下载的资源的主要结构，

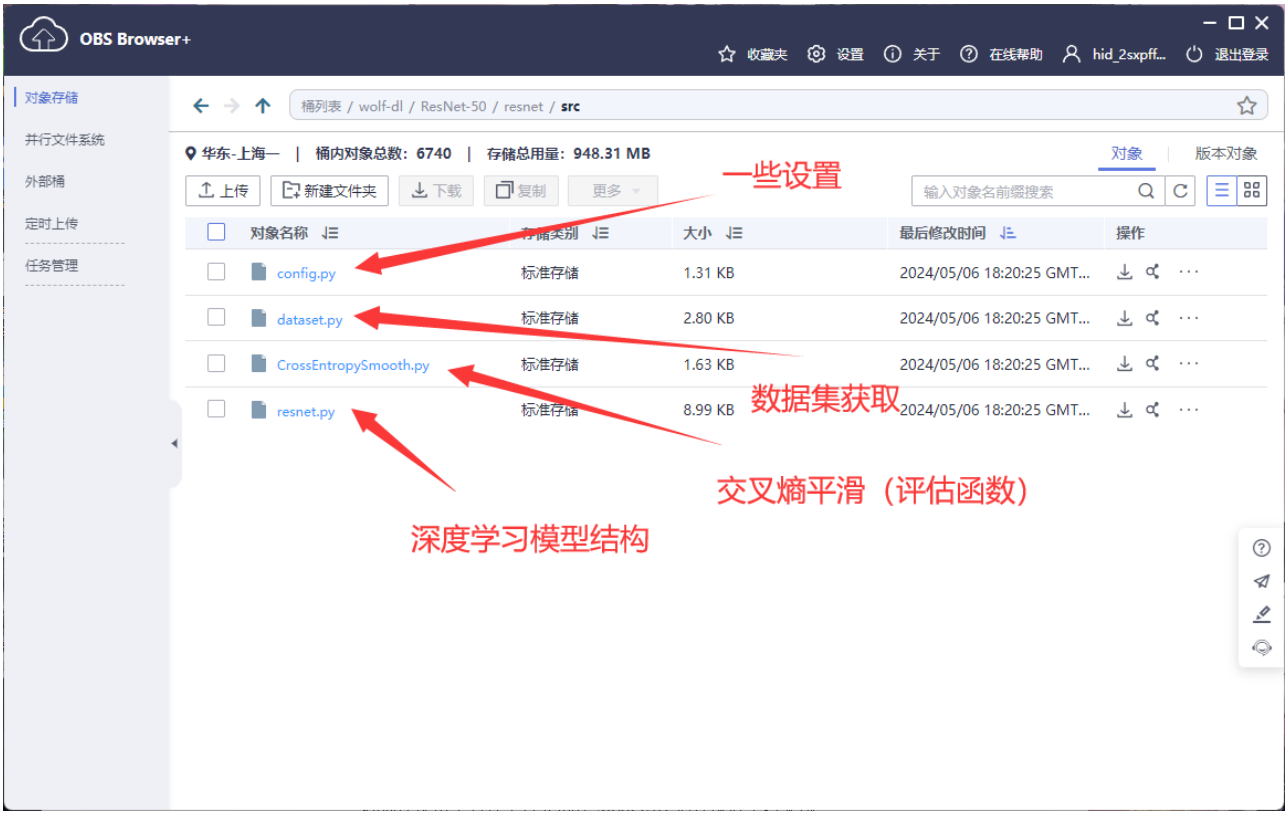


我们在服务器上“创建算法”流程会指定一些参数。开始训练之后，服务器会按照我们的设置来调用这里的文件进行训练。

这是代码文件夹中的资源，train、eval、predict分别是训练、验证、测试。对应流程的三个阶段。训练时服务器会运行train文件，之后会运行eval文件对训练结果做一个测试。做预测的时候调用predict。这三个py文件的作用是按照我们任务的需求组织并调用src中模型文件的代码。



src文件夹中主要保存深度学习模型的代码。



3.4 【代码解析】train/eval/predict.py

这三个代码有重合的部分，也有独立的部分，依据功能而定。

3.4.1 主要函数

train：

- `resnet50_train`: 使用MindSpore训练ResNet50模型。
- `get_lr`: 生成学习率数组。
- `PerformanceCallback`: 训练性能回调，用于显示训练性能指标。
- `data_preprocess`: 数据预处理函数，用于对图像进行预处理。
- `resnet50_train`: ResNet50模型的训练函数，包括数据准备、模型创建、损失函数定义、优化器定义、模型训练等过程。

eval：

- `resnet50_eval`: 使用MindSpore进行ResNet50模型的评估。
- `resnet50_eval`: ResNet50模型的评估函数，包括数据准备、模型加载、损失函数定义、模型评估等过程。

predict：

- `resnet50_predict`: 使用MindSpore进行ResNet50模型的预测。
- `_crop_center`: 图像裁剪函数，用于从图像中心裁剪指定大小的图像。
- `_normalize`: 图像归一化函数，用于对图像进行归一化处理。
- `data_preprocess`: 数据预处理函数，用于对图像进行预处理。
- `resnet50_predict`: ResNet50模型的预测函数，包括数据准备、模型加载、图像预处理、模型推理等过程。

3.4.2 异同点比较

相同部分：

- 代码开头导入了必要的模块和库。
- 设置了随机种子。
- 使用了 MindSpore 框架。
- 使用了 `moxing` 库来处理数据的下载和上传。

- 定义了 `resnet50` 模型，包括模型结构和数据预处理。
- 设置了图模式（`GRAPH_MODE`）和设备目标（`Ascend`）。
- 使用了预训练的模型参数进行预测和评估。

不同部分：

1. 数据处理部分：
    - 训练代码中使用了 `create_dataset` 函数来创建训练和评估数据集。
    - 预测和评估代码中使用了自定义的数据预处理函数 `data_preprocess` 来处理待预测和评估的图像数据。
2. 模型训练部分：
    - 训练代码中使用了 `Model` 类来定义模型，并且进行了模型训练。
    - 预测和评估代码中并未使用 `Model` 类，而是直接调用模型进行预测和评估。
3. 损失函数部分：
    - 训练和评估代码中使用了不同的损失函数。训练代码中使用了 `SoftmaxCrossEntropyWithLogits`，而评估代码中使用了 `CrossEntropySmooth`。
4. 评估指标部分：
    - 训练代码中未定义评估指标，而是在模型定义时指定了 `metrics` 参数，包括了准确率（`'acc'`）。
    - 评估代码中定义了评估指标，包括了 top-1 和 top-5 准确率。
5. 命令行参数部分：
    - 三段代码都使用了命令行参数来指定数据、训练输出和模型参数的路径，以及设备目标等。

这些代码共同构成了一个完整的模型训练、预测和评估的流程，使用了 MindSpore 框架和相关的工具库来完成。

3.5 【代码解析】ResNet模型

resnet.py定义了ResNet（深度残差网络）的模型结构，主要包括如下

1. 卷积层和批归一化层的定义：
    - `_conv3x3`：3x3卷积层的定义。
    - `_conv1x1`：1x1卷积层的定义。
    - `_conv7x7`：7x7卷积层的定义。

- `_bn`：标准批归一化层的定义。
- `_bn_last`：最后一层的批归一化层的定义。
- `_fc`：全连接层的定义。

2. 残差块（Residual Block）的定义：

- `ResidualBlock`：ResNet V1版本的残差块定义，包括两个3x3卷积层和一个1x1卷积层，以及对输入进行下采样的模块。

3. ResNet模型的定义：

- `ResNet`：ResNet模型的定义，包括卷积层、残差块、全局平均池化层、Flatten层和全连接层。

4. ResNet模型的导出函数：

- `resnet50`：导出ResNet50模型。【这是我们模型采用的】
- `resnet101`：导出ResNet101模型。

更为具体的解释在前面ResNet原理的部分，这段代码主要是在复现论文。

3.6 【代码解析】损失函数（交叉熵）

```
"""define loss function for network"""
import mindspore.nn as nn
from mindspore import Tensor
from mindspore.common import dtype as mstype
from mindspore.nn.loss.loss import _Loss
from mindspore.ops import functional as F
from mindspore.ops import operations as P


class CrossEntropySmooth(_Loss):
    """CrossEntropy"""

    def __init__(self, sparse=True, reduction='mean', smooth_factor=0., num_classes=1000):
        super(CrossEntropySmooth, self).__init__()
        self.onehot = P.OneHot()
        self.sparse = sparse
        self.on_value = Tensor(1.0 - smooth_factor, mstype.float32)
        self.off_value = Tensor(1.0 * smooth_factor / (num_classes - 1), mstype.float32)
```

```
        self.ce = nn.SoftmaxCrossEntropyWithLogits(reduction=reduction)

    def construct(self, logit, label):
        if self.sparse:
            label = self.onehot(label, F.shape(logit)[1], self.on_value, self.off_value)
        loss = self.ce(logit, label)
        return loss

```

主要就是计算交叉熵，并做一个平滑处理

1. 使用 `SoftmaxCrossEntropyWithLogits` 函数计算交叉熵损失。
2. 如果 `sparse=True`，则将标签进行 one-hot 编码，并对编码后的标签进行平滑处理。
3. 平滑处理使用的参数 `smooth_factor` 用于控制平滑的程度。
4. 支持设置损失函数的计算模式，如 `reduction='mean'` 表示计算损失的平均值。

平滑处理是通过修改交叉熵损失函数的标签表示来实现的。标准的交叉熵损失函数使用的是一个独热编码的标签，其中正确类别的位置上的值为1，其余位置上的值为0。而在平滑处理中，我们将正确类别的位置上的值设为一个较大的值（通常是 1 - smooth_factor），并将其他位置上的值设置为较小的值（通常是 smooth_factor / (num_classes - 1)）。

这样做的目的是使得模型在训练时对于正确类别的预测更加自信，同时对于其他类别的预测也有一定的概率。这样可以提高模型的泛化能力，防止过拟合。

具体来说，平滑处理的过程如下：

1. 将原始的整数类型标签进行 one-hot 编码。
2. 将正确类别的位置上的值设为 `1 - smooth_factor`，将其他位置上的值设为 `smooth_factor / (num_classes - 1)`。

3.7 【代码解析】dataset代码

这段代码用于创建训练集或验证集的数据集对象。

1. `create_dataset` 函数用于创建数据集对象，接收以下参数：

- `dataset_path`：数据集路径。
- `do_train`：布尔值，指示数据集是否用于训练。
- `repeat_num`：数据集重复次数，默认为 1。
- `batch_size`：批处理大小，默认为 32。

2. `create_dataset` 函数首先根据当前的环境获取设备数量和设备编号。
3. 接着根据设备数量和设备编号，使用 `de.ImageFolderDataset` 创建数据集对象。如果只有一个设备，数据集将被简单地划分为多个 batch，并且将设置 `shuffle=True`。如果有多个设备，数据集将被划分为多个部分，并且每个设备负责处理一个部分，同时保持数据集的 shuffle 状态。
4. 定义了图像的预处理操作。如果 `do_train` 为 `True`，表示数据集用于训练，采用了以下预处理操作：

    - `C.RandomCropDecodeResize`：随机裁剪和调整图像大小。
    - `C.RandomHorizontalFlip`：以一定的概率水平翻转图像。
    - `C.Normalize`：对图像进行归一化。
    - `C.HWC2CHW`：将图像的通道维度从 HWC 转换为 CHW。

    如果 `do_train` 为 `False`，表示数据集用于验证，采用了以下预处理操作：

    - `C.Decode`：解码图像。
    - `C.Resize`：调整图像大小。
    - `C.CenterCrop`：中心裁剪图像。
    - `C.Normalize`：对图像进行归一化。
    - `C.HWC2CHW`：将图像的通道维度从 HWC 转换为 CHW。
5. 对标签进行类型转换操作，将标签的数据类型转换为 `int32`。
6. 对数据集应用 batch 操作，并且使用 `drop_remainder=True` 丢弃最后一个不完整的 batch。
7. 对数据集应用 `repeat` 操作，指定数据集重复次数为 `repeat_num`。

最后返回数据集对象 `ds`。

3.8 【代码解析】config.py

解释如下：

```python
from easydict import EasyDict as ed

# config for resnet50, imagenet2012
```

```
cfg = ed({
    "class_num": 9,                     # 类别数量
    "batch_size": 32,                   # 批处理大小
    "loss_scale": 1024,                 # 损失放缩比例
    "momentum": 0.9,                    # 动量
    "weight_decay": 1e-4,               # 权重衰减
    "epoch_size": 90,                   # 训练的总轮数
    "pretrain_epoch_size": 0,           # 预训练的轮数
    "save_checkpoint": True,            # 是否保存检查点
    "save_checkpoint_epochs": 5,        # 多少个 epoch 保存一次检查点
    "keep_checkpoint_max": 10,          # 最多保存的检查点数量
    "save_checkpoint_path": "./",       # 检查点保存路径
    "warmup_epochs": 0,                 # warmup 轮数
    "lr_decay_mode": "linear",          # 学习率衰减模式
    "use_label_smooth": True,           # 是否使用标签平滑处理
    "label_smooth_factor": 0.1,         # 标签平滑处理因子
    "lr_init": 0,                       # 初始学习率
    "lr_max": 0.8,                      # 最大学习率
    "lr_end": 0.0                       # 结束学习率
})

```

四、实验操作步骤

选用一个实验平台，采用开源深度学习工具求解实际人工智能应用问题，例如计算机视觉、图像识别、文字识别、自然语言处理、无人驾驶、语音识别等。

实验平台推荐：华为云平台（推荐使用昇腾910、ModelArts、OCR识别、图像识别等完成综合实验） https://www.huaweicloud.com/ 【记得问助教要代金券】

我选择使用华为云平台完成在 ModelArt 运行深度学习案例

★完整的实验代码，包括我训练得到的checkpoint，放在github上，链接：

https://github.com/wolfvoid/ResNet-mushroom-classifier

【直接下载使用我的chenkpoint可以绕过训练过程】

4.0 环境准备（OBS存储）

4.0.1 下载数据到本地

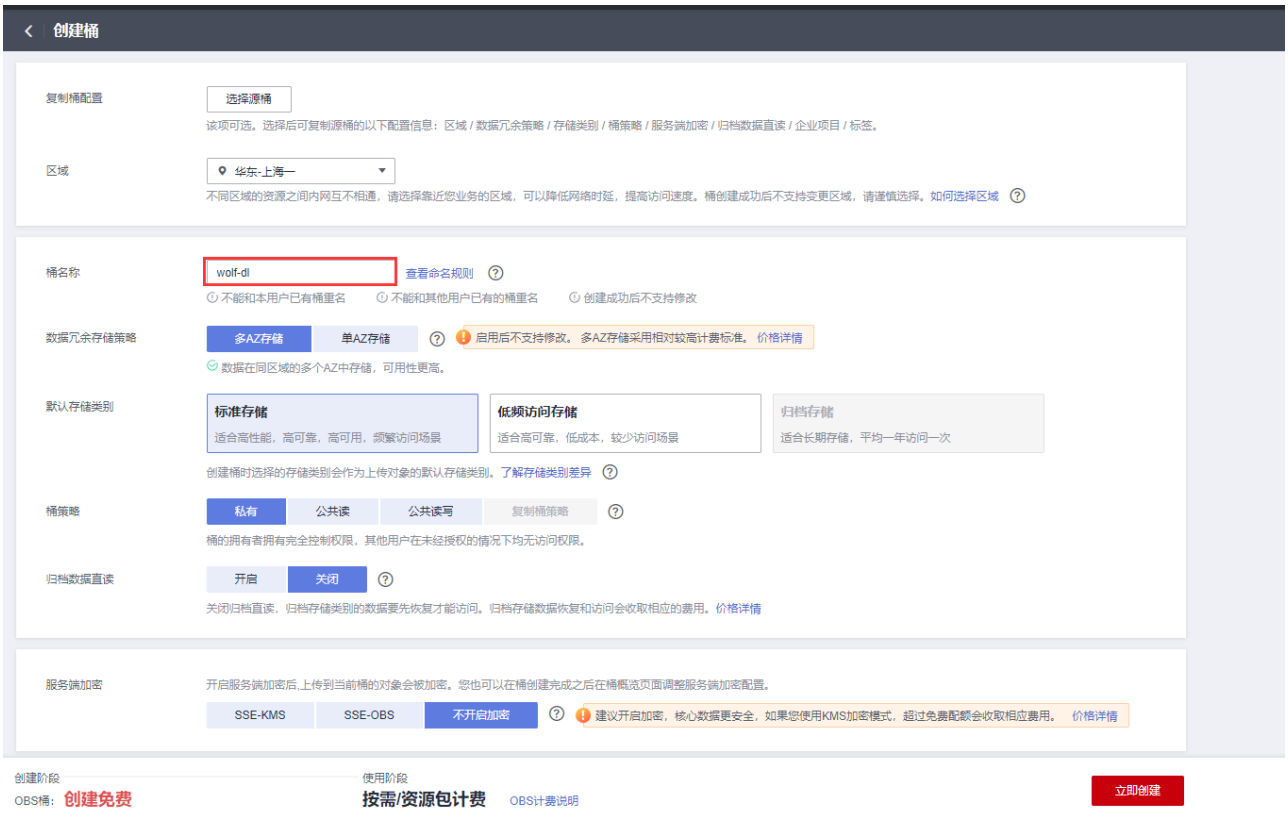
提供的参考的代码

由于提供的github链接上路径发生变更，按照github上“进阶作业文档”所示继续操作。

文档

4.0.2 在华为云创建桶

桶就相当于在服务器上分配的一块硬盘区域，供我们存储数据。



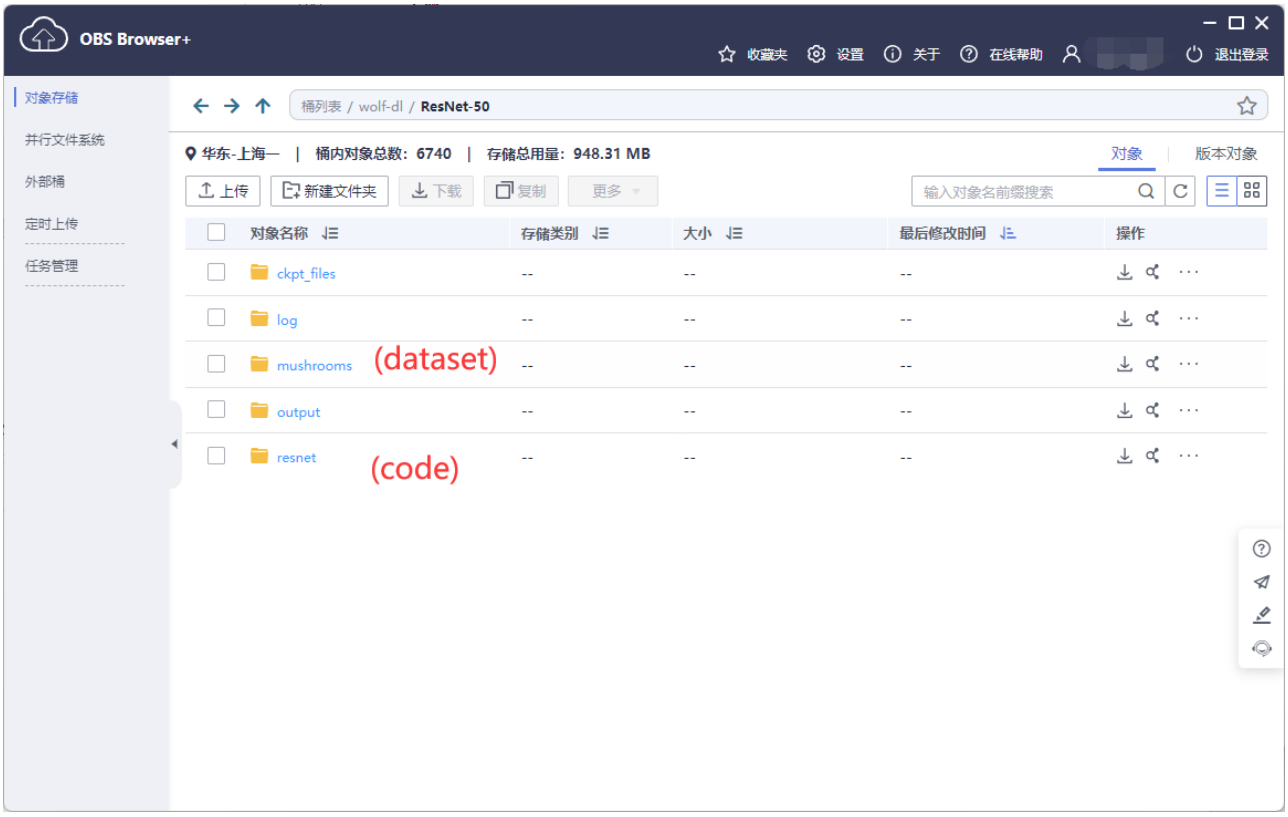
由于使用华为云服务器环境，我们需要上传的资源数较大，故需下载OBS Browser

https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsbrowserplus/win64/OBSBrowserPlus-HEC-win64.zip

4.0.3 上传数据到华为云

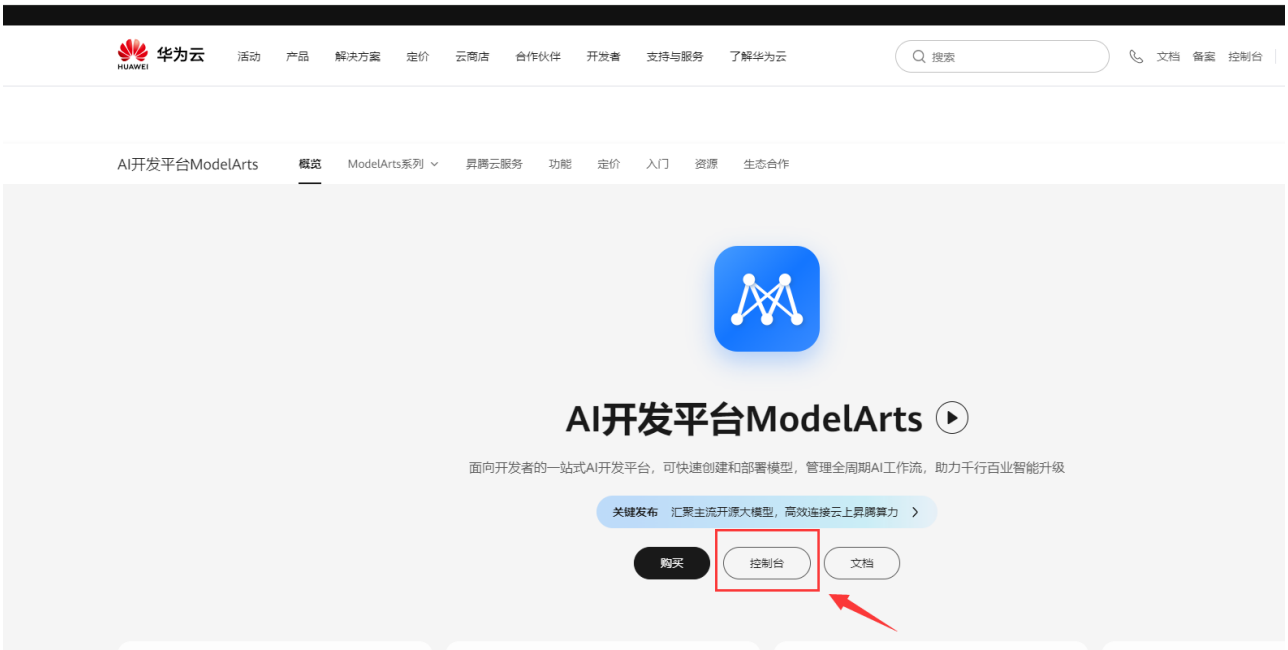
使用OBS Browser（需要生成并使用AccessKey登录），上传数据。

【请时刻注意自己的服务器是哪个】

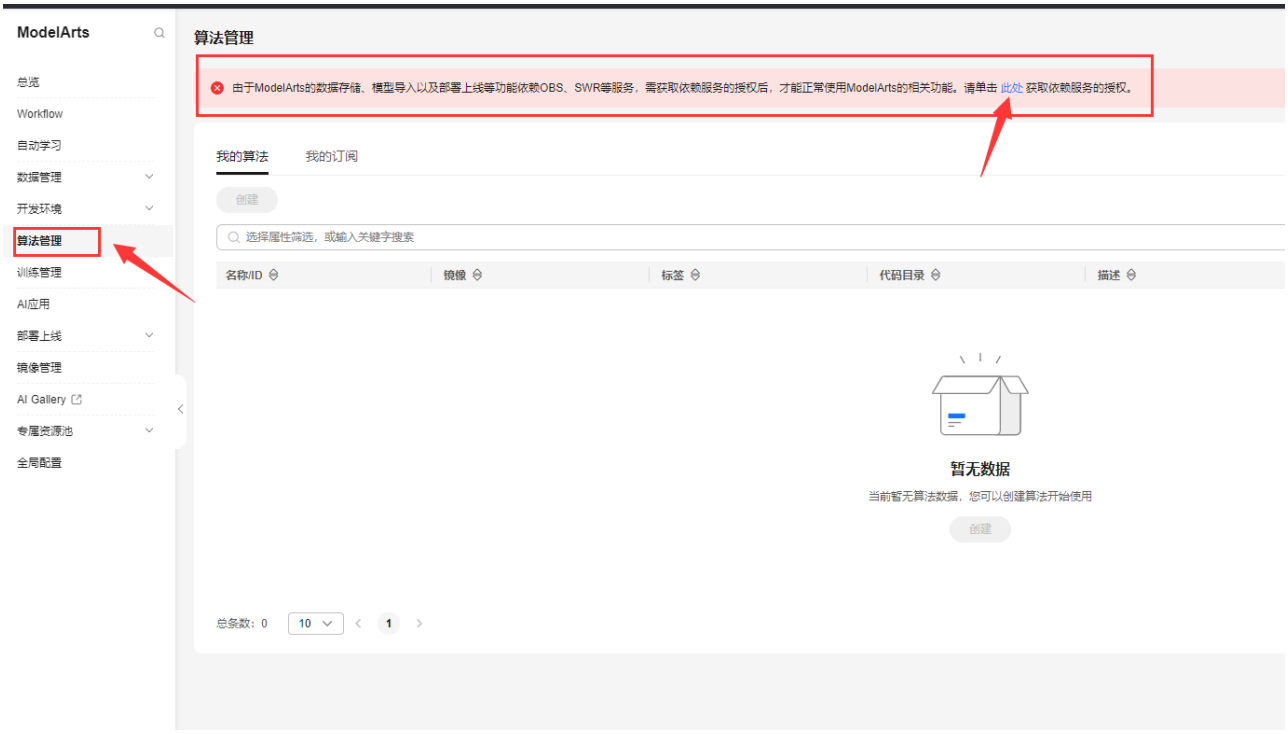


4.0.4 授予操作权限

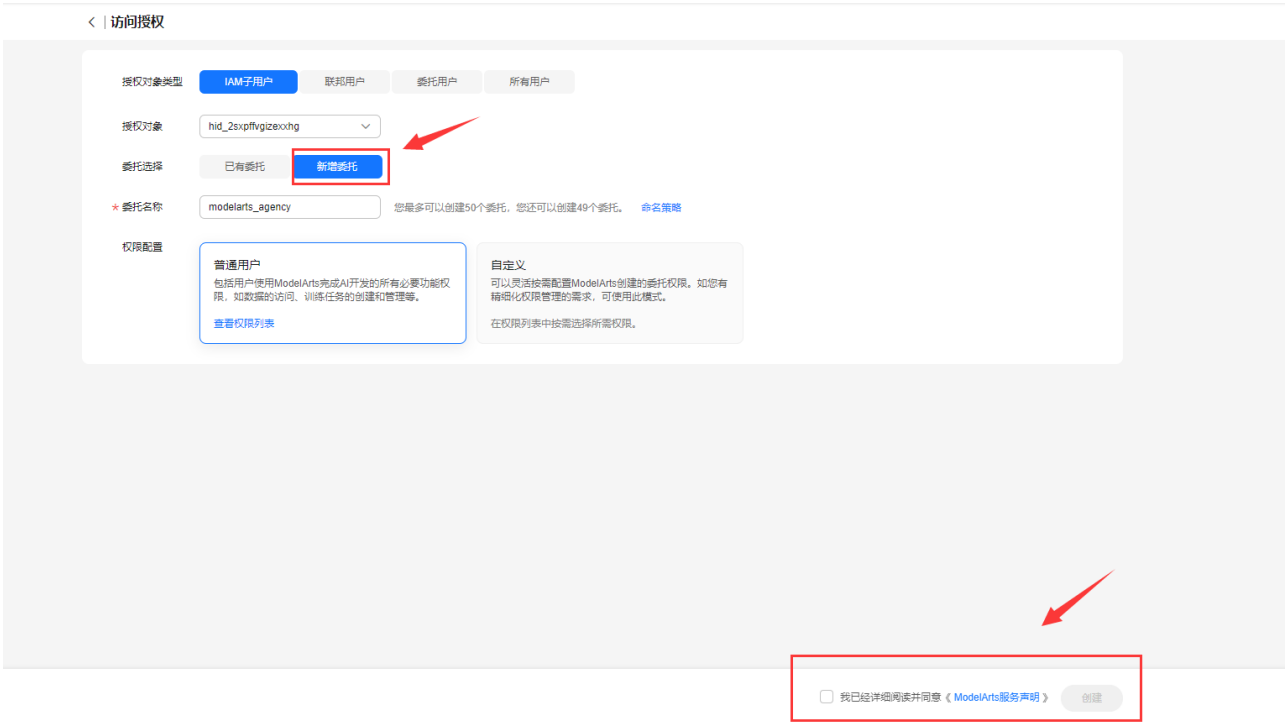
搜索ModelArts，并进入控制台



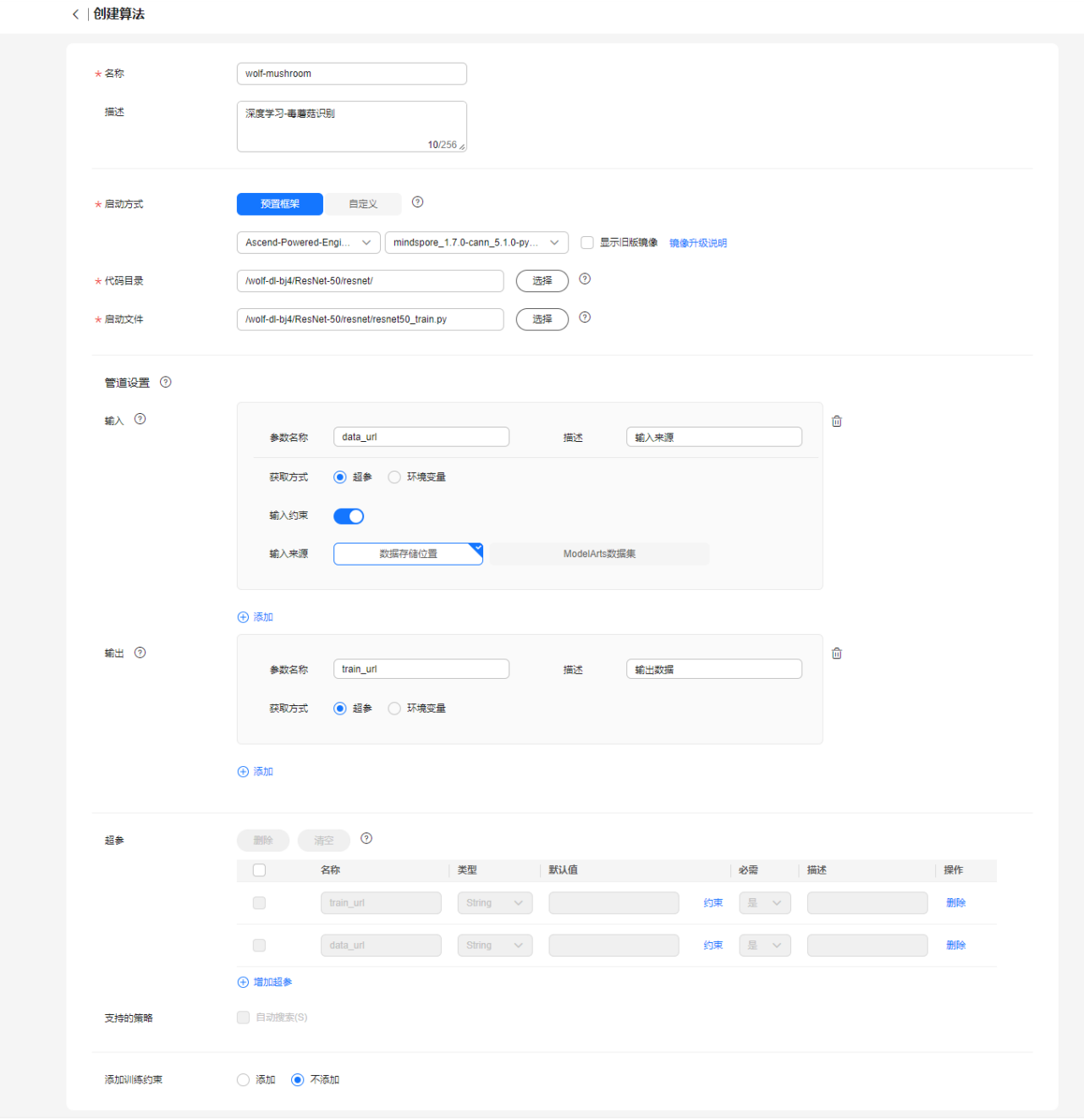
进入“算法管理”，发现这里需要我们获得ModelArts模块对于OBS模块的访问权限。



按照提示进行操作，授予权限。



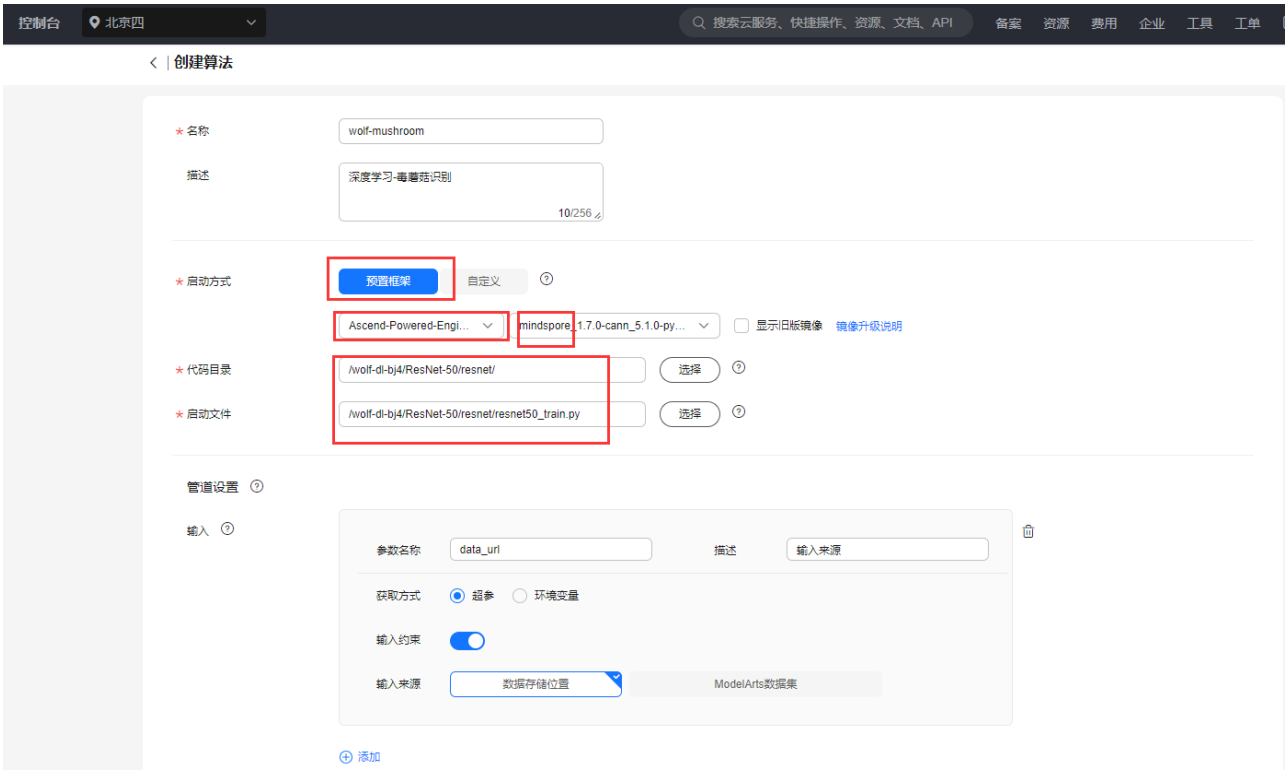
4.1 创建算法



提示：这里的路径一定要选择正确。

启动方式选择“预置框架”，选择Ascend这项，然后后面版本选择相应的版本。

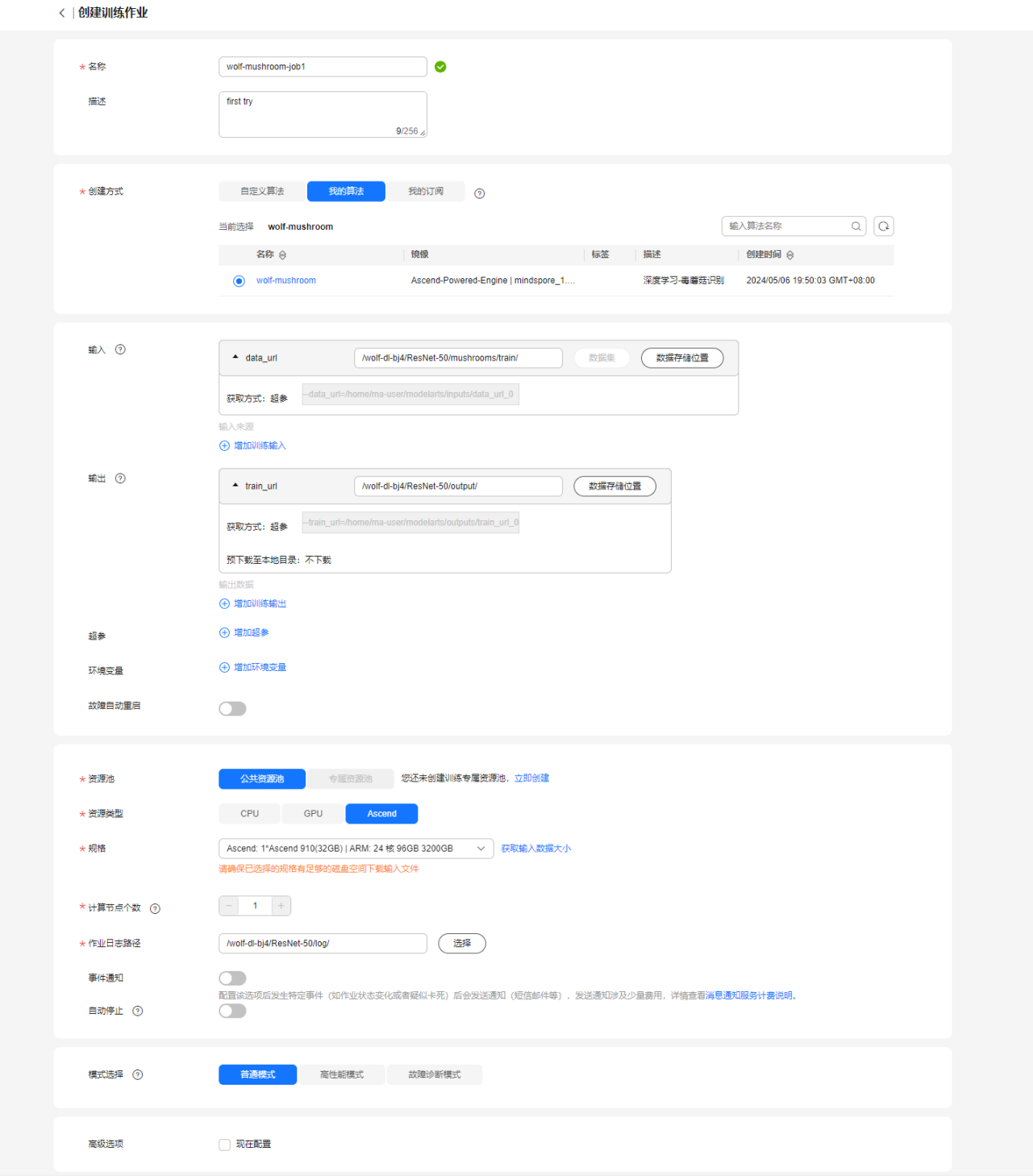
如果没有找到Aecend框架，可以换一下服务器，如我这里使用的是“北京四”，之前使用“上海一”就没有这个Ascend框架，后来更换的这个，但是请注意，若“北京四”服务器的ModelArts只能使用同样在“北京四”服务器上的OBS，即上传存储的服务器必须与运行代码的服务器保持一致。



4.2 训练

【注意】训练过程可以跳过，使用预训练的checkpoint结果即可。我是走了一遍训练过程。

4.2.1 创建训练作业



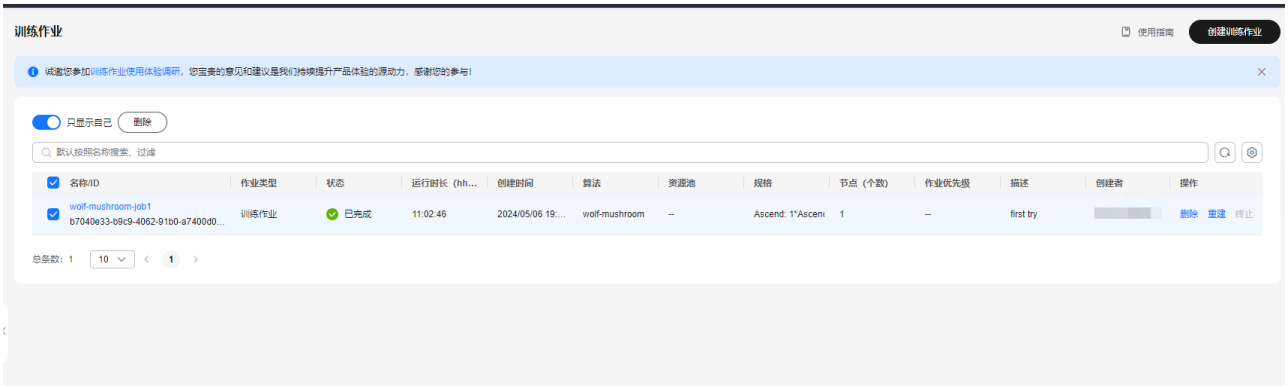
【报错】ModelArts.2840: 该规格不兼容选用的引擎或计算策略：下面类型得选择Ascend，不要选择GPU或者CPU，否则会报这个错误。

4.2.2 等待训练完成

我等待了半个小时才进入训练任务，并且任务训练的时间很长，我一直在思考要不要用预训练模型直接替代（我的代金券呜呜呜），但最后还是让它跑完了，真的好贵。11个小时的训练花了200多块钱（代金券扣掉了）。

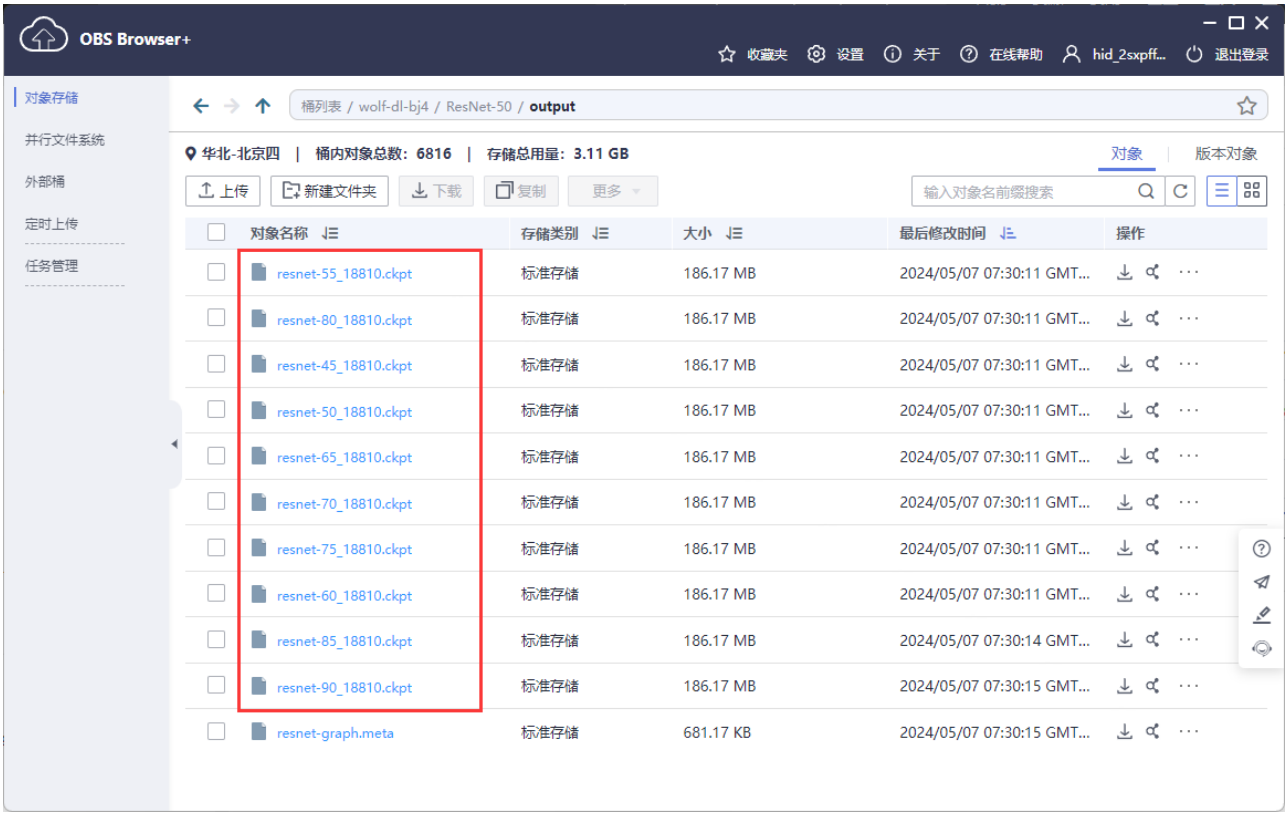
【提示】如何查看自己的模型已经训练到什么程度了：log是实时更新的，若不更改初始参数，应该是90个epoch，所以只要看log显示现在是第几个epoch，再稍加计算即可知道自己现在在哪里。

训练完的结果是这样：



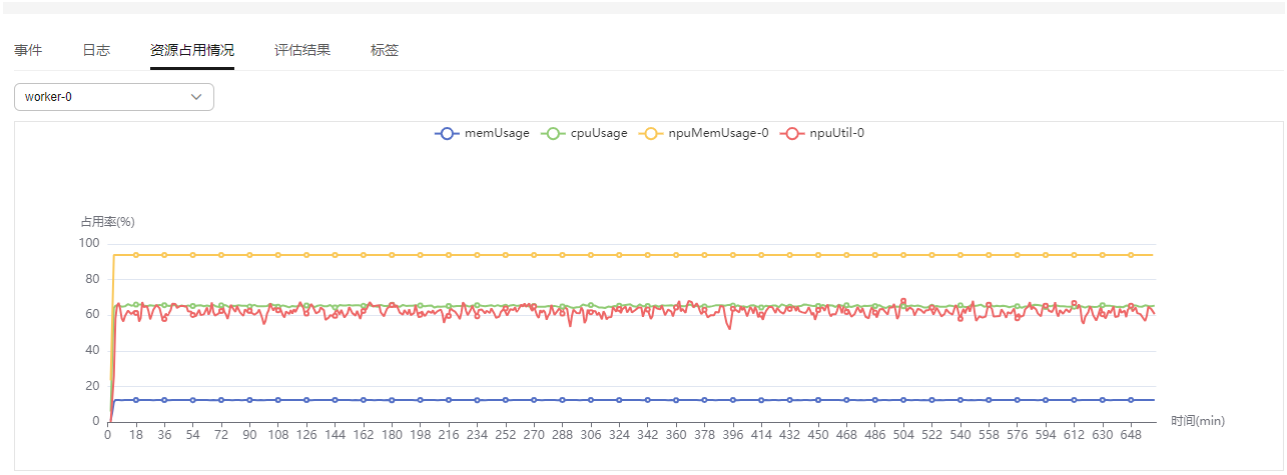
这些checkpoint文件保存了训练指定轮数后的模型参数(权重)。（这里的任意一个cpkt都是可以用来使用的一套权重数据，只不过轮数越高一般情况下效果是越好，因此它们的大小都是相同的）

【操作】我们需要将这些权重文件拷贝到上级目录的ckpt_files文件夹下。



4.2.3 资源占用情况监测

这里还可以看到训练的资源占用情况



4.2.4 关注日志文件

根据我们的设置，日志文件生成在log文件夹下

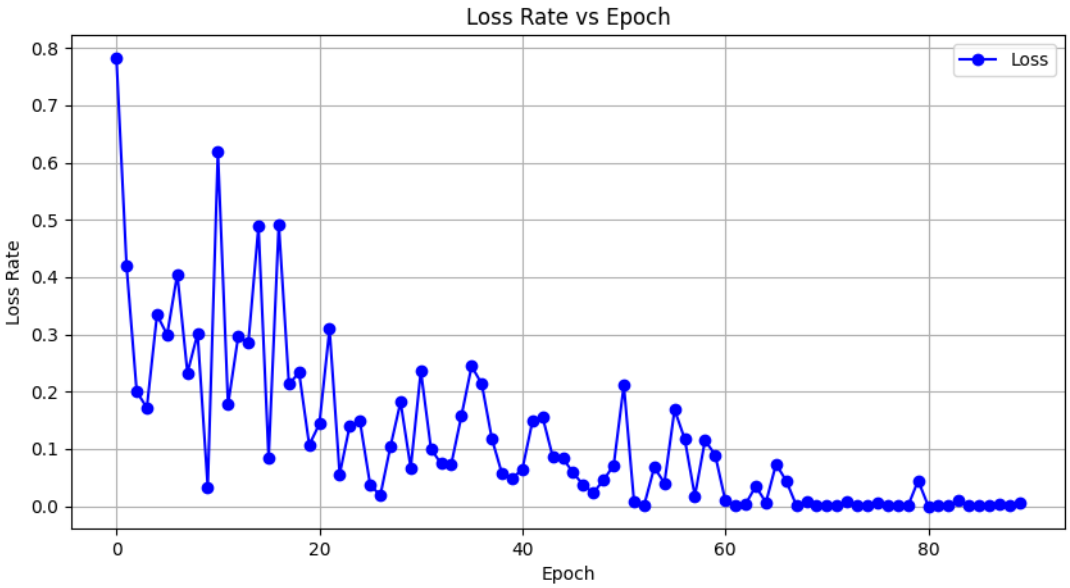
```
Start run training, total epoch: 90.

```

```
epoch 1 cost time = 508.7149701118469, train step num: 18810, one step time: 27.044921324393776 ms, train samples per second of cluster: 1183.2

epoch: 1 step: 18810, loss is 0.7837822437286377
epoch time: 515333.027 ms, per step time: 27.397 ms

epoch 2 cost time = 441.9296247959137, train step num: 18810, one step time: 23.494397915784887 ms, train samples per second of cluster: 1362.0

epoch: 2 step: 18810, loss is 0.41933563351631165
epoch time: 441933.526 ms, per step time: 23.495 ms

.....
中间省略
.....

epoch 89 cost time = 435.6134955883026, train step num: 18810, one step time: 23.158612205651387 ms, train samples per second of cluster: 1381.8

epoch: 89 step: 18810, loss is 0.00016484444495290518
epoch time: 435617.344 ms, per step time: 23.159 ms

epoch 90 cost time = 440.81839871406555, train step num: 18810, one step time: 23.435321569062495 ms, train samples per second of cluster: 1365.5

epoch: 90 step: 18810, loss is 0.004747382830828428
epoch time: 442179.918 ms, per step time: 23.508 ms
Upload checkpoint.
ResNet50 training success
```

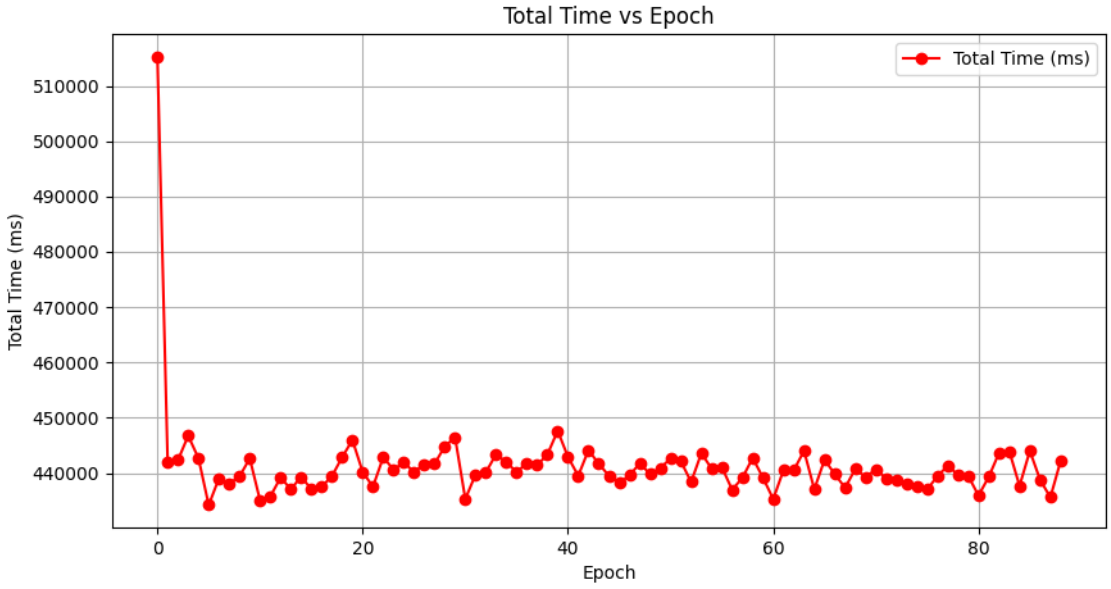
在完成之后，我们可以分析日志文件得到训练的损失率变化与收敛趋势。

我根据这个log的特点写了一个python程序，可以获取loss rate和epoch time，并绘制表格。

这是损失率迭代曲线（可见最后应该是收敛了）



这是训练时间迭代曲线



这是处理该log的python程序（正则逻辑只适用于该log）

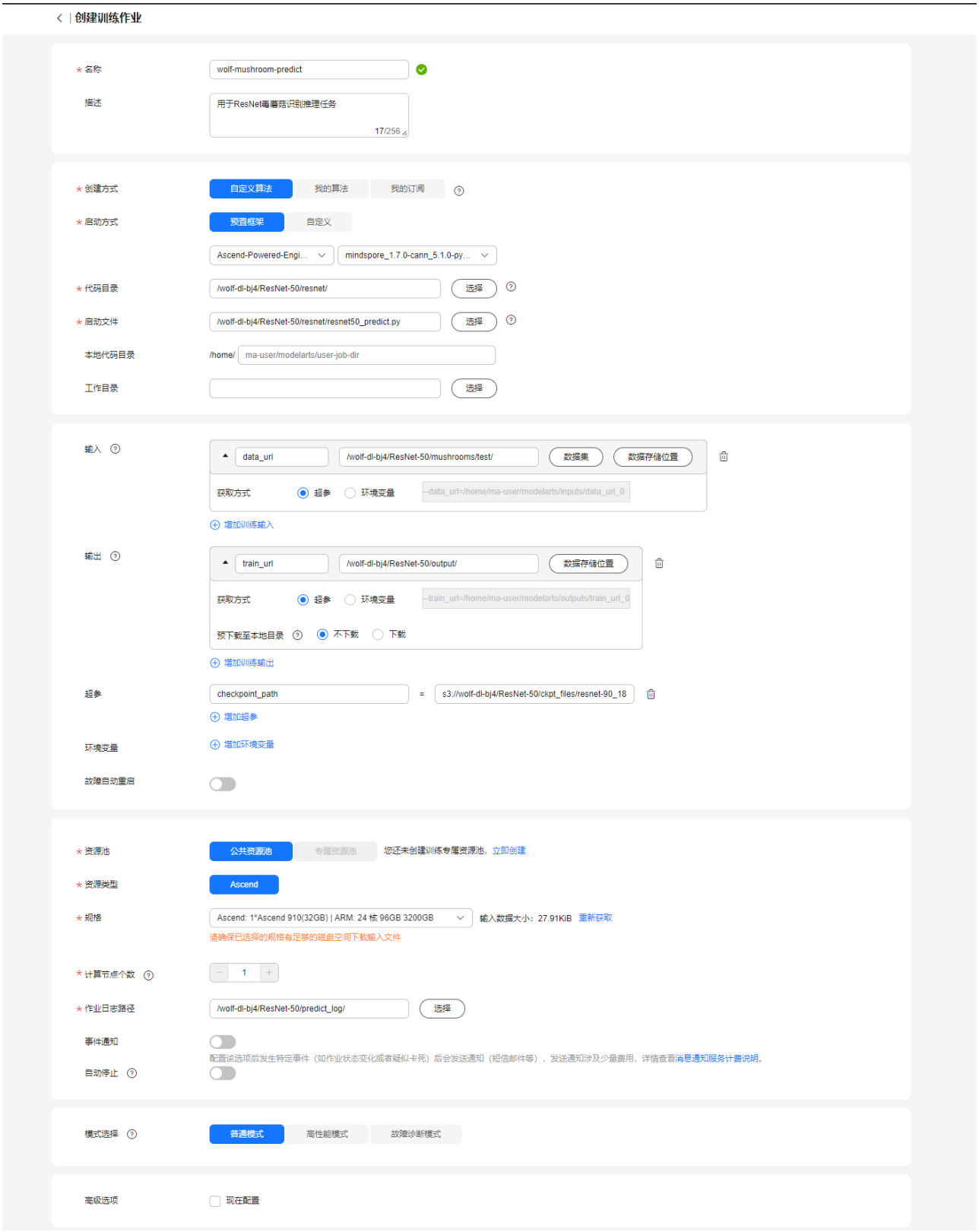
```python
import re
import matplotlib.pyplot as plt

# 读取 log 文件
with open("log.txt", "r") as file:
    lines = file.readlines()

# 初始化列表来存储损失率和总耗时
losses = []
times = []
```

```python

# 匹配损失率和总耗时的正则表达式
loss_pattern = re.compile(r"loss is (\d+\.\d+(?:e[-+]?\d+)?)")
time_pattern = re.compile(r"epoch time: (\d+\.\d+)")

# 提取损失率和总耗时
for line in lines:
    loss_match = loss_pattern.search(line)
    time_match = time_pattern.search(line)
    if loss_match:
        losses.append(float(loss_match.group(1)))
    if time_match:
        times.append(float(time_match.group(1)))

print(losses)
print(times)

# 绘制损失率随 epoch 变化图像
plt.figure(figsize=(10, 5))
plt.plot(losses, marker='o', color='b', label='Loss')
plt.title('Loss Rate vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss Rate')
plt.grid(True)
plt.legend()
plt.show()

# 绘制总耗时随 epoch 变化图像
plt.figure(figsize=(10, 5))
plt.plot(times, marker='o', color='r', label='Total Time (ms)')
plt.title('Total Time vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Total Time (ms)')
plt.grid(True)
plt.legend()
plt.show()

```

4.3 推理

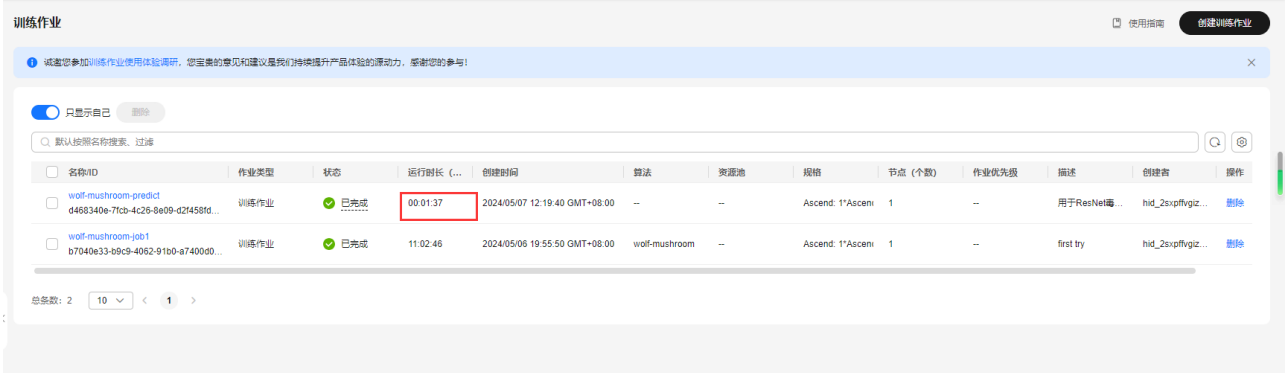
4.3.1 创建推理作业

步骤与之前类似，推理主要是根据权重和待预测数据来预测结果。所以应该比较快的，设置如下：



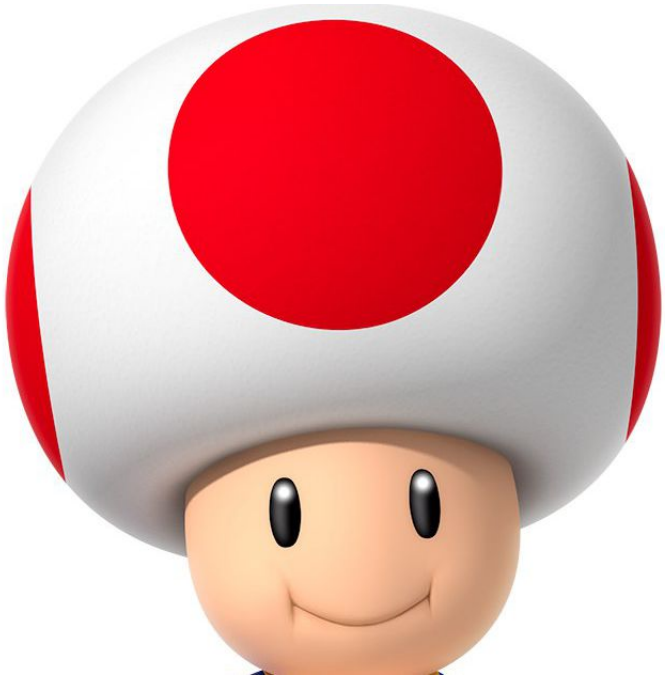
4.3.2 等待推理完成

本身推理任务时间是很短的，但是进入任务的等待时间太长了。如下，推理只花了1分钟左右，与训练的11小时形成鲜明的对比。



4.4 推理结果分析

我要预测的蘑菇是这个（卡通形象可爱捏）



推理完成后，进入log，查看推理结果

```
预测的蘑菇标签为:
    Hygrocybe浅黄褐湿伞,伞菌目,蜡伞科,湿伞属,分布于香港(见于松仔园),有毒

ResNet50 prediction success!
```

与我们常识相符，这是一个毒蘑菇。

六、思考题

> 深度算法参数的设置对算法性能的影响？

主要针对ResNet中出现的参数进行分析

1. **Residual Block**中的通道数设置：
    - 参数：`in_channel`, `out_channel`。
    - 影响：通道数的设置直接影响了模型的表示能力和复杂度。如果通道数过小，模型可能无法学习到足够复杂的特征，导致欠拟合；如果通道数过大，模型可能会过拟合。
2. 网络的层数和宽度：
    - 参数：`layer_nums`, `in_channels`, `out_channels`。
    - 影响：层数和宽度是衡量模型复杂度的重要指标。增加层数和宽度可以增加模型的表示能力，但也会增加训练和推理的计算成本。
3. 学习率和学习率衰减策略：
    - 参数：`lr_init`, `lr_max`, `lr_end`, `lr_decay_mode`。
    - 影响：学习率决定了模型参数更新的步长和速度。学习率太大会导致模型无法收敛，学习率太小会导致收敛速度过慢。学习率衰减策略影响着模型在训练过程中学习率的变化规律，合适的学习率衰减策略能够加快模型的收敛速度，提高模型性能。
4. 优化器和参数初始化：
    - 参数：优化器的选择和参数初始化方法。
    - 影响：优化器和参数初始化方法对模型的训练过程和性能有很大影响。合适的优化器和参数初始化方法能够加速模型的收敛，提高模型的性能。
5. 标签平滑：
    - 参数：`use_label_smooth`, `label_smooth_factor`。
    - 影响：标签平滑是一种正则化方法，用于减少模型对训练数据的过拟合。标签平滑能够提高模型的泛化能力，降低模型的波动性，提高模型的性能。
6. 数据增强策略：
    - 影响：数据增强是提高模型泛化能力的关键手段之一。合适的数据增强策略能够增加训练数据的多样性，提高模型的泛化能力。

七、 实验总结

在本次实验中，我选择了使用 ResNet（深度残差网络）模型来进行毒蘑菇的分类任务。ResNet 最早于 2015 年在《Deep Residual Learning for Image Recognition》论文中提出，其核心思想是通过跳跃连接将输入直接传递到输出，从而允许网络学习残差（residual）映射，而不是直接学习原始映射。这种残差学习的方式使得网络能够更容易地学习到恒等映射（identity mapping），从而解决了深度网络训练中的梯度消失和梯度爆炸问题。

通过对 ResNet 的研究，我学到了以下几点内容：

1. **ResNet** 模型原理：
    - 了解了 ResNet 的基本原理，包括跳跃连接和残差学习的概念，以及如何通过残差学习解决深度网络训练中的梯度消失和梯度爆炸问题。
2. **ResNet** 模型结构：
    - 深入研究了 ResNet 的模型结构，包括残差块（Residual Block）的设计和堆叠方式，以及全局平均池化层和全连接层的作用。
3. **ResNet** 在深度学习任务中的应用：
    - 了解了 ResNet 在图像分类等深度学习任务中的应用情况，并学会了如何使用 ResNet 模型来解决实际问题。
4. **ResNet** 在华为云平台上的应用：
    - 通过在华为云平台上运行 ResNet 模型进行毒蘑菇分类任务的实验，掌握了如何在云平台上配置并训练深度学习模型的技能。

通过对 ResNet 的研究和实践，我不仅提升了对深度学习模型原理和结构的理解，还学会了如何在云平台上应用和部署深度学习模型，为今后在深度学习领域的研究和实践打下了坚实的基础。

参考文献

Resnet的学习部分有参考

- https://blog.csdn.net/lsb2002/article/details/130748991
- https://www.cnblogs.com/shine-lee/p/12363488.html
- 《Deep Residual Learning for Image Recognition》