

《机器学习基础》实验报告

年级、专业、班级	2021 级计算机科学与技术 04 班	姓名	胡鑫
实验题目	基于 MindSpore 的图像识别全流程		
实验时间	2023 年 5 月 25 日	实验地点	DS3402
实验成绩	优秀/良好/中等	实验性质	<input type="checkbox"/> 验证性 <input type="checkbox"/> 设计性 <input checked="" type="checkbox"/> 综合性
教师评价： <input checked="" type="checkbox"/> 算法/实验过程正确； <input checked="" type="checkbox"/> 源程序/实验内容提交； <input checked="" type="checkbox"/> 程序结构/实验步骤合理； <input checked="" type="checkbox"/> 实验结果正确； <input checked="" type="checkbox"/> 语法、语义正确； <input checked="" type="checkbox"/> 报告规范； 其他： <div>评价教师: 何静媛</div>			
实验目的 掌握图像识别相关知识原理。			

报告完成时间: 2023 年 6 月 6 日

1 实验内容

1. 图像分类在我们的日常生活中广泛使用,比如拍照识物,还有手机的 AI 拍照,在学术界,每年也有很多图像分类的比赛,本实验要求利用华为云提供的 flower_photos 开源数据集来帮助大家学习如何构建自己的图像识别模型。
2. 要求基于 MindSpore 来构建图像识别模型,然后将模型部署 ModelArts 上提供在线识别服务,完成对拓展实验 1 测试图片的三种花卉图像的测试。

2 实验过程或算法(源程序)

2.1 导入数据集压缩包到相同路径下,并解压 flower_photos.zip 文件

```
import zipfile
import os

def un_zip(file_name, dst):
    """解压 zip 文件"""
    zip_file = zipfile.ZipFile(file_name)
    if os.path.isdir(dst):
        pass
    else:
        os.mkdir(dst)
    for names in zip_file.namelist():
        zip_file.extract(names, dst)
    zip_file.close()

if __name__ == '__main__':
    file_name = r"flower_photos.zip"
    dst = r"flower"
    un_zip(file_name, dst)
```

2.2 实验环境导入

```
# 隐藏警告
import warnings
warnings.filterwarnings('ignore')
#easydict模块用于以属性的方式访问字典的值
from easydict import EasyDict as edict
#os模块主要用于处理文件和目录
import os
import numpy as np
```

```

import matplotlib.pyplot as plt
import mindspore
#导入mindspore框架数据集
import mindspore.dataset as ds
#vision.c_transforms模块是处理图像增强的高性能模块，用于数据增强图像数据改进训练模型。
from mindspore.dataset.vision import c_transforms as vision
from mindspore import context
import mindspore.nn as nn
from mindspore.train import Model
from mindspore.nn.optim.momentum import Momentum
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor
from mindspore import Tensor
from mindspore.train.serialization import export
from mindspore.train.loss_scale_manager import FixedLossScaleManager
from mindspore.train.serialization import load_checkpoint, load_param_into_net
import mindspore.ops as ops
# 设置MindSpore的执行模式和设备
context.set_context(mode=context.GRAPH_MODE, device_target="GPU")

```

2.3 定义模型的相关变量

```

cfg = edict({
    'data_path': 'output/train', #训练数据集
    'test_path': 'output/val', #测试数据集
    'data_size': 3616,
    'HEIGHT': 224, # 图片高度
    'WIDTH': 224, # 图片宽度
    '_R_MEAN': 123.68,
    '_G_MEAN': 116.78,
    '_B_MEAN': 103.94,
    '_R_STD': 1,
    '_G_STD': 1,
    '_B_STD': 1,
    '_RESIZE_SIDE_MIN': 256,
    '_RESIZE_SIDE_MAX': 512,
    'batch_size': 32,
    'num_class': 5, # 分类类别
    'epoch_size': 150, # 训练次数
    'loss_scale_num': 1024,
    'prefix': 'resnet-ai',
    'directory': './model_resnet',
    'save_checkpoint_steps': 10,
})

```

2.4 数据集读取及预处理

数据处理

```
def read_data(path,config,usage="train"):
    #从目录中读取图像的源数据集。
    dataset = ds.ImageFolderDataset(path,class_indexing
                                    {'daisy':0, 'dandelion':1, 'roses':2, 'sunflowers':3, 'tulips':4})
    # define map operations
    decode_op = vision.Decode()
    normalize_op = vision.Normalize(mean=[cfg._R_MEAN, cfg._G_MEAN,cfg._B_MEAN],
                                    std=[cfg._R_STD, cfg._G_STD, cfg._B_STD])
    resize_op = vision.Resize(cfg._RESIZE_SIDE_MIN)
    center_crop_op = vision.CenterCrop((cfg.HEIGHT, cfg.WIDTH))
    horizontal_flip_op = vision.RandomHorizontalFlip()
    channelswap_op = vision.HWC2CHW()
    random_crop_decode_resize_op = vision.RandomCropDecodeResize((cfg.HEIGHT,cfg.WIDTH),
                                                                    (0.5, 1.0), (1.0, 1.0), max_attempts=100)
    if usage == 'train':
        dataset =
            dataset.map(input_columns="image",operations=random_crop_decode_resize_op)
        dataset = dataset.map(input_columns="image",operations=horizontal_flip_op)
    else:
        dataset = dataset.map(input_columns="image", operations=decode_op)
        dataset = dataset.map(input_columns="image", operations=resize_op)
        dataset = dataset.map(input_columns="image", operations=center_crop_op)

    dataset = dataset.map(input_columns="image", operations=normalize_op)
    dataset = dataset.map(input_columns="image", operations=channelswap_op)

    if usage == 'train':
        dataset = dataset.shuffle(buffer_size=10000) # 10000 as in imageNet train script
        dataset = dataset.batch(cfg.batch_size, drop_remainder=True)
    else:
        dataset = dataset.batch(1, drop_remainder=True)

    dataset = dataset.repeat(1)
    dataset.map_model = 4
    return dataset

de_train = read_data(cfg.data_path,cfg,usage="output/train")
de_test = read_data(cfg.test_path,cfg,usage="output/val")
step_size=de_train.get_dataset_size()
print('训练数据集数量:',de_train.get_dataset_size()*cfg.batch_size)#get_dataset_size()获取批处理的大小。
print('测试数据集数量:',de_test.get_dataset_size())
de_dataset = de_train
```

```

data_next = de_dataset.create_dict_iterator(output_numpy=True).__next__()
print('通道数/图像长/宽:', data_next['image'][0,...].shape)
print('一张图像的标签样式:', data_next['label'][0]) # 一共5类, 用0-4的数字表达类别。
plt.figure()
plt.imshow(data_next['image'][0,0,...])
plt.colorbar()
plt.grid(False)
plt.show()

```

2.5 模型构建训练

#构建网络

#构建残差网络结构

```

from typing import Type, Union, List, Optional
from mindvision.classification.models.blocks import ConvNormActivation
from mindvision.classification.models.classifiers import BaseClassifier
from mindvision.classification.models.head import DenseHead
from mindvision.classification.models.neck import GlobalAvgPooling
from mindvision.classification.utils.model_urls import model_urls
from mindvision.utils.load_pretrained_model import LoadPretrainedModel
from mindspore import nn

```

#构建ResidualBlockBase类实现Building Block结构

```

class ResidualBlockBase(nn.Cell):
    expansion: int = 1 # 最后一个卷积核数量与第一个卷积核数量相等

    def __init__(self, in_channel: int, out_channel: int,
                 stride: int = 1, norm: Optional[nn.Cell] = None,
                 down_sample: Optional[nn.Cell] = None) -> None:
        super(ResidualBlockBase, self).__init__()
        if not norm:
            norm = nn.BatchNorm2d

        self.conv1 = ConvNormActivation(in_channel, out_channel, kernel_size=3, stride=stride,
                                       norm=norm)
        self.conv2 = ConvNormActivation(out_channel, out_channel, kernel_size=3, norm=norm,
                                       activation=None)
        self.relu = nn.ReLU()
        self.down_sample = down_sample

    def construct(self, x):
        """ResidualBlockBase construct."""
        identity = x # shortcuts分支

```

```

out = self.conv1(x) # 主分支第一层：3*3卷积层
out = self.conv2(out) # 主分支第二层：3*3卷积层

if self.down_sample:
    identity = self.down_sample(x)
    out += identity # 输出为主分支与shortcuts之和
    out = self.relu(out)

return out
#构建ResidualBlock类实现Bottleneck结构
class ResidualBlock(nn.Cell):
    expansion = 4 # 最后一个卷积核的数量是第一个卷积核数量的4倍

    def __init__(self, in_channel: int, out_channel: int,
                 stride: int = 1, norm: Optional[nn.Cell] = None,
                 down_sample: Optional[nn.Cell] = None) -> None:
        super(ResidualBlock, self).__init__()
        if not norm:
            norm = nn.BatchNorm2d

        self.conv1 = ConvNormActivation(in_channel, out_channel, kernel_size=1, norm=norm)
        self.conv2 = ConvNormActivation(out_channel, out_channel, kernel_size=3, stride=stride,
                                         norm=norm)
        self.conv3 = ConvNormActivation(out_channel, out_channel * self.expansion,
                                         kernel_size=1, norm=norm, activation=None)
        self.relu = nn.ReLU()
        self.down_sample = down_sample

    def construct(self, x):
        identity = x # shortcuts分支

        out = self.conv1(x) # 主分支第一层：1*1卷积层
        out = self.conv2(out) # 主分支第二层：3*3卷积层
        out = self.conv3(out) # 主分支第三层：1*1卷积层

        if self.down_sample:
            identity = self.down_sample(x)

        out += identity # 输出为主分支与shortcuts之和
        out = self.relu(out)

    return out
#定义make_layer实现残差块的构建
def make_layer(last_out_channel, block: Type[Union[ResidualBlockBase,

```

```

        ResidualBlock]], channel: int, block_nums: int, stride: int = 1):
down_sample = None # shortcuts分支

if stride != 1 or last_out_channel != channel * block.expansion:
    down_sample = ConvNormActivation(last_out_channel, channel *
        block.expansion, kernel_size=1, stride=stride, norm=nn.BatchNorm2d, activation=None)

layers = []
layers.append(block(last_out_channel, channel, stride=stride, down_sample=down_sample,
    norm=nn.BatchNorm2d))

in_channel = channel * block.expansion
# 堆叠残差网络
for _ in range(1, block_nums):
    layers.append(block(in_channel, channel, norm=nn.BatchNorm2d))

return nn.SequentialCell(layers)

#构建ResNet50模型
class ResNet(nn.Cell):
    def __init__(self, block: Type[Union[ResidualBlockBase, ResidualBlock]], layer_nums: List[int],
        norm: Optional[nn.Cell] = None) -> None:
        super(ResNet, self).__init__()
        if not norm:
            norm = nn.BatchNorm2d
        # 第一个卷积层, 输入channel为3 (彩色图像), 输出channel为64
        self.conv1 = ConvNormActivation(3, 64, kernel_size=7, stride=2, norm=norm)
        # 最大池化层, 缩小图片的尺寸
        self.max_pool = nn.MaxPool2d(kernel_size=3, stride=2, pad_mode='same')
        # 各个残差网络结构块定义,
        self.layer1 = make_layer(64, block, 64, layer_nums[0])
        self.layer2 = make_layer(64 * block.expansion, block, 128, layer_nums[1], stride=2)
        self.layer3 = make_layer(128 * block.expansion, block, 256, layer_nums[2], stride=2)
        self.layer4 = make_layer(256 * block.expansion, block, 512, layer_nums[3], stride=2)

    def construct(self, x):
        x = self.conv1(x)
        x = self.max_pool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x

def _resnet(arch: str, block: Type[Union[ResidualBlockBase, ResidualBlock]], layers: List[int],

```

```

    num_classes: int, pretrained: bool, input_channel: int):
    backbone = ResNet(block, layers)
    neck = GlobalAvgPooling() # 平均池化层
    head = DenseHead(input_channel=input_channel, num_classes=num_classes) # 全连接层
    model = BaseClassifier(backbone, neck, head) # 将backbone层、neck层和head层连接起来

    if pretrained:
        # 下载并加载预训练模型
        LoadPretrainedModel(model, model_urls[arch]).run()

    return model

def resnet50(num_classes: int = 1000, pretrained: bool = False):
    "ResNet50模型"
    return _resnet("resnet50", ResidualBlock, [3, 4, 6, 3], num_classes, pretrained, 2048)

```

2.6 模型训练与评估

```

from mindspore.train import Model
from mindvision.engine.callback import ValAccMonitor

# 定义ResNet50网络
network = resnet50(pretrained=True)

# 全连接层输入层的大小
in_channel = network.head.dense.in_channels
head = DenseHead(input_channel=in_channel, num_classes=5)
# 重置全连接层
network.head = head
# 设置学习率
num_epochs = 20
lr = nn.cosine_decay_lr(min_lr=0.00001, max_lr=0.001, total_step=step_size *
    num_epochs, step_per_epoch=step_size, decay_epoch=num_epochs)
# 定义优化器和损失函数
opt = nn.Momentum(params=network.trainable_params(), learning_rate=lr, momentum=0.9)
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
# 实例化模型
model = Model(network, loss, opt, metrics={"Accuracy": nn.Accuracy()})
# 模型训练
model.train(num_epochs, de_train, callbacks=[ValAccMonitor(model, de_test, num_epochs)])

```

2.7 可视化模型预测 (第一版, 在.ipynb 文件中已被替换)

```
import matplotlib.pyplot as plt
from mindspore import Tensor
from mindspore import load_checkpoint, load_param_into_net

def visualize_model(best_ckpt_path, val_ds):
    num_class = 5
    net = resnet50(num_class)
    # 加载模型参数
    param_dict = load_checkpoint(best_ckpt_path)
    load_param_into_net(net, param_dict)
    model = Model(net)
    # 加载验证集的数据进行验证
    data = next(val_ds.create_dict_iterator())
    images = data["image"].asnumpy()
    labels = data["label"].asnumpy()
    # 预测图像类别
    output = model.predict(Tensor(data['image']))
    pred = np.argmax(output.asnumpy(), axis=1)

    # 显示图像及图像的预测值
    plt.figure()
    for i in range(len(images)):
        plt.subplot(2, 3, i+1)
        # 若预测正确, 显示为蓝色; 若预测错误, 显示为红色
        color = 'blue' if pred[i] == labels[i] else 'red'
        plt.title('predict:{0}'.format(pred[i]), color=color)
        picture_show = np.transpose(images[i], (1, 2, 0))
        mean = np.array([0.4914, 0.4822, 0.4465])
        std = np.array([0.2023, 0.1994, 0.2010])
        picture_show = std * picture_show + mean
        picture_show = np.clip(picture_show, 0, 1)
        plt.imshow(picture_show)
        plt.axis('off')

    plt.show()

# 使用测试数据集进行验证
visualize_model('best.ckpt', de_test)
```

2.8 可视化模型预测 (第二版)

```

import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from mindspore import Tensor
from mindspore import load_checkpoint, load_param_into_net

def visualize_model(best_ckpt_path, image_paths):
    num_class = 5
    net = resnet50(num_class) # 加载模型参数
    param_dict = load_checkpoint(best_ckpt_path)
    load_param_into_net(net, param_dict)
    model = Model(net) # 预处理图片并进行预测
    # 标签名称映射
    class_names = {'daisy': 0, 'dandelion': 1, 'roses': 2, 'sunflowers': 3, 'tulips': 4}
    images = []
    labels = [] # 存储标签名称
    for path in image_paths:
        image = Image.open(path)
        image = image.resize((224, 224))
        image = np.array(image)
        image = np.transpose(image, (2, 0, 1))
        image = image.astype(np.float32)
        image = (image / 255 - 0.5) / 0.5 # 标准化
        images.append(image)
        label = path.split('/')[-1].split('.')[0] # 获取标签名称
        labels.append(label)
    images = np.array(images)
    output = model.predict(Tensor(images))
    preds = np.argmax(output.asnumpy(), axis=1) # 显示图片及预测结果
    plt.figure()
    for i in range(len(image_paths)):
        plt.subplot(2, 3, i+1)
        if preds[i] == class_names[labels[i]]:
            color = 'blue' # 蓝色代表预测正确
        else:
            color = 'red' # 红色代表预测错误
        plt.title('predict:{0}'.format(list(class_names.keys())[list(class_names.values()).index(preds[i])]),
            color=color) # 输出预测正确的名称
        image = Image.open(image_paths[i])
        plt.imshow(image)
        plt.axis('off')
    plt.show()

image_paths = ['roses.jpg', 'tulips.jpg', 'sunflowers.jpg']

```

```
visualize_model('best.ckpt', image_paths)
```

2.9 模型保存和转换

2.9.1 将模型保存为 onnx 格式文件

```
import os
#创建文件夹
if not os.path.exists('./flowers/'):
    os.mkdir('./flowers/')
param_dict = load_checkpoint(os.path.join('best.ckpt'))
# load the parameter into net
resnet=resnet50(num_classes=cfg.num_class)
load_param_into_net(resnet, param_dict)
x = np.random.uniform(-1.0, 1.0, size = [1, 3, cfg.HEIGHT,cfg.WIDTH]).astype(np.float32)
export(resnet, Tensor(x), file_name = './flowers/best_model.onnx',file_format = 'ONNX')
```

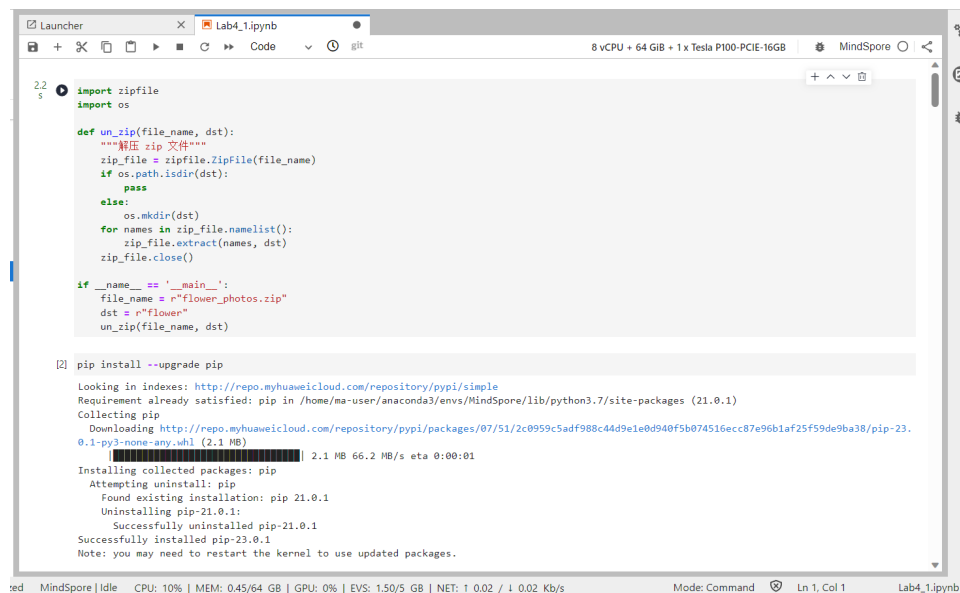
2.9.2 将模型存储至自己创建的 obs 桶中

```
import moxing
moxing.file.copy_parallel(src_url='./flowers/best_model.onnx',
                          dst_url='s3://wlm-obs-hx/flower/onnx/best_model.onnx')
```

3 实验结果及分析

3.1 导入数据集

导入数据集压缩包,对压缩包进行解压缩,并划分训练集和测试集



```
2.2
s
import zipfile
import os

def un_zip(file_name, dst):
    """解压 zip 文件"""
    zip_file = zipfile.ZipFile(file_name)
    if os.path.isdir(dst):
        pass
    else:
        os.mkdir(dst)
    for names in zip_file.namelist():
        zip_file.extract(names, dst)
    zip_file.close()

if __name__ == '__main__':
    file_name = r"flower_photos.zip"
    dst = r"flower"
    un_zip(file_name, dst)

[2] pip install --upgrade pip

Looking in indexes: http://repo.myhuaweicloud.com/repository/pypi/simple
Requirement already satisfied: pip in /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages (21.0.1)
Collecting pip
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/07/51/2c0959c5adf988c44d9e1e0d940f5b074516ecc87e96b1af25f59de9ba38/pip-23.0.1-py3-none-any.whl (2.1 MB)
    |#####| 2.1 MB 66.2 MB/s eta 0:00:01
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.0.1
    Uninstalling pip-21.0.1:
      Successfully uninstalled pip-21.0.1
  Successfully installed pip-23.0.1
Note: you may need to restart the kernel to use updated packages.
```

图 1: 解压缩代码展示图

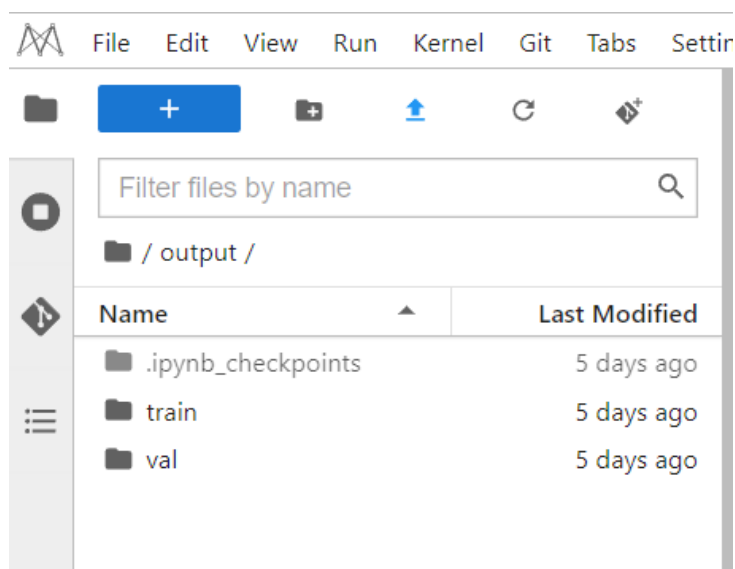


图 2: 划分训练集和测试集成功后生成文件夹

3.2 数据集读取及预处理

MindSpore 的 `mindspore.dataset` 提供了 `ImageFolderDataset` 函数,可以直接读取文件夹图片数据并映射文件夹名字为其标签 (label)。此处使用 `ImageFolderDataset` 函数读取 'daisy', 'dandelion', 'roses', 'sunflowers', 'tulips' 数据。并将这五类标签映射为: 'daisy':0, 'dandelion':1, 'roses':2, 'sunflowers':3, 'tulips':4, 使用 `RandomCropDecodeResize`, `HWC2CHW`, `shuffle` 进行数据预处理。

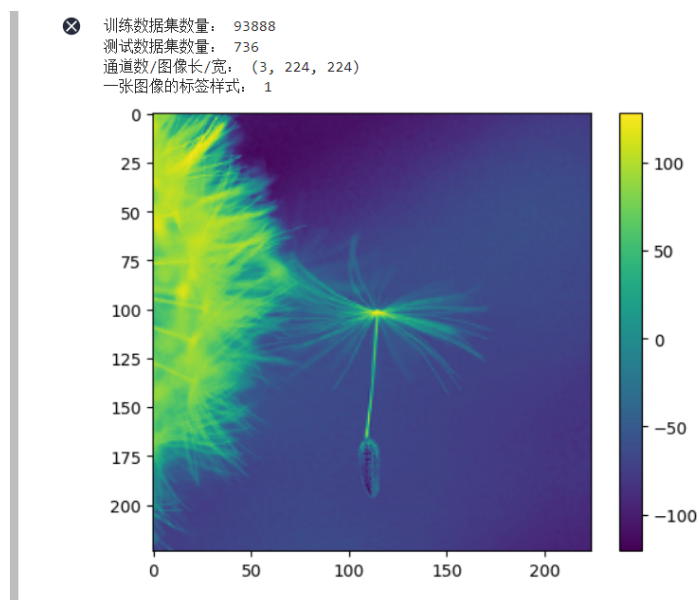
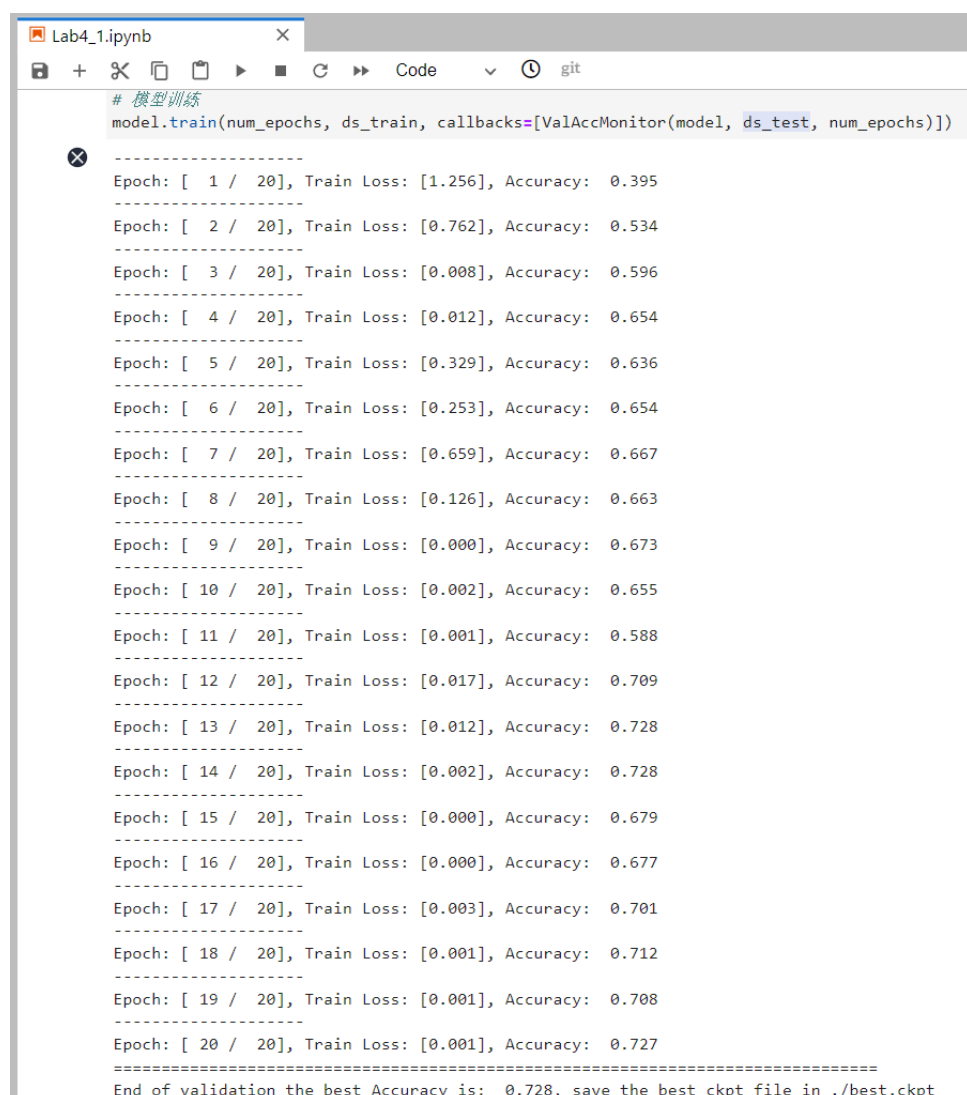


图 3: 预处理绘制图

3.3 模型训练与评估

调用 resnet50 构造 ResNet50 模型,定义优化器和损失函数,通过 model.train 接口对网络进行训练,将 MindSpore Vision 中的 mindvision.engine.callback.ValAccMonitor 接口传入回调函数中,将会打印训练的损失值和评估精度,并把评估精度最高的 ckpt 文件 (best.ckpt) 保存到当前目录下。



```
Lab4_1.ipynb
# 模型训练
model.train(num_epochs, ds_train, callbacks=[ValAccMonitor(model, ds_test, num_epochs)])

Epoch: [ 1 / 20], Train Loss: [1.256], Accuracy: 0.395
Epoch: [ 2 / 20], Train Loss: [0.762], Accuracy: 0.534
Epoch: [ 3 / 20], Train Loss: [0.008], Accuracy: 0.596
Epoch: [ 4 / 20], Train Loss: [0.012], Accuracy: 0.654
Epoch: [ 5 / 20], Train Loss: [0.329], Accuracy: 0.636
Epoch: [ 6 / 20], Train Loss: [0.253], Accuracy: 0.654
Epoch: [ 7 / 20], Train Loss: [0.659], Accuracy: 0.667
Epoch: [ 8 / 20], Train Loss: [0.126], Accuracy: 0.663
Epoch: [ 9 / 20], Train Loss: [0.000], Accuracy: 0.673
Epoch: [10 / 20], Train Loss: [0.002], Accuracy: 0.655
Epoch: [11 / 20], Train Loss: [0.001], Accuracy: 0.588
Epoch: [12 / 20], Train Loss: [0.017], Accuracy: 0.709
Epoch: [13 / 20], Train Loss: [0.012], Accuracy: 0.728
Epoch: [14 / 20], Train Loss: [0.002], Accuracy: 0.728
Epoch: [15 / 20], Train Loss: [0.000], Accuracy: 0.679
Epoch: [16 / 20], Train Loss: [0.000], Accuracy: 0.677
Epoch: [17 / 20], Train Loss: [0.003], Accuracy: 0.701
Epoch: [18 / 20], Train Loss: [0.001], Accuracy: 0.712
Epoch: [19 / 20], Train Loss: [0.001], Accuracy: 0.708
Epoch: [20 / 20], Train Loss: [0.001], Accuracy: 0.727
=====
End of validation the best Accuracy is: 0.728, save the best ckpt file in ./best.ckpt
```

图 4: 输出迭代次数、对应的损失和精度

3.4 可视化模型预测

使用验证精度最高的模型对测试数据集进行预测,并将预测结果可视化。若预测字体颜色为蓝色表示为预测正确,预测字体颜色为红色则表示预测错误。



图 5: 预测正确显示图



图 6: 预测错误显示图



图 7: 使用提供的测试图片预测结果图

3.5 模型保存和转换

3.5.1 将模型保存为 onnx 格式文件,并存储至 obs 桶中



图 8: onnx 格式文件展示

3.5.2 将 onnx 格式模型转换为 om 格式

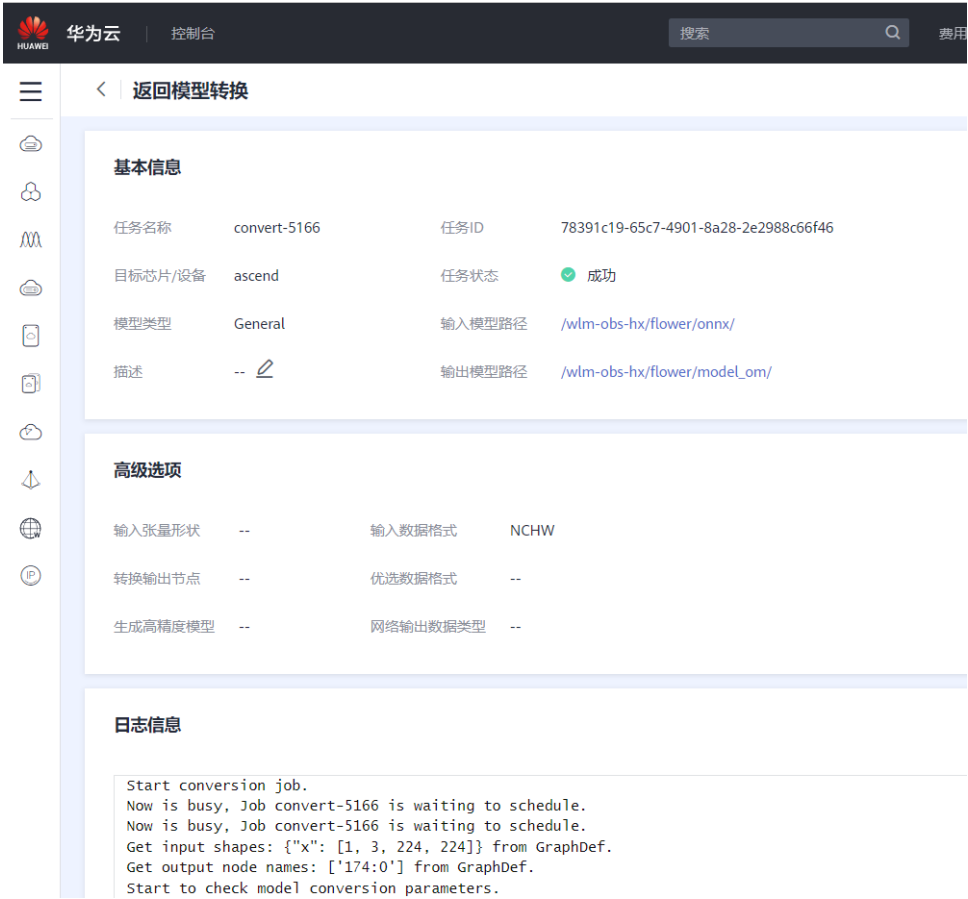


图 9: 模型转换成功

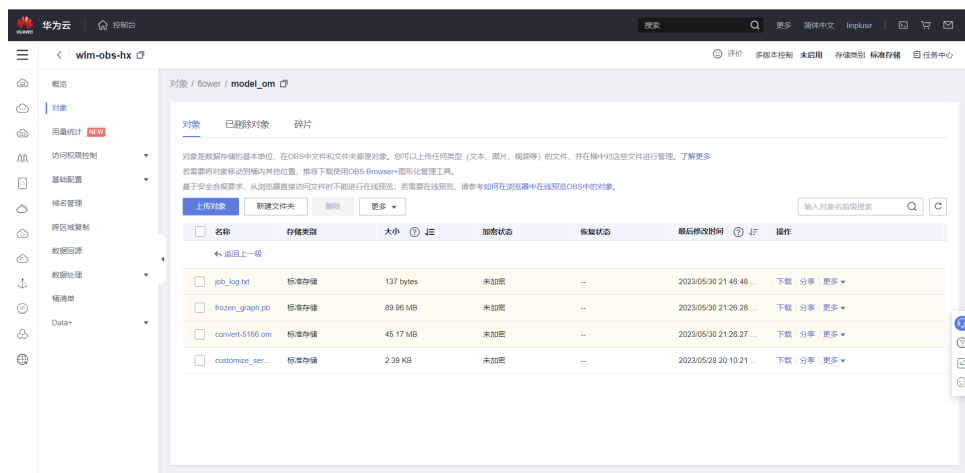


图 10: om 格式文件展示

3.6 模型部署上线

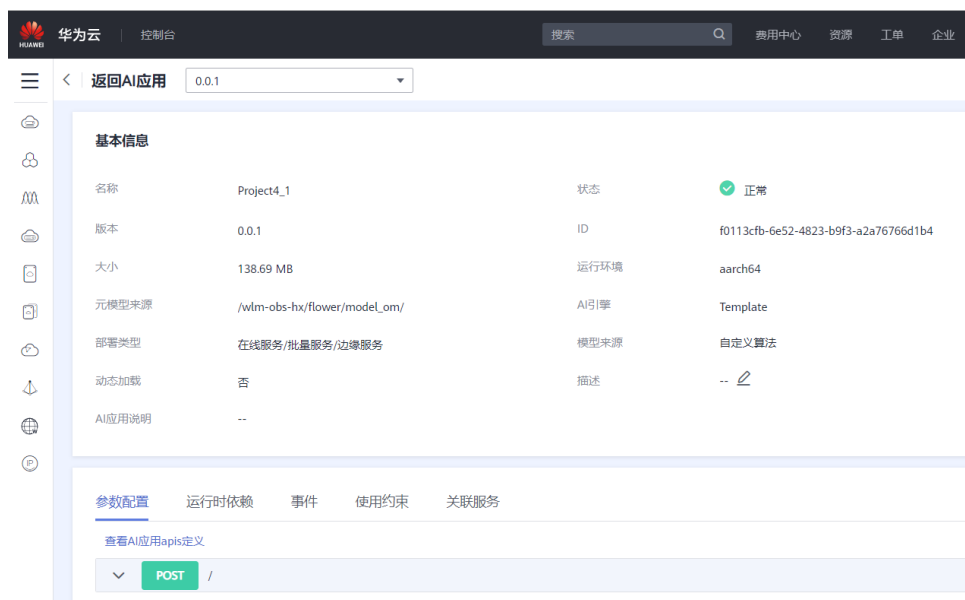


图 11: 模型创建成功

4 遇到的问题及解决办法

问题:模型部署失败

解决办法: 创建工单, 咨询华为云工程师, 最后, 采取修改模型可视化预测部分的代码, 直接在这个步骤实现对提供的图片的可视化预测, 放弃华为云平台的模型部署

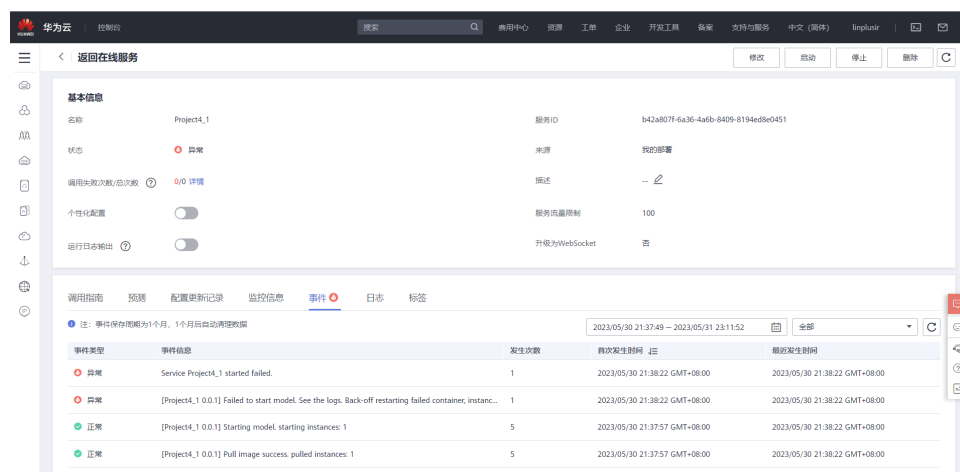


图 12: 在线服务部署异常

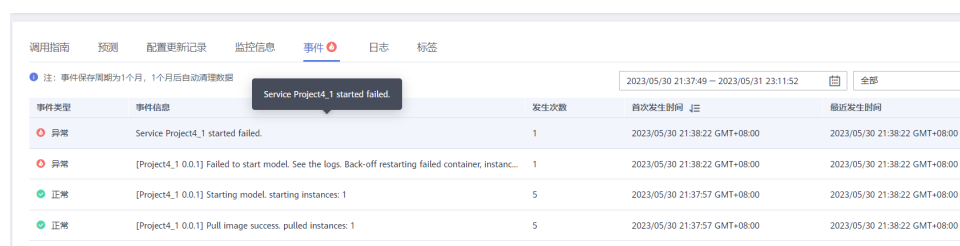


图 13: 异常信息展示



图 14: 异常信息展示

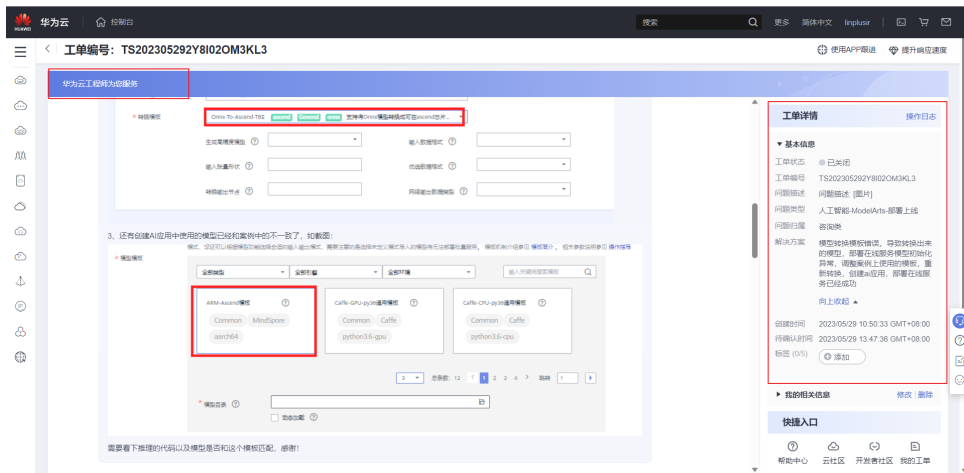


图 15: 咨询界面截图

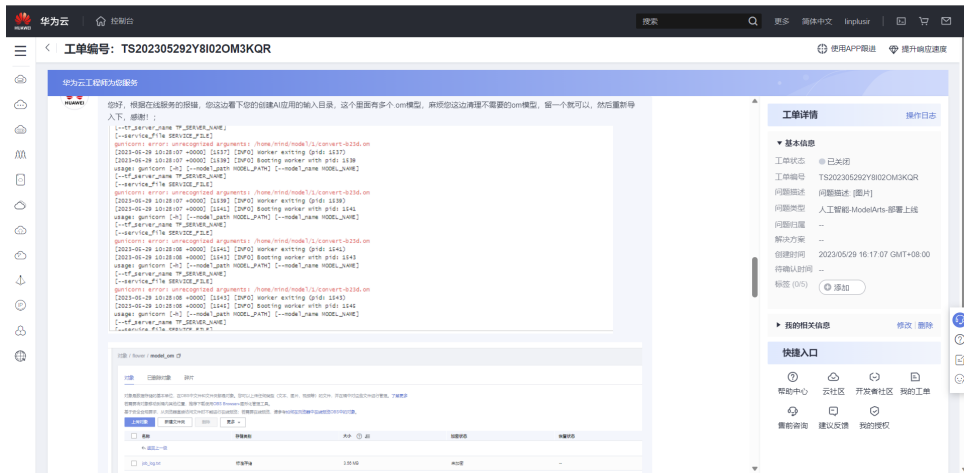


图 16: 咨询界面截图

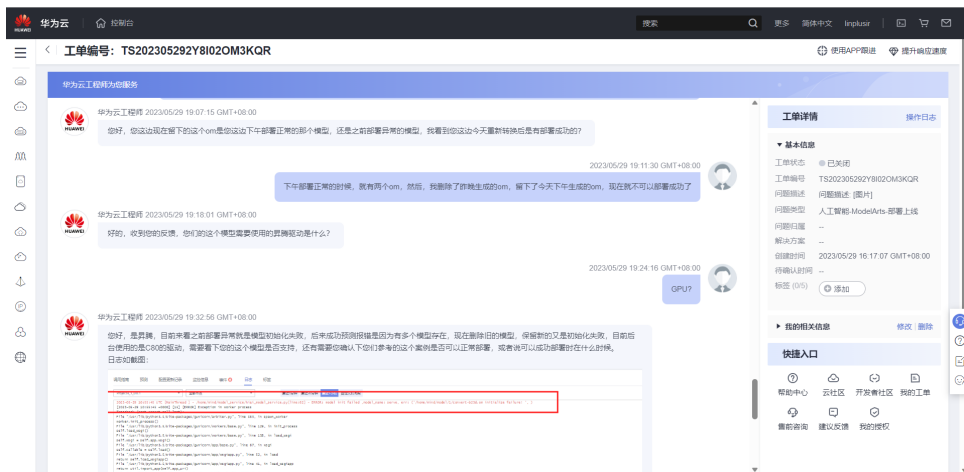


图 17: 咨询界面截图



图 18: 咨询界面截图