基于MindSpore的ResNet50图像分类迁移学习实验手册



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2022。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 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。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验介绍 2](#_Toc177056699)

[1.1 实验目的 2](#_Toc177056700)

[1.2 实验清单 2](#_Toc177056701)

[1.3 开发平台介绍 2](#_Toc177056702)

[1.4 开发环境搭建 3](#_Toc177056703)

[2 基于MindSpore的ResNet50图像分类迁移学习 4](#_Toc177056704)

[2.1 ResNet网络介绍背景知识 4](#_Toc177056705)

[2.2 数据集准备、加载与可视化 5](#_Toc177056706)

[2.2.1 数据集准备 5](#_Toc177056707)

[2.2.2 数据集加载 6](#_Toc177056708)

[2.2.3 数据集可视化 7](#_Toc177056709)

[2.3 模型训练 9](#_Toc177056710)

[2.3.1 构建Resnet50网络 9](#_Toc177056711)

[2.3.2 模型微调 12](#_Toc177056712)

[2.3.3 训练和评估 13](#_Toc177056713)

[2.3.4 可视化模型预测 15](#_Toc177056714)

[2.4 网络迁移学习（固定特征进行训练） 17](#_Toc177056715)

[2.4.1 模型微调 17](#_Toc177056716)

[2.4.2 训练与评估 18](#_Toc177056717)

[2.4.3 可视化模型预测 20](#_Toc177056718)

[2.5 实验小结 21](#_Toc177056719)

# 实验介绍

图像分类是最基础的计算机视觉应用，属于有监督学习类别，如给定一张图像(猫、狗、飞机、汽车等等)，判断图像所属的类别。在实际应用场景中，往往由于训练数据集不足，很少有人会从头开始训练整个网络。普遍的做法是，在一个非常大的基础数据集上训练得到一个预训练模型，然后使用该模型来初始化网络的权重参数或作为固定特征提取器应用于特定的任务中。本实验基于MindSpore框架，将使用迁移学习的方法对ImageNet数据集中的狼和狗图像进行分类。

## 实验目的

本实验基于MindSpore框架，将使用迁移学习的方法对ImageNet数据集中的狼和狗图像进行分类。通过本实验可以了解到如何对预训练模型进行模型微调，以快速获得符合实际任务的模型，进一步可以了解到如何固定特征进行模型训练，以快速获得符合实际任务的模型。

## 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验** | **简述** | **难度** | **软件环境** | **开发环境** |
| 基于MindSpore的ResNet50图像分类迁移学习 | 本实验基于MindSpore框架，将使用迁移学习的方法对ImageNet数据集中的狼和狗图像进行分类。 | 中级 | Python3.9、MindSpore2.2 | ModelArts、 Ascend910 |

## 开发平台介绍

昇腾910（Ascend 910）是华为自研的一款基于达芬奇架构的AI处理器，具有超高算力的和极高的能效比，最高可达320TFLOPS（FP16）的浮点计算能力。昇腾910的片上系统（SoC）还集成了多个CPU、DVPP和任务调度器，因而具有自我管理能力，可以充分发挥其高算力的特点。强大的矩阵、向量运行能力，在进行海量数据运算的神经网络训练场景有巨大的优势。

昇思MindSpore（官方网站：<https://www.mindspore.cn/>）是一种适用于端边云场景的新型开源深度学习训练/推理框架。 MindSpore提供了友好的设计和高效的执行，旨在提升数据科学家和算法工程师的开发体验，并为Ascend AI处理器提供原生支持，以及软硬件协同优化。同时，MindSpore作为全球AI开源社区，致力于进一步开发和丰富AI软硬件应用生态。

MindSpore Vision是基于MindSpore的开源计算机视觉研究工具库，MindSpore Vision已构筑基础的AI复用能力，支持部分主流、前沿网络，例如Efficientnet、Lenet、Mobilenetv2、Resnet、Vit等，提供手机侧推理与部署，欢迎开发者们参与贡献，共同拓展和维护MindSpore Vision，力争将其打造为易用易理解的主流工具库。

ModelArts（官方网站：<https://console.huaweicloud.com/modelarts/>）是面向AI开发者的一站式开发平台，提供海量数据预处理及半自动化标注、大规模分布式训练、自动化模型生成及模型按需部署能力，帮助用户快速创建和部署AI应用，管理全周期AI工作流。

## 开发环境搭建

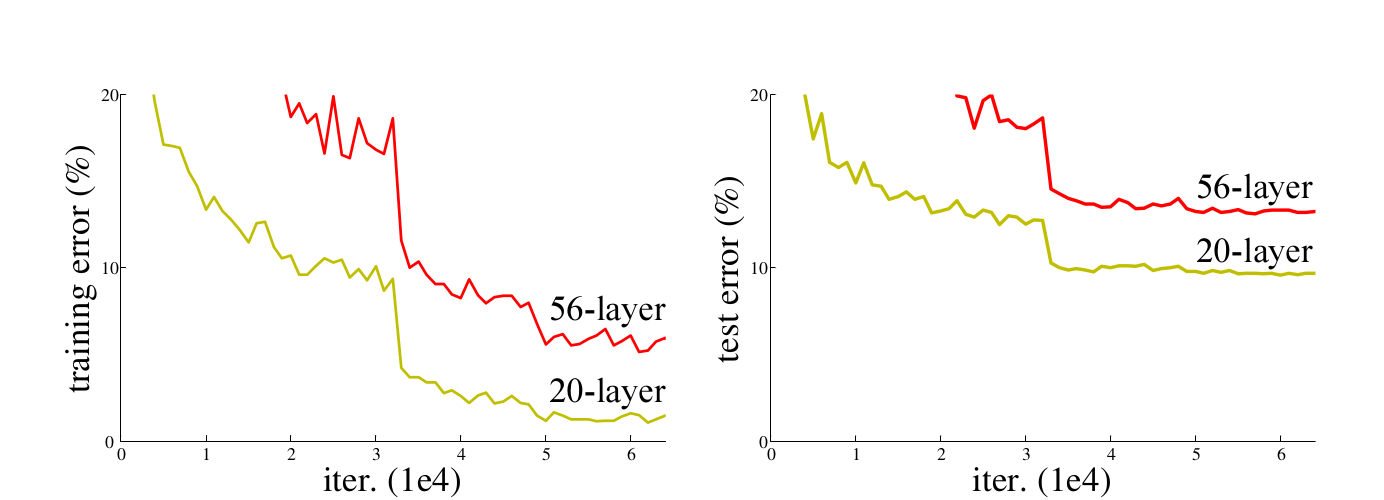
本实验开发环境为华为云ModelArts平台，环境搭建方式请参考如下手册：



# 基于MindSpore的ResNet50图像分类迁移学习

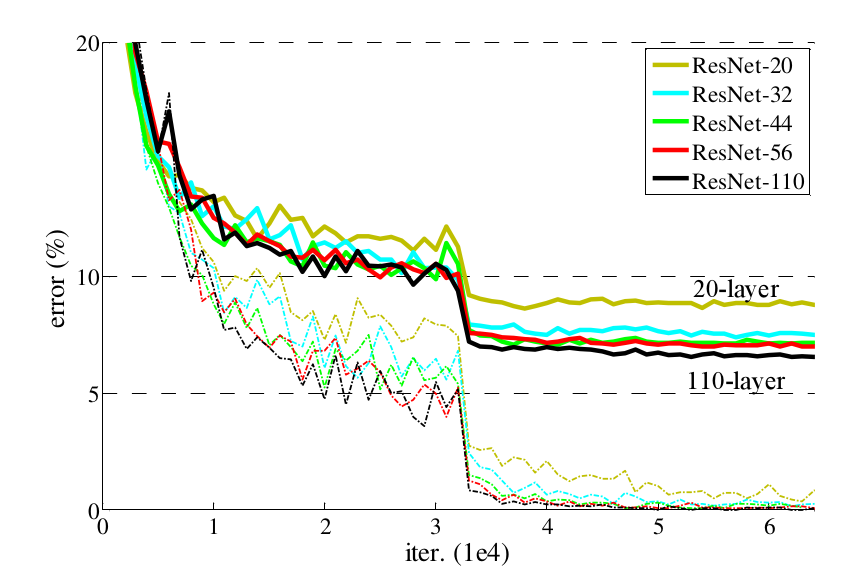
## ResNet网络介绍背景知识

ResNet50网络是2015年由微软实验室的何恺明提出，获得ILSVRC2015图像分类竞赛第一名。在ResNet网络提出之前，传统的卷积神经网络都是将一系列的卷积层和池化层堆叠得到的，但当网络堆叠到一定深度时，就会出现退化问题。图2-1是在CIFAR-10数据集上使用56层网络与20层网络训练误差和测试误差图，由图中数据可以看出，56层网络比20层网络训练误差和测试误差更大，随着网络的加深，其误差并没有如预想的一样减小。



传统卷积网络，不同深度下错误率对比

ResNet网络提出了残差网络结构(Residual Network)来减轻退化问题，使用ResNet网络可以实现搭建较深的网络结构（突破1000层）。论文中使用ResNet网络在CIFAR-10数据集上的训练误差与测试误差对比如图2-2所示，图中虚线表示训练误差，实线表示测试误差。由图中数据可以看出，ResNet网络层数越深，其训练误差和测试误差越小。



添加残差结构的卷积网络，不同深度下的错误率对比

了解ResNet网络更多详细内容，参见[ResNet论文](https://arxiv.org/pdf/1512.03385.pdf)。

## 数据集准备、加载与可视化

### 数据集准备

本实验使用[狗与狼分类数据集](https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/datasets/intermediate/Canidae_data.zip" \t "_blank)，数据集中的图像来自于ImageNet，每个分类有大约120张训练图像与30张验证图像。本实验使用download接口下载数据集，并将下载后的数据集自动解压到当前目录下。

from download import download

dataset\_url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/datasets/intermediate/Canidae\_data.zip"

download(dataset\_url, "./datasets-Canidae", kind="zip", replace=True)

狗与狼分类数据集文件的目录结构如下：

datasets-Canidae/data/

└── Canidae

├── train

│ ├── dogs

│ └── wolves

└── val

├── dogs

└── wolves

### 数据集加载

狼狗数据集提取自ImageNet分类数据集，本实验使用mindspore.dataset.

ImageFolderDataset接口来加载数据集，并进行相关图像增强操作。

首先执行过程定义一些输入：

batch\_size = 18 # 批量大小

image\_size = 224 # 训练图像空间大小

num\_epochs = 10 # 训练周期数

lr = 0.001 # 学习率

momentum = 0.9 # 动量

workers = 4 # 并行线程个数

import mindspore as ms

import mindspore.dataset as ds

import mindspore.dataset.vision as vision

# 数据集目录路径

data\_path\_train = "./datasets-Canidae/data/Canidae/train/"

data\_path\_val = "./datasets-Canidae/data/Canidae/val/"

# 创建训练数据集

def create\_dataset\_canidae(dataset\_path, usage):

"""数据加载"""

data\_set = ds.ImageFolderDataset(dataset\_path,

num\_parallel\_workers=workers,

shuffle=True,)

# 数据增强操作

mean = [0.485 \* 255, 0.456 \* 255, 0.406 \* 255]

std = [0.229 \* 255, 0.224 \* 255, 0.225 \* 255]

scale = 32

if usage == "train":

# Define map operations for training dataset

trans = [

vision.RandomCropDecodeResize(size=image\_size, scale=(0.08, 1.0), ratio=(0.75, 1.333)),

vision.RandomHorizontalFlip(prob=0.5),

vision.Normalize(mean=mean, std=std),

vision.HWC2CHW()

]

else:

# Define map operations for inference dataset

trans = [

vision.Decode(),

vision.Resize(image\_size + scale),

vision.CenterCrop(image\_size),

vision.Normalize(mean=mean, std=std),

vision.HWC2CHW()

]

# 数据映射操作

data\_set = data\_set.map(

operations=trans,

input\_columns='image',

num\_parallel\_workers=workers)

# 批量操作

data\_set = data\_set.batch(batch\_size)

return data\_set

dataset\_train = create\_dataset\_canidae(data\_path\_train, "train")

step\_size\_train = dataset\_train.get\_dataset\_size()

dataset\_val = create\_dataset\_canidae(data\_path\_val, "val")

step\_size\_val = dataset\_val.get\_dataset\_size()

### 数据集可视化

从mindspore.dataset.ImageFolderDataset接口中加载的训练数据集返回值为字典，用户可通过 create\_dict\_iterator 接口创建数据迭代器，使用 next 迭代访问数据集。本章中 batch\_size 设为18，所以使用 next 一次可获取18个图像及标签数据。

data = next(dataset\_train.create\_dict\_iterator())

images = data["image"]

labels = data["label"]

print("Tensor of image", images.shape)

输出结果：

Tensor of image (18, 3, 224, 224)

Labels: [1 0 0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0]

对获取到的图像及标签数据进行可视化，标题为图像对应的label名称。

import matplotlib.pyplot as plt

import numpy as np

# class\_name对应label，按文件夹字符串从小到大的顺序标记label

class\_name = {0: "dogs", 1: "wolves"}

plt.figure(figsize=(5, 5))

for i in range(4):

# 获取图像及其对应的label

data\_image = images[i].asnumpy()

data\_label = labels[i]

# 处理图像供展示使用

data\_image = np.transpose(data\_image, (1, 2, 0))

mean = np.array([0.485, 0.456, 0.406])

std = np.array([0.229, 0.224, 0.225])

data\_image = std \* data\_image + mean

data\_image = np.clip(data\_image, 0, 1)

# 显示图像

plt.subplot(2, 2, i+1)

plt.imshow(data\_image)

plt.title(class\_name[int(labels[i].asnumpy())])

plt.axis("off")

plt.show()

输出结果：



## 模型训练

本环节使用ResNet50模型进行训练。搭建好模型框架后，通过将pretrained参数设置为True来下载ResNet50的预训练模型并将权重参数加载到网络中。

### 构建Resnet50网络

from typing import Type, Union, List, Optional

from mindspore import nn, train

from mindspore.common.initializer import Normal

weight\_init = Normal(mean=0, sigma=0.02)

gamma\_init = Normal(mean=1, sigma=0.02)

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:

self.norm = nn.BatchNorm2d(out\_channel)

else:

self.norm = norm

self.conv1 = nn.Conv2d(in\_channel, out\_channel,

kernel\_size=3, stride=stride,

weight\_init=weight\_init)

self.conv2 = nn.Conv2d(in\_channel, out\_channel,

kernel\_size=3, weight\_init=weight\_init)

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.norm(out)

out = self.relu(out)

out = self.conv2(out) # 主分支第二层：3\*3卷积层

out = self.norm(out)

if self.down\_sample is not None:

identity = self.down\_sample(x)

out += identity # 输出为主分支与shortcuts之和

out = self.relu(out)

return out

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, down\_sample: Optional[nn.Cell] = None) -> None:

super(ResidualBlock, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channel, out\_channel,

kernel\_size=1, weight\_init=weight\_init)

self.norm1 = nn.BatchNorm2d(out\_channel)

self.conv2 = nn.Conv2d(out\_channel, out\_channel,

kernel\_size=3, stride=stride,

weight\_init=weight\_init)

self.norm2 = nn.BatchNorm2d(out\_channel)

self.conv3 = nn.Conv2d(out\_channel, out\_channel \* self.expansion,

kernel\_size=1, weight\_init=weight\_init)

self.norm3 = nn.BatchNorm2d(out\_channel \* self.expansion)

self.relu = nn.ReLU()

self.down\_sample = down\_sample

def construct(self, x):

identity = x # shortscuts分支

out = self.conv1(x) # 主分支第一层：1\*1卷积层

out = self.norm1(out)

out = self.relu(out)

out = self.conv2(out) # 主分支第二层：3\*3卷积层

out = self.norm2(out)

out = self.relu(out)

out = self.conv3(out) # 主分支第三层：1\*1卷积层

out = self.norm3(out)

if self.down\_sample is not None:

identity = self.down\_sample(x)

out += identity # 输出为主分支与shortcuts之和

out = self.relu(out)

return out

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 = nn.SequentialCell([

nn.Conv2d(last\_out\_channel, channel \* block.expansion,

kernel\_size=1, stride=stride, weight\_init=weight\_init),

nn.BatchNorm2d(channel \* block.expansion, gamma\_init=gamma\_init)

])

layers = []

layers.append(block(last\_out\_channel, channel, stride=stride, down\_sample=down\_sample))

in\_channel = channel \* block.expansion

# 堆叠残差网络

for \_ in range(1, block\_nums):

layers.append(block(in\_channel, channel))

return nn.SequentialCell(layers)

from mindspore import load\_checkpoint, load\_param\_into\_net

class ResNet(nn.Cell):

def \_\_init\_\_(self, block: Type[Union[ResidualBlockBase, ResidualBlock]],

layer\_nums: List[int], num\_classes: int, input\_channel: int) -> None:

super(ResNet, self).\_\_init\_\_()

self.relu = nn.ReLU()

# 第一个卷积层，输入channel为3（彩色图像），输出channel为64

self.conv1 = nn.Conv2d(3, 64, kernel\_size=7, stride=2, weight\_init=weight\_init)

self.norm = nn.BatchNorm2d(64)

# 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)

# 平均池化层

self.avg\_pool = nn.AvgPool2d()

# flattern层

self.flatten = nn.Flatten()

# 全连接层

self.fc = nn.Dense(in\_channels=input\_channel, out\_channels=num\_classes)

def construct(self, x):

x = self.conv1(x)

x = self.norm(x)

x = self.relu(x)

x = self.max\_pool(x)

x = self.layer1(x)

x = self.layer2(x)

x = self.layer3(x)

x = self.layer4(x)

x = self.avg\_pool(x)

x = self.flatten(x)

x = self.fc(x)

return x

def \_resnet(model\_url: str, block: Type[Union[ResidualBlockBase, ResidualBlock]],

layers: List[int], num\_classes: int, pretrained: bool, pretrianed\_ckpt: str,

input\_channel: int):

model = ResNet(block, layers, num\_classes, input\_channel)

if pretrained:

# 加载预训练模型

download(url=model\_url, path=pretrianed\_ckpt, replace=True)

param\_dict = load\_checkpoint(pretrianed\_ckpt)

load\_param\_into\_net(model, param\_dict)

return model

def resnet50(num\_classes: int = 1000, pretrained: bool = False):

"ResNet50模型"

resnet50\_url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/models/application/resnet50\_224\_new.ckpt"

resnet50\_ckpt = "./LoadPretrainedModel/resnet50\_224\_new.ckpt"

return \_resnet(resnet50\_url, ResidualBlock, [3, 4, 6, 3], num\_classes,

pretrained, resnet50\_ckpt, 2048)

### 模型微调

由于ResNet50中的预训练模型是针对ImageNet数据集中的1000个类别进行分类的，本实验只对狼和狗两个类别进行分类，所以需要重置预训练模型中的分类器，然后重新微调网络。

import mindspore as ms

network = resnet50(pretrained=True)

# 全连接层输入层的大小

in\_channels = network.fc.in\_channels

# 输出通道数大小为狼狗分类数2

head = nn.Dense(in\_channels, 2)

# 重置全连接层

network.fc = head

# 平均池化层kernel size为7

avg\_pool = nn.AvgPool2d(kernel\_size=7)

# 重置平均池化层

network.avg\_pool = avg\_pool

import mindspore as ms

# 定义优化器和损失函数

opt = nn.Momentum(params=network.trainable\_params(), learning\_rate=lr, momentum=momentum)

loss\_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 实例化模型

model = train.Model(network, loss\_fn, opt, metrics={"Accuracy": train.Accuracy()})

def forward\_fn(inputs, targets):

logits = network(inputs)

loss = loss\_fn(logits, targets)

return loss

grad\_fn = ms.value\_and\_grad(forward\_fn, None, opt.parameters)

def train\_step(inputs, targets):

loss, grads = grad\_fn(inputs, targets)

opt(grads)

return loss

### 训练和评估

训练并评估网络，且在训练完成后，保存评估精度最高的ckpt文件(resnet50-best.ckpt)到当前路径的/BestCheckpoint下，保存路径和ckpt文件名可自行调整。

# 创建迭代器

data\_loader\_train = dataset\_train.create\_tuple\_iterator(num\_epochs=num\_epochs)

# 最佳模型保存路径

best\_ckpt\_dir = "./BestCheckpoint"

best\_ckpt\_path = "./BestCheckpoint/resnet50-best.ckpt"

import os

import time

# 开始循环训练

print("Start Training Loop ...")

best\_acc = 0

for epoch in range(num\_epochs):

losses = []

network.set\_train()

epoch\_start = time.time()

# 为每轮训练读入数据

for i, (images, labels) in enumerate(data\_loader\_train):

labels = labels.astype(ms.int32)

loss = train\_step(images, labels)

losses.append(loss)

# 每个epoch结束后，验证准确率

acc = model.eval(dataset\_val)['Accuracy']

epoch\_end = time.time()

epoch\_seconds = (epoch\_end - epoch\_start) \* 1000

step\_seconds = epoch\_seconds/step\_size\_train

print("-" \* 20)

print("Epoch: [%3d/%3d], Average Train Loss: [%5.3f], Accuracy: [%5.3f]" % (

epoch+1, num\_epochs, sum(losses)/len(losses), acc

))

print("epoch time: %5.3f ms, per step time: %5.3f ms" % (

epoch\_seconds, step\_seconds

))

if acc > best\_acc:

best\_acc = acc

if not os.path.exists(best\_ckpt\_dir):

os.mkdir(best\_ckpt\_dir)

ms.save\_checkpoint(network, best\_ckpt\_path)

print("=" \* 80)

print(f"End of validation the best Accuracy is: {best\_acc: 5.3f}, "

f"save the best ckpt file in {best\_ckpt\_path}", flush=True)

输出结果：

--------------------

Epoch: [ 1 / 10], Train Loss: [0.469], Accuracy: 1.000

epoch time: 6525.242 ms, per step time: 501.942 ms

--------------------

Epoch: [ 2 / 10], Train Loss: [0.134], Accuracy: 1.000

epoch time: 2549.441 ms, per step time: 196.111 ms

--------------------

Epoch: [ 3 / 10], Train Loss: [0.069], Accuracy: 1.000

epoch time: 2561.402 ms, per step time: 197.031 ms

--------------------

Epoch: [ 4 / 10], Train Loss: [0.131], Accuracy: 1.000

epoch time: 2564.437 ms, per step time: 197.264 ms

--------------------

Epoch: [ 5 / 10], Train Loss: [0.097], Accuracy: 1.000

epoch time: 2563.061 ms, per step time: 197.159 ms

--------------------

Epoch: [ 6 / 10], Train Loss: [0.037], Accuracy: 1.000

epoch time: 2569.943 ms, per step time: 197.688 ms

--------------------

Epoch: [ 7 / 10], Train Loss: [0.011], Accuracy: 1.000

epoch time: 2577.678 ms, per step time: 198.283 ms

--------------------

Epoch: [ 8 / 10], Train Loss: [0.018], Accuracy: 1.000

epoch time: 2574.261 ms, per step time: 198.020 ms

--------------------

Epoch: [ 9 / 10], Train Loss: [0.036], Accuracy: 1.000

epoch time: 2568.633 ms, per step time: 197.587 ms

--------------------

Epoch: [ 10 / 10], Train Loss: [0.016], Accuracy: 1.000

epoch time: 2558.043 ms, per step time: 196.773 ms

================================================================================

End of validation the best Accuracy is: 1.000, save the best ckpt file in ./best.ckpt

真实训练过程中，Accuracy值可能不高，属于正常现象，可增大num\_epochs参数，使网路训练轮数增多，以提高准确率。

### 可视化模型预测

定义 visualize\_mode 函数，可视化模型预测。

import matplotlib.pyplot as plt

import mindspore as ms

def visualize\_model(best\_ckpt\_path, val\_ds):

net = resnet50()

# 全连接层输入层的大小

in\_channels = net.fc.in\_channels

# 输出通道数大小为狼狗分类数2

head = nn.Dense(in\_channels, 2)

# 重置全连接层

net.fc = head

# 平均池化层kernel size为7

avg\_pool = nn.AvgPool2d(kernel\_size=7)

# 重置平均池化层

net.avg\_pool = avg\_pool

# 加载模型参数

param\_dict = ms.load\_checkpoint(best\_ckpt\_path)

ms.load\_param\_into\_net(net, param\_dict)

model = train.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()

class\_name = {0: "dogs", 1: "wolves"}

# 预测图像类别

output = model.predict(ms.Tensor(data['image']))

pred = np.argmax(output.asnumpy(), axis=1)

# 显示图像及图像的预测值

plt.figure(figsize=(5, 5))

for i in range(4):

plt.subplot(2, 2, 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:{}'.format(class\_name[pred[i]]), color=color)

picture\_show = np.transpose(images[i], (1, 2, 0))

mean = np.array([0.485, 0.456, 0.406])

std = np.array([0.229, 0.224, 0.225])

picture\_show = std \* picture\_show + mean

picture\_show = np.clip(picture\_show, 0, 1)

plt.imshow(picture\_show)

plt.axis('off')

plt.show()

使用模型微调得到的best.ckpt文件对验证集的狼和狗图像数据进行预测。若预测字体为蓝色表示预测正确，若预测字体为红色表示预测错误。

visualize\_model(best\_ckpt\_path, dataset\_val)

结果如下：



## 网络迁移学习（固定特征进行训练）

### 模型微调

使用固定特征进行训练的时候，需要冻结除最后一层之外的所有网络层。通过设置 requires\_grad = False 冻结参数，以便不在反向传播中计算梯度。

net\_work = resnet50(pretrained=True)

# 全连接层输入层的大小

in\_channels = net\_work.fc.in\_channels

# 输出通道数大小为狼狗分类数2

head = nn.Dense(in\_channels, 2)

# 重置全连接层

net\_work.fc = head

# 平均池化层kernel size为7

avg\_pool = nn.AvgPool2d(kernel\_size=7)

# 重置平均池化层

net\_work.avg\_pool = avg\_pool

# 冻结除最后一层外的所有参数

for param in net\_work.get\_parameters():

if param.name not in ["fc.weight", "fc.bias"]:

param.requires\_grad = False

# 定义优化器和损失函数

opt = nn.Momentum(params=net\_work.trainable\_params(), learning\_rate=lr, momentum=0.5)

loss\_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

def forward\_fn(inputs, targets):

logits = net\_work(inputs)

loss = loss\_fn(logits, targets)

return loss

grad\_fn = ms.value\_and\_grad(forward\_fn, None, opt.parameters)

def train\_step(inputs, targets):

loss, grads = grad\_fn(inputs, targets)

opt(grads)

return loss

# 实例化模型

model1 = train.Model(net\_work, loss\_fn, opt, metrics={"Accuracy": train.Accuracy()})

### 训练与评估

开始训练模型，与没有预训练模型相比，将节约一大半时间，因为此时可以不用计算部分梯度。保存评估精度最高的ckpt文件于当前路径的./BestCheckpoint/resnet50-best-freezing-param.ckpt。

dataset\_train = create\_dataset\_canidae(data\_path\_train, "train")

step\_size\_train = dataset\_train.get\_dataset\_size()

dataset\_val = create\_dataset\_canidae(data\_path\_val, "val")

step\_size\_val = dataset\_val.get\_dataset\_size()

num\_epochs = 10

# 创建迭代器

data\_loader\_train = dataset\_train.create\_tuple\_iterator(num\_epochs=num\_epochs)

data\_loader\_val = dataset\_val.create\_tuple\_iterator(num\_epochs=num\_epochs)

best\_ckpt\_dir = "./BestCheckpoint"

best\_ckpt\_path = "./BestCheckpoint/resnet50-best-freezing-param.ckpt"

# 开始循环训练

print("Start Training Loop ...")

best\_acc = 0

for epoch in range(num\_epochs):

losses = []

net\_work.set\_train()

epoch\_start = time.time()

# 为每轮训练读入数据

for i, (images, labels) in enumerate(data\_loader\_train):

labels = labels.astype(ms.int32)

loss = train\_step(images, labels)

losses.append(loss)

# 每个epoch结束后，验证准确率

acc = model1.eval(dataset\_val)['Accuracy']

epoch\_end = time.time()

epoch\_seconds = (epoch\_end - epoch\_start) \* 1000

step\_seconds = epoch\_seconds/step\_size\_train

print("-" \* 20)

print("Epoch: [%3d/%3d], Average Train Loss: [%5.3f], Accuracy: [%5.3f]" % (

epoch+1, num\_epochs, sum(losses)/len(losses), acc

))

print("epoch time: %5.3f ms, per step time: %5.3f ms" % (

epoch\_seconds, step\_seconds

))

if acc > best\_acc:

best\_acc = acc

if not os.path.exists(best\_ckpt\_dir):

os.mkdir(best\_ckpt\_dir)

ms.save\_checkpoint(net\_work, best\_ckpt\_path)

print("=" \* 80)

print(f"End of validation the best Accuracy is: {best\_acc: 5.3f}, "

f"save the best ckpt file in {best\_ckpt\_path}", flush=True)

输出结果：

--------------------

Epoch: [ 1 / 10], Train Loss: [0.598], Accuracy: 0.981

epoch time: 3602.971 ms, per step time: 277.152 ms

--------------------

Epoch: [ 2 / 10], Train Loss: [0.516], Accuracy: 0.870

epoch time: 1422.890 ms, per step time: 109.453 ms

--------------------

Epoch: [ 3 / 10], Train Loss: [0.388], Accuracy: 1.000

epoch time: 2254.670 ms, per step time: 173.436 ms

--------------------

Epoch: [ 4 / 10], Train Loss: [0.362], Accuracy: 1.000

epoch time: 2181.150 ms, per step time: 167.781 ms

--------------------

Epoch: [ 5 / 10], Train Loss: [0.332], Accuracy: 1.000

epoch time: 2173.812 ms, per step time: 167.216 ms

--------------------

Epoch: [ 6 / 10], Train Loss: [0.306], Accuracy: 1.000

epoch time: 2205.381 ms, per step time: 169.645 ms

--------------------

Epoch: [ 7 / 10], Train Loss: [0.285], Accuracy: 1.000

epoch time: 2180.671 ms, per step time: 167.744 ms

--------------------

Epoch: [ 8 / 10], Train Loss: [0.244], Accuracy: 1.000

epoch time: 2166.159 ms, per step time: 166.628 ms

--------------------

Epoch: [ 9 / 10], Train Loss: [0.259], Accuracy: 1.000

epoch time: 2169.718 ms, per step time: 166.901 ms

--------------------

Epoch: [ 10 / 10], Train Loss: [0.280], Accuracy: 1.000

epoch time: 2182.844 ms, per step time: 167.911 ms

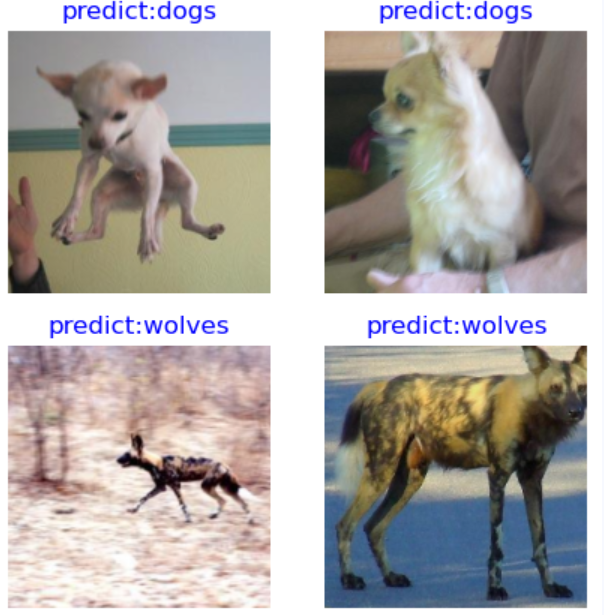
================================================================================

End of validation the best Accuracy is: 1.000, save the best ckpt file in ./best.ckpt

### 可视化模型预测

使用固定特征得到的resnet50-best-freezing-param.ckpt文件对验证集的狼和狗图像数据进行预测。若预测字体为蓝色即为预测正确，若预测字体为红色则预测错误。

visualize\_model(best\_ckpt\_path, dataset\_val)



## 实验小结

本案例通过在ModelArts平台创建拥有昇腾910 AI加速芯片的NoteBook开发环境，基于MindSpore深度学习框架并使用狗与狼分类数据集对预训练好的ResNet50模型进行微调(fine tuning)，使用微调后的模型对测试集数据进行了预测，并输出预测结果。通过这个实验，我们可以了解MindSpore框架和ModelArts平台的使用，了解如何使用迁移学习的方式来解决图像分类的具体任务。