MindSpore搭建 ResNet网络实现图像分类



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 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 MindSpore搭建ResNet网络实现图像分类 2](#_Toc150154572)

[1.1 实验介绍 2](#_Toc150154573)

[1.1.1 实验背景 2](#_Toc150154574)

[1.1.2 实验目的 2](#_Toc150154575)

[1.1.3 实验清单 3](#_Toc150154576)

[1.2 实验环境 3](#_Toc150154577)

[1.2.1 实验环境介绍 3](#_Toc150154578)

[1.3 数据与模型 3](#_Toc150154579)

[1.3.1 数据集介绍 3](#_Toc150154580)

[1.3.2 ResNet网络简介 4](#_Toc150154581)

[1.4 MindSpore训练 8](#_Toc150154582)

[1.4.1 环境准备 8](#_Toc150154583)

[1.4.2 项目介绍 8](#_Toc150154584)

[1.4.3 配置文件参数 10](#_Toc150154585)

[1.5 模型训练 11](#_Toc150154586)

[1.5.2 模型评估 16](#_Toc150154587)

[1.6 实验小结 17](#_Toc150154588)

# MindSpore搭建ResNet50网络实现图像分类

## 实验介绍

### 实验背景

图像分类是图像处理在人工智能领域的基本研究主题之一，是计算机视觉领域的核心问题。简而言之就是对于一个给定的图像，预测它属于的那个分类标签（或者给出属于一系列不同标签的可能性）。在应用领域，人脸、场景的识别等都可以归为分类任务。

这一任务是最简单、最基础的图像理解任务，也是深度学习模型最先取得突破和实现大规模应用的任务。

卷积网络模型最早的商业应用就是图像分类，89年开始LeNet系列网络在识别手写数字和手写邮编方面取得成功，应用于几乎全美的邮政系统，并于1998年诞生了我们熟知的LeNet-5，今天LeNet-5已成为理解卷积神经网络的“Hello World”，对应常用数据集MNIST。

随后80 million tiny images dataset、ImageNet Large Scale Visual Recognition Challenge等项目的开展，产生了诸如CIFAR-10、CIFAR100、ImageNet等大型基准数据集，与此同时AlexNet、ResNet、Inception、VGG等大量优秀的深度网络结构也产生，并在上述项目中获得优异的表现。

残差神经网络（ResNet）由微软研究院何凯明等五位华人提出，通过ResNet单元，成功训练152层神经网络，赢得了ILSVRC2015冠军。ResNet前五项的误差率为3.57%，参数量低于VGGNet，因此效果非常显著。传统的卷积网络或全连接网络或多或少存在信息丢失的问题，还会造成梯度消失或爆炸，导致深度网络训练失败，ResNet则在一定程度上解决了这个问题。通过将输入信息传递给输出，确保信息完整性。整个网络只需要学习输入和输出的差异部分，简化了学习目标和难度。ResNet的结构大幅提高了神经网络训练的速度，并且大大提高了模型的准确率。正因如此，ResNet十分受欢迎，甚至可以直接用于ConceptNet网络。

### 实验目的

本案例使用MindSpore框架，以CIFAR-10数据集为例，搭建ResNet网络进行计算机视觉中图像分类问题的处理。

通过本手册练习，读者能够：

理解图像分类任务

掌握如何定义和初始化基于ResNet结构的ResNet网络

掌握使用MindSpore框架进行模型构建、模型验证（验证正确率）

### 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验** | **简述** | **难度** | **软件环境** | **运行环境** |
| MindSpre搭建ResNet网络实现图像分类任务 | 本案例使用MindSpore框架，以CIFAR-10数据集为例，搭建ResNet网络进行计算机视觉中图像分类问题的处理。 | 中级 | MindSpore2.2，Python3.9 | ModelArts |

## 实验环境

### 实验环境介绍

实验平台：Modelarts

AI框架：MindSpore2.2

实验平台介绍：

华为云ModelArts平台：<https://www.huaweicloud.com/product/modelarts.html> ModelArts 是面向开发者的一站式 AI 平台，为机器学习与深度学习提供海量数据预处理及交互式智能标注、大规模分布式训练、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期 AI 工作流。

AI计算框架介绍：

MindSpore深度学习框架：<https://www.mindspore.cn/mindspore>

昇思MindSpore是一个全场景深度学习框架，旨在实现易开发、高效执行、全场景覆盖三大目标，提供支持异构加速的张量可微编程能力，支持云、服务器、边和端多种硬件平台。

## 数据与模型

### 数据集介绍

CIFAR-10数据集共有60000张32\*32的彩色照片，每个像素点包括RGB三个数值，数值范围0~255，分为10个类，分别是'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'，每类6000张图。

手机屏幕截图

描述已自动生成

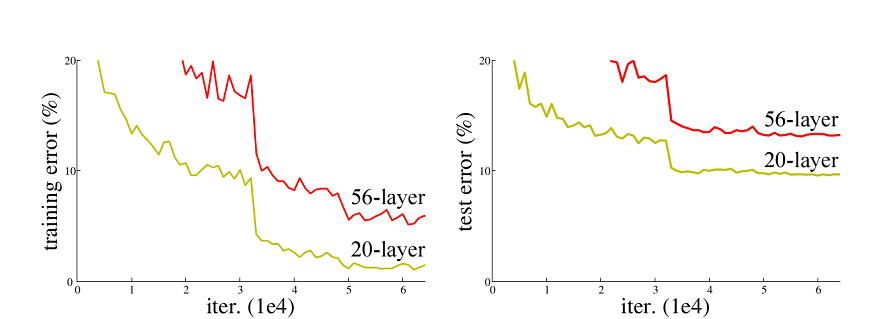
CIFAR-10数据集示意图

数据集被分为5个训练批次和一个测试批次，每个批次含10000张图片。

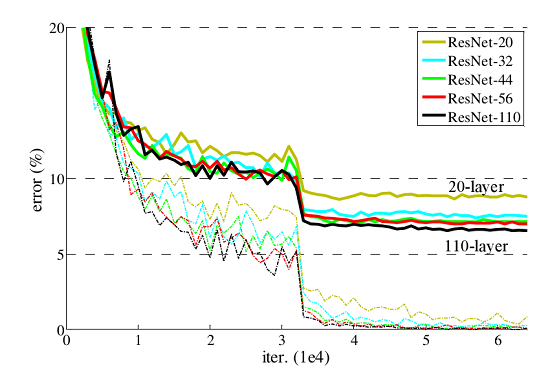
测试批次的数据取自10类中的每一类，每一类随机取1000张。抽剩下的就随机排列组成了训练批次。注意一个训练批中的各类图像并不一定数量相同。

### ResNet网络简介

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。下图是在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。



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图如下图所示，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。



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

## MindSpore训练

### 环境准备

在开始本实验前，需要完成实验环境搭建工作。

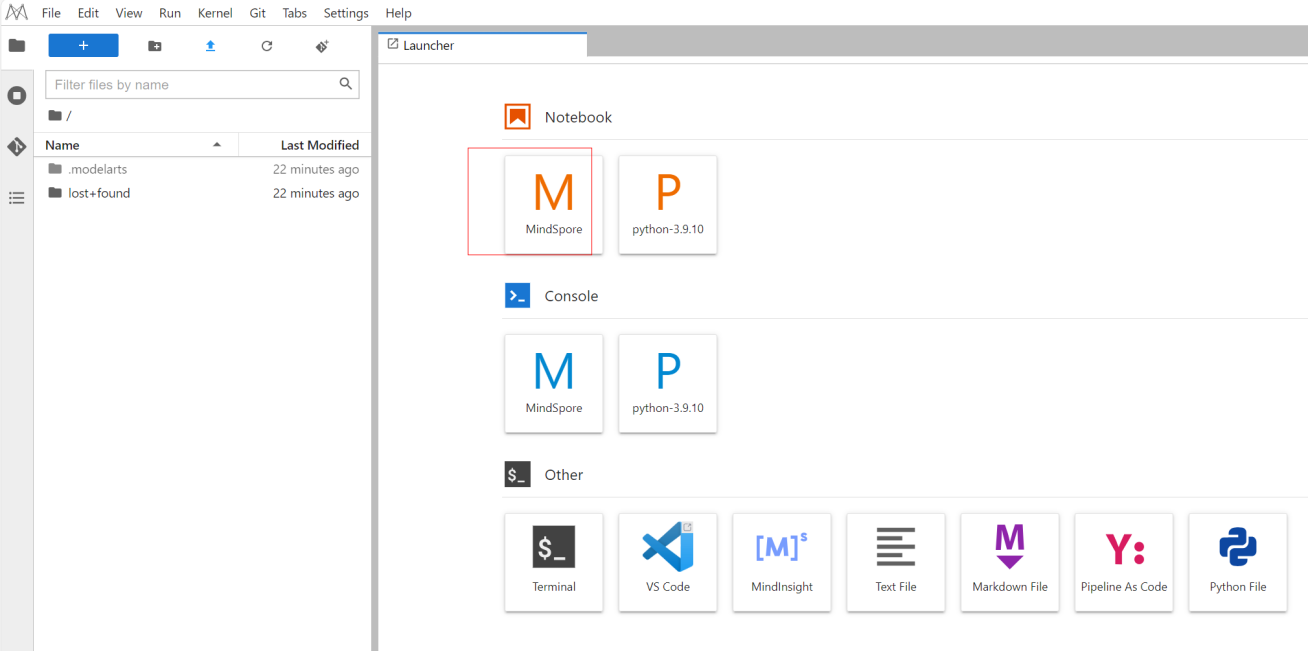
进入ModelArts开发环境

参考下方《华为云ModelArts环境搭建手册》，完成开发环境搭建。



打开Notebook

打开Notebook控制台后，新建ipynb文件，选择MindSpore环境作为Kernel，即可开始编辑实验代码。



创建ipynb文件

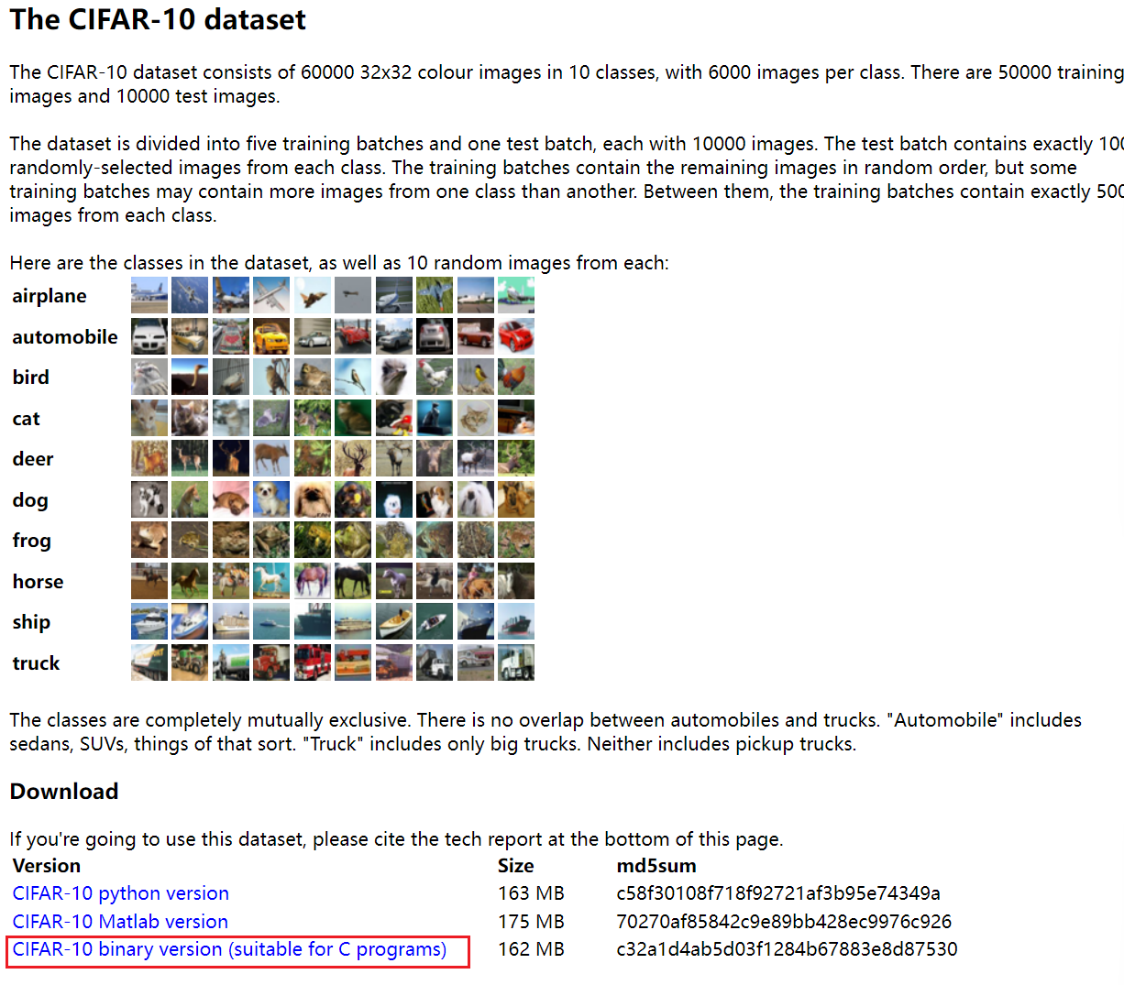
\*注意：Notebook环境内上传、创建和编辑的文件均在/home/ma-user/work目录下。

### 开始实验

#### 数据集准备与加载

数据下载

方式一：[官网](http://www.cs.toronto.edu/~kriz/cifar.html)下载，上传至云上notebook开发环境下



方式二：使用download接口下载并解压

打开Resnet文件夹，新建MindSpore 核的notebook，输入如下命令进行下载数据集

%env no\_proxy='a.test.com,127.0.0.1,2.2.2.2'

!pip install download

from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/datasets/cifar-10-binary.tar.gz"

download(url, "./datasets-cifar10-bin", kind="tar.gz", replace=True)

下载后的数据集目录结构如下：

datasets-cifar10-bin/cifar-10-batches-bin

├── batches.meta.text

├── data\_batch\_1.bin

├── data\_batch\_2.bin

├── data\_batch\_3.bin

├── data\_batch\_4.bin

├── data\_batch\_5.bin

├── readme.html

└── test\_batch.bin

数据加载

使用mindspore.dataset.Cifar10Dataset接口来加载数据集，并进行相关图像增强操作。

import mindspore as ms

import mindspore.dataset as ds

import mindspore.dataset.vision as vision

import mindspore.dataset.transforms as transforms

from mindspore import dtype as mstype

data\_dir = "./datasets-cifar10-bin/cifar-10-batches-bin" # 数据集根目录

batch\_size = 256 # 批量大小

image\_size = 32 # 训练图像空间大小

workers = 4 # 并行线程个数

num\_classes = 10 # 分类数量

def create\_dataset\_cifar10(dataset\_dir, usage, resize, batch\_size, workers):

data\_set = ds.Cifar10Dataset(dataset\_dir=dataset\_dir,

usage=usage,

num\_parallel\_workers=workers,

shuffle=True)

trans = []

if usage == "train":

trans += [

vision.RandomCrop((32, 32), (4, 4, 4, 4)),

vision.RandomHorizontalFlip(prob=0.5)

]

trans += [

vision.Resize(resize),

vision.Rescale(1.0 / 255.0, 0.0),

vision.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010]),

vision.HWC2CHW()

]

target\_trans = transforms.TypeCast(mstype.int32)

# 数据映射操作

data\_set = data\_set.map(operations=trans,

input\_columns='image',

num\_parallel\_workers=workers)

data\_set = data\_set.map(operations=target\_trans,

input\_columns='label',

num\_parallel\_workers=workers)

# 批量操作

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

return data\_set

# 获取处理后的训练与测试数据集

dataset\_train = create\_dataset\_cifar10(dataset\_dir=data\_dir,

usage="train",

resize=image\_size,

batch\_size=batch\_size,

workers=workers)

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

dataset\_val = create\_dataset\_cifar10(dataset\_dir=data\_dir,

usage="test",

resize=image\_size,

batch\_size=batch\_size,

workers=workers)

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

数据可视化

对CIFAR-10训练数据集进行可视化

import matplotlib.pyplot as plt

import numpy as np

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

images = data\_iter["image"].asnumpy()

labels = data\_iter["label"].asnumpy()

print(f"Image shape: {images.shape}, Label shape: {labels.shape}")

# 训练数据集中，前六张图片所对应的标签

print(f"Labels: {labels[:6]}")

classes = []

with open(data\_dir + "/batches.meta.txt", "r") as f:

for line in f:

line = line.rstrip()

if line:

classes.append(line)

# 训练数据集的前六张图片

plt.figure()

for i in range(6):

plt.subplot(2, 3, i + 1)

image\_trans = 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])

image\_trans = std \* image\_trans + mean

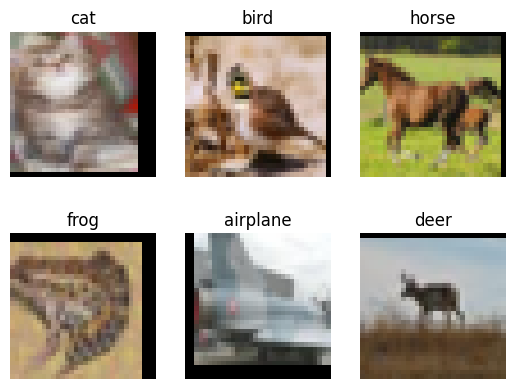
image\_trans = np.clip(image\_trans, 0, 1)

plt.title(f"{classes[labels[i]]}")

plt.imshow(image\_trans)

plt.axis("off")

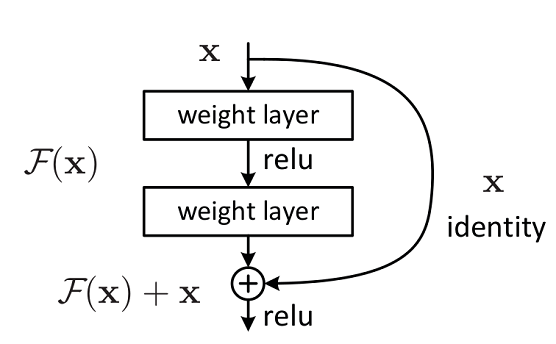
plt.show()



#### 构建网络

残差网络结构(Residual Network)是ResNet网络的主要亮点，ResNet使用残差网络结构后可有效地减轻退化问题，实现更深的网络结构设计，提高网络的训练精度。本节首先讲述如何构建残差网络结构，然后通过堆叠残差网络来构建ResNet50网络。

##### 构建残差网络结构

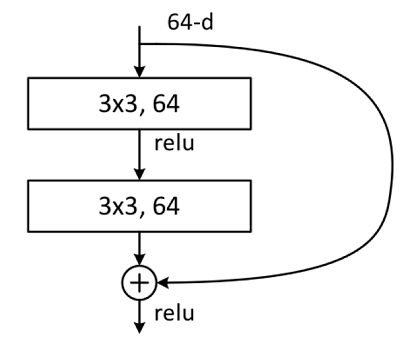
残差网络结构图如下图所示，残差网络由两个分支构成：一个主分支，一个shortcuts（图中弧线表示）。主分支通过堆叠一系列的卷积操作得到，shotcuts从输入直接到输出，主分支输出的特征矩阵加上shortcuts输出的特征矩阵，通过Relu激活函数后即为残差网络最后的输出。

残差网络结构主要由两种，一种是Building Block，适用于较浅的ResNet网络，如ResNet18和ResNet34；另一种是Bottleneck，适用于层数较深的ResNet网络，如ResNet50、ResNet101和ResNet152。

Building Block

Building Block结构图如下图所示，主分支有两层卷积网络结构：

主分支第一层网络以输入channel为64为例，首先通过一个3\*3的卷积层，然后通过Batch Normalization层，最后通过Relu激活函数层，输出channel为64；主分支第二层网络的输入channel为64，首先通过一个3\*3的卷积层，然后通过Batch Normalization层，输出channel为64。最后将主分支输出的特征矩阵与shortcuts输出的特征矩阵相加，通过Relu激活函数即为Building Block最后的输出。



主分支与shortcuts输出的特征矩阵相加时，需要保证主分支与shortcuts输出的特征矩阵shape相同。如果主分支与shortcuts输出的特征矩阵shape不相同，如输出channel是输入channel的一倍时，shortcuts上需要使用数量与输出channel相等，大小为1\*1的卷积核进行卷积操作；若输出的图像较输入图像缩小一倍，则要设置shortcuts中卷积操作中的stride为2，主分支第一层卷积操作的stride也需设置为2。

构建Building Block结构

如下代码定义ResidualBlockBase类实现Building Block结构。

from typing import Type, Union, List, Optional

import mindspore.nn as nn

from mindspore.common.initializer import Normal

# 初始化卷积层与BatchNorm的参数

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)

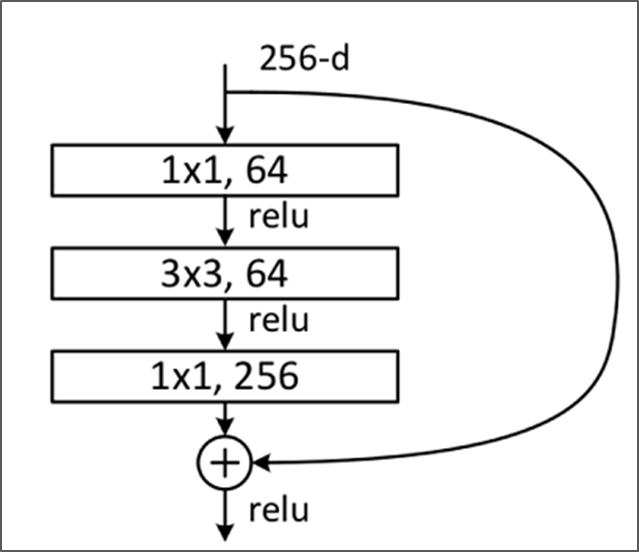
return out

Bottleneck

Bottleneck结构图如下图所示，在输入相同的情况下Bottleneck结构相对Building Block结构的参数数量更少，更适合层数较深的网络，ResNet50使用的残差结构就是Bottleneck。该结构的主分支有三层卷积结构，分别为1\*1的卷积层、3\*3卷积层和1\*1的卷积层，其中1\*1的卷积层分别起降维和升维的作用。

主分支第一层网络以输入channel为256为例，首先通过数量为64，大小为1\*1的卷积核进行降维，然后通过Batch Normalization层，最后通过Relu激活函数层，其输出channel为64；

主分支第二层网络通过数量为64，大小为3\*3的卷积核提取特征，然后通过Batch Normalization层，最后通过Relu激活函数层，其输出channel为64；主分支第三层通过数量为256，大小1\*1的卷积核进行升维，然后通过Batch Normalization层，其输出channel为256。最后将主分支输出的特征矩阵与shortcuts输出的特征矩阵相加，通过Relu激活函数即为Bottleneck最后的输出。



主分支与shortcuts输出的特征矩阵相加时，需要保证主分支与shortcuts输出的特征矩阵shape相同。如果主分支与shortcuts输出的特征矩阵shape不相同，如输出channel是输入channel的一倍时，shortcuts上需要使用数量与输出channel相等，大小为1\*1的卷积核进行卷积操作；若输出的图像较输入图像缩小一倍，则要设置shortcuts中卷积操作中的stride为2，主分支第二层卷积操作的stride也需设置为2。

Bottleneck结构实现

如下代码定义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, 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)

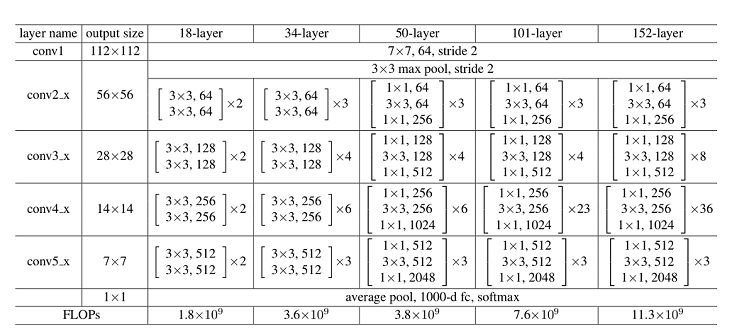
out += identity # 输出为主分支与shortcuts之和

out = self.relu(out)

return out

##### 构建ResNet50网络

ResNet网络层结构如下图所示，以输入彩色图像224\*224为例，首先通过数量64，卷积核大小为7\*7，stride为2的卷积层conv1，该层输出图片大小为112\*112，输出channel为64；然后通过一个3\*3的最大下采样池化层，该层输出图片大小为56\*56，输出channel为64；再堆叠4个残差网络块（conv2\_x、conv3\_x、conv4\_x和conv5\_x），此时输出图片大小为7\*7，输出channel为2048；最后通过一个平均池化层、全连接层和softmax，得到分类概率。



构建残差块

对于每个残差网络块，以ResNet50网络中的conv2\_x为例，其由3个Bottleneck结构堆叠而成，每个Bottleneck输入的channel为64，输出channel为256。

def make\_layer(last\_out\_channel, block: Type[Union[ResidualBlockBase, ResidualBlock]],

channel: int, block\_nums: int, stride: int = 1):

‘’’

last\_out\_channel：上一个残差网络输出的通道数。

block：残差网络的类别，分别为ResidualBlockBase和ResidualBlock。

channel：残差网络输入的通道数。

block\_nums：残差网络块堆叠的个数。

stride：卷积移动的步幅。

‘’’

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)

构建Resnet50模型

ResNet50网络共有5个卷积结构，一个平均池化层，一个全连接层，以CIFAR-10数据集为例：

conv1：输入图片大小为32\*32，输入channel为3。首先经过一个卷积核数量为64，卷积核大小为7\*7，stride为2的卷积层；然后通过一个Batch Normalization层；最后通过Reul激活函数。该层输出feature map大小为16\*16，输出channel为64。

conv2\_x：输入feature map大小为16\*16，输入channel为64。首先经过一个卷积核大小为3\*3，stride为2的最大下采样池化操作；然后堆叠3个[1\*1，64；3\*3，64；1\*1，256]结构的Bottleneck。该层输出feature map大小为8\*8，输出channel为256。

conv3\_x：输入feature map大小为8\*8，输入channel为256。该层堆叠4个[1\*1，128；3\*3，128；1\*1，512]结构的Bottleneck。该层输出feature map大小为8\*8，输出channel为512。

conv4\_x：输入feature map大小为4\*4，输入channel为512。该层堆叠6个[1\*1，256；3\*3，256；1\*1，1024]结构的Bottleneck。该层输出feature map大小为2\*2，输出channel为1024。

conv5\_x：输入feature map大小为2\*2，输入channel为1024。该层堆叠3个[1\*1，512；3\*3，512；1\*1，2048]结构的Bottleneck。该层输出feature map大小为1\*1，输出channel为2048。

average pool & fc：输入channel为2048，输出channel为分类的类别数。

如下示例代码实现ResNet50模型的构建，通过用调函数resnet50即可构建ResNet50模型。

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

加载预训练网络

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

layers: List[int], num\_classes: int, pretrained: bool, pretrained\_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=pretrained\_ckpt, replace=True)

param\_dict = load\_checkpoint(pretrained\_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)

##### 模型训练与评估

本节使用ResNet50预训练模型进行微调。调用resnet50构造ResNet50模型，并设置pretrained参数为True，将会自动下载ResNet50预训练模型，并加载预训练模型中的参数到网络中。然后定义优化器和损失函数，逐个epoch打印训练的损失值和评估精度，并保存评估精度最高的ckpt文件（resnet50-best.ckpt）到当前路径的./BestCheckPoint下。

由于预训练模型全连接层（fc）的输出大小（对应参数num\_classes）为1000， 为了成功加载预训练权重，我们将模型的全连接输出大小设置为默认的1000。CIFAR10数据集共有10个分类，在使用该数据集进行训练时，需要将加载好预训练权重的模型全连接层输出大小重置为10。

实例化ResNet50

# 定义ResNet50网络

network = resnet50(pretrained=True)

# 全连接层输入层的大小

in\_channel = network.fc.in\_channels

fc = nn.Dense(in\_channels=in\_channel, out\_channels=10)

# 重置全连接层

network.fc = fc

设置学习率、损失函数、优化器

# 设置学习率

num\_epochs = 5

lr = nn.cosine\_decay\_lr(min\_lr=0.00001, max\_lr=0.001, total\_step=step\_size\_train \* num\_epochs,

step\_per\_epoch=step\_size\_train, 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\_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

##MindSpore2.0 混合FP编程

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

设置模型存储路径

import os

# 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)

# 最佳模型存储路径

best\_acc = 0

best\_ckpt\_dir = "./BestCheckpoint"

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

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

os.mkdir(best\_ckpt\_dir)

定义模型训练、验证

import mindspore.ops as ops

def train(data\_loader, epoch):

"""模型训练"""

losses = []

network.set\_train(True)#设置成True训练

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

loss = train\_step(images, labels)

if i % 100 == 0 or i == step\_size\_train - 1:

print('Epoch: [%3d/%3d], Steps: [%3d/%3d], Train Loss: [%5.3f]' %

(epoch + 1, num\_epochs, i + 1, step\_size\_train, loss))

losses.append(loss)

return sum(losses) / len(losses)

def evaluate(data\_loader):

"""模型验证"""

network.set\_train(False)#模型推理

correct\_num = 0.0 # 预测正确个数

total\_num = 0.0 # 预测总数

for images, labels in data\_loader:

logits = network(images)

pred = logits.argmax(axis=1) # 预测结果

correct = ops.equal(pred, labels).reshape((-1, ))

correct\_num += correct.sum().asnumpy()

total\_num += correct.shape[0]

acc = correct\_num / total\_num # 准确率

return acc

开始训练

# 开始循环训练

print("Start Training Loop ...")

for epoch in range(num\_epochs):

curr\_loss = train(data\_loader\_train, epoch)

curr\_acc = evaluate(data\_loader\_val)

print("-" \* 50)

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

epoch+1, num\_epochs, curr\_loss, curr\_acc

))

print("-" \* 50)

# 保存当前预测准确率最高的模型

if curr\_acc > best\_acc:

best\_acc = curr\_acc

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)

训练日志输出：

Start Training Loop ...

Epoch: [ 1/ 5], Steps: [ 1/196], Train Loss: [2.389]

Epoch: [ 1/ 5], Steps: [101/196], Train Loss: [1.467]

Epoch: [ 1/ 5], Steps: [196/196], Train Loss: [1.093]

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

Epoch: [ 1/ 5], Average Train Loss: [1.641], Accuracy: [0.595]

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

Epoch: [ 2/ 5], Steps: [ 1/196], Train Loss: [1.253]

Epoch: [ 2/ 5], Steps: [101/196], Train Loss: [0.974]

Epoch: [ 2/ 5], Steps: [196/196], Train Loss: [0.832]

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

Epoch: [ 2/ 5], Average Train Loss: [1.019], Accuracy: [0.685]

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

Epoch: [ 3/ 5], Steps: [ 1/196], Train Loss: [0.917]

Epoch: [ 3/ 5], Steps: [101/196], Train Loss: [0.879]

Epoch: [ 3/ 5], Steps: [196/196], Train Loss: [0.743]

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

Epoch: [ 3/ 5], Average Train Loss: [0.852], Accuracy: [0.721]

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

Epoch: [ 4/ 5], Steps: [ 1/196], Train Loss: [0.911]

Epoch: [ 4/ 5], Steps: [101/196], Train Loss: [0.703]

Epoch: [ 4/ 5], Steps: [196/196], Train Loss: [0.768]

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

Epoch: [ 4/ 5], Average Train Loss: [0.777], Accuracy: [0.737]

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

Epoch: [ 5/ 5], Steps: [ 1/196], Train Loss: [0.793]

Epoch: [ 5/ 5], Steps: [101/196], Train Loss: [0.809]

Epoch: [ 5/ 5], Steps: [196/196], Train Loss: [0.734]

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

Epoch: [ 5/ 5], Average Train Loss: [0.745], Accuracy: [0.742]

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

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

End of validation the best Accuracy is: 0.742, save the best ckpt file in ./BestCheckpoint/resnet50-best.ckpt

备注：此处我们展示了5个epochs的训练过程，如果想要达到理想的训练效果，建议训练80个epochs。

可视化模型预测

定义visualize\_model函数，使用上述验证精度最高的模型对CIFAR-10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。

import matplotlib.pyplot as plt

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

num\_class = 10 # 对狼和狗图像进行二分类

net = resnet50(num\_class)

# 加载模型参数

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

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

# 加载验证集的数据进行验证

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

images = data["image"]

labels = data["label"]

# 预测图像类别

output = net(data['image'])

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

# 图像分类

classes = []

with open(data\_dir + "/batches.meta.txt", "r") as f:

for line in f:

line = line.rstrip()

if line:

classes.append(line)

# 显示图像及图像的预测值

plt.figure()

for i in range(6):

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.asnumpy()[i] else 'red'

plt.title('predict:{}'.format(classes[pred[i]]), color=color)

picture\_show = np.transpose(images.asnumpy()[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\_path=best\_ckpt\_path, dataset\_val=dataset\_val)



## 实验小结

本案例使用MindSpore框架进行图像分类处理问题，理解了如何通过定义和初始化ResNet网络进行训练模型及模型验证（验证正确率）。