



**机器学习课程设计**

**MindSpore下基于GooleNet网络的物体识别**

**学 院 计算机学院**

**专 业 人工智能**

**学 号 3121005358**

**学生姓名 欧炜标**

**编 号**

**2023 年 5 月**

目录

[摘要 5](#_Toc136552407)

[1 实验介绍 5](#_Toc136552408)

[1.1 实验背景 5](#_Toc136552409)

[1.2 实验目的 6](#_Toc136552410)

[1.3 实验清单 6](#_Toc136552411)

[1.4 实验环境 6](#_Toc136552412)

[1.4.1 环境介绍 6](#_Toc136552413)

[1.5 数据与模型 8](#_Toc136552414)

[1.5.1 数据集介绍 8](#_Toc136552415)

[1.5.2 模型介绍 9](#_Toc136552416)

[2 具体实验操作 10](#_Toc136552417)

[2.1 下载项目和数据集 10](#_Toc136552418)

[2.2 数据集预处理 11](#_Toc136552419)

[2.3 设置超参数 14](#_Toc136552420)

[2.4 模型训练 16](#_Toc136552421)

[2.4.1 GoogLeNet网络 16](#_Toc136552422)

[2.4.2 损失函数 17](#_Toc136552423)

[2.4.3 优化器 18](#_Toc136552424)

[2.4.4 模型编译 18](#_Toc136552425)

[2.4.5 回调函数 19](#_Toc136552426)

[2.4.6 执行训练 20](#_Toc136552427)

[2.5 模型验证 21](#_Toc136552428)

[2.6 模型保存 22](#_Toc136552429)

[2.7 实验总结 23](#_Toc136552430)

[参考文献 24](#_Toc136552431)

摘要

本实验旨在使用MindSpore深度学习框架实现基于GoogLeNet网络的物体识别任务。物体识别是计算机视觉领域的关键任务之一，它可以帮助计算机系统自动识别和分类图像中的物体。

本次我们选择了GoogLeNet作为基础网络架构，GoogLeNet是一种经典的深度卷积神经网络模型，具有较强的图像识别能力和参数效率。我们将使用MindSpore框架来构建、训练和评估这个网络模型。

首先，我们将介绍MindSpore框架的基本概念和特性，包括张量操作、模型定义和训练过程等。然后，我们将详细讲解GoogLeNet网络的结构和原理，并在MindSpore中实现该网络模型。

接下来，我们将使用公开的数据集（例如ImageNet）进行物体识别的实验。我们将介绍数据预处理步骤，包括图像加载、数据增强和批处理等。然后，我们将使用MindSpore提供的优化器和损失函数来训练GoogLeNet网络模型，并监控训练过程中的指标和准确率。

最后，我们将评估训练好的模型在测试集上的性能，并展示物体识别的结果。我们将比较不同参数设置和优化技巧对模型性能的影响，并探讨如何进一步改进模型的准确率和效率。

通过完成本课程设计，学生将获得以下能力和收获：熟悉MindSpore深度学习框架的使用，了解GoogLeNet网络的原理和结构，掌握物体识别任务的实现步骤，以及对模型性能进行评估和优化的能力。

# 实验介绍

## 实验背景

计算机视觉，是计算机领域的重要研究对象，目前计算机视觉已经深度融合了人工智能的技术，尤其深度学习，在计算机视觉中起到了至关重要的作用。本章主要围绕计算机视觉当中的物体识别模块进行介绍。

本实验使用昇思MindSpore深度学习框架搭建GoogleNet网络，使用CIFAR数据集，利用华为云上昇腾910资源进行训练，得到物体识别模型；通过本实验可以体验GoogleNet网络在物体识别的应用。

## 实验目的

本实验使用MindSpore深度学习框架构建GoogleNet网络，进行模型训练。

掌握GoogleNet网络模型结构。

掌握MindSpore模型训练的流程。

## 实验清单

|  |  |  |  |
| --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 开发环境 |
| GoogleNet网络实现物体识别实验 | 本实验用CIFAR10数据集，实现MindSpore在物体识别应用 | 中级 | ModelArts：MindSpore1.7，Python3.7，  Ascend 910 +ARM |

## 实验环境

### 环境介绍

MindSpore训练：在华为云ModelArts上的开发环境中执行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验环境 | 实验平台 | AI 计算框架 | AI处理器/算力 | 软件 |
| MindSpore训练 | 华为云 ModelArts | MindSpore1.7 | Ascend 910 | Notebook环境，  Python3.7.5，MindSpore1.7 |

环境要求

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。

NPU芯片介绍：

Ascend 910：<https://e.huawei.com/cn/products/cloud-computing-dc/atlas/ascend-910>

昇腾910是一款具有超高算力的AI处理器，其最大功耗为310W，华为自研的达芬奇架构大大提升了其能效比。八位整数精度（INT8）下的性能达到640TOPS，16位浮点数（FP16）下的性能达到320 TFLOPS。

Ascend 310：<https://e.huawei.com/cn/products/cloud-computing-dc/atlas/ascend-310>

昇腾310是一款高效、灵活、可编程的AI处理器。基于典型配置，八位整数精度（INT8）下的性能达到22TOPS，16位浮点数（FP16）下的性能达到11 TFLOPS，而其功耗仅为8W。昇腾310芯片采用华为自研的达芬奇架构，集成了丰富的计算单元，在各个领域得到广泛应用。随着全AI业务流程的加速，昇腾310芯片能够使智能系统的性能大幅提升，部署成本大幅降低。

其他资源介绍：

CANN（异构计算架构）：<https://www.hiascend.com/zh/software/cann>

AI场景的异构计算架构，通过提供多层次的编程接口，支持用户快速构建基于昇腾平台的AI应用和业务。

npu-driver（固件与驱动）：<https://www.hiascend.com/hardware/firmware-drivers?tag=community>

使用华为提供的固件包、驱动包完成昇腾设备的基础环境部署。

MobaXterm：<https://mobaxterm.mobatek.net/>

MobaXterm是一款功能强大的远程连接工具。

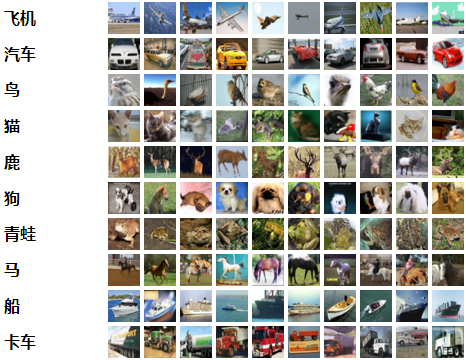
## 数据与模型

### 数据集介绍

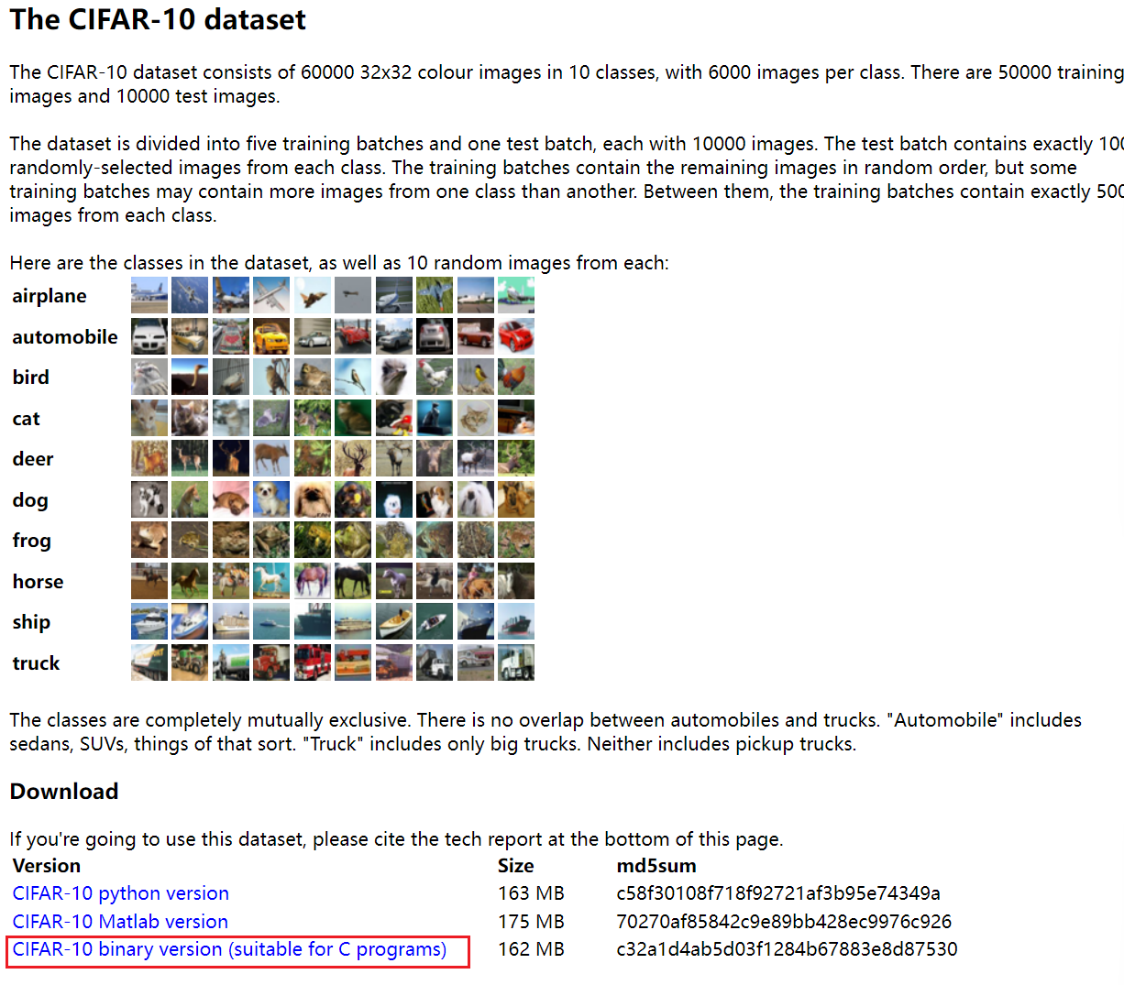
CIFAR-10数据集

CIFAR-10数据集由10个类的60000个32x32彩色图像组成，每个类有6000个图像。有50000个训练图像和10000个测试图像。数据集分为五个训练批次和一个测试批次，每个批次有10000个图像。测试批次包含来自每个类别的恰好1000个随机选择的图像。训练批次以随机的顺序输入图像，但一些训练批次可能包含来自一个类别的图像比另一个更多。总体来说，五个训练集之和包含来自每个类的正好5000张图像。

10个类完全相互排斥，且类之间没有重叠，汽车和卡车之间没有重叠。“汽车”包括轿车，SUV等。“卡车”只包括大卡车，不包括皮卡车。  
以下是10个类别类的名字：airplane/automobile/bird/cat/deer/dog/frog/horse/ship/truck



Cifar10数据示例



CIFAR数据目录结构,如下：

./cifar-10-batches-bin

├── batches.meta.txt

├── data\_batch\_1.bin

├── data\_batch\_2.bin

├── data\_batch\_3.bin

├── data\_batch\_4.bin

├── data\_batch\_5.bin

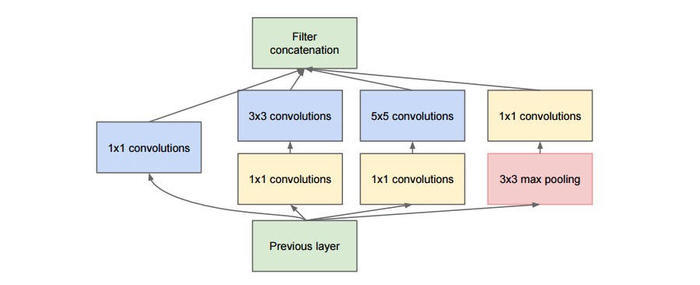
├── readme.html

└── test\_batch.bin

### 模型介绍

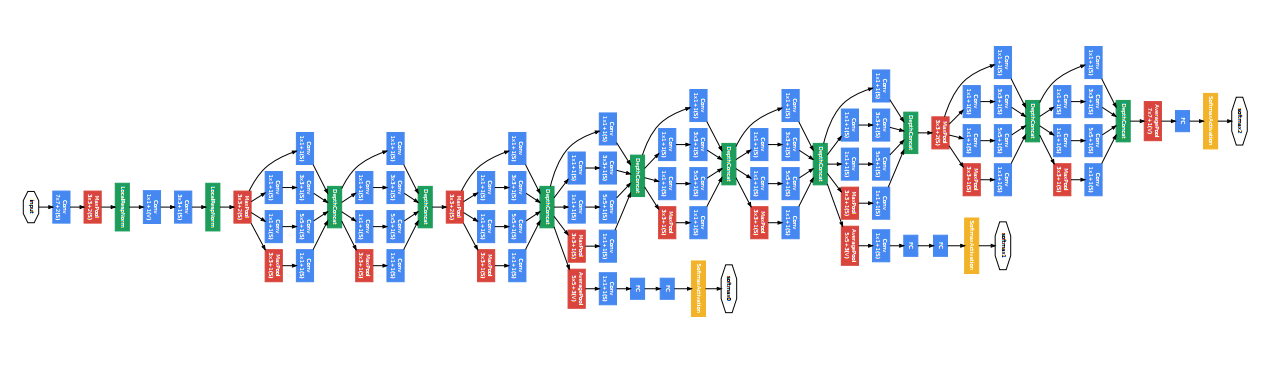
GoogleNet是2014年提出的22层深度网络，在2014年ImageNet大型视觉识别挑战赛（ILSVRC14）中获得第一名。 GoogleNet，也称Inception v1，比ZFNet（2013年获奖者）和AlexNet（2012年获奖者）改进明显，与VGGNet相比，错误率相对较低。

深度学习网络包含的参数更多，更容易过拟合。网络规模变大也会增加使用计算资源。为了解决这些问题，GoogleNet采用1\*1卷积核来降维，从而进一步减少计算量。在网络末端使用全局平均池化，而不是使用全连接的层。

Inception就是把多个卷积或池化操作，放在一起组装成一个网络模块，设计神经网络时以模块为单位去组装整个网络结构。inception结构的主要贡献有两个：一是使用1x1的卷积来进行升降维；二是在多个尺寸上同时进行卷积再聚合。

GoogleNet网络结构图

GoogleNet由多个inception模块串联起来，可以更加深入。 降维的inception模块一般包括1×1卷积、3×3卷积、5×5卷积和3×3最大池化，同时完成前一次的输入，并在输出处再次堆叠在一起。



GoogleNet整体结构[1]

# 具体实验操作

## 下载项目和数据集

下载项目代码

使用git从modelzoo下载训练脚本的源码。

!git clone -b r1.5 <https://gitee.com/mindspore/models.git>

项目文件结构（本实验所用代码，均以加粗显示）

├── googlenet

├── README.md // googlenet相关说明

├── ascend310\_infer // 实现310推理源代码

├── scripts

│ ├──run\_train.sh // 分布式到Ascend的shell脚本

│ ├──run\_train\_gpu.sh // 分布式到GPU处理器的shell脚本

│ ├──run\_train\_cpu.sh // CPU处理器训练的shell脚本

│ ├──run\_eval.sh // Ascend评估的shell脚本

│ ├──run\_infer\_310.sh // Ascend推理shell脚本

│ ├──run\_eval\_gpu.sh // GPU处理器评估的shell脚本

│ ├──run\_eval\_cpu.sh // CPU处理器评估的shell脚本

├── src

│ ├──**dataset.py**  // 创建数据集

│ ├─**─googlenet.py** // googlenet架构

│ ├─**─…..**

├── **train.py**  // 训练脚本

├── **eval.py** // 评估脚本

├── **postprocess.py**  // 推理后处理脚本

├── **preprocess.py**  // 预处理脚本

├── **cifar10\_config.yaml**  // 配置文件

├── **export.py**  // 将checkpoint文件导出到air/mindir

…..

下载数据集

!wget http://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz

!mkdir -p data/cifar-10-verify-bin

!tar -xvzf cifar-10-binary.tar.gz -C ./data/

!cp -r ./data/cifar-10-batches-bin/test\_batch.bin ./data/cifar-10-verify-bin/

## 数据集预处理

代码在googlenet/src/dataset.py，进行图像数据集的加载、通道的转换、旋转、缩放、随机裁剪等操作。

import os

import mindspore.common.dtype as mstype

import mindspore.dataset as ds

import mindspore.dataset.transforms.c\_transforms as C

import mindspore.dataset.vision.c\_transforms as vision

def create\_dataset\_cifar10(data\_home, repeat\_num=1, training=True, cifar\_cfg=None):

"""Data operations."""

data\_dir = os.path.join(data\_home, "cifar-10-batches-bin")

if not training:

data\_dir = os.path.join(data\_home, "cifar-10-verify-bin")

if training:

rank\_size, rank\_id = \_get\_rank\_info()

**data\_set = ds.Cifar10Dataset(data\_dir, num\_shards=rank\_size, shard\_id=rank\_id, shuffle=True)**

else:

data\_set = ds.Cifar10Dataset(data\_dir, shuffle=False)

resize\_height = cifar\_cfg.image\_height

resize\_width = cifar\_cfg.image\_width

# define map operations

random\_crop\_op = vision.RandomCrop((32, 32), (4, 4, 4, 4)) # padding\_mode default CONSTANT

random\_horizontal\_op = vision.RandomHorizontalFlip()

resize\_op = vision.Resize((resize\_height, resize\_width)) # interpolation default BILINEAR

rescale\_op = vision.Rescale(1.0 / 255.0, 0.0)

normalize\_op = vision.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))

changeswap\_op = vision.HWC2CHW()

type\_cast\_op = C.TypeCast(mstype.int32)

c\_trans = []

if training:

c\_trans = [random\_crop\_op, random\_horizontal\_op]

c\_trans += [resize\_op, rescale\_op, normalize\_op, changeswap\_op]

# apply map operations on images

data\_set = data\_set.map(operations=type\_cast\_op, input\_columns="label")

data\_set = data\_set.map(operations=c\_trans, input\_columns="image")

# apply batch operations

data\_set = data\_set.batch(batch\_size=cifar\_cfg.batch\_size, drop\_remainder=True)

# apply repeat operations

data\_set = data\_set.repeat(repeat\_num)

return data\_set

Cifar10Dataset()主要参数：

dataset\_dir (str): 包含数据集文件的根目录路径。

num\_parallel\_workers (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用mindspore.dataset.config中配置的线程数。

shuffle (bool, 可选) - 是否混洗数据集。默认值：None

sampler (Sampler, 可选) - 指定从数据集中选取样本的采样器，默认值：None

num\_shards (int, 可选) - 指定分布式训练时将数据集进行划分的分片数，默认值：None。指定此参数后， num\_samples 表示每个分片的最大样本数。

shard\_id (int, 可选) - 指定分布式训练时使用的分片ID号，默认值：None。只有当指定了 num\_shards 时才能指定此参数。

|  |  |
| --- | --- |
| API接口 | 功能 |
| mindspore.dtype | 数据 type形态转变、权重初始化等的常规工具。 |
| mindspore.dataset | 数据集的载入与处理。 |
| mindspore.dataset.Cifar10Dataset | 读取和解析CIFAR-10数据集的源文件构建数据集。 |
| mindspore.dataset.vision | 提供了c\_transforms模块和py\_transforms模块供用户进行多种数据增强操作, c\_transforms：基于C++的OpenCV实现，提供了多种图像增强功能，具有较高的性能；py\_transforms：基于Python的PIL实现，提供了多种图像增强功能，并提供了PIL Image和NumPy数组之间的传输方法。 |
| [mindspore.dataset.vision.RandomCrop](https://www.mindspore.cn/docs/zh-CN/master/api_python/dataset_vision/mindspore.dataset.vision.RandomCrop.html#mindspore.dataset.vision.RandomCrop) | 对输入图像进行随机区域的裁剪。 |
| [mindspore.dataset.vision.RandomHorizontalFlip](https://www.mindspore.cn/docs/zh-CN/master/api_python/dataset_vision/mindspore.dataset.vision.RandomHorizontalFlip.html#mindspore.dataset.vision.RandomHorizontalFlip) | 对输入图像按给定的概率进行水平随机翻转。 |
| [mindspore.dataset.vision.Rescale](https://www.mindspore.cn/docs/zh-CN/master/api_python/dataset_vision/mindspore.dataset.vision.Rescale.html#mindspore.dataset.vision.Rescale) | 基于给定的缩放和平移因子调整图像的像素大小。 |
| [mindspore.dataset.vision.Normalize](https://www.mindspore.cn/docs/zh-CN/master/api_python/dataset_vision/mindspore.dataset.vision.Normalize.html#mindspore.dataset.vision.Normalize) | 根据均值和标准差对输入图像进行归一化。 |
| [mindspore.dataset.vision.HWC2CHW](https://www.mindspore.cn/docs/zh-CN/r1.8/api_python/dataset_vision/mindspore.dataset.vision.HWC2CHW.html#mindspore.dataset.vision.HWC2CHW) | 将输入图像的shape从 <H, W, C> 转换为 <C, H, W>。 |
| [mindspore.dataset.transforms.TypeCast](https://www.mindspore.cn/docs/zh-CN/master/api_python/dataset_transforms/mindspore.dataset.transforms.TypeCast.html#mindspore.dataset.transforms.TypeCast) | 将输入的Tensor转换为指定的数据类型。 |
| mindspore.dataset.transforms | 用于通用数据增强，包括 c\_transforms 和 py\_transforms 两个子模块。c\_transforms 是一个高性能数据增强模块，基于C++的OpenCV实现。py\_transforms 提供了一种基于Python和NumPy的实现方式。 |
| mindspore.dataset.map() | 给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。每个数据增强操作将数据集对象中的一个或多个数据列作为输入，将数据增强的结果输出为一个或多个数据列。 |

## 设置超参数

超参数文件为cifar10\_config.yaml，可以同时配置训练参数和评估参数。

超参是可以调整的参数，可以控制模型训练优化的过程，不同的超参数值可能会影响模型训练和收敛速度。目前深度学习模型多采用批量随机梯度下降算法进行优化，随机梯度下降算法的原理如下：

是批量大小(batch size)，是学习率(learning rate)；另外，为训练轮次中权重参数，为损失函数的导数。可知道除了梯度本身，这两个因子直接决定了模型的权重更新，从优化本身来看它们是影响模型性能收敛最重要的参数。

一般会定义以下用于训练的超参：

|  |  |  |
| --- | --- | --- |
| 超参数名称 | 超参数释义 | 本实验超参数值 |
| 训练轮次（epoch） | 训练时遍历数据集的次数。 | 本实验训练轮数设置epoch\_size：125#（大概用时50min，可以自行设置，建议设置20） |
| 批次大小（batch size） | 数据集进行分批读取训练，设定每个批次数据的大小。batch size过小，花费时间多，同时梯度震荡严重，不利于收敛；batch size过大，不同batch的梯度方向没有任何变化，容易陷入局部极小值，因此需要选择合适的batch size，可以有效提高模型精度、全局收敛。 | 本实验批次大小设置batch\_size: 128，特别注意的：导出模型时，需要设置batch\_size 为1。 |
| 学习率（learning rate） | 如果学习率偏小，会导致收敛的速度变慢，如果学习率偏大则可能会导致训练不收敛等不可预测的结果。梯度下降法是一个广泛被用来最小化模型误差的参数优化算法。梯度下降法通过多次迭代，并在每一步中最小化损失函数来估计模型的参数。学习率在迭代过程中，会控制模型的学习进度。 | 本实验中学习率初始值lr\_init: 0.1。 |
| 分类类别数 | 物体识别实验中，目标物体的类别数。 | num\_classes: 10 # 分类类别数 |
| 图片宽、高 | 经过图像预处理后，输入神经网络的图片大小。 | image\_height: 224 # 图片高  image\_width: 224 # 图片宽 |

修改cifar\_config.yaml超参数

打开/home/ma-user/work/models/official/cv/googlenet/cifar\_config.yaml文件，修改以下超参数：

train\_data\_path: "/home/ma-user/work/data/" # 训练集读取路径

val\_data\_path: "/home/ma-user/work/data/" # 验证集读取路径

checkpoint\_path: "/home/ma-user/work/models/official/cv/googlenet/ckpt/train\_googlenet\_cifar10-20\_468.ckpt" #做模型验证时，读取ckpt的路径，与ckpt\_save\_dir保持一致

ckpt\_save\_dir: " /home/ma-user/work/models/official/cv/googlenet/ckpt/" # 模型保存路径

## 模型训练

### GoogLeNet网络

完整实验代码：googlenet/src/googlenet.py。

神经网络模型由多个数据操作层组成，mindspore.nn提供了各种网络基础模块。

MindSpore的Cell类是构建所有网络的基类，也是网络的基本单元。当用户需要神经网络时，需要继承Cell类，并重写\_\_init\_\_方法和construct方法。

import numpy as np

import mindspore

import mindspore.nn as nn

from mindspore import Tensor

class GoogleNet(nn.Cell):

"""

Googlenet architecture

"""

def \_\_init\_\_(self, num\_classes, include\_top=True):

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

self.conv1 = Conv2dBlock(3, 64, kernel\_size=7, stride=2, padding=0)

self.maxpool1 = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode="same")

self.conv2 = Conv2dBlock(64, 64, kernel\_size=1)

self.conv3 = Conv2dBlock(64, 192, kernel\_size=3, padding=0)

self.maxpool2 = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode="same")

self.block3a = Inception(192, 64, 96, 128, 16, 32, 32)

self.block3b = Inception(256, 128, 128, 192, 32, 96, 64)

self.maxpool3 = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode="same")

self.block4a = Inception(480, 192, 96, 208, 16, 48, 64)

self.block4b = Inception(512, 160, 112, 224, 24, 64, 64)

self.block4c = Inception(512, 128, 128, 256, 24, 64, 64)

self.block4d = Inception(512, 112, 144, 288, 32, 64, 64)

self.block4e = Inception(528, 256, 160, 320, 32, 128, 128)

self.maxpool4 = nn.MaxPool2d(kernel\_size=2, stride=2, pad\_mode="same")

self.block5a = Inception(832, 256, 160, 320, 32, 128, 128)

self.block5b = Inception(832, 384, 192, 384, 48, 128, 128)

self.dropout = nn.Dropout(keep\_prob=0.8)

self.include\_top = include\_top

if self.include\_top:

self.mean = P.ReduceMean(keep\_dims=True)

self.flatten = nn.Flatten()

self.classifier = nn.Dense(1024, num\_classes, weight\_init=weight\_variable(),

bias\_init=weight\_variable())

def construct(self, x):

"""construct"""

x = self.conv1(x)

x = self.maxpool1(x)

x = self.conv2(x)

x = self.conv3(x)

x = self.maxpool2(x)

x = self.block3a(x)

x = self.block3b(x)

x = self.maxpool3(x)

x = self.block4a(x)

x = self.block4b(x)

x = self.block4c(x)

x = self.block4d(x)

x = self.block4e(x)

x = self.maxpool4(x)

x = self.block5a(x)

x = self.block5b(x)

if not self.include\_top:

return x

x = self.mean(x, (2, 3))

x = self.flatten(x)

x = self.classifier(x)

return x

### 损失函数

完整实验代码： googlenet/train.py。

损失函数用来评价模型的预测值和真实值不一样的程度。

mindspore.nn.loss也提供了许多常用的损失函数，如SoftmaxCrossEntropyWithLogits、MSELoss、SmoothL1Loss等。

本实验采用的是交叉熵损失（SoftmaxCrossEntropyWithLogits），即把网络输出层的值经过softmax函数之后计算真实值与预测值之间的交叉熵损失

#mindspore.nn.SoftmaxCrossEntropyWithLogits(sparse=False, reduction='none')

主要参数解释：

sparse (bool) - 指定目标值是否使用稀疏格式。默认值：False。

reduction (str) - 指定应用于输出结果的计算方式。取值为”mean”，”sum”，或”none”。取值为”none”，则不执行reduction。默认值：”none”。

### 优化器

完整实验代码： googlenet/train.py。

优化器函数用于计算和更新梯度，模型优化算法的选择直接关系到最终模型的性能。有时候最终模型效果不好，未必是特征或者模型设计的问题，很有可能是优化算法的问题。优化器内部定义了模型的参数优化过程（即梯度如何更新至模型参数），MindSpore所有优化逻辑都封装在Optimizer对象中。本实验，我们使用Momentum优化器。mindspore.nn也提供了许多其他常用的优化器，如Adam、SGD、RMSProp等。

我们需要构建一个Optimizer对象，这个对象能够保持当前参数状态并基于计算得到的梯度进行参数更新。为了构建一个Optimizer，我们需要给它一个包含可优化的参数（必须是Variable对象）的迭代器，如网络中所有可以训练的parameter，将params设置为net.trainable\_params()即可。然后，你可以设置Optimizer的参数选项，比如学习率、权重衰减等等。

“””

From mindspore import nn

network = Net()

opt = nn.Momentum(network.trainable\_params(), lrs, config.momentum, config.weight\_decay, loss\_scale=LOSS\_SCALE)

“””

主要参数解释：

momentum (float) - 浮点数类型的超参，表示移动平均的动量。必须等于或大于0.0。

weight\_decay (Union[float, int, Cell]) - 权重衰减（L2 penalty）。默认值：0.0。

float: 固定值，必须大于或者等于0。int: 固定值，必须大于或者等于0，会被转换成float。

loss\_scale (float) - 梯度缩放系数，必须大于0。如果 loss\_scale 是整数，它将被转换为浮点数。

### 模型编译

完整实验代码： googlenet/train.py。

[MindSpore.Model](https://www.mindspore.cn/docs/zh-CN/master/api_python/train/mindspore.train.Model.html#mindspore.train.Model) 模型训练或推理的高阶接口。 Model 会根据用户传入的参数封装可训练或推理的实例。

from mindspore import nn, Tensor, Model

from mindspore import dtype as mstype

from mindspore.train.callback import LossMonitor

# 输入训练轮次和数据集进行训练

**model = Model(net, loss\_fn=loss, optimizer=optim)**

**model.train(epoch=epochs, train\_dataset=dataset, callbacks=cb)**

主要参数说明：

net-用于训练或推理的神经网络。

loss\_fn - 损失函数。如果 loss\_fn 为None，network 中需要进行损失函数计算，必要时也需要进行并行计算。默认值：None。

optimizer - 用于更新网络权重的优化器。如果 optimizer 为None， network 中需要进行反向传播和网络权重更新。默认值：None。

### 回调函数

完整实验代码： googlenet/train.py。

在模型训练过程中，可以添加检查点（Checkpoint,ckpt）用于保存模型的参数，以便进行推理及中断后再训练使用。

方式一：直接保存模型

这种方式的优点是接口简单易用，但是只保留执行命令时候的网络模型状态；使用MindSpore提供的save\_checkpoint保存模型，传入网络和保存路径。

net = Net()

save\_checkpoint(net, "googlenet.ckpt")

方式二：训练过程中保存模型

在网络模型训练中进行保存，MindSpore在网络模型训练的过程中，自动保存训练时候设定好的epoch数和step数的参数，也就是把模型训练过程中产生的中间权重参数也保存下来，方便进行网络微调和停止训练。

本实验采用该方式进行保存模型文件。使用model.train里面的callbacks参数传入保存模型的对象 ModelCheckpoint，可以保存模型参数，生成CheckPoint(简称ckpt)文件。

“”

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

config\_ck = CheckpointConfig(save\_checkpoint\_steps=32, keep\_checkpoint\_max=10) #  首先需要初始化一个CheckpointConfig类对象，用来设置保存策略。

ckpt\_cb = ModelCheckpoint(prefix='googlenet', directory=None, config=config\_ck)# 创建一个ModelCheckpoint对象把它传递给model.train方法，就可以在训练过程中使用CheckPoint功能了。

model.train(epoch\_num, dataset, callbacks=ckpoint\_cb)

“””

参数解释：

save\_checkpoint\_steps表示每隔多少个step保存一次。

keep\_checkpoint\_max表示最多保留CheckPoint文件的数量。

prefix表示生成CheckPoint文件的前缀名。

directory表示存放文件的目录。

生成的CheckPoint文件如下：

XXnet-graph.meta # 编译后的计算图

XXnet-1\_32.ckpt # CheckPoint文件后缀名为'.ckpt'

XXnet-2\_32.ckpt # 文件的命名方式表示保存参数所在的epoch和step数

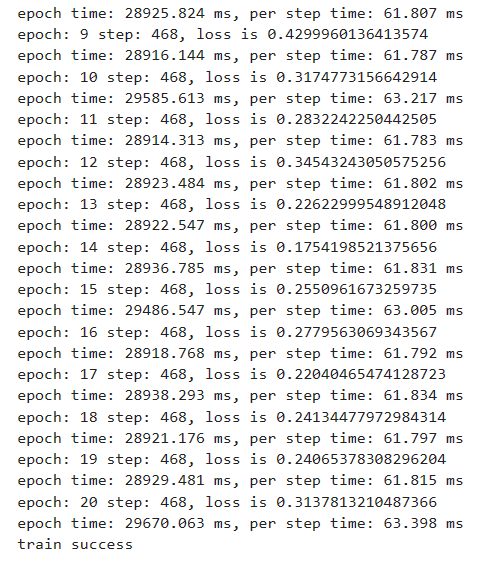
XXnet-3\_32.ckpt # 表示保存的是第3个epoch的第32个step的模型参数

### 执行训练

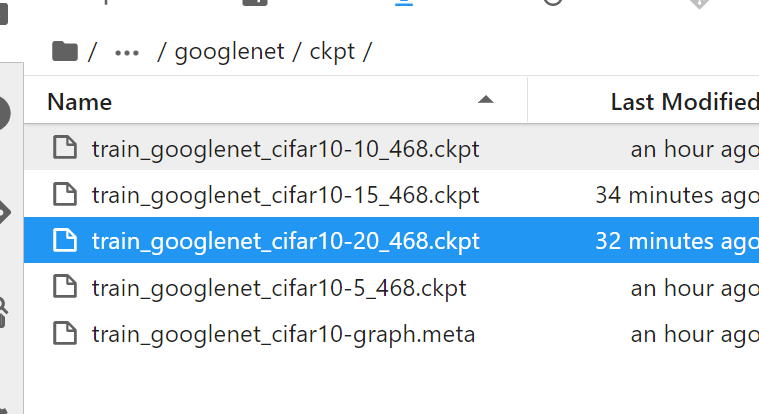
输入下方命令，启动模型训练

!python /home/ma-user/work/models/official/cv/googlenet/train.py

训练日志：



训练完成后，在ckpt文件夹下生成模型文件。



## 模型验证

完整实验代码： googlenet/eval.py。

model.eval接口进行推理验证，先要加载模型权重，创建相同模型的实例，然后使用load\_checkpoint和load\_param\_into\_net方法加载参数。

load\_checkpoint方法会把参数文件中的网络参数加载到字典param\_dict中。

load\_param\_into\_net方法会把字典param\_dict中的参数加载到网络或者优化器中，加载后，网络中的参数就是CheckPoint保存的

dataset = create\_dataset\_cifar10(config.val\_data\_path, 1, False, cifar\_cfg=config)

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

net = GoogleNet(num\_classes=config.num\_classes)

opt = Momentum(filter(lambda x: x.requires\_grad, net.get\_parameters()), 0.01, config.momentum,

weight\_decay=config.weight\_decay)

model = Model(net, loss\_fn=loss, optimizer=opt, metrics={'acc'})

param\_dict = load\_checkpoint(config.checkpoint\_path)

print("load checkpoint from [{}].".format(config.checkpoint\_path))

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

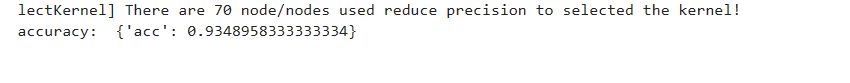
acc = model.eval(dataset)

print("accuracy: ", acc)

输入下方命令，启动模型评估

!python /home/ma-user/work/models/official/cv/googlenet/eval.py

推理日志：



可见，该模型整体的精确度很高，超过了90%。

## 模型保存

再次修改 cifar10\_config.yaml 超参数

再次打开/home/ma-user/work/models/official/cv/googlenet/cifar\_config.yaml文件，修改以下超参数：

ckpt\_file: "/home/ma-user/work/models/official/cv/googlenet/ckpt/train\_googlenet\_cifar10-20\_468.ckpt" # 模型读取路径

file\_format: "AIR" # 模型保存格式

batch\_size: 1 # 取消注释，推理图片输入批次为1。

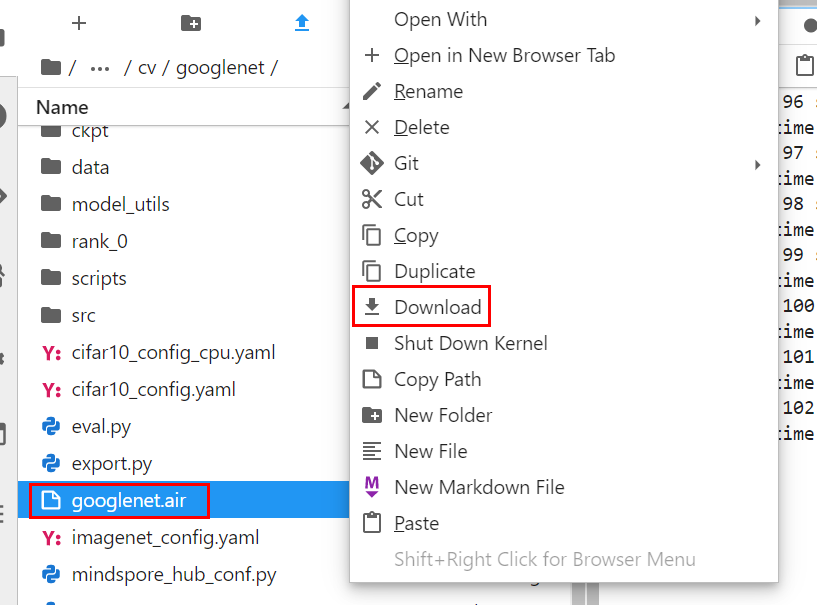


输入下方命令，导出模型

!python /home/ma-user/work/models/official/cv/googlenet/export.py

成功后，在文件夹下生成air模型文件。

下载模型文件，进行后续推理。



模型保存

# 实验总结

在本次课程设计中，通过MindSpore深度学习框架实现了基于GoogLeNet网络的物体识别任务。以下是对实验的总结：

1. 实验目标： 通过训练GoogLeNet网络，实现对图像中物体的识别和分类。
2. 实验步骤：
   * 数据准备：使用了公开的数据集（如CIFAR-10）作为训练和测试数据。数据集经过预处理，包括图像加载、数据增强和批处理等。
   * 网络构建：在MindSpore框架中实现了GoogLeNet网络模型，并配置了合适的超参数和损失函数。
   * 训练过程：使用训练数据对GoogLeNet模型进行了训练，并通过调整学习率和优化器等参数来优化模型的性能。
   * 模型评估：在训练完成后，使用测试数据对训练好的模型进行了评估，计算了准确率和其他指标。
3. 实验结果： 在本次实验中，可以观察到GoogLeNet模型在物体识别任务上表现出色。通过训练和优化，获得了具有较高准确率的模型。
4. 实验总结： 本次课程设计中，我熟悉了MindSpore框架的使用，并实践了基于GoogLeNet网络的物体识别任务。了解了数据准备、网络构建、训练和评估的基本步骤，以及调整超参数和优化模型的方法。通过这个实验，不仅提高了对深度学习和计算机视觉的理解，还培养了实际问题解决和代码实现的能力。

总而言之，本次实验为我们提供了一个完整的物体识别任务实践，展示了MindSpore框架和GoogLeNet网络的应用。通过这个实验，我能够进一步探索和扩展深度学习在计算机视觉领域的应用。

# 参考文献

1. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich."Going deeper with convolutions."Proceedings of the IEEE conference on computer vision and pattern recognition.2015.