昇腾创新实践课

CIFAR-10图像分类实验手册



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 实验环境介绍 2](#_Toc112870215)

[1.1 实验介绍 2](#_Toc112870216)

[1.1.1 关于本实验 2](#_Toc112870217)

[1.1.2 实验环境介绍 2](#_Toc112870218)

[2 CIFAR-10图像分类实验 3](#_Toc112870219)

[2.1 实验介绍 3](#_Toc112870220)

[2.1.1 关于本实验 3](#_Toc112870221)

[2.1.2 实验目的 4](#_Toc112870222)

[2.1.3 背景知识 4](#_Toc112870223)

[2.1.4 实验设计 4](#_Toc112870224)

[2.2 实验过程 4](#_Toc112870225)

[2.2.1 环境和数据准备 4](#_Toc112870226)

[2.2.2 数据展示 7](#_Toc112870227)

[2.2.3 数据处理 8](#_Toc112870228)

[2.2.4 网络定义 10](#_Toc112870229)

[2.2.5 模型训练 11](#_Toc112870230)

[2.2.6 模型评估 15](#_Toc112870231)

[2.2.7 模型优化 16](#_Toc112870232)

[2.2.8 重新训练与评估 18](#_Toc112870233)

[2.2.9 效果展示 20](#_Toc112870234)

[2.3 实验总结 21](#_Toc112870235)

[2.4 思考题 22](#_Toc112870236)

[2.5 挑战 22](#_Toc112870237)

# 实验环境介绍

## 实验介绍

### 关于本实验

本实验使用MindSpore深度学习框架，演示一个完整的图像分类模型开发流程。

### 实验环境介绍

实验、介绍、难度、软件环境、硬件环境：

实验环境介绍

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 实验介绍 | 难度 | 软件环境 | 开发环境 |
| CIFAR-10图像分类实验 | 基于MindSpore的进阶操作，使用CIFAR-10数据集演示一个完整的图像分类模型开发流程； | 普通 | Python3.7 MindSpore 1.7  Numpy 1.17.5  matplotlib 3.3.4 | PC机 |

# CIFAR-10图像分类实验

## 实验介绍

### 关于本实验

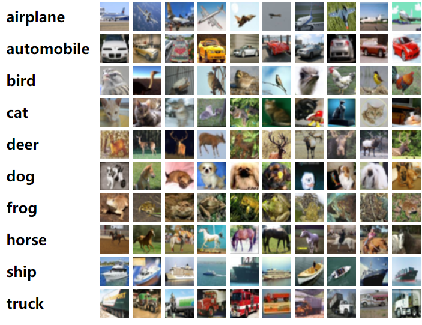
本实验使用MindSpore深度学习框架，演示一个完整的图像分类模型开发流程。

数据集介绍：

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。

CIFAR-10数据集官网：<https://www.cs.toronto.edu/~kriz/cifar.html>



CIFAR-10 dataset

### 实验目的

理解MindSpore开发全部流程。

理解MindSpore常用模块的功能。

掌握MindSpore的进阶操作。

掌握卷积神经网络的搭建。

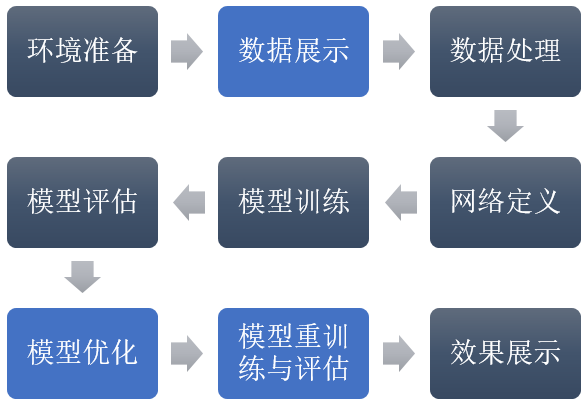
掌握模型调优。

### 背景知识

神经网络知识，MindSpore基础知识，MindSpore进阶知识，图像数据预处理，LeNet5卷积神经网络，模型调优。

### 实验设计

新增数据展示、模型优化、模型重训练预评估：



实验步骤

## 实验过程

### 环境和数据准备

获取数据集

从官网（<https://www.cs.toronto.edu/~kriz/cifar.html>）下载数据：

图形用户界面, 表格

描述已自动生成

下载后解压至工作环境目录；

由于训练和测试数据放在一起，因此需另建一个文件夹如“cifar-10-verify-bin”，将原始解压缩后“cifar-10-batches-bin”文件夹下的“test\_batch.bin”文件转移放入。

注意：

请确保存储路径只有英文，否则可能报错；

第二次运行请先清空保存模型的results文件夹，否则可能报错。

安装MindSpore环境

请参考《MindSpore环境搭建手册》或官网（<https://www.mindspore.cn/install>）在PC本地搭建MindSpore r1.7环境。

MindSpore模块主要用于本次实验卷积神经网络的构建，包括很多子模块。

mindspore.dataset：包括CIFAR-10数据集的载入与处理，也可以自定义数据集。

mindspore.common：包中会有诸如type形态转变、权重初始化等的常规工具。

mindspore.nn：主要包括网络可能涉及到的各类网络层，诸如卷积层、池化层、全连接层，也包括损失函数，激活函数等。

mindspore.train.callback：涉及到各类回调函数，如checkpoint，lossMonitor等，也可以自定义Callback。

context：设定mindspore的运行环境与运行设备。

Model：承载网络结构，并能够调用优化器、损失函数、评价指标。

save\_checkpoint, load\_checkpoint：保存与读取最佳网络参数。

Python环境导入

代码：

import mindspore

# mindspore.dataset

import mindspore.dataset as ds # 数据集的载入

import mindspore.dataset.transforms.c\_transforms as C # 常用转化算子

import mindspore.dataset.vision.c\_transforms as CV # 图像转化算子

# mindspore.common

from mindspore.common import dtype as mstype # 数据形态转换

from mindspore.common.initializer import Normal # 参数初始化

# mindspore.nn

import mindspore.nn as nn # 各类网络层都在nn里面

from mindspore.nn.metrics import Accuracy, Loss # 测试模型用

# mindspore.train.callback

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor, Callback # 回调函数

from mindspore import Model # 承载网络结构

from mindspore import save\_checkpoint, load\_checkpoint # 保存与读取最佳参数

from mindspore import context # 设置mindspore运行的环境

import numpy as np # numpy

import matplotlib.pyplot as plt # 可视化用

import copy # 保存网络参数用

# 数据路径处理

import os, stat

MindSpore 环境设置

MindSpore支持两种运行模式，在调试或者运行方面做了不同的优化：

PYNATIVE模式：也称动态图模式，将神经网络中的各个算子逐一下发执行，方便用户编写和调试神经网络模型。

GRAPH模式：也称静态图模式或者图模式，将神经网络模型编译成一整张图，然后下发执行。该模式利用图优化等技术提高运行性能，同时有助于规模部署和跨平台运行。

代码：

device\_target = context.get\_context('device\_target')

# 获取运行装置（CPU，GPU，Ascend）

dataset\_sink\_mode = True if device\_target in ['Ascend','GPU'] else False

# 是否将数据通过pipeline下发到装置上

context.set\_context(mode = context.GRAPH\_MODE, device\_target = device\_target)

# 设置运行环境，静态图context.GRAPH\_MODE指向静态图模型，即在运行之前会把全部图建立编译完毕

print(f'device\_target: {device\_target}')

print(f'dataset\_sink\_mode: {dataset\_sink\_mode}')

输出：

device\_target: CPU

dataset\_sink\_mode: False

### 数据展示

查看数据集

代码：

# 数据路径

train\_path = os.path.join('cifar-10-batches-bin') # 训练集路径

test\_path = os.path.join('cifar-10-verify-bin') # 测试集路径

print(f'训练集路径：{train\_path}')

print(f'测试集路径：{test\_path}')

输出：

训练集路径：cifar-10-batches-bin

测试集路径：cifar-10-verify-bin

展示数据集

代码：

# 创建图像标签列表

category\_dict = {0:'airplane',1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',

6:'frog',7:'horse',8:'ship',9:'truck'}

# 载入展示用数据

demo\_data = ds.Cifar10Dataset(test\_path)

# 设置图像大小

plt.figure(figsize=(6, 6))

# 打印9张子图

i = 1

for dic in demo\_data.create\_dict\_iterator():

plt.subplot(3,3,i)

plt.imshow(dic['image'].asnumpy()) # asnumpy：将 MindSpore tensor 转换成 numpy

plt.axis('off')

plt.title(category\_dict[dic['label'].asnumpy().item()])

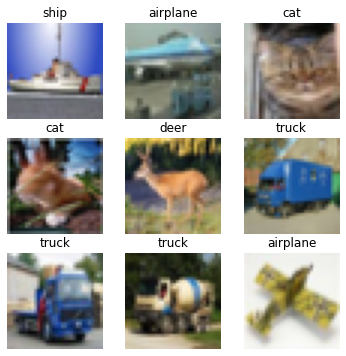
i +=1

if i > 9 :

break

plt.show()

输出：



### 数据处理

计算数据集平均数和标准差

计算数据集平均数和标准差，数据标准化时使用

ds\_train = ds.Cifar10Dataset(train\_path)

#计算数据集平均数和标准差，数据标准化时使用

tmp = np.asarray( [x['image'] for x in ds\_train.create\_dict\_iterator(output\_numpy=True)] )

RGB\_mean = tuple(np.mean(tmp, axis=(0, 1, 2)))

RGB\_std = tuple(np.std(tmp, axis=(0, 1, 2)))

print(RGB\_mean)

print(RGB\_std)

输出：

(125.306918046875, 122.950394140625, 113.86538318359375)

(62.99321927774456, 62.08870764035727, 66.70489964064619)

定义数据预处理函数。

函数功能包括：

1. 加载数据集

2. 打乱数据集

3. 图像特征处理（包括尺寸大小变更、平移、标准化、训练时的随机裁剪、随机翻转等）

4. 批量输出数据

5. 重复

代码：

def create\_dataset(data\_path, batch\_size = 32, repeat\_num=1, usage = 'train'):

"""

数据处理

Args:

data\_path (str): 数据路径

batch\_size (int): 批量大小

usage (str): 训练或测试

Returns:

Dataset对象

"""

# 载入数据集

data = ds.Cifar10Dataset(data\_path)

# 打乱数据集

data = data.shuffle(buffer\_size=10000)

# 定义算子

if usage=='train':

trans = [

CV.Normalize(RGB\_mean, RGB\_std), # 数据标准化

# 数据增强

CV.RandomCrop([32, 32], [4, 4, 4, 4]), # 随机裁剪

CV.RandomHorizontalFlip(), # 随机翻转

CV.HWC2CHW() # 通道前移（为配适网络，CHW的格式可最佳发挥昇腾芯片算力）

]

else:

trans = [

CV.Normalize(RGB\_mean, RGB\_std), # 数据标准化

CV.HWC2CHW() # 通道前移（为配适网络，CHW的格式可最佳发挥昇腾芯片算力）

]

typecast\_op = C.TypeCast(mstype.int32) # 原始数据的标签是unint，计算损失需要int

# 算子运算

data = data.map(input\_columns='label', operations=typecast\_op)

data = data.map(input\_columns='image', operations=trans)

# 批处理

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

# 重复

data = data.repeat(repeat\_num)

return data

### 网络定义

使用“MNIST手写体识别实验”中介绍的网络，LeNet5。

代码：

class LeNet5(nn.Cell):

"""

LeNet5网络

Args:

num\_class (int): 输出分类数

num\_channel (int): 输入通道数

Returns:

Tensor, 输出张量

Examples:

>>> LeNet5(10, 3)

"""

# 定义算子

def \_\_init\_\_(self, num\_class=10, num\_channel=3):

super(LeNet5, self).\_\_init\_\_()

# 卷积层

self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode='valid')

self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode='valid')

# 全连接层

self.fc1 = nn.Dense(16 \* 5 \* 5, 120, weight\_init=Normal(0.02))

self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))

self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))

# 激活函数

self.relu = nn.ReLU()

# 最大池化成

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

# 网络展开

self.flatten = nn.Flatten()

# 建构网络

def construct(self, x):

x = self.conv1(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv2(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

x = self.relu(x)

x = self.fc3(x)

return x

### 模型训练

载入数据集

代码：

train\_data = create\_dataset(train\_path, batch\_size = 32, usage = 'train') # 训练数据集

test\_data = create\_dataset(test\_path, batch\_size = 50, usage= 'test') # 测试数据集

构建网络

构建网络、损失函数、优化器、模型。

这里损失函数采用交叉熵损失函数：<https://www.mindspore.cn/docs/zh-CN/r1.7/api_python/nn/mindspore.nn.SoftmaxCrossEntropyWithLogits.html>

主要参数解释：

sparse (bool) - 指定目标值是否使用稀疏格式。当标签数据不是one-hot编码形式时，需要输入参数sparse为True。

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。为mean，返回loss平均值结果

这里优化器采用Momentum：<https://www.mindspore.cn/docs/zh-CN/r1.7/api_python/nn/mindspore.nn.Momentum.html>

代码：

# 网络

network1 = LeNet5(10)

# 损失函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 优化器

net\_opt = nn.Momentum(params=network1.trainable\_params(), learning\_rate=0.01, momentum=0.9)

# 模型

model = Model(network = network1, loss\_fn=net\_loss, optimizer=net\_opt, metrics={'accuracy': Accuracy(), 'loss':Loss()})

记录模型每个epoch的loss

定义记录loss的Callback，每个epoch结束时记录。

Callback介绍：<https://www.mindspore.cn/docs/zh-CN/r1.7/api_python/mindspore.train.html#mindspore.train.callback.Callback>

自定义Callback教程：[https://www.mindspore.cn/tutorials/zh-CN/r1.7/advanced/train/callback.html](https://www.mindspore.cn/tutorials/zh-CN/r1.7/advanced/train/callback.html#%E8%87%AA%E5%AE%9A%E4%B9%89%E5%9B%9E%E8%B0%83%E6%9C%BA%E5%88%B6)

代码：

# 记录模型每个epoch的loss

class TrainHistroy(Callback):

"""

记录模型训练时每个epoch的loss的回调函数

Args:

history (list): 传入list以保存模型每个epoch的loss

"""

def \_\_init\_\_(self, history):

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

self.history = history

# 每个epoch结束时执行

def epoch\_end(self, run\_context):

cb\_params = run\_context.original\_args()

loss = cb\_params.net\_outputs.asnumpy()

self.history.append(loss)

# 测试并记录模型在测试集的loss和accuracy，每个epoch结束时进行模型测试并记录结果，跟踪并保存准确率最高的模型网络参数

class EvalHistory(Callback):

"""

记录模型训练时每个epoch在测试集的loss和accuracy的回调函数，并保存准确率最高的模型网络参数

Args:

model (Cell): 模型，评估loss和accuracy用

loss\_history (list): 传入list以保存模型每个epoch在测试集的loss

acc\_history (list): 传入list以保存模型每个epoch在测试集的accuracy

eval\_data (Dataset): 测试集，评估模型loss和accuracy用

"""

#保存accuracy最高的网络参数

best\_param = None

def \_\_init\_\_(self, model, loss\_history, acc\_history, eval\_data):

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

self.loss\_history = loss\_history

self.acc\_history = acc\_history

self.eval\_data = eval\_data

self.model = model

# 每个epoch结束时执行

def epoch\_end(self, run\_context):

cb\_params = run\_context.original\_args()

res = self.model.eval(self.eval\_data, dataset\_sink\_mode=False)

if len(self.acc\_history)==0 or res['accuracy']>=max(self.acc\_history):

self.best\_param = copy.deepcopy(cb\_params.network)

self.loss\_history.append(res['loss'])

self.acc\_history.append(res['accuracy'])

print('acc\_eval: ',res['accuracy'])

# 训练结束后执行

def end(self, run\_context):

# 保存最优网络参数

best\_param\_path = os.path.join(ckpt\_path, 'best\_param.ckpt')

if os.path.exists(best\_param\_path):

# best\_param.ckpt已存在时MindSpore会覆盖旧的文件，这里修改文件读写权限防止报错

os.chmod(best\_param\_path, stat.S\_IWRITE)

save\_checkpoint(self.best\_param, best\_param\_path)

设置回调函数（Callback）

代码：

ckpt\_path = os.path.join('.','results') # 网络参数保存路径

hist = {'loss':[], 'loss\_eval':[], 'acc\_eval':[]} # 训练过程记录

# 网络参数自动保存，这里设定每2000个step保存一次，最多保存10次

config\_ck = CheckpointConfig(save\_checkpoint\_steps=2000,

keep\_checkpoint\_max=10)

ckpoint\_cb = ModelCheckpoint(prefix='checkpoint\_lenet', directory=ckpt\_path, config=config\_ck)

# 监控每次迭代的时间

time\_cb = TimeMonitor(data\_size=ds\_train.get\_dataset\_size())

# 监控loss值

loss\_cb = LossMonitor(per\_print\_times=500)

# 记录每次迭代的模型损失值

train\_hist\_cb = TrainHistroy(hist['loss'])

# 测试并记录模型在验证集的loss和accuracy，并保存最优网络参数

eval\_hist\_cb = EvalHistory(model = model,

loss\_history = hist['loss\_eval'],

acc\_history = hist['acc\_eval'],

eval\_data = test\_data)

训练模型

代码：

epoch = 10 # 迭代次数

# 开始训练

model.train(epoch, train\_data, callbacks=[train\_hist\_cb, eval\_hist\_cb, time\_cb, ckpoint\_cb, loss\_cb], dataset\_sink\_mode=dataset\_sink\_mode)

输出：

epoch: 1 step: 500, loss is 2.2974353

epoch: 1 step: 1000, loss is 2.1961524

epoch: 1 step: 1500, loss is 2.032713

acc\_eval: 0.3245

epoch time: 26303.570 ms, per step time: 16.840 ms

epoch: 2 step: 438, loss is 1.8902785

epoch: 2 step: 938, loss is 1.8267795

epoch: 2 step: 1438, loss is 1.8329716

acc\_eval: 0.4489

…

epoch time: 28064.980 ms, per step time: 17.967 ms

epoch: 10 step: 442, loss is 1.5684246

epoch: 10 step: 942, loss is 1.1709045

epoch: 10 step: 1442, loss is 1.0168015

acc\_eval: 0.5905

epoch time: 28536.139 ms, per step time: 18.269 ms

### 模型评估

查看模型在测试集的准确率。

【注意】：测试集不会进行随机裁剪与翻转。

观察模型loss的变化

代码：

# 定义loss记录绘制函数

def plot\_loss(hist):

plt.plot(hist['loss'], marker='.')

plt.plot(hist['loss\_eval'], marker='.')

plt.title('loss record')

plt.xlabel('epoch')

plt.ylabel('loss')

plt.grid()

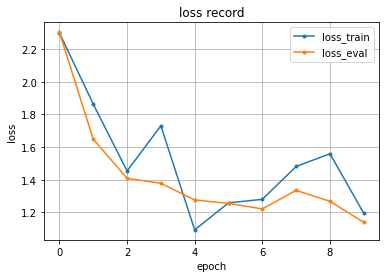
plt.legend(['loss\_train', 'loss\_eval'], loc='upper right')

plt.show()

plt.close()

plot\_loss(hist)

输出：



观察模型accuracy变化

代码：

def plot\_accuracy(hist):

plt.plot(hist['acc\_eval'], marker='.')

plt.title('accuracy history')

plt.xlabel('epoch')

plt.ylabel('acc\_eval')

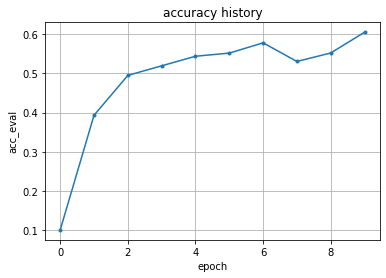
plt.grid()

plt.show()

plt.close()

plot\_accuracy(hist)

输出：



载入最佳网络参数，并测试其accuracy与loss

代码：

# 使用准确率最高的参数组合建立模型，并测试其在验证集上的效果

load\_checkpoint(os.path.join(ckpt\_path, 'best\_param.ckpt'), net=network1)

res = model.eval(test\_data, dataset\_sink\_mode=dataset\_sink\_mode)

print(res)

输出：

{'accuracy': 0.6052, 'loss': 1.1393188083171844}

### 模型优化

重新定义网络

Lenet网络本身的复杂度并不足以对CIFAR-10的图像分类任务产生出足够的拟合效果，因此需要做进一步改进。总的来说，网络基本维持了lenet的网络结构，增加卷积的个数与卷积核的大小，同时略微增加了网络的深度。

所有的卷积核从5∗5变成3∗3。

增加了一层网络的深度，提升模型的非线性映射能力。

提升了卷积核数量，使模型可以提取更多的特征，如32核，64核，128核。

代码：

class LeNet5\_2(nn.Cell):

# 定义算子

def \_\_init\_\_(self, num\_class=10, num\_channel=3):

super(LeNet5\_2, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(num\_channel, 32, 3, pad\_mode='valid', weight\_init=Normal(0.02))

self.conv2 = nn.Conv2d(32, 64, 3, pad\_mode='valid', weight\_init=Normal(0.02))

self.conv3 = nn.Conv2d(64, 128, 3, pad\_mode='valid', weight\_init=Normal(0.02))

self.fc1 = nn.Dense(128 \* 2 \* 2, 120, weight\_init=Normal(0.02))

self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))

self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))

self.relu = nn.ReLU()

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

self.num\_class = num\_class

# 构建网络

def construct(self, x):

x = self.conv1(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv2(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv3(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

x = self.relu(x)

x = self.fc3(x)

return x

重新处理数据集

数据预处理不做调整，重新创建训练集合测试集。

代码：

# 训练数据集预处理

train\_data = create\_dataset(train\_path, batch\_size = 32, usage = 'train')

# 测试数据集预处理

test\_data = create\_dataset(test\_path, batch\_size = 50, usage = 'test')

重新构建网络

构建新的网络、损失函数、优化器、模型。

代码：

# 网络

network2 = LeNet5\_2(10)

# 损失函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 优化器

net\_opt = nn.Adam(params=network2.trainable\_params())

# 模型

model = Model(network = network2, loss\_fn=net\_loss, optimizer=net\_opt, metrics={'accuracy': Accuracy(), 'loss':Loss()})

重新定义回调函数

定义新的Callback回调函数。

代码：

hist = {'loss':[], 'loss\_eval':[], 'acc\_eval':[]} # 训练过程记录

# 网络参数自动保存，这里设定每2000个step保存一次，最多保存10次

config\_ck = CheckpointConfig(save\_checkpoint\_steps=2000, keep\_checkpoint\_max=10)

ckpoint\_cb = ModelCheckpoint(prefix='checkpoint\_lenet\_2', directory=ckpt\_path, config=config\_ck)

# 记录每次迭代的模型准确率

train\_hist\_cb = TrainHistroy(hist['loss'])

# 测试并记录模型在验证集的loss和accuracy，并保存最优网络参数

eval\_hist\_cb = EvalHistory(model = model,

loss\_history = hist['loss\_eval'],

acc\_history = hist['acc\_eval'],

eval\_data = test\_data)

### 重新训练与评估

重新训练模型

优化了模型的网络结构并改用Adam优化器，数据预处理、模型训练的超参数与损失函数没做调整。

代码：

epoch = 10 # 迭代次数

# 开始训练

model.train(epoch, train\_data,

callbacks=[train\_hist\_cb, eval\_hist\_cb, time\_cb, ckpoint\_cb, LossMonitor(per\_print\_times=500)],

dataset\_sink\_mode=dataset\_sink\_mode)

输出：

epoch: 1 step: 500, loss is 1.7009366

epoch: 1 step: 1000, loss is 1.5922877

epoch: 1 step: 1500, loss is 1.6888745

acc\_eval: 0.458

epoch time: 87964.925 ms, per step time: 56.316 ms

epoch: 2 step: 438, loss is 1.4446156

epoch: 2 step: 938, loss is 0.927093

epoch: 2 step: 1438, loss is 1.3724257

acc\_eval: 0.5705

…

epoch time: 96208.182 ms, per step time: 61.593 ms

epoch: 10 step: 442, loss is 0.6328895

epoch: 10 step: 942, loss is 0.9332195

epoch: 10 step: 1442, loss is 0.9274956

acc\_eval: 0.7291

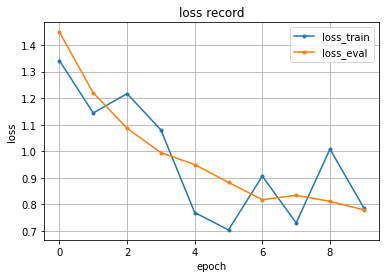
epoch time: 91424.528 ms, per step time: 58.530 ms

观察新模型loss的变化

代码：

plot\_loss(hist)

输出：

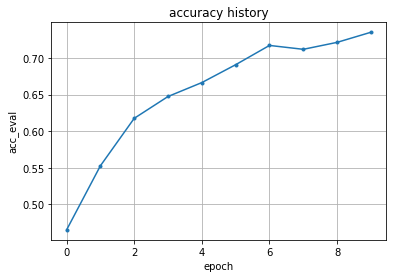


观察新模型accuracy变化

代码：

plot\_accuracy(hist)

输出：



载入最佳网络参数，并测试其accuracy与loss

代码：

# 使用准确率最高的参数组合建立模型，并测试其在验证集上的效果

best\_param = mindspore.load\_checkpoint(os.path.join(ckpt\_path, 'best\_param.ckpt'), net=network2)

res = model.eval(test\_data, dataset\_sink\_mode=dataset\_sink\_mode)

print(res)

输出：

{'accuracy': 0.735, 'loss': 0.7793407985568046}

### 效果展示

代码：

#创建图像标签列表

category\_dict = {0:'airplane',1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',

6:'frog',7:'horse',8:'ship',9:'truck'}

data\_path=os.path.join('data', '10-verify-bin')

demo\_data = create\_dataset(test\_path, batch\_size=1, usage='test')

# 将数据标准化至0~1区间

def normalize(data):

\_range = np.max(data) - np.min(data)

return (data - np.min(data)) / \_range

# 设置图像大小

plt.figure(figsize=(10,10))

i = 1

# 打印9张子图

for dic in demo\_data.create\_dict\_iterator():

# 预测单张图片

input\_img = dic['image']

output = model.predict(input\_img)

predict = np.argmax(output.asnumpy(),axis=1)[0] # 反馈可能性最大的类别

# 可视化

plt.subplot(3,3,i)

input\_image = np.squeeze(input\_img.asnumpy(),axis=0) # 删除batch维度，方便可视化

input\_image = input\_image.transpose(1,2,0) # CHW转HWC，方便可视化

input\_image = normalize(input\_image) # 重新标准化，方便可视化

plt.imshow(input\_image)

plt.axis('off')

plt.title('True: %s\n Predict: %s'%(category\_dict[dic['label'].asnumpy().item()],category\_dict[predict]))

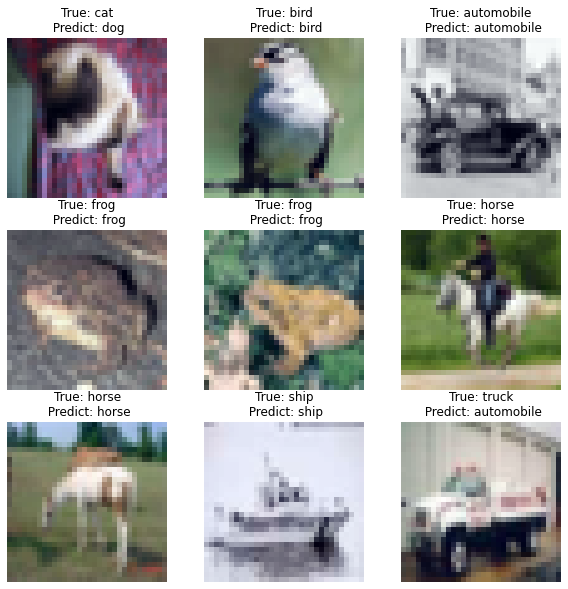
i +=1

if i > 9 :

break

plt.show()

输出：



## 实验总结

本章提供了一个基于开源框架MindSpore的图像识别实验。该实验演示了如何利用开源框架MindSpore完成CIFAR-10图像识别任务。本章对实验做了详尽的剖析，阐明了整个实验功能、结构与流程，详细解释了如何解析数据、如何构建深度学习模型、如何自定义回调函数以及保存等内容，并且展示了模型的优化与调参。学员可以在该实验的基础上开发更有针对性的应用实验。

## 思考题

1. 为何在数据增强前要加入 if usage == 'train' 的判断式？

答：数据增强的目的是提供模型更多不同的训练数据，评估是不需要做数据增强。

2. 有哪些数据增强的方法（举例）？

答：随机剪裁、随机翻转、随机旋转等

3. 什么是一个step？

答：一个step指一个batch送入网络中完成一次前向计算及反向传播的过程。

4. 编写自定义回调函数需继承什么基类？

答：Callback基类

5. 如何在自定义回调函数中获取训练过程中的重要信息（损失函数、优化器、当前的epoch数等）？

答：通过run\_context.original\_args()方法可以获取到cb\_params字典，字典里会包含训练过程中的重要信息。

6. 本实验的模型欠拟合，可通过增加网络复杂度优化模型。如果模型过拟合，常见的处理方法有哪些？

答：L1，L2正则化，Early stopping，增加数据集，增加噪声，添加dropout层等。

7. 如何理解dataset sink mode？

dataset\_sink\_mode=True时，可以这样简单的理解：Model中仅建立数据通道与执行网络之间的连接关系，不会直接将数据喂给网络。数据会通过数据通道(pipeline)下发到卡上，网络在卡上执行时会直接从对应的数据通道中获取数据。在这种模式下，数据下发与网络执行可以并行，单个epoch的训练过程中host与device之间不会交互，因此能提升性能，也因此打屏以epoch递增。

dataset\_sink\_mode=False，没有使用数据下沉模式。这时候数据不会通过通道直接向Device下发，Model会将数据一个batch一个batch得取出，喂给网络。在这种模式下，每个step结束，host都可以获取device上网络的执行结果，因此打屏以step递增。

8. 为什么本实验的模型在训练集的loss波动较大，且比在测试集高？

答：因为训练集做了数据增强，有些经过剪裁或翻转的图片较难辨认。

## 挑战

本实验的最终模型准确率仍然不高，尝试将模型准确率提升到90%以上。