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

商标声明

 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.huawei.com

目录

1 实验环境介绍

1.1 实验介绍

1.1.1 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。

1.1.2 实验环境介绍

实验、介绍、难度、软件环境、硬件环境：

表 1-1 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 1.1.1 Numpy 1.17.5 matplotlib 3.3.4	PC机

2 CIFAR-10 图像分类实验

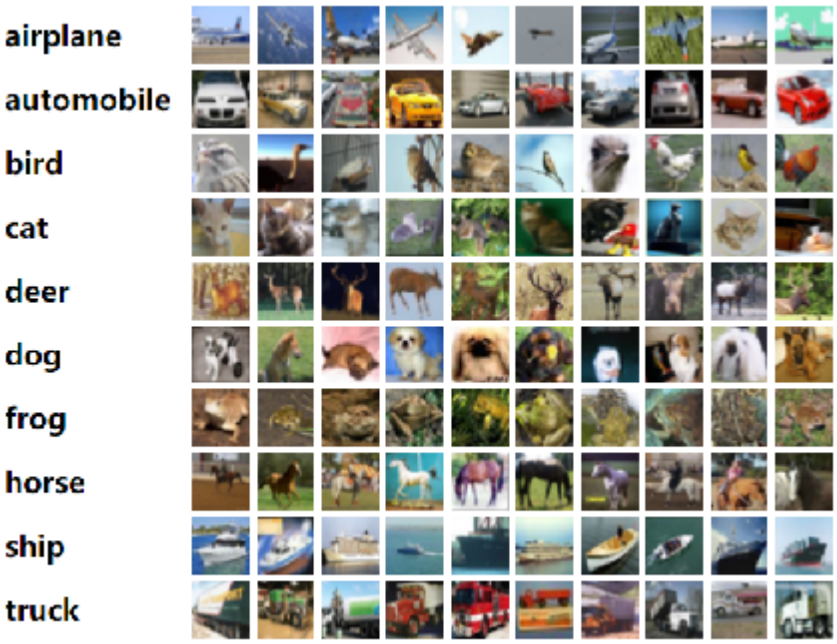
2.1 实验介绍

2.1.1 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



2.1.2 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。

2.1.3 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。

2.1.4 实验设计

新增数据展示、模型优化、模型重训练预评估：

2.2 实验过程

2.2.1 环境准备

注意：1. 请确保存储路径只有英文，否则可能报错；

2. 第二次运行请先清空 results 文件夹，否则可能报错。

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。

步骤 1 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
```

步骤 2 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
```

2.2.2 数据展示

步骤 1 查看数据集

代码：

```
# 数据路径
train_path = os.path.join('data','10-batches-bin') # 训练集路径
test_path = os.path.join('data','********************') #请填写测试集路径
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：data\10-batches-bin
测试集路径：data\********************
```

步骤 2 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: '****',
                 6: '****',7: '****',8: '****',9:'****'}# 请补充图片的类别

# 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：

2.2.3 数据处理

步骤 1 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(****** (tmp, axis=(0, 1, 2))) #请补充 np 函数以计算标准差


print(RGB_mean)
```

```
print(RGB_std)
```

输出：

```
(125.306918046875, 122.950394140625, 113.86538318359375)
(62.99321927774456, 62.08870764035727, 66.70489964064619)
```

步骤 2 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

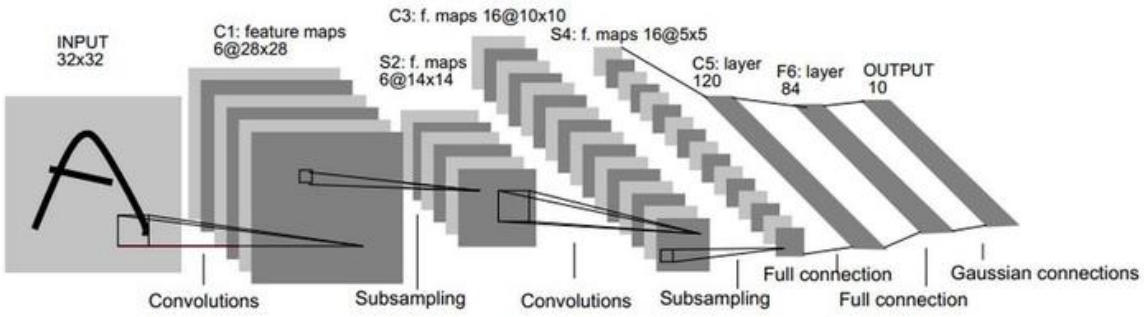
代码：

```
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
```

2.2.4 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
    # 请根据 Lenet 网络结构，构建如下模型，请留意，卷积层后需要加激活函数
    def construct(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        **********
        **********
        **********
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        **********
        **********
        x = self.fc3(x)
        return x
```

2.2.5 模型训练

步骤 1 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(*********, batch_size = 50, usage= 'test') # 测试数据集，请补充测试数据集路径
```

步骤 2 构建网络

构建网络、损失函数、优化器、模型。

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，请补充模型的超参数
model = Model(network = network1, loss_fn=*******, optimizer=*******, metrics={'accuracy': Accuracy(), 'loss':Loss()})
```

步骤 3 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/doc/programming_guide/zh-CN/r1.1/callback.html
- 自定义 Callback 教程：https://www.mindspore.cn/tutorial/training/zh-CN/r1.1/advanced_use/custom_debugging_info.html#id3

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)
```

步骤 4 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)
```

步骤 5 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
```

2.2.6 模型评估

查看模型在测试集的准确率。

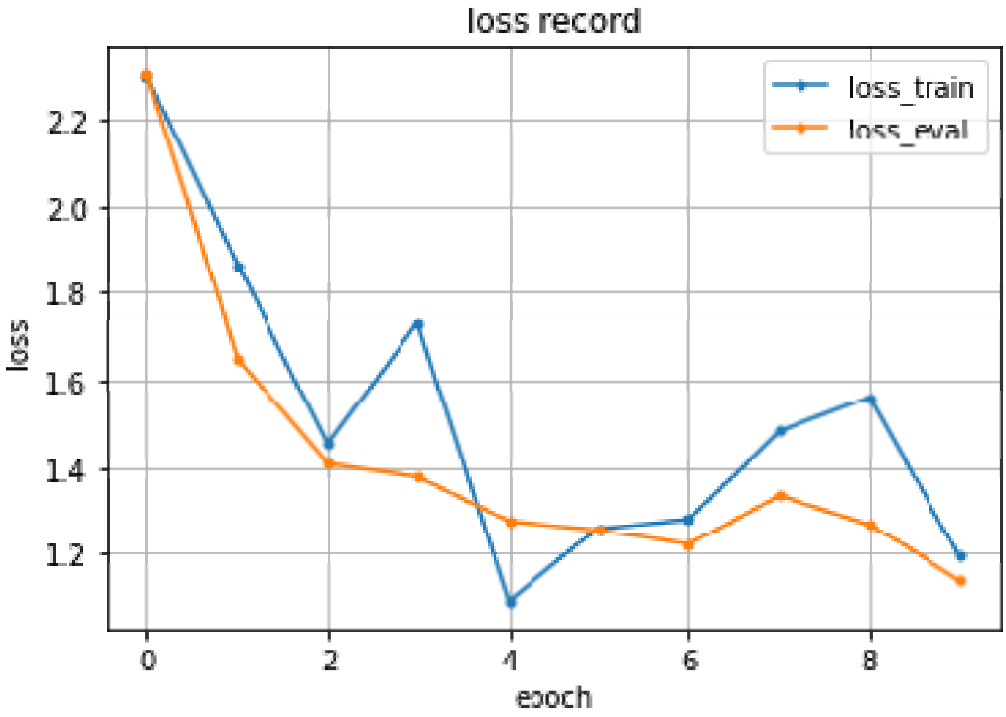
【注意】：测试集不会进行随机裁剪与翻转。

步骤 1 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：



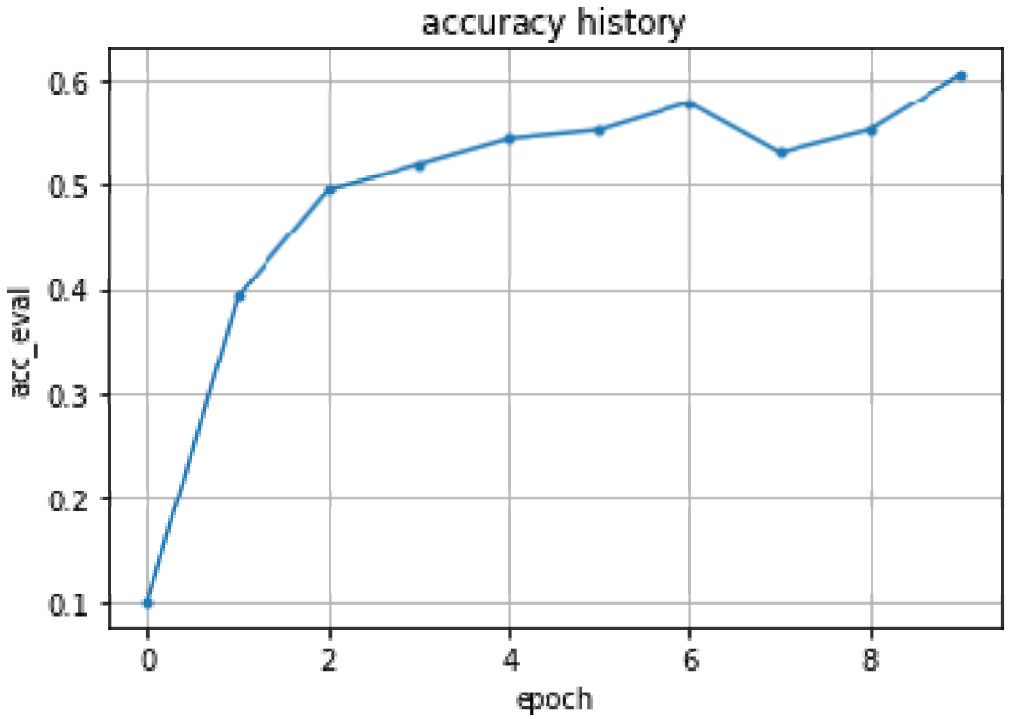
步骤 2 观察模型 accuracy 变化

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：



步骤 3　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}
```

2.2.7 模型优化

步骤 1　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
```

步骤 2 重新处理数据集

数据预处理不做调整，可沿用旧的数据。

但如果已将数据通过 pipeline 下发到装置上（dataset_sink_mode==True），使用新的 model 需要重新载入数据。

代码：

```
# 如果已将数据通过 pipeline 下发到装置上，使用新的 model 需要重新载入数据
if not dataset_sink_mode:
    # 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')
```

步骤 3 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，请补充 model 的超参数
model = Model(network = ********, loss_fn=********, optimizer=********, metrics={'accuracy': Accuracy(),
'loss':Loss()})
```

步骤 4 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)
```

2.2.8 重新训练与评估

步骤 1 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：

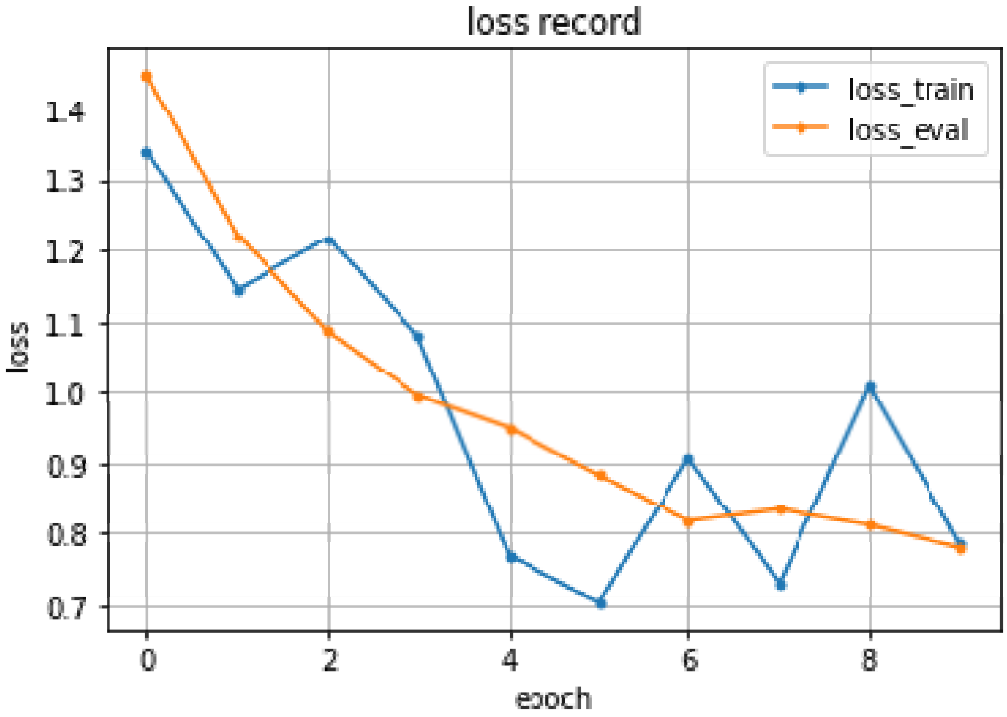
```
epoch: 1 step: 500, loss is 1.7009366
epoch: 1 step: 1000, loss is 1.5922877
epoch: 1 step: 1500, loss is 1.6888745
acc_eval:    0.458
epoch time: 87964.925 ms, per step time: 56.316 ms
epoch: 2 step: 438, loss is 1.4446156
epoch: 2 step: 938, loss is 0.927093
epoch: 2 step: 1438, loss is 1.3724257
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
```

步骤 2 观察新模型 loss 的变化

代码：

```
plot_loss(hist)
```

输出：

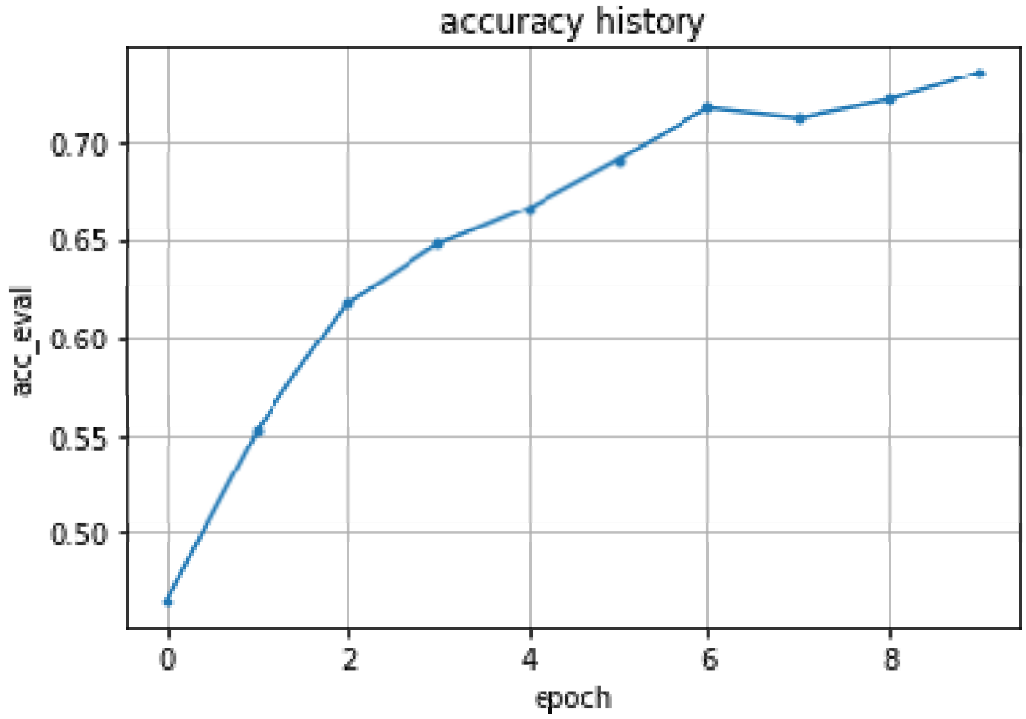


步骤 3 观察新模型 accuracy 变化

代码：

```
plot_accuracy(hist)
```

输出：



步骤 4 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}
```

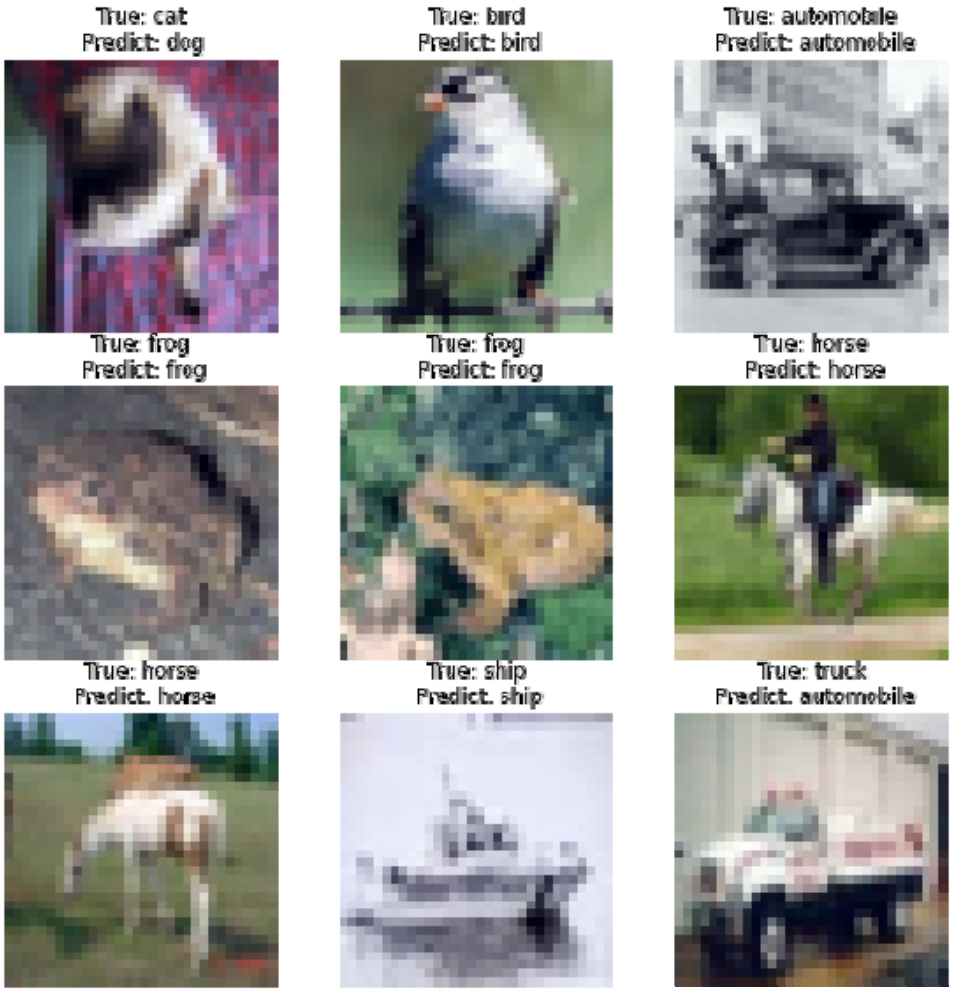
2.2.9 效果展示

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：



2.3 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。

2.4 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。

2.5 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。