《计算机视觉》
-物体识别

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 实验介绍

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的模块进行介绍。本章实验难度分为初级和高级。

初级实验：CIFAR-10 分类任务

1.1 实验目的

本章实验的主要目的是掌握计算机视觉中物体识别相关基础知识点，了解物体识别中的图像识别与视频识别，同时掌握深度学习相关基础知识，尤其是卷积神经网络。掌握不同相关的物体识别中比较成熟的物体识别网络的设计原理，熟悉使用 MindSpore 深度学习框架。

1.2 实验清单

实验	简述	难度	软件环境	开发环境
CIFAR-10分类任务	基于cifar10数据集，使用MindSpore和Ascend芯片，实现图像分类。	初级	MindSpore1.5	ModelArts

2 CIFAR-10 分类任务

2.1 实验介绍

图像中存在大量信息，所谓一图胜千言，就是在表达这个意思。在众多图像处理中，对图像进行分类将是最基本的任务。本实验将利用卷积神经网络进行手写体识别，实验中使用深度学习框架 Mindspore 构建卷积神经网络模型解决图像分类问题。

实验使用的数据集是 CIFAR-10，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，学员们将通过本实验理解物体识别的基本知识。

2.2 实验环境要求

Python 3.7.5

Mindspore 1.5

Matplotlib 3.2.2

Numpy 1.18.5

2.3 实验目的

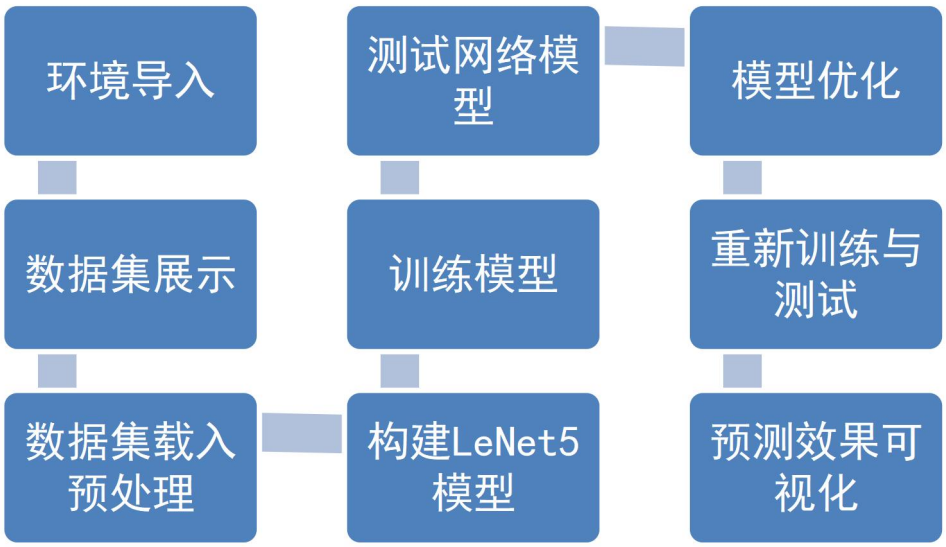
加强对基于 Mindspore 的神经网络模型构建流程的理解。

掌握如何用 Mindspore 实现卷积神经网络的构建。

学会利用 checkpoint 函数保存模型参数。

掌握如何利用模型预测单张图像的分类结果。

2.4 实验总体设计



2.5 实验过程

2.5.1 创建华为云 Notebook

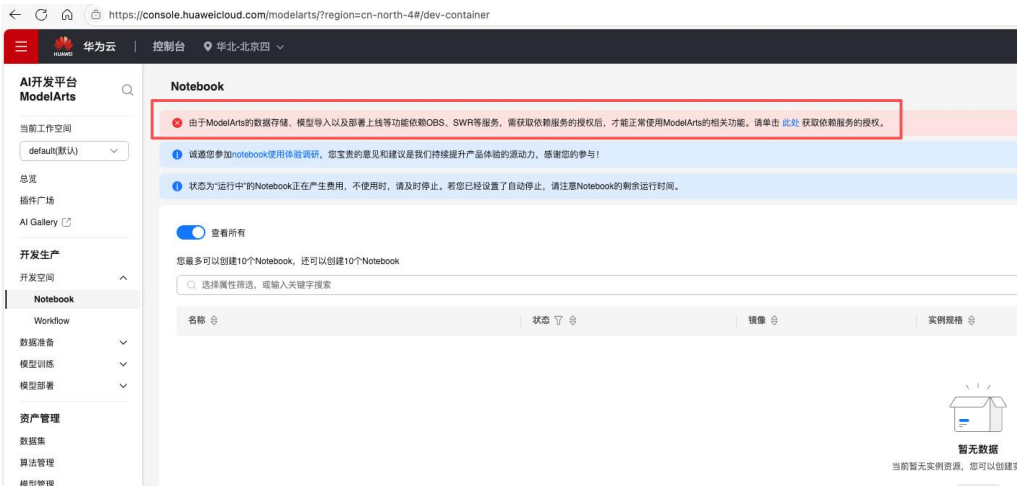
步骤 1 进入 Modelarts，选择 Notebook

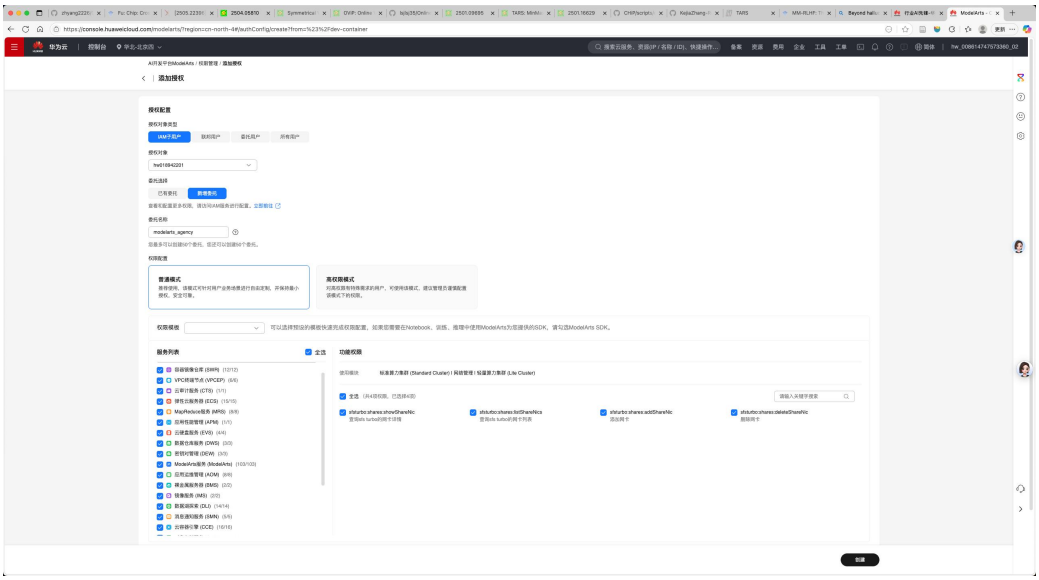
在华为云主页搜索 Modelarts 并点击“进入控制台”，或者通过以下网址进入：

https://console.huaweicloud.com/modelarts/?region=cn-north-4#/dashboard

点击左侧导航栏的“开发环境”，选择“Notebook”。

在使用前需要进行授权。





步骤 2 创建 Notebook

点击创建按钮来创建一个新的 Notebook，选择如下配置：

名称：建议使用 CIFAR10

工作环境：选择 tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64

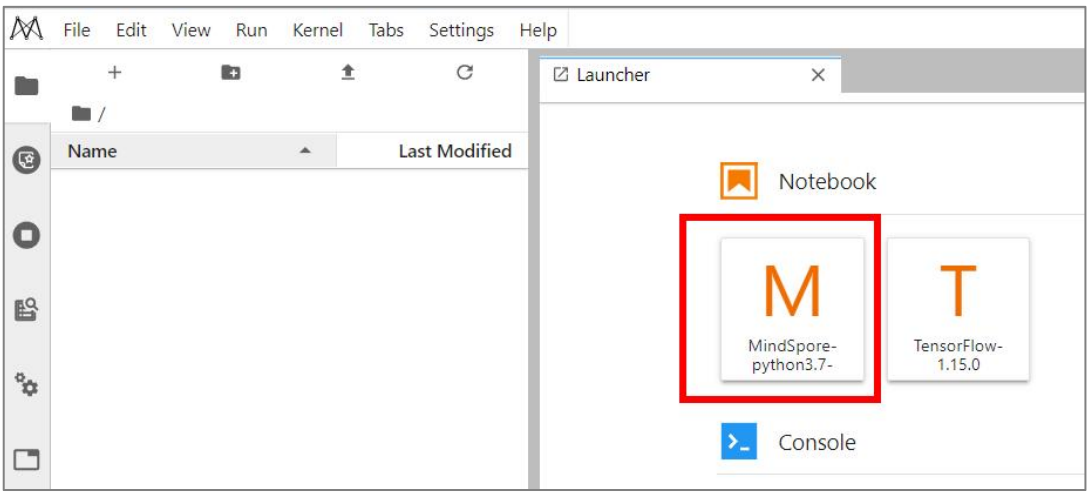
规格：Ascend: 1*Ascend910|CPU:24 核 96GB

存储配置：云硬盘 EVS 规格：5GB

点击“下一步”，确认规格如下后选择提交：

步骤 3 启动 Notebook

当 Notebook 状态变为“运行中”时，点击右侧“打开”按钮打开 Notebook。打开后选择右侧“MindSpore-python3.7-aarch64”按钮，进入 Notebook 环境：



2.5.2 导入相关实验模块

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 主要包括 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.Tensor 提供了 mindspore 网络可用的张量，context 用于设定 mindspore 的运行环境与运行设备，Model 用来承载网络结构，并能够调用优化器，损失函数，评价指标。

mindspord.nn 当中主要会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 下面会涉及到各类回调函数，如 checkpoint,lossMonitor 等，主要用于在每个 epoch 训练完的时候自动执行。

其他 numpy 用来处理数组问题，matplotlib 用于画图。

```
import mindspore
# 载入 mindspore 的默认数据集
```

```
import mindspore.dataset as ds
# 常用转化用算子
import mindspore.dataset.transforms.c_transforms as C
# 图像转化用算子
####____####
import mindspore.dataset.vision.c_transforms as CV
from mindspore.common import dtype as mstype
# mindspore 的 tensor
from mindspore import Tensor


# 各类网络层都在 nn 里面
import mindspore.nn as nn
# 参数初始化的方式

from mindspore.common.initializer import TruncatedNormal
# 设置 mindspore 运行的环境
from mindspore import context
# 引入训练时候会使用到回调函数，如 checkpoint, lossMoniter
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor
# 引入模型
from mindspore.train import Model
# 引入评估模型的包
from mindspore.nn.metrics import Accuracy

# numpy
import numpy as np
# 画图用
import matplotlib.pyplot as plt


####____####
# 下载数据相关的包
import os
import requests
import zipfile
```

2.5.3 数据集展示与数据初始化

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

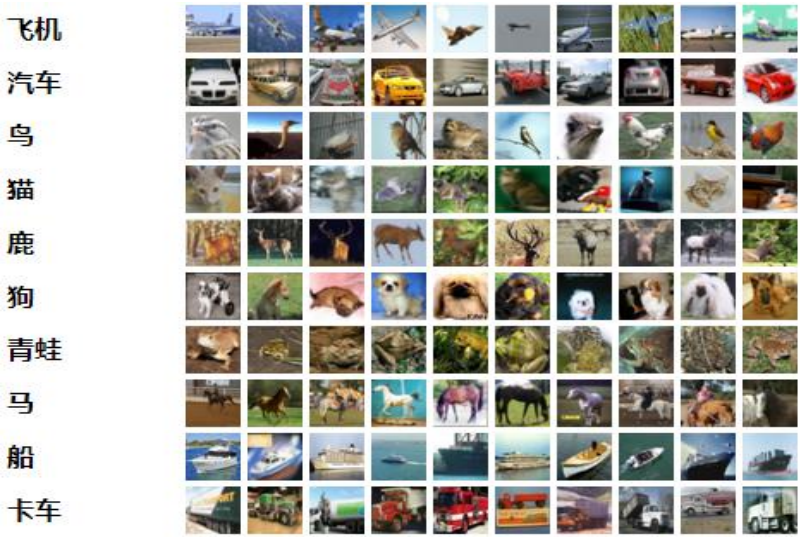


图 2-1 Cifar10 数据示例

下方代码包含下载步骤，您也可从官网下载后上传至云上 notebook 开发环境下。

- 官网下载：http://www.cs.toronto.edu/~kriz/cifar.html 注意选择二进制版本文件。

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Download

If you're going to use this dataset, please cite the tech report at the bottom of this page.

Version	Size	md5sum
CIFAR-10 python version	163 MB	c58f30108f718f92721af3b95e74349a
CIFAR-10 Matlab version	175 MB	70270af85842c9e89bb428ec9976c926
CIFAR-10 binary version (suitable for C programs)	162 MB	c32a1d4ab5d03f1284b67883e8d87530

2.5.3.2 数据集下载

```
!wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/ComputerVision/cifar10_mindspore.zip
!unzip cifar10_mindspore.zip
```

2.5.3.3 查看数据集

注意：这里每次提取出来的数据是随机的。

```
#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'}
####____####
current_path = os.getcwd()
data_path = os.path.join(current_path, 'data/10-verify-bin')
```

```
cifar_ds = ds.Cifar10Dataset(data_path)
# 设置图像大小
plt.figure(figsize=(8,8))
i = 1
# 打印 9 张子图
for dic in cifar_ds.create_dict_iterator():
    plt.subplot(3,3,i)
    ####____####
    plt.imshow(dic['image'].asnumpy())
    plt.xticks([])
    plt.yticks([])
    plt.axis('off')
    plt.title(category_dict[dic['label'].asnumpy().sum()])
    i +=1
    if i > 9 :
        break
plt.show()
```

输出：

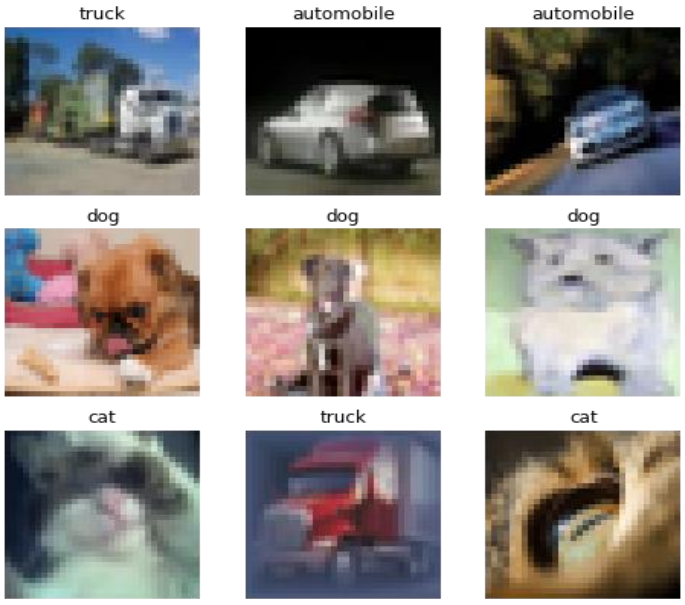


图 2-2 Cifar10 图像示例

2.5.3.4 思考题

1. 彩色图像有几个颜色通道？
2. 请列举常见的颜色空间。

【答案】

1. 3 个颜色通道，一般为 RGB。
2. RGB, HSV，Lab，CMYK 等。

2.5.3.5 定义数据预处理的步骤

定义了两个函数，getdata 用于数据集的读取，使用 dataset.Cifar10Dataset()来完成（数据需要预下载）；第二个函数 process_dataset 是对于图像数据特征处理，其中主要包括尺寸大小变更、平移、归一化与标准化、训练时的随机裁剪、随机翻转等，并且内部对于数据集进行了 shuffle，变更了一个批量输出的 generator。

```
def get_data(datapath):
    cifar_ds = ds.Cifar10Dataset(datapath)
    return cifar_ds

def process_dataset(cifar_ds,batch_size =32,status="train"):
    '''
    ---- 定义算子 ----
    '''
    # 归一化
    rescale = 1.0 / 255.0
    # 平移
    shift = 0.0

    resize_op = CV.Resize((32, 32))
    rescale_op = CV.Rescale(rescale, shift)
    # 对于 RGB 三通道分别设定 mean 和 std
    normalize_op = CV.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    if status == "train":
        # 随机裁剪
        random_crop_op = CV.RandomCrop([32, 32], [4, 4, 4, 4])
        # 随机翻转
        random_horizontal_op = CV.RandomHorizontalFlip()
    # 通道变化
    channel_swap_op = CV.HWC2CHW()
    # 类型变化
    typecast_op = C.TypeCast(mstype.int32)
    '''
    ---- 算子运算 ----
    '''
    cifar_ds = cifar_ds.map(input_columns="label", operations=typecast_op)
    if status == "train":
        cifar_ds = cifar_ds.map(input_columns="image", operations=random_crop_op)
        cifar_ds = cifar_ds.map(input_columns="image", operations=random_horizontal_op)
    cifar_ds = cifar_ds.map(input_columns="image", operations=resize_op)
    cifar_ds = cifar_ds.map(input_columns="image", operations=rescale_op)
    cifar_ds = cifar_ds.map(input_columns="image", operations=normalize_op)
    cifar_ds = cifar_ds.map(input_columns="image", operations=channel_swap_op)

    # shuffle
    cifar_ds = cifar_ds.shuffle(buffer_size=1000)
    # 切分数据集到 batch_size
    cifar_ds = cifar_ds.batch(batch_size, drop_remainder=True)
    return cifar_ds
```

2.5.3.6 生成训练数据集

引用了上述定义的函数，训练集的位置在'./data/10-batches-bin'中，我们设置的 batch_size=32。

```
data_path = os.path.join(current_path, 'data/10-batches-bin')
batch_size=32
status="train"
# 生成训练数据集
cifar_ds = get_data(data_path)
ds_train = process_dataset(cifar_ds,batch_size =batch_size, status=status)
```

2.5.3.7 思考题

1. 为什么我们在训练时会使用随机裁剪或者翻转的方式来处理图片？
2. 将图片归一化有什么好处？

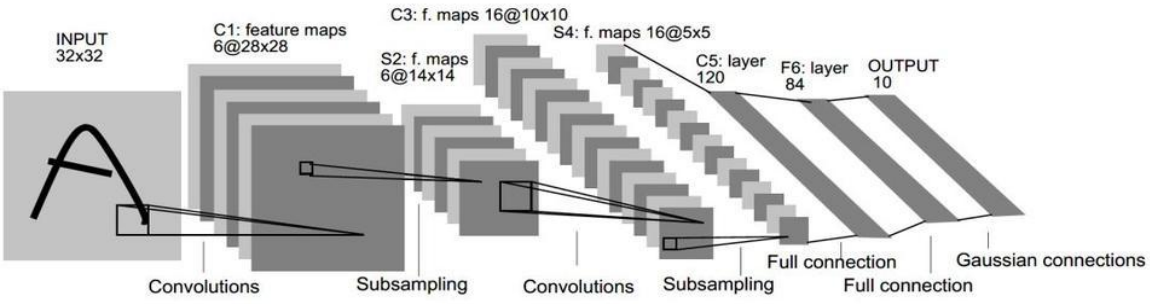
【答案】

1. 为了增加网络的泛化能力，即使图片只有部分或者翻转了，也依然有能力识别出来。
2. 加快模型的训练时候的收敛速度，也可以在一定程度上避免梯度消失或者爆炸。

2.5.4 构建网络模型

2.5.4.1 定义 Lenet 网络结构，构建网络

LeNet-5 出自论文《Gradient-Based Learning Applied to Document Recognition》，原本是一种用于手写体字符识别的非常高效的卷积神经网络，包含了深度学习的基本模块：卷积层，池化层，全连接层。



其网络结构如下：

1. INPUT（输入层）：输入 32*32 的图片。
2. C1（卷积层）：选取 6 个 5*5 卷积核(不包含偏置)，得到 6 个特征图，每个特征图的一个边为 32-5+1=28，也就是神经元的个数由 32*32=1024 减小到了 28*28=784。
3. S2（池化层）：池化层是一个下采样层，输出 14*14*6 的特征图。
4. C3（卷积层）：选取 16 个大小为 5*5 卷积核，得到特征图大小为 10*10*16。
5. S4（池化层）：窗口大小为 2*2，输出 5*5*16 的特征图。

6. F5（全连接层）：120 个神经元。
7. F6（全连接层）：84 个神经元。
8. OUTPUT（输出层）：10 个神经元，10 分类问题。

```
"""LeNet."""


def conv(in_channels, out_channels, kernel_size, stride=1, padding=0):
    """weight initial for conv layer"""
    weight = weight_variable()
    return nn.Conv2d(in_channels, out_channels,
                     kernel_size=kernel_size, stride=stride, padding=padding,
                     weight_init=weight, has_bias=False, pad_mode="same")


def fc_with_initialize(input_channels, out_channels):
    """weight initial for fc layer"""
    weight = weight_variable()
    bias = weight_variable()
    return nn.Dense(input_channels, out_channels, weight, bias)


def weight_variable():
    """weight initial"""
    return TruncatedNormal(0.02)


class LeNet5(nn.Cell):
    """
    Lenet network

    Args:
        num_class (int): Num classes. Default: 10.

    Returns:
        Tensor, output tensor
    Examples:
        >>> LeNet(num_class=10)

    """
    def __init__(self, num_class=10, channel=3):
        super(LeNet5, self).__init__()
        self.num_class = num_class
        self.conv1 = conv(channel, 6, 5)
        self.conv2 = conv(6, 16, 5)
        self.fc1 = fc_with_initialize(16 * 8 * 8, 120)
```

```
        self.fc2 = fc_with_initialize(120, 84)
        self.fc3 = fc_with_initialize(84, self.num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    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
# 构建网络
network = LeNet5(10)
```

2.5.5 模型训练与测试

2.5.5.1 定义损失函数与优化器

这一部分主要给出了用于训练网络的损失函数优化器，本次使用的损失函数为 nn.SoftmaxCrossEntropyWithLogits 损失函数，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。优化器使用了 Adam，即动量优化器。

另外我们同时设置了 mindspore 网络的设备与图的模型，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。设备指定为 CPU。

```
# 返回当前设备
device_target = mindspore.context.get_context('device_target')
# 确定图模型是否下沉到芯片上
dataset_sink_mode = True if device_target in ['Ascend','GPU'] else False
# 设置模型的设备与图的模式
context.set_context(mode=context.GRAPH_MODE, device_target=device_target)
# 使用交叉熵函数作为损失函数
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
# 优化器为 Adam
net_opt = nn.Adam(params=network.trainable_params(), learning_rate=0.001)
# 监控每个 epoch 训练的时间
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
```

2.5.5.2 定义保存路径与训练

这一部分主要包括训练时候用的 callback 函数 CheckpointConfig，ModelCheckpoint。Model 函数中，确定网络模型、损失函数、优化器、评估指标。

```
from mindspore.train.callback import Callback

class EvalCallBack(Callback):
    def __init__(self, model, eval_dataset, eval_per_epoch, epoch_per_eval):
        self.model = model
        self.eval_dataset = eval_dataset
        self.eval_per_epoch = eval_per_epoch
        self.epoch_per_eval = epoch_per_eval

    def epoch_end(self, run_context):
        cb_param = run_context.original_args()
        cur_epoch = cb_param.cur_epoch_num
        if cur_epoch % self.eval_per_epoch == 0:
            acc = self.model.eval(self.eval_dataset, dataset_sink_mode=False)
            self.epoch_per_eval["epoch"].append(cur_epoch)
            self.epoch_per_eval["acc"].append(acc["Accuracy"])
            print(acc)


# 设置 CheckpointConfig，callback 函数。save_checkpoint_steps=训练总数/batch_size
config_ck = CheckpointConfig(save_checkpoint_steps=1562,
                             keep_checkpoint_max=10)
ckpoint_cb = ModelCheckpoint(prefix="checkpoint_lenet_original", directory='./results',config=config_ck)
# 建立可训练模型
model = Model(network = network, loss_fn=net_loss,optimizer=net_opt, metrics={"Accuracy": Accuracy()})
eval_per_epoch = 1
epoch_per_eval = {"epoch": [], "acc": []}
eval_cb = EvalCallBack(model, ds_train, eval_per_epoch, epoch_per_eval)
print("============== Starting Training ==============")
model.train(100, ds_train,callbacks=[ckpoint_cb,
LossMonitor(per_print_times=1),eval_cb],dataset_sink_mode=dataset_sink_mode)
```

输出：

```
============== Starting Training ==============
epoch: 1 step: 312, loss is 1.9112499
{'Accuracy': 0.25470753205128205}
epoch: 2 step: 312, loss is 1.7392981
{'Accuracy': 0.3444511217948718}
epoch: 3 step: 312, loss is 1.5038271
{'Accuracy': 0.386318108974359}
epoch: 4 step: 312, loss is 1.7462614
{'Accuracy': 0.4170673076923077}
```

```
epoch: 5 step: 312, loss is 1.2543318
{'Accuracy': 0.4378004807692308}
......
......
epoch: 95 step: 312, loss is 0.9264654
{'Accuracy': 0.6858974358974359}
epoch: 96 step: 312, loss is 1.0020974
{'Accuracy': 0.6957131410256411}
epoch: 97 step: 312, loss is 0.7703911
{'Accuracy': 0.6965144230769231}
epoch: 98 step: 312, loss is 1.0507934
{'Accuracy': 0.6821915064102564}
epoch: 99 step: 312, loss is 0.75813186
{'Accuracy': 0.6972155448717948}
epoch: 100 step: 312, loss is 0.99650973
{'Accuracy': 0.6905048076923077}
```

2.5.5.3 设置测试集参数并测试

注意：测试集不会进行随机裁剪与翻转

```
data_path = os.path.join(current_path, 'data/10-verify-bin')
batch_size=32
status="test"
# 生成测试数据集
cifar_ds = ds.Cifar10Dataset(data_path)
ds_eval = process_dataset(cifar_ds,batch_size=batch_size,status=status)

res = model.eval(ds_eval, dataset_sink_mode=dataset_sink_mode)
# 评估测试集
print('test results:',res)
```

输出：

```
test results: {'Accuracy': 0.73046875}
```

说明：可以发现，不管是训练集还是测试集准确率都只有 70%左右，可见模型处于欠拟合状态，因此需要想办法增加模型的拟合能力。

2.5.5.4 图片类别预测与可视化

```
#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'}

cifar_ds = get_data('./data/10-verify-bin')
df_test = process_dataset(cifar_ds,batch_size=1,status='test')
```

```
def normalization(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 df_test:
    # 预测单张图片
    input_img = dic[0]
    output = model.predict(Tensor(input_img))
    output = nn.Softmax()(output)
    # 反馈可能性最大的类别
    predicted = np.argmax(output.asnumpy(),axis=1)[0]

    # 可视化
    plt.subplot(3,3,i)
    # 删除 batch 维度
    input_image = np.squeeze(input_img.asnumpy(),axis=0).transpose(1,2,0)
    # 重新归一化，方便可视化
    input_image = normalization(input_image)
    plt.imshow(input_image)
    plt.xticks([])
    plt.yticks([])
    plt.axis('off')
    plt.title('True label:%s,\n Predicted:%s'%(category_dict[dic[1].asnumpy().sum()],category_dict[predicted]))
    i +=1
    if i > 9 :
        break

plt.show()
```

输出：

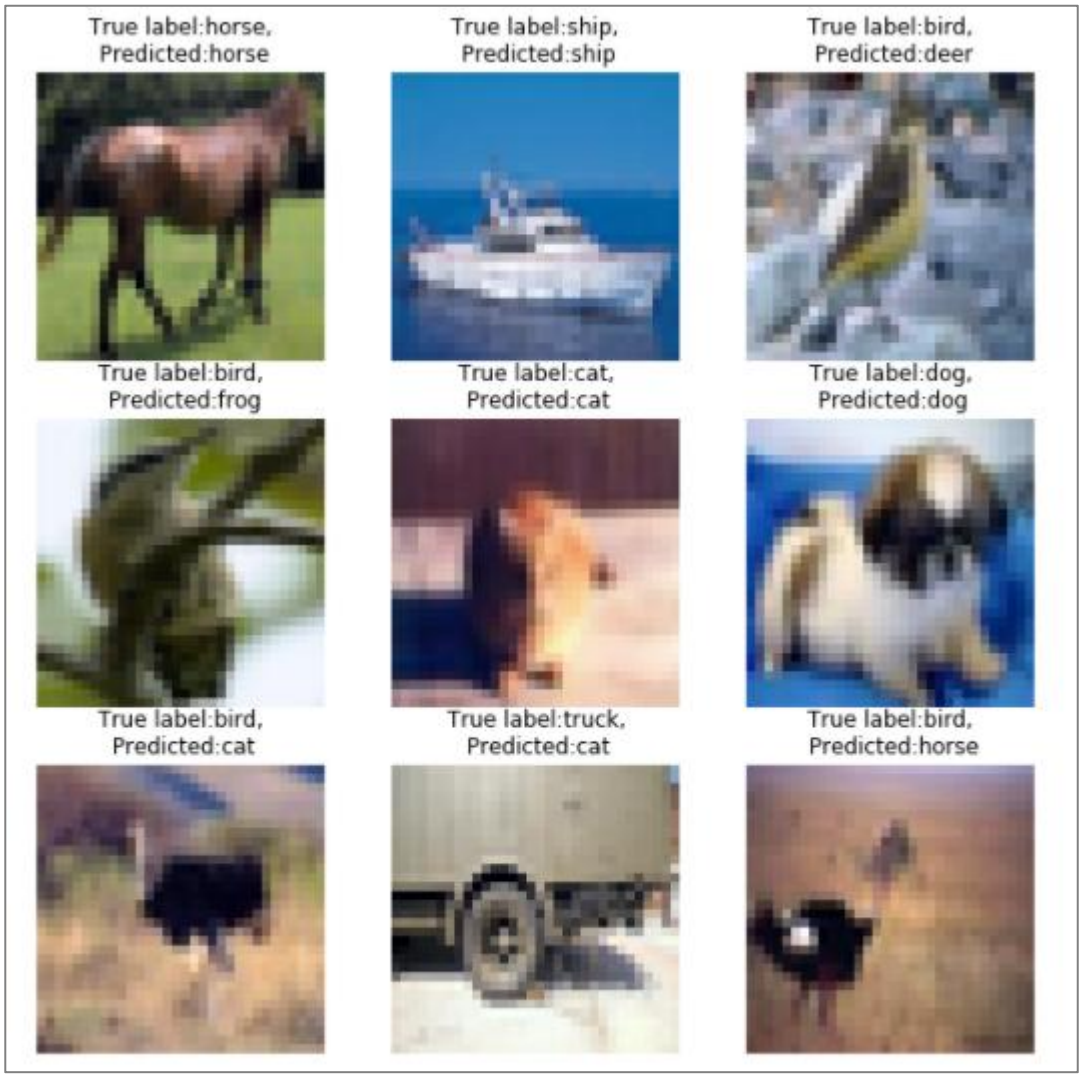


图 2-3 预测结果

2.5.5.5 思考题

1. 除了 Adam 优化器，请列举其他常见的优化器?
2. 什么叫一个 epoch?

【答案】

1. SGD, BGD, MBGD, Momentum, NAG(Nesterov Accelerated Gradient)，Adagrad，Adadelta，RMSprop。
2. 一个 epoch 指代所有的数据送入网络中完成一次前向计算及反向传播的过程。

2.5.6 模型优化与重新训练

2.5.6.1 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，增加卷积的个数且降低卷积核的大小，同时略微增加了网络的深度。最后，为了帮助训练，在每一层网络层后都加入了批标准化层（BatchNormalization）以加快训练且减少过拟合。

1. 所有的卷积核从 5*5 变成 3*3。
2. 增加了两层卷积层，提升模型的非线性映射能力。
3. 提升了卷积核数量，使模型可以提取更多的特征，如 64 核，128 核，256 核。
4. 在每一层网络层中加入 BatchNormalization 层。

```
class LeNet5_2(nn.Cell):
    """
    Lenet network



    Args:
        num_class (int): Num classes. Default: 10.

    Returns:
        Tensor, output tensor
    Examples:
        >>> LeNet(num_class=10)

    """
    def __init__(self, num_class=10, channel=3):
        super(LeNet5_2, self).__init__()
        self.num_class = num_class
        self.conv1_1 = conv(channel, 8, 3)
        self.bn2_1 = nn.BatchNorm2d(num_features=8)
        self.conv1_2 = conv(8, 16, 3)
        self.bn2_2 = nn.BatchNorm2d(num_features=16)
        self.conv2_1 = conv(16, 32, 3)
        self.bn2_3 = nn.BatchNorm2d(num_features=32)
        self.conv2_2 = conv(32, 64, 3)
        self.bn2_4 = nn.BatchNorm2d(num_features=64)
        self.fc1 = fc_with_initialize(64*8*8, 120)
        self.bn1_1 = nn.BatchNorm1d(num_features=120)
        self.fc2 = fc_with_initialize(120, 84)
        self.bn1_2 = nn.BatchNorm1d(num_features=84)
        self.fc3 = fc_with_initialize(84, self.num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
```

```
        self.flatten = nn.Flatten()



    def construct(self, x):
        x = self.conv1_1(x)
        x = self.bn2_1(x)
        x = self.relu(x)
        x = self.conv1_2(x)
        x = self.bn2_2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2_1(x)
        x = self.bn2_3(x)
        x = self.relu(x)
        x = self.conv2_2(x)
        x = self.bn2_4(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1_1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.bn1_2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

2.5.6.2 用新网络进行训练

其余损失函数与优化器等保持不变，以及模型训练的参数保持不变，仅仅将 ModelCheckpoint 中保存模型的前缀 prefix 改为"checkpoint_lenet_2_verified"。

```
data_path = os.path.join(current_path, 'data/10-batches-bin')
batch_size=32
status="train"

# 生成训练数据集
cifar_ds = get_data(data_path)
ds_train = process_dataset(cifar_ds,batch_size =batch_size, status=status)
network = LeNet5_2(10)
#network = resnet50(10)
# 返回当前设备
device_target = mindspore.context.get_context('device_target')
# 确定图模型是否下沉到芯片上
dataset_sink_mode = True if device_target in ['Ascend','GPU'] else False
```

```
# 设置模型的设备与图的模式
context.set_context(mode=context.GRAPH_MODE, device_target=device_target)
# 使用交叉熵函数作为损失函数
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
# 优化器为 momentum
#net_opt = nn.Momentum(params=network.trainable_params(), learning_rate=0.01, momentum=0.9)
net_opt = nn.Adam(params=network.trainable_params(), learning_rate=0.001)
# 时间监控，反馈每个 epoch 的运行时间
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
# 设置 callback 函数。
config_ck = CheckpointConfig(save_checkpoint_steps=1562,
                             keep_checkpoint_max=10)
ckpoint_cb = ModelCheckpoint(prefix="checkpoint_lenet_2_verified",directory='./results', config=config_ck)
# 建立可训练模型
model = Model(network = network, loss_fn=net_loss,optimizer=net_opt, metrics={"Accuracy": Accuracy()})
eval_per_epoch = 1
epoch_per_eval = {"epoch": [], "acc": []}
eval_cb = EvalCallBack(model, ds_train, eval_per_epoch, epoch_per_eval)
print("============== Starting Training ==============")

model.train(100, ds_train,callbacks=[ckpoint_cb,
LossMonitor(per_print_times=1),eval_cb],dataset_sink_mode=dataset_sink_mode)
```

输出：

```
============== Starting Training ==============
epoch: 1 step: 312, loss is 1.7387633
{'Accuracy': 0.44611378205128205}
epoch: 2 step: 312, loss is 1.5458272
{'Accuracy': 0.501802884615384 6}
epoch: 3 step: 312, loss is 1.271571
{'Accuracy': 0.5406650641025641}
epoch: 4 step: 312, loss is 1.5671562
{'Accuracy': 0.6051682692307693}
epoch: 5 step: 312, loss is 0.9090654
{'Accuracy': 0.6235977564102564}
……
……
epoch: 95 step: 312, loss is 0.44285166
{'Accuracy': 0.9346955128205128}
epoch: 96 step: 312, loss is 0.17368981
{'Accuracy': 0.9334935897435898}
epoch: 97 step: 312, loss is 0.45997486
{'Accuracy': 0.9360977564102564}
epoch: 98 step: 312, loss is 0.6351193
{'Accuracy': 0.9377003205128205}
```

```
epoch: 99 step: 312, loss is 0.32450855
{'Accuracy': 0.9397035256410257}
epoch: 100 step: 312, loss is 0.2047475
{'Accuracy': 0.9345953525641025}
```

2.5.7 模型测试与可视化

2.5.7.1 评估模型的有效性

```
data_path = os.path.join(current_path, 'data/10-verify-bin')
batch_size=32
status="test"
# 生成测试数据集
cifar_ds = ds.Cifar10Dataset(data_path)
ds_eval = process_dataset(cifar_ds,batch_size=batch_size,status=status)

res = model.eval(ds_eval, dataset_sink_mode=dataset_sink_mode)
# 评估测试集
print('test results:',res)
```

输出：

```
test results: {'Accuracy': 0.9730568910256411}
```

2.5.7.2 图片类别预测与可视化

```
#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'}

cifar_ds = get_data('./data/10-verify-bin')
df_test = process_dataset(cifar_ds,batch_size=1,status='test')

def normalization(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 df_test:
    # 预测单张图片
    input_img = dic[0]
    output = model.predict(Tensor(input_img))
    output = nn.Softmax()(output)
    # 反馈可能性最大的类别
```

```
        predicted = np.argmax(output.asnumpy(),axis=1)[0]

        # 可视化
        plt.subplot(3,3,i)
        # 删除 batch 维度
        input_image = np.squeeze(input_img.asnumpy(),axis=0).transpose(1,2,0)
        # 重新归一化，方便可视化
        input_image = normalization(input_image)
        plt.imshow(input_image)
        plt.xticks([])
        plt.yticks([])
        plt.axis('off')
        plt.title('True label:%s,\n Predicted:%s'%(category_dict[dic[1].asnumpy().sum()],category_dict[predicted]))
        i +=1
        if i > 9 :
            break

plt.show()
```

输出：

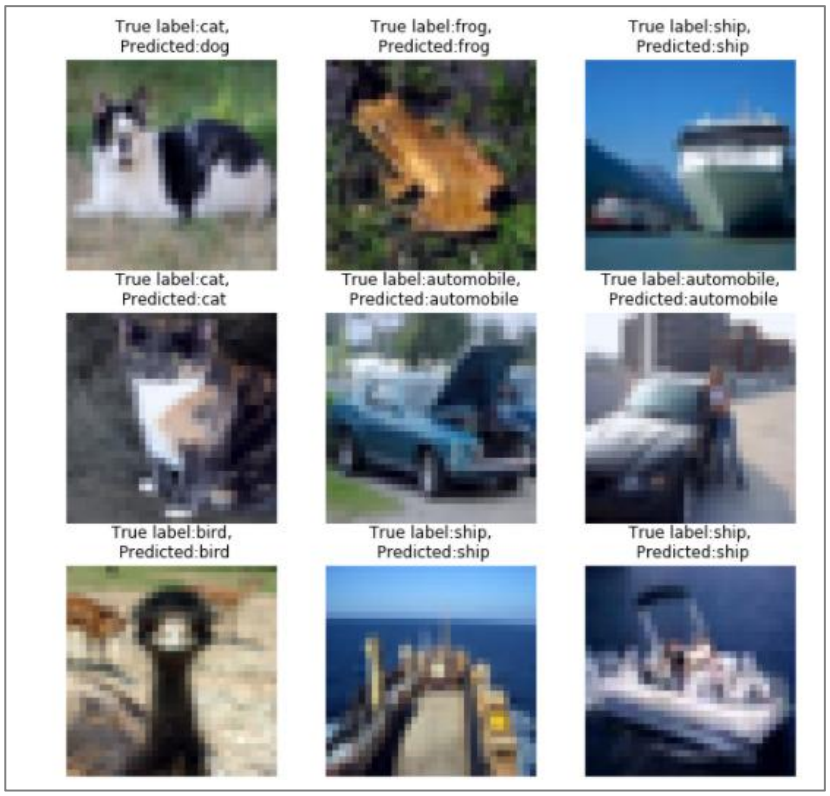


图 2-4 预测结果 2

2.5.7.3 思考题

1. 如果模型过拟合，常见的处理方法有哪些?
2. 【开放题】请设计的一个网络让测试集准确率达到 99%以上。

【答案】

1. L1，L2 正则化，Early stopping，数据增强，dropout 等。

2.6 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、如何保存模型等内容，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。

2.7 思考题汇总

- 彩色图像有几个颜色通道？
- 请列举一下常见的颜色空间。
- 为什么我们在训练时会使用随机裁剪或者翻转的方式来处理图片？
- 将图片归一化有什么好处？
- 除了 Adam 优化器，请列举其他常见的优化器?
- 什么叫一个 epoch?
- 如果模型过拟合，常见的处理方法有哪些?

【答案】

- 3 个颜色通道，一般为 RGB。
- RGB, HSV，Lab，CMYK 等。
- 为了增加网络的泛化能力，即使图片只有部分或者翻转了，也依然有能力识别出来。
- 加快模型的训练时候的收敛速度，也可以在一定程度上避免梯度消失或者爆炸。
- SGD, BGD, MBGD, Momentum, NAG(Nesterov Accelerated Gradient)，Adagrad，Adadelta，RMSprop。
- 一个 epoch 指代所有的数据送入网络中完成一次前向计算及反向传播的过程。
- L1，L2 正则化，Early stopping，数据增强，dropout 等。

2.8 开放题

请设计的一个网络让测试集准确率达到 99%以上。