《计算机视觉》

-物体识别



华为技术有限公司

目录

[1 实验介绍 2](#_Toc95470291)

[1.1 实验目的 2](#_Toc95470292)

[1.2 实验清单 2](#_Toc95470293)

[2 CIFAR-10分类任务 3](#_Toc95470294)

[2.1 实验介绍 3](#_Toc95470295)

[2.2 实验环境要求 3](#_Toc95470296)

[2.3 实验目的 3](#_Toc95470297)

[2.4 实验总体设计 4](#_Toc95470298)

[2.5 实验过程 4](#_Toc95470299)

[2.5.1 创建华为云Notebook 4](#_Toc95470300)

[2.5.2 导入相关实验模块 5](#_Toc95470301)

[2.5.3 数据集展示与数据初始化 7](#_Toc95470302)

[2.5.4 构建网络模型 11](#_Toc95470303)

[2.5.5 模型训练与测试 13](#_Toc95470304)

[2.5.6 模型优化与重新训练 17](#_Toc95470305)

[2.5.7 模型测试与可视化 20](#_Toc95470306)

[2.6 实验总结 22](#_Toc95470307)

[2.7 思考题汇总 23](#_Toc95470308)

[2.8 开放题 23](#_Toc95470309)

# 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. 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. 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 |

# CIFAR-10分类任务

## 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CIFAR-10分类任务，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。

## 实验环境要求

Python 3.7.5

Mindspore 1.5

Matplotlib 3.2.2

Numpy 1.18.5

## 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。

## 实验总体设计

## 实验过程

* + 1. 创建华为云Notebook

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”。



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-MindSpore1.5-cann5.0.2-euler2.8-aarch64

规格：Ascend: 1\*Ascend910|CPU:24核96GB

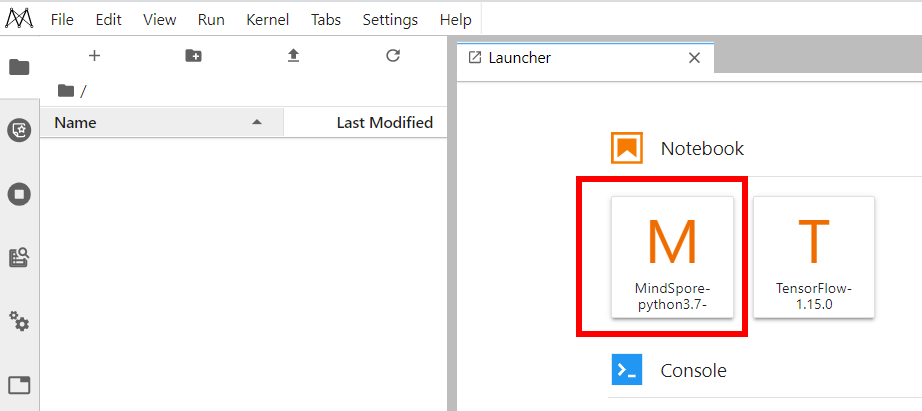
存储配置：云硬盘EVS 规格：5GB

点击“下一步”，确认规格如下后选择提交：



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：



### 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

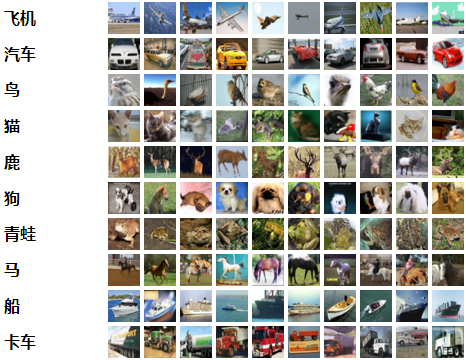
import requests

import zipfile

### 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。

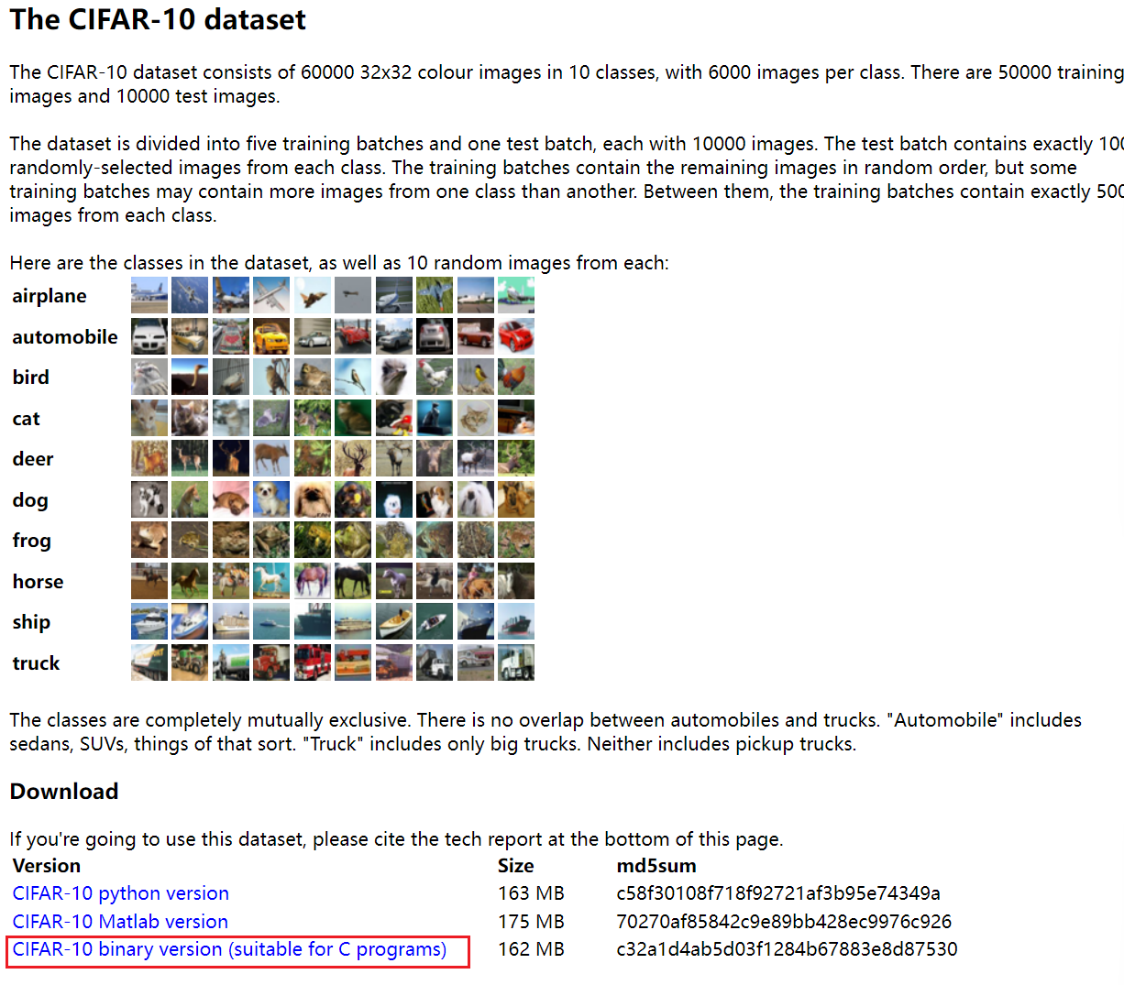
10个类完全相互排斥，且类之间没有重叠，汽车和卡车之间没有重叠。“汽车”包括轿车，SUV等。“卡车”只包括大卡车，不包括皮卡车。  
以下是10个类别类的名字：airplane/automobile/bird/cat/deer/dog/frog/horse/ship/truck



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。



#### 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

#### 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()])

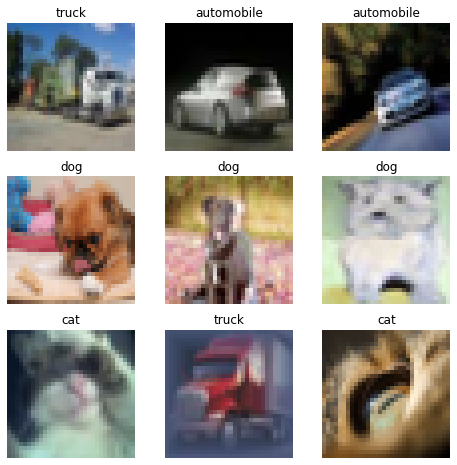
i +=1

if i > 9 :

break

plt.show()

输出：



Cifar10图像示例

#### 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。

#### 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

#### 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)

#### 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？

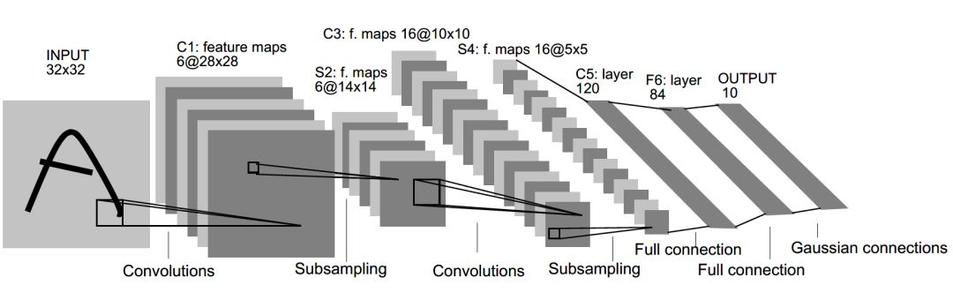
【答案】

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。

### 构建网络模型

#### 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)

### 模型训练与测试

#### 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。

# 设置模型的设备与图的模式

context.set\_context(mode=context.GRAPH\_MODE)

# 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())

#### 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=True)

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}

#### 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=True)

# 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。

#### 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]))

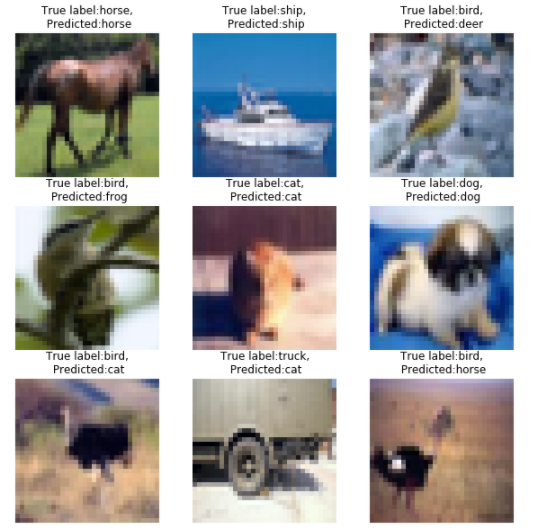
i +=1

if i > 9 :

break

plt.show()

输出：



预测结果

#### 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。

### 模型优化与重新训练

#### 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

#### 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)

# 设置模型的设备与图的模式

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

# 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=True)

输出：

============== Starting Training ==============

epoch: 1 step: 312, loss is 1.7387633

{'Accuracy': 0.44611378205128205}

epoch: 2 step: 312, loss is 1.5458272

{'Accuracy': 0.5018028846153846}

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}

### 模型测试与可视化

#### 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=True)

# 评估测试集

print('test results:',res)

输出：

test results: {'Accuracy': 0.9730568910256411}

#### 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：



预测结果2

## 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。