实验四 深度学习任务

姓名：张辰菁 班级：计算机2103 学号：2215015048

一、实验目的

1.学习使用mindspore.nn库搭建VGG11和Resnet18，并在cifar-10训练集上训练。

2.学习使用mindspore.nn库搭建LSTM网络，并在Cifar-10训练集上训练。

二、实验过程

1.神经网络搭建

1.1 VGG11

VGG网络是经典的卷积神经网络之一，主要的思想是用连续的多个3×3的卷积核代替单层的5×5、7×7卷积核，从而达到减少网络参数、加深网络深度的目的。根据不同的用法，VGG网络有着不同的网络结构：

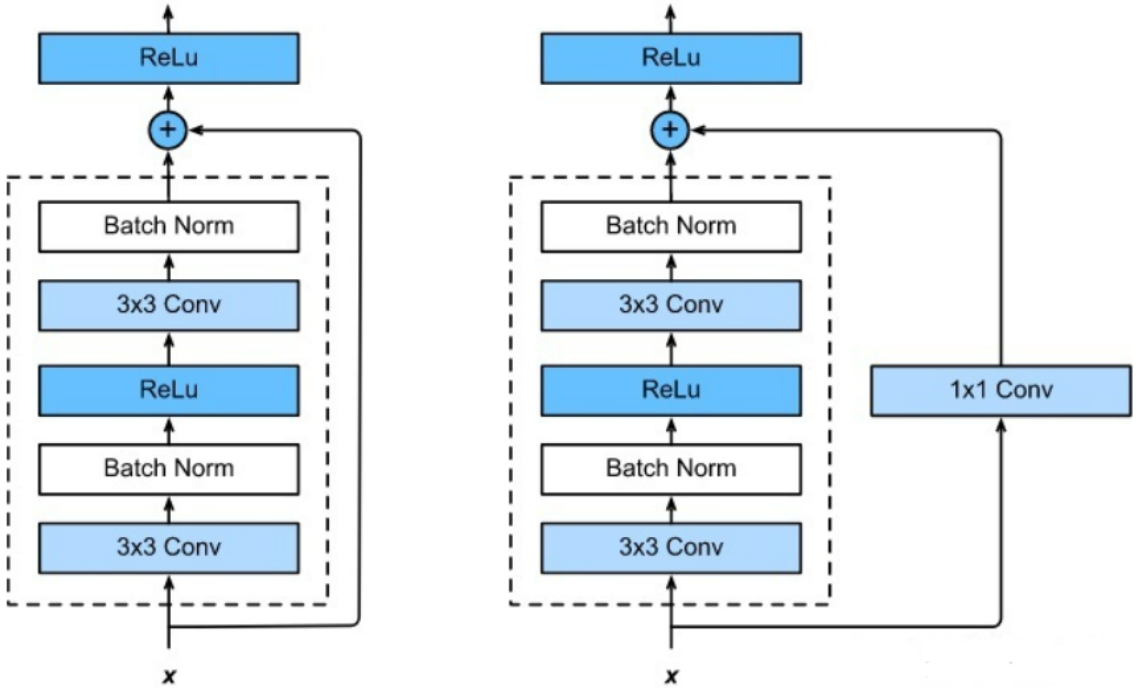
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

本次实验中搭建VGG11网络架构。

1.2 ResNet18

ResNet网络也是经典的卷积网络之一。与VGG中笔直的数据流不同，在ResNet中，独有着“残差连接”这个概念。在某些层中，网络会将未经运算的数据直接加到经过运算的结果上，这部分数据就叫做“残差”。因此，ResNet与其他卷积神经网络相比，更注重了线性变换与非线性变换的平衡，在各种任务中取得了很好的效果，是现在最常用的神经网络之一。常见的ResNet网络有着不同的规格，如ResNet18、ResNet34、ResNet50等。

有残差连接和无残差连接的残差块如下：



常见ResNet网络规格如下：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

本次实验中搭建Resnet18网络架构。

1.3 LSTM

LSTM（Long Short-Term Memory，长短时记忆网络）是一种特殊的循环神经网络（RNN），用于处理并记住长期的序列信息。与传统 RNN 不同，LSTM 在设计上可以更有效地捕捉长距离的依赖关系，防止梯度消失或爆炸的问题。

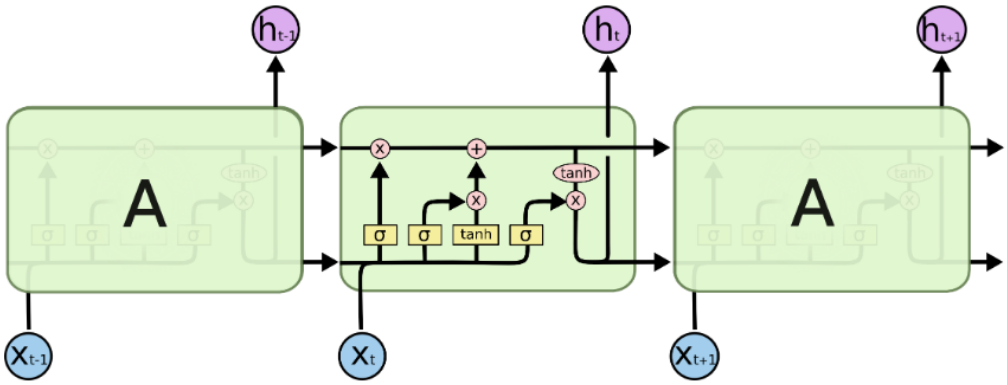
LSTM 由以下几部分组成：

输入门（Input Gate）： 控制新输入的信息进入当前状态的多少。

遗忘门（Forget Gate）： 控制当前状态的哪些部分需要被保留或遗忘。

输出门（Output Gate）： 控制当前状态中的哪些部分将作为当前时间步的输出。

细胞状态（Cell State）： 长期记忆信息的传递，负责保留序列信息。



LSTM 中的重复模块包含四个交互层。

本次实验使用给出的LSTMClassifier类构建LSTM神经网络，主要参数为Input size（输入维度），hidden_size（隐层维度），num_layers（LSTM层数），num_classes（线性层输出维度，即分类的类别数）。

2.数据读取、加载和预处理

使用ds.Cifar10Dataset读取数据集，先定义c_trans（包含归一化、图像增强等函数），然后使用cifar_ds.map(operations=c_trans, input_columns="image", num_parallel_workers=8)或cifar_ds.map(operations=type_cast_op, input_columns="label", num_parallel_workers=8)来对输入数据集进行归一化处理和数据增强，如可以使用裁剪、反转等方法，最后获取数据集的加载器。

3.训练神经网络

1.首先使用上面的方法准备好训练集加载器和测试集加载器。

2.然后设置网络结构（VGG11/ResNet18）、损失函数(crossentropy和优化器（adam），然后调用model = Model(net, loss_fn=loss_fn, optimizer=optimizer, metrics={"Accuracy": Accuracy()})函数来初始化模型。

3.之后调用model.train(epoch_size, ds_train, callbacks=[ckpoint_cb, loss_cb, step_loss_acc_info], dataset_sink_mode=False)来进行训练，其中epoch_size为训练的epoch数，ds_train为训练集加载器，callbacks为训练过程中打印loss信息、保存模型参数等操作，dataset_sink_mode为是否开启数据下沉模式，影响的是中间的loss等信息是否打印出来，如果为True则只会打印最后一个要求打印的step的相关信息。

4.分类结果评价

使用正确率来对分类结果进行评价。绘制训练过程中的学习曲线来观察神经网络的训练情况。

正确率即为所有样本中预测类别正确所占比例。

使用res = model.eval(ds_test) print("result: ", res)，查看模型在测试集上的正确率。

保存每一个epoch指定steps的loss和正确率，使用matplotlib绘制学习曲线查看网络的训练情况。

三、实验内容及实验结果

1.VGG11

按照图中的架构搭建VGG11网络。

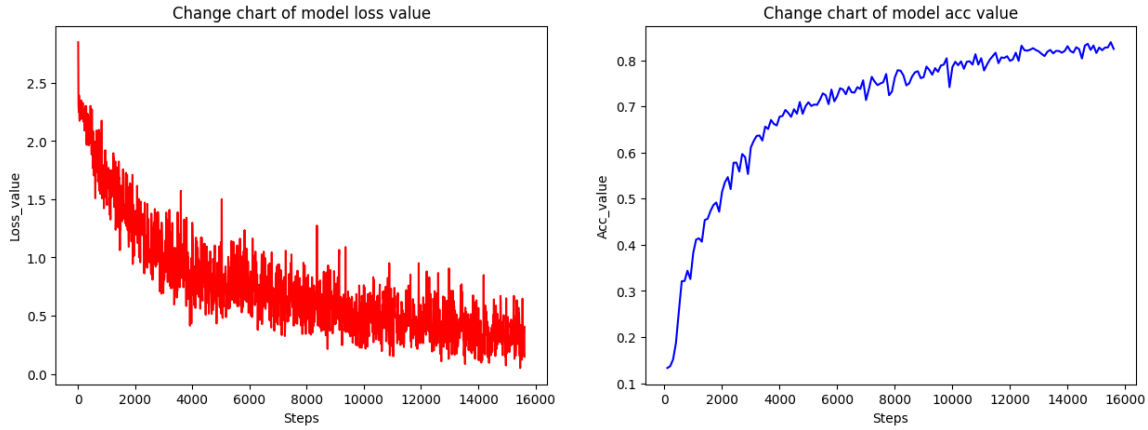
```
import mindspore.nn as nn


class VGG11(nn.Cell):
    """
    网络结构
    """
    '''
    任务一补全
    '''
    def __init__(self, num_class=10, num_channel=3):
        super(VGG11,self).__init__()
        self.conv1 = self.conv_block(num_channel, 64)
        self.conv2 = self.conv_block(64, 128)
        self.conv3 = self.conv_block(128, 256)
        self.conv4 = self.conv_block(256,256)
        self.conv5 = self.conv_block(256, 512)
        self.conv6 = self.conv_block(512, 512)
        self.conv7 = self.conv_block(512, 512)
        self.conv8 = self.conv_block(512, 512)
        self.fc9 = nn.Dense(25088, 6272)
        self.fc10 = nn.Dense(6272, 1568)
        self.fc11 = nn.Dense(1568, num_class)
        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)

    def conv_block(self, in_channel, out_channel):
        seq = nn.SequentialCell(
            [
                nn.Conv2d(in_channel, out_channel, 3, padding=1,
pad_mode='pad'),
                nn.BatchNorm2d(out_channel),
                nn.ReLU()
            ])
        return seq

    def construct(self, x):
        # 使用定义好的运算构建前向网络
        x = self.conv1(x)
        x = self.max_pool2d(x)

        x = self.conv2(x)
        x = self.max_pool2d(x)

        x = self.conv3(x)
        x = self.conv4(x)
        x = self.max_pool2d(x)

        x = self.conv5(x)
        x = self.conv6(x)
        x = self.max_pool2d(x)

        x = self.conv7(x)
        x = self.conv8(x)
        x = self.max_pool2d(x)
```

```
        x = self.flatten(x)

        x = self.fc9(x)
        x = self.relu(x)
        x = self.fc10(x)
        x = self.relu(x)
        x = self.fc11(x)

        return x
```

训练10个epoch。

训练结果

```
epoch: 10 step: 1420, loss is 0.19228297472000122
epoch: 10 step: 1562, loss is 0.14636282622814178
result:  {'Accuracy': 0.8299278846153846}
```



可以看到开始模型的拟合速度较快，准确率上升和loss下降幅度都比较快，越往后，loss和准确率的整体下降/上升幅度都变慢。并且，loss和准确率是波动变化的，而不是只上升/下降。

2.ResNet18

按照图中的架构搭建ResNet18网络。

```
import mindspore.nn as nn


class ResBlockDS(nn.Cell):
#在这里进行补全！！
    def __init__(self, in_channel, out_channel):
        super(ResBlockDS, self).__init__()
        self.seq = nn.SequentialCell(
            [
                nn.Conv2d(in_channel, out_channel, 3, stride=2, padding=1,
pad_mode='pad'),
                nn.BatchNorm2d(out_channel),
                nn.ReLU(),
                nn.Conv2d(out_channel, out_channel, 3, stride=1, padding=1,
pad_mode='pad'),
                nn.BatchNorm2d(out_channel),
            ])
        self.relu = nn.ReLU()

        self.shortup = nn.SequentialCell(
            [
                nn.Conv2d(in_channel, out_channel, 1, stride=2),
```

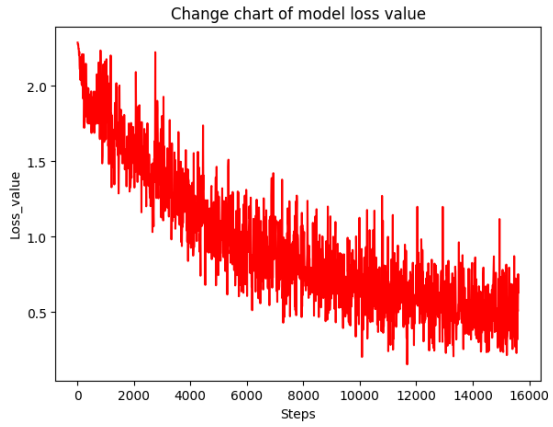
```
                nn.BatchNorm2d(out_channel),
            ])
    def construct(self, x):
        y = self.seq(x) + self.shortup(x)
        #先relu后maxpooling
        y = self.relu(y)
        return y

class ResBlock(nn.Cell):
#在这里进行补全！！
    def __init__(self, in_channel, out_channel):
        super(ResBlock, self).__init__()
        self.seq = nn.SequentialCell(
            [
                nn.Conv2d(in_channel, out_channel, 3, stride=1, padding=1,
pad_mode='pad'),
                nn.BatchNorm2d(out_channel),
                nn.ReLU(),
                nn.Conv2d(out_channel, out_channel, 3, stride=1, padding=1,
pad_mode='pad'),
                nn.BatchNorm2d(out_channel),
            ])
        self.relu = nn.ReLU()

    def construct(self, x):
        y = self.seq(x)

        y = self.relu(y)
        return y

class ResNet18(nn.Cell):
    def __init__(self, class_num=10, in_channel=3):
        super(ResNet18, self).__init__()
        self.conv = nn.Conv2d(in_channel, 64, 7, stride=2, padding=3,
pad_mode='pad')
        self.max_pool = nn.MaxPool2d(kernel_size=3, stride=2)
        self.ResBlock1 = ResBlock(64, 64)
        self.ResBlock2 = ResBlock(64, 64)

        self.ResBlockDS3 = ResBlockDS(64, 128)

        self.ResBlock4 = ResBlock(128, 128)
        self.ResBlockDS5 = ResBlockDS(128, 256)

        self.ResBlock6 = ResBlock(256, 256)
        self.ResBlockDS7 = ResBlockDS(256, 512)
        self.ResBlock8 = ResBlock(512, 512)
        self.LastProcess = nn.SequentialCell([
            nn.AvgPool2d(7),
            nn.Flatten(),
            nn.Dense(512, 1024),
            nn.ReLU(),
            nn.Dense(1024, 512),
            nn.ReLU(),
            nn.Dropout(keep_prob=0.5),
            nn.Dense(512, class_num),
        ])
```

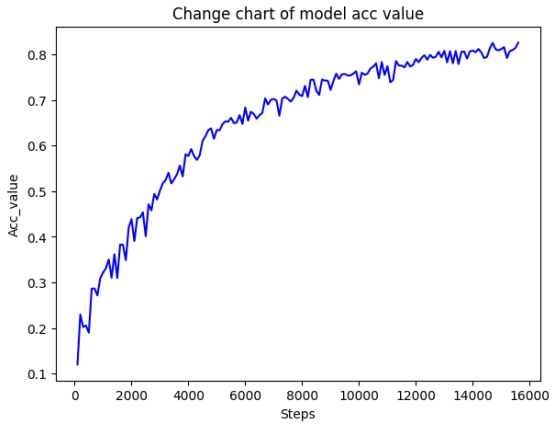
```
    def construct(self, x):
        x = self.conv(x)
        x = self.max_pool(x)
        x = self.ResBlock1(x)
        x = self.ResBlock2(x)
        x = self.ResBlockDS3(x)
        x = self.ResBlock4(x)
        x = self.ResBlockDS5(x)
        x = self.ResBlock6(x)
        x = self.ResBlockDS7(x)
        x = self.ResBlock8(x)
        y = self.LastProcess(x)
        return y
```

训练10个epoch。

训练结果

```
epoch: 10 step: 1278, loss is 0.48370566964149475
epoch: 10 step: 1420, loss is 0.47877851128578186
epoch: 10 step: 1562, loss is 0.6301215887069702
result:  {'Accuracy': 0.8135016025641025}
```





可以发现Resnet18的训练结果与VGG11的训练结果相差无几。同样，可以看到开始模型的拟合速度较快，准确率上升和loss下降幅度都比较快，越往后，loss和准确率的整体下降/上升幅度都变慢。并且，loss和准确率是波动变化的，而不是只上升/下降。训练时间上，VGG11慢于ResNet18。

3.LSTM

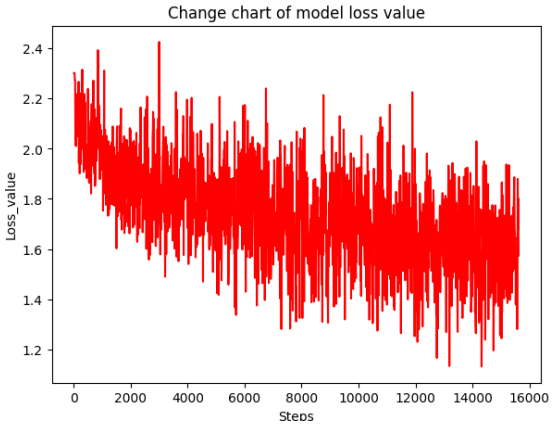
按照所给函数搭建LSTM网络。

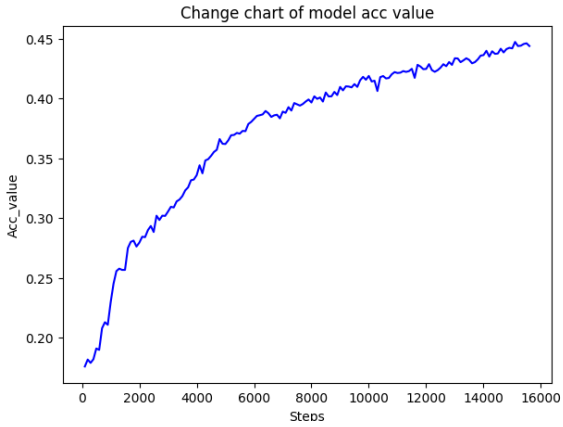
```
# 创建 LSTM 模型
class LSTMClassifier(nn.Cell):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(LSTMClassifier, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers=num_layers,
batch_first=True)
        self.fc = nn.Dense(hidden_size, num_classes)

    def construct(self, x):
        # 将图像展平到序列形式
        batch_size, channels, height, width = x.shape
        flattened_input_size = channels * height * width

        # 展平数据并将其转换为 LSTM 所期望的形状
        x = x.view(batch_size, -1, flattened_input_size)

        # LSTM 前向传播
```

```
        lstm_out, _ = self.lstm(x)

        # 取最后一个时间步的输出进行分类
        output = self.fc(lstm_out[:, -1, :])

        return output

# 实例化 LSTM 模型
net=LSTMClassifier(3072,100,5,10)
# 定义损失函数和优化器
loss_func = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
optimizer = nn.Adam(net.trainable_params(), learning_rate=1e-3)

# 创建模型
model = Model(net, loss_fn=loss_func, optimizer=optimizer, metrics={"Accuracy":
Accuracy()})
# 训练模型
epoch_size = 10

# 设置模型保存的路径，删除掉之前训练保存的模型
model_path = "./models/ckpt/mindspore_vision_application/"
os.system('rm -f {0}*.ckpt {0}*.meta {0}*.pb'.format(model_path))
# 得到每个batch的训练步数
batch_num = train_dataset.get_dataset_size()
# 设置与模型保存相关的参数
net_type = 'LSTM'
config_ck = CheckpointConfig(save_checkpoint_steps=batch_num,
keep_checkpoint_max=1)
ckpoint_cb = ModelCheckpoint(prefix="train_" + net_type + "_cifar10",
directory=model_path, config=config_ck)
# 设置损失记录器，参数是打印Loss信息的步长
loss_cb = LossMonitor(200)
# 设置记录损失的数据结构，用于在后面打印曲线图像
steps_loss = {"step": [], "loss_value": []}
steps_eval = {"step": [], "acc": []}
step_loss_acc_info = StepLossAccInfo(model, test_dataset, steps_loss,
steps_eval)
# 执行模型的训练
print("begin training...")
model.train(epoch_size, train_dataset, callbacks=[ckpoint_cb, loss_cb,
step_loss_acc_info], dataset_sink_mode=False)
# model.train(epoch_size, ds_train, callbacks=[ckpoint_cb, loss_cb,
step_loss_acc_info])
# 训练完后，对模型进行测试，打印正确率
res = model.eval(test_dataset)
print("result: ", res)
```

训练结果

```
epoch: 10 step: 1142, loss is 1.5775002241134644
epoch: 10 step: 1342, loss is 1.6633403301239014
epoch: 10 step: 1542, loss is 1.6749581098556519
result:  {'Accuracy': 0.44224151728553135}
```





可以看到LSTM对于分类任务的训练结果明显差于VGG11和ResNet18，这是因为对于分类任务来说不存在上一个输入对下一个输入的影响，所以LSTM的网络架构并不适合分类任务的训练。

同样，可以看到开始模型的拟合速度较快，准确率上升和loss下降幅度都比较快，越往后，loss和准确率的整体下降/上升幅度都变慢。并且，loss和准确率是波动变化的，而不是只上升/下降。

4.加载训练好的模型

使用load_param_into_net()函数加载训练好的参数到对应模型中，然后使用该模型进行对应操作。

下面以VGG11为例,加载模型后，在测试集上查看准确率。

```
ds_test_path = "./datasets/cifar10/test/"
vgg = VGG11()
loss = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

#将模型参数存入parameter的字典中

param_dict =
load_checkpoint("models/ckpt/mindspore_vision_application/train_vgg11_cifar10-10_1562.ckpt")

#将参数加载到网络中

load_param_into_net(vgg, param_dict)
model = Model(vgg, loss, metrics={"accuracy"})
dataset_eval = create_dataset(ds_test_path)
acc = model.eval(dataset_eval)
print(acc)
```

测试结果

```
param_dict = load_checkpoint("models/ckpt/mindspore_vision_applic
# 将参数加载到网络中
load_param_into_net(vgg, param_dict)
model = Model(vgg, loss, metrics={"accuracy"})
dataset_eval = create_dataset(ds_test_path)
acc = model.eval(dataset_eval)
print(acc)
```

```
{'accuracy': 0.8293269230769231}
```

可以看到结果与先前的测试结果相差无几。

也可以用这个方法对任意真实图片进行预测，但需要提前对真实图片进行resize，归一化等操作。

四、总结

1.本次实验通过使用mindspore.nn库搭建VGG11、ResNet18和LSTM网络进行训练。

2.在搭建神经网络的过程中一个比较容易出错的地方就是在卷积完进入全连接层时对于输入flatten后的维度不太了解而导致无法进行计算，一个可行的方法是在flatten之后立即打印x.shape,根据打印信息修改神经网络，再进行训练，可以免去繁杂的计算过程。

3.如果在训练过程中报错，而根据报错信息修改相关代码后仍然报相同的错，这可能是没有手动重启内核造成的，在保存代码后，重启内核后再次尝试，一般可以解决上述问题。

五、完整源代码

```
import mindspore.nn as nn
from mindspore import dtype as mstype
import mindspore.dataset as ds
import mindspore.dataset.vision.c_transforms as C
import mindspore.dataset.transforms.c_transforms as C2
from mindspore import context
import matplotlib.pyplot as plt
from mindspore.train.callback import Callback
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig,
LossMonitor
import os
from mindspore import Model
from mindspore.nn import Accuracy
from mindspore.nn import SoftmaxCrossEntropyWithLogits
from VGG11 import VGG11
from Resnet18 import ResNet18

context.set_context(mode=context.GRAPH_MODE, device_target="GPU")


def create_dataset(data_home, repeat_num=1, batch_size=32, do_train=True,
device_target="GPU"):
    """
    create data for next use such as training or inferring
    """

    cifar_ds = ds.Cifar10Dataset(data_home, num_parallel_workers=8,
shuffle=True)

    c_trans = []
    if do_train:
        c_trans += [
            C.Resize((224, 224)),
            C.RandomHorizontalFlip(prob=0.5)
        ]

    c_trans += [
        C.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010]),
        C.HWC2CHW()
    ]
```

```
    type_cast_op = C2.TypeCast(mstype.int32)

    #规范输出格式
    cifar_ds = cifar_ds.map(operations=type_cast_op, input_columns="label",
num_parallel_workers=8)
    cifar_ds = cifar_ds.map(operations=c_trans, input_columns="image",
num_parallel_workers=8)

    cifar_ds = cifar_ds.batch(batch_size, drop_remainder=True)
    cifar_ds = cifar_ds.repeat(repeat_num)

    return cifar_ds


class StepLossAccInfo(Callback):
    def __init__(self, model, eval_dataset, steps_loss, steps_eval):
        self.model = model
        self.eval_dataset = eval_dataset
        self.steps_loss = steps_loss
        self.steps_eval = steps_eval

    def step_end(self, run_context):
        cb_params = run_context.original_args()
        # cur_epoch = cb_params.cur_epoch_num
        # cur_step = (cur_epoch-1)*1562 + cb_params.cur_step_num
        cur_step = cb_params.cur_step_num
        if cur_step % 10 == 0:
            self.steps_loss["loss_value"].append(str(cb_params.net_outputs))
            self.steps_loss["step"].append(str(cur_step))
        if cur_step % 100 == 0:
            acc = self.model.eval(self.eval_dataset, dataset_sink_mode=False)
            self.steps_eval["step"].append(cur_step)
            self.steps_eval["acc"].append(acc["Accuracy"])


def train(net_type, epoch_size):
    # 设置网络结构
    if net_type == "vgg11":
        net = VGG11()
    elif net_type == 'resnet18':
        net = ResNet18()
    # 设置损失函数和优化器
    loss_fn = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
    optimizer = nn.Adam(net.trainable_params(), learning_rate=1e-3)
    # 初始化模型
    model = Model(net, loss_fn=loss_fn, optimizer=optimizer, metrics=
{"Accuracy": Accuracy()})
    # 设置训练数据集和测试数据集
    ds_train_path = "./datasets/cifar10/train/"
    ds_test_path = "./datasets/cifar10/test/"
    ds_train = create_dataset(ds_train_path)
    ds_test = create_dataset(ds_test_path)
    # 设置模型保存的路径，删除掉之前训练保存的模型
    model_path = "./models/ckpt/mindspore_vision_application/"
    os.system('rm -f {0}*.ckpt {0}*.meta {0}*.pb'.format(model_path))
    # 得到每个batch的训练步数
    batch_num = ds_train.get_dataset_size()
    # 设置与模型保存相关的参数
```

```
    config_ck = CheckpointConfig(save_checkpoint_steps=batch_num,
keep_checkpoint_max=1)
    ckpoint_cb = ModelCheckpoint(prefix="train_" + net_type + "_cifar10",
directory=model_path, config=config_ck)
    # 设置损失记录器，参数是打印Loss信息的步长，142的意思是每142步打印一次loss信息
    loss_cb = LossMonitor(142)
    # 设置记录损失的数据结构，用于在后面打印曲线图像
    steps_loss = {"step": [], "loss_value": []}
    steps_eval = {"step": [], "acc": []}
    step_loss_acc_info = StepLossAccInfo(model, ds_test, steps_loss, steps_eval)
    # 执行模型的训练
    print("begin training...")
    model.train(epoch_size, ds_train, callbacks=[ckpoint_cb, loss_cb,
step_loss_acc_info], dataset_sink_mode=False)
    # model.train(epoch_size, ds_train, callbacks=[ckpoint_cb, loss_cb,
step_loss_acc_info])
    # 训练完后，对模型进行测试，打印正确率
    res = model.eval(ds_test)
    print("result: ", res)
    # 利用训练中保存的信息，打印训练损失曲线和训练正确率曲线
    steps = steps_loss["step"]
    loss_value = steps_loss["loss_value"]
    steps = list(map(int, steps))
    loss_value = list(map(float, loss_value))
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 2, 1)
    plt.plot(steps, loss_value, color="red")
    plt.xlabel("Steps")
    plt.ylabel("Loss_value")
    plt.title("Change chart of model loss value")
    steps = steps_eval["step"]
    acc_value = steps_eval["acc"]
    steps = list(map(int, steps))
    loss_value = list(map(float, acc_value))
    plt.subplot(1, 2, 2)
    plt.plot(steps, acc_value, color="blue")
    plt.xlabel("Steps")
    plt.ylabel("Acc_value")
    plt.title("Change chart of model acc value")
    plt.show()
```

```
train("resnet18", 10)
```

```
train("vgg11", 10)
```

```
from mindspore.train.callback import ModelCheckpoint
from mindspore import load_checkpoint, load_param_into_net
from mindspore import context
from VGG11 import VGG11
from Resnet18 import ResNet18
from mindspore import Model
from mindspore.nn import Accuracy
from mindspore.nn import SoftmaxCrossEntropyWithLogits
context.set_context(mode=context.GRAPH_MODE, device_target="GPU")
ds_test_path = "./datasets/cifar10/test/"
```

```
vgg = VGG11()
loss = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
# 将模型参数存入parameter的字典中
param_dict =
load_checkpoint("models/ckpt/mindspore_vision_application/train_vgg11_cifar10-10_1562.ckpt")
# 将参数加载到网络中
load_param_into_net(vgg, param_dict)
model = Model(vgg, loss, metrics={"accuracy"})
dataset_eval = create_dataset(ds_test_path)
acc = model.eval(dataset_eval)
print(acc)
```

```
import mindspore.nn as nn
from mindspore import  context
from mindspore import dtype as mstype
import mindspore.dataset as ds
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as VC
from mindspore.train import Model
from mindspore.train.callback import LossMonitor
from mindspore.nn import SoftmaxCrossEntropyWithLogits
from mindspore.nn import Accuracy
import numpy as np
from mindspore.train.callback import Callback
import matplotlib.pyplot as plt


from mindspore.train.callback import ModelCheckpoint, CheckpointConfig
import os
```

```
'''
任务四补全
'''
# 实例化 LSTM 模型
net=LSTMClassifier(3072,100,5,10)
# 定义损失函数和优化器
loss_func = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
optimizer = nn.Adam(net.trainable_params(), learning_rate=1e-3)

# 创建模型
model = Model(net, loss_fn=loss_func, optimizer=optimizer, metrics={"Accuracy": Accuracy()})
# 训练模型
epoch_size = 10

# 设置模型保存的路径，删除掉之前训练保存的模型
model_path = "./models/ckpt/mindspore_vision_application/"
os.system('rm -f {0}*.ckpt {0}*.meta {0}*.pb'.format(model_path))
# 得到每个batch的训练步数
batch_num = train_dataset.get_dataset_size()
# 设置与模型保存相关的参数
net_type = 'LSTM'
config_ck = CheckpointConfig(save_checkpoint_steps=batch_num, keep_checkpoint_max=1)
```

```
ckpoint_cb = ModelCheckpoint(prefix="train_" + net_type + "_cifar10",
directory=model_path, config=config_ck)
# 设置损失记录器，参数是打印Loss信息的步长
loss_cb = LossMonitor(200)
# 设置记录损失的数据结构，用于在后面打印曲线图像
steps_loss = {"step": [], "loss_value": []}
steps_eval = {"step": [], "acc": []}
step_loss_acc_info = StepLossAccInfo(model, test_dataset, steps_loss,
steps_eval)
# 执行模型的训练
print("begin training...")
model.train(epoch_size, train_dataset, callbacks=[ckpoint_cb, loss_cb,
step_loss_acc_info], dataset_sink_mode=False)
# model.train(epoch_size, ds_train, callbacks=[ckpoint_cb, loss_cb,
step_loss_acc_info])
# 训练完后，对模型进行测试，打印正确率
res = model.eval(test_dataset)
print("result: ", res)
# 利用训练中保存的信息，打印训练损失曲线和训练正确率曲线
steps = steps_loss["step"]
loss_value = steps_loss["loss_value"]
steps = list(map(int, steps))
loss_value = list(map(float, loss_value))
plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
plt.plot(steps, loss_value, color="red")
plt.xlabel("Steps")
plt.ylabel("Loss_value")
plt.title("Change chart of model loss value")
steps = steps_eval["step"]
acc_value = steps_eval["acc"]
steps = list(map(int, steps))
loss_value = list(map(float, acc_value))
plt.subplot(1, 2, 2)
plt.plot(steps, acc_value, color="blue")
plt.xlabel("Steps")
plt.ylabel("Acc_value")
plt.title("Change chart of model acc value")
plt.show()
```