

基于 Lenet 的 CIFAR-10 分类任务

1、实验目的

- ① 学会使用 MindSpore 进行简单卷积神经网络的开发。
- ② 学会使用 MindSpore 进行 CIFAR-10 数据集分类任务的训练和测试。
- ③ 可视化学习到的特征表达器，和手工定义的特征进行分析和比较









2、实验原理

① MindSpore

MindSpore 是华为在 2019 年发布的，并于 2020 年正式开源的全场景 AI 计算框架。是一个较为年轻的框架，框架的开发态友好（例如显著减少训练时间和成本）和运行态高效（例如最少资源和最高能效比）、适应每个场景（包括端、边缘和云）这三个方面有较大进步，

② CIFAR-10 数据集

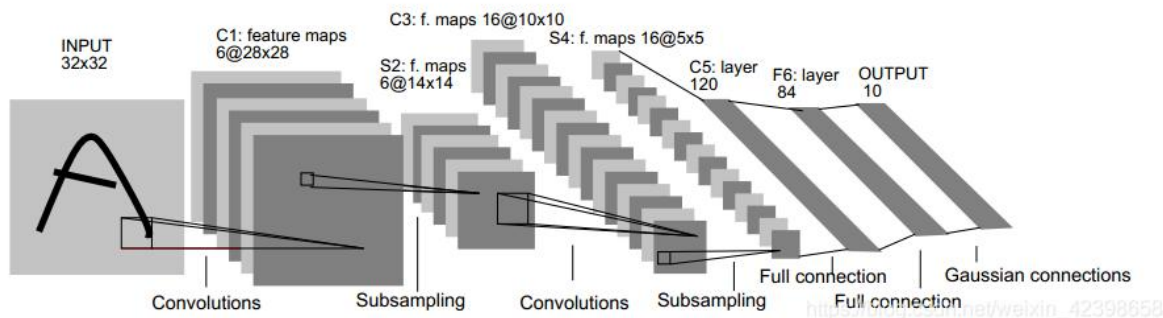
CIFAR-10 是一个更接近普适物体的彩色图像数据集。一共包含 10 个类别的 RGB 彩色图片：飞机（ airplane ）、汽车（ automobile ）、鸟类（ bird ）、猫（ cat ）、鹿（ deer ）、狗（ dog ）、蛙类（ frog ）、马（ horse ）、船（ ship ）和卡车（ truck ）。每个图片的尺寸为 32×32 ，每个类别有 6000 个图像，数据集中一共有 50000 张训练图片和 10000 张测试图片。

名称	修改日期	类型	大小
 batches.meta.txt	2009/6/5 3:44	文本文档	1 KB
 data_batch_1.bin	2009/6/5 3:36	BIN 文件	30,010 KB
 data_batch_2.bin	2009/6/5 3:37	BIN 文件	30,010 KB
 data_batch_3.bin	2009/6/5 3:38	BIN 文件	30,010 KB
 data_batch_4.bin	2009/6/5 3:37	BIN 文件	30,010 KB
 data_batch_5.bin	2009/6/5 3:35	BIN 文件	30,010 KB
 readme.html	2009/6/5 4:46	Microsoft Edge ...	1 KB
 test_batch.bin	2009/6/5 3:36	BIN 文件	30,010 KB

打开下载的 CIFAR-10 数据集可以看到里面有六个 bin 文件。文件 data_batch_1.bin、data_batch_2.bin 、 、 data_batch_5.bin 和 test_batch.bin 中各有 10000 个样本。一个样本由 3073 个字节组成，第一个字节为标签 label，剩下 3072 个字节为图像数据。样本和样本之间没高多余的字节分割，因此这几个二进制文件的大小都是 30730000 字节。

③ LeNet 网络

LeNet 网络是一个比较简单的卷积神经网络，出自论文 Gradient-Based Learning Applied to Document Recognition，将一个输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用 softmax 分类作为输出层。



(LeNet 网络结构图)

可以看到上面一共有八层网络结构。

A. 输入层

输入层没什么好说的，输入是一张图像，注意图中的图像是一张单通道的手写字符图像，不过在本次实验中我们要使用的是 32×32 的三通道的彩色图像。

B. 卷积层 C1

卷积层的作用是提取特征。这里可以看到生成了六个特征平面，且尺寸大小由 32×32 变为了 28×28 ，故可知卷积核的尺寸为 5×5

C. 池化层 S2

池化的作用是降低数据的维度。原来是 28×28 的，采样后就是 14×14 ，故可知使用了 2×2 进行采样，一共有六个采样平面。

D. 卷积层 C3

这里仍然是一个卷积层，但是我们观察后发现生成了 16 个特征平面，是怎样从 6 个池化平面生成了 16 个特征平面的呢。我们把 C3 的卷积层特征平面编号即 0, 1, 2, ..., 15，把池化层 S2 也编号为 0, 1, 2, 3, 4, 5，其对应关系如下：

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

由于池化层的尺寸为 14×14 卷积后的大小为 10×10 ，可知，卷积核的大小为 5×5 。上图我们可以清楚的看到前 6 个特征平面对应池化层的三个平面即 0, 1, 2, 3, 4, 5，而 6 到 14 每张特征平面对应 4 个卷积层，此时每个特征平面的一个神经元的连接数为 $5 \times 5 \times 4$ ，最后一个特征平面是对应池化层所有的样本平面。

这样设计的**目的**为：主要是为了打破对称性，提取深层特征，因为特征不是对称的，因此需要打破这种对称，以提取到更重要的特征

E. 池化层 S4

这里同样是进行 2×2 进行采样，将尺寸变为 5×5 。

F. 卷积层 C5

这里进行了卷积后，变成了长度为 120 的顺序排列的神经元。这里是因为使用了 5×5 大小的卷积核，该卷积核与池化层的所有层连接，卷积后生成了一个神经元。这样最终的结果为 120 个神经元进行排列。

G. 全连接层 F6

这里是一个全连接层，将 120 大小的维度转化为 84 大小的维度。

H. 输出层

输出层最终输出 10 个结果，输出的结果为 10 个概率，我们选取概率最大的作为我们的最总输出。

3、实验过程

1、数据集的下载

因为这是一个很常用的数据集，非常成熟，直接从官网下载并解压就行。

2、代码编写

① Mindspore 库的下载

在 mindspore 官网上可以下载比较根据其提供的命令下载 mindspore 不过在 window 系统上只有 CPU 版本。这里只支持在 Python3.9 版本安装 mindspore1.5 版本和在 Python3.7.5 上安装 1.3 版本。

② 导入实验相关模块

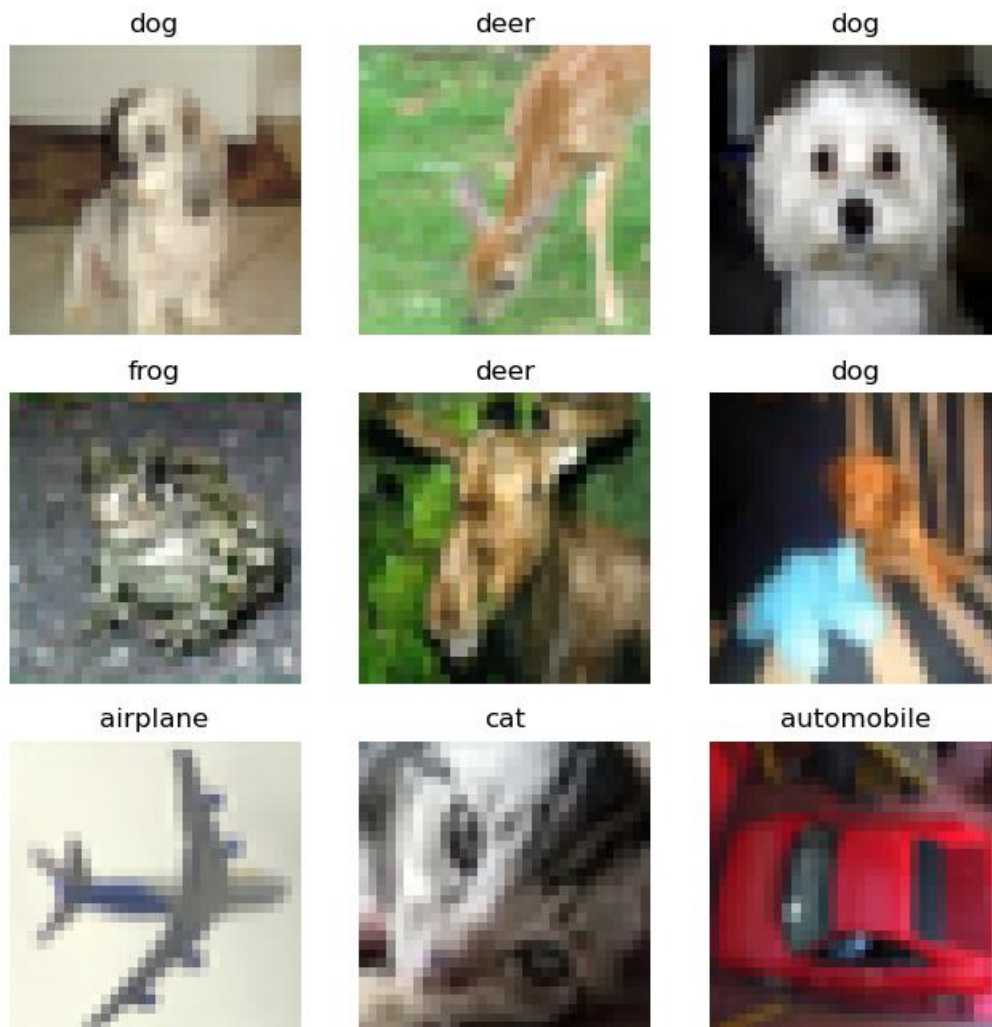
```
import numpy as np
import matplotlib.pyplot as plt
import mindspore # 载入mindspore的默认数据集
import mindspore.nn as nn # 各类网络层都在nn里面
from mindspore import Tensor # mindspore的tensor
from mindspore import context # 设置mindspore运行的环境
import mindspore.dataset as ds # 常用转化用算子
from mindspore.train import Model # 引入模型
from mindspore.nn.metrics import Accuracy # 引入评估模型的包
from mindspore.common import dtype as mstype
import mindspore.dataset.vision.c_transforms as CV
import mindspore.dataset.transforms.c_transforms as C # 图像转化用算子
from mindspore.common.initializer import TruncatedNormal # 参数初始化的方式
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor
```

③ 查看数据集图像

我们的数据集已经下载成功并放在了与该 python 文件统计的目录下，所以可以先加载出其中的几个图片简单观察以下

```
# 定义图像和标签
images = data['image']
labels = data['label']
# 设置图像大小
plt.figure(figsize=(8,8))
count = 1
for img in images:
    plt.subplot(3, 3, count)
    picture_show = np.transpose(img.asnumpy(), (1, 2, 0))
    picture_show = picture_show/np.amax(picture_show)
    picture_show = np.clip(picture_show, 0, 1)
    plt.imshow(picture_show)
    plt.xticks([])
    plt.axis("off")
    plt.title(category_dict[int(labels[count-1].asnumpy())])
    count += 1
    if count > 9 :
        break
plt.show()
```

效果图：



④ 定义数据预处理函数

A. 数据加载函数

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

封装成 `get_data` 函数来获取指定路径中的 `cifar-10` 数据

B. 数据处理函数


```
def process_dataset(cifar_ds, batch_size=32, status="train"):
    # 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)

    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
```

对数据进行了归一化，随机翻转，随机裁剪等处理

⑤ 构建网络模型

```
class LeNet5(nn.Cell):
    """
    LeNet网络结构
    """
    def __init__(self, num_class=10, num_channel=3):
        super(LeNet5, self).__init__()
        # 定义所需要的运算
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        self.fc1 = nn.Dense(16 * 5 * 5, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

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

这里根据 Lenet-5 的网络结构一点点搭建即可

⑥ 模型的训练与测试

A. 训练集数据集的加载与处理

```
data_path='cifar-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)
```

这里对从路径中加载训练数据集，并使用上面定义的预处理函数进行数据的处理

B. 模型参数的设置

```
# 设置模型的设备与图的模式
context.set_context(mode=context.GRAPH_MODE, device_target='CPU')
# 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)
# 监控每个epoch训练的时间
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
```

这里设置了模型在 CPU 上计算（这是在自己的电脑上跑，所以设置为 CPU，切换到华为云上时会改回去），同时定义了模型的损失函数和优化器

C. 模型的保存与建立

```
# 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)
ckptpoint_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()})
```

这里定义了模型的保存，其保存步长设置为 1562，同时保存的最大数量为 10，最后一行使用 Model 函数创建了一个 lenet-5 网络结构的模型

D. 模型的训练与测试

```
model.train(10, ds_train,callbacks=[time_cb, ckpoint_cb, LossMonitor(per_print_times=200)],dataset_sink_mode=False)
model.eval(ds_train, dataset_sink_mode=False)

data_path='test'
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=False)
print(res)
```

在训练集上进行训练，上面代码中的 epoch=10，当然在最后训练时，这个少了点，加载到华为云上时会适当增加。训练集上训练结束后会加载测试集，对准确度进行测试。

至此，一个简单的网络训练和测试已经完成，后面进行一些细节的丰富与补充。

3、实验补充

① 预测结果可视化

```
def visualize_model(best_ckpt_path, val_ds):
    # 定义网络并加载参数，对验证集进行预测
    net = LeNet5_2()
    param_dict = load_checkpoint(best_ckpt_path)
    load_param_into_net(net, param_dict)
    loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
    model = Model(net, loss, metrics={"Accuracy": nn.Accuracy()})
    data = next(val_ds.create_dict_iterator())
    images = data["image"].asnumpy()
    labels = data["label"].asnumpy()
    class_name = {0: "dogs", 1: "wolves"}
    output = model.predict(Tensor(data["image"]))
    pred = np.argmax(output.asnumpy(), axis=1)

    # 可视化模型预测
    for i in range(len(labels)):
        plt.subplot(3, 8, i+1)
        color = 'blue' if pred[i] == labels[i] else 'red'
        plt.title('pre:{}'.format(class_name[pred[i]]), color=color)
        picture_show = np.transpose(images[i], (1, 2, 0))
        picture_show = picture_show / np.amax(picture_show)
        picture_show = np.clip(picture_show, 0, 1)
        plt.imshow(picture_show)
        plt.axis('off')
    plt.show()
```

这里的代码实现的功能是从测试集中加载出 9 张图像，做出预测，然后我们会进行预测结果和实际结果之间的比较，如果预测正确了，那我们输出的标签为蓝色，预测错误输出标签颜色则为红色。

② 网络结构优化

```
class LeNet5_2(nn.Cell):
    def __init__(self, num_class=10, channel=3):
        super(LeNet5_2, self).__init__()
        self.num_class = num_class
        self.conv1 = nn.Conv2d(channel, 64, 3, pad_mode='valid')
        self.conv2 = nn.Conv2d(64, 128, 3, pad_mode='valid')
        self.conv3 = nn.Conv2d(128, 128, 3, pad_mode='valid')
        self.fc1 = nn.Dense(128 * 2 * 2, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(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.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
```


经过多次训练后，发现最终得到的效果始终不好，在网上查阅资料后得知，可以将网络结构进行一些小小的改进，改进后的结果如上，可以看到与第一次的网络结构相比，这里卷积核的大小由 5×5 变为了 3×3 ，每一次卷积时使用的卷积核数目也更多了，同时网络结构也有原来的两个卷积层，变为了现在的三个卷积层。

③ 模型准确度可视化

```
# 每一个epoch后，打印训练集的损失值和验证集的模型精度，并保存精度最好的ckpt文件
def epoch_end(self, run_context):
    cb_params = run_context.original_args()
    cur_epoch = cb_params.cur_epoch_num
    loss_epoch = cb_params.net_outputs
    if cur_epoch >= self.eval_start_epoch and (cur_epoch - self.eval_start_epoch) % self.interval == 0:
        res = self.eval_function(self.eval_param_dict)
        print('Epoch {}/{}'.format(cur_epoch, num_epochs))
        print('-' * 10)
        print('train Loss: {}'.format(loss_epoch))
        print('val Acc: {}'.format(res))
        if res >= self.best_res:
            self.best_res = res
            self.best_epoch = cur_epoch
            if self.save_best_ckpt:
                if os.path.exists(self.best_ckpt_path):
                    self.remove_checkpoint_file(self.best_ckpt_path)
                save_checkpoint(cb_params.train_network, self.best_ckpt_path)
```

注意这里红框内的代码，我们实现的功能是将每次 epoch 的准确度和损失值在训练结束后打印出来，事实上我们在后续的代码改进中也将准确度保存在一个列表中，最终绘图，如下：

```
print(accuracy_show)
print(loss_show)
x = np.arange(0, 50)+1
plt.plot(x, accuracy_show, "r", linewidth=1)
plt.title('accuracy')
plt.show()
```

④ 模型的不断更新和保存

```
# 每一个epoch后，打印训练集的损失值和验证集的模型精度，并保存精度最好的ckpt文件
def epoch_end(self, run_context):
    cb_params = run_context.original_args()
    cur_epoch = cb_params.cur_epoch_num
    loss_epoch = cb_params.net_outputs
    if cur_epoch >= self.eval_start_epoch and (cur_epoch - self.eval_start_epoch) % self.interval == 0:
        res = self.eval_function(self.eval_param_dict)
        print('Epoch {}/{}'.format(cur_epoch, num_epochs))
        print('-' * 10)
        print('train Loss: {}'.format(loss_epoch))
        print('val Acc: {}'.format(res))
        if res >= self.best_res:
            self.best_res = res
            self.best_epoch = cur_epoch
            if self.save_best_ckpt:
                if os.path.exists(self.best_ckpt_path):
                    self.remove_checkpoint_file(self.best_ckpt_path)
                save_checkpoint(cb_params.train_network, self.best_ckpt_path)

# 训练结束后，打印最好的精度和对应的epoch
def end(self, run_context):
    print("End training, the best {0} is: {1}, the best {0} epoch is {2}".format(self.metrics_name, self.best_res, self.best_epoch), flush=True)
```

这里的代码实现的是，每一个 epoch 训练结束后，打印出训练集上的损失值和测试集上的准确度，同时我们将当前的模型准确度与当前循环下准确度最高的模型进行对比，保存下准确度最高的模型，这样我们就实现了保存训练过程中准确度最高的模型的功能。

4、实验结果

① 准确度可视化


```

model = Model(network = net, loss_fn=net_loss,optimizer=net_opt, metrics={"Accuracy": Accuracy()})

Delete parameter from checkpoint: moments.fc3.weight
Delete parameter from checkpoint: moments.fc3.bias

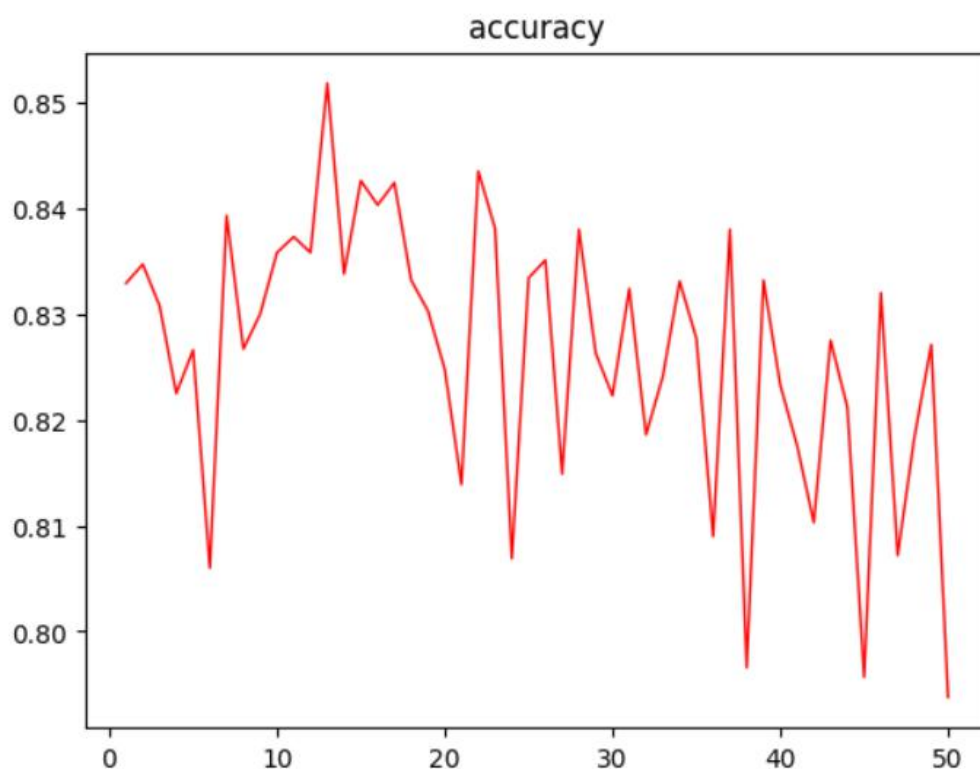
[6]: # 模型训练
eval_param_dict = {"model":model,"dataset":ds_eval,"metrics_name":"Accuracy"}
eval_cb = EvalCallback(apply_eval, eval_param_dict,)

# visualize_model('checkpoint_lenet_verified_3-118_1875.ckpt', ds_eval)
print("===== Starting Training =====")
# 训练模型
model.train(num_epochs, ds_train, callbacks=[eval_cb, TimeMonitor()], dataset_sink_mode=True)

<class 'int'>
<class 'int'>
===== Starting Training =====
Epoch 1/50
-----
train Loss: 0.20916429
val Acc: 0.8127003205128205
epoch time: 9449.184 ms, per step time: 5.040 ms
Epoch 2/50
-----
train Loss: 0.7609746
val Acc: 0.8361378205128205
epoch time: 5543.432 ms, per step time: 2.956 ms
Epoch 3/50
-----
train Loss: 0.20568538
val Acc: 0.8435496794871795
epoch time: 5485.104 ms, per step time: 2.925 ms

```

注意到在每次 epoch 训练结束后，会对应输出该次训练后的准确度，同时我们会把每次的准确度保存下来，最后会绘制成一个折线图如下。



② 预测可视化结果

```

    if i >= 8:
        break
plt.show()

```

```
[24]: visualize_model('best.ckpt', ds_eval)
```



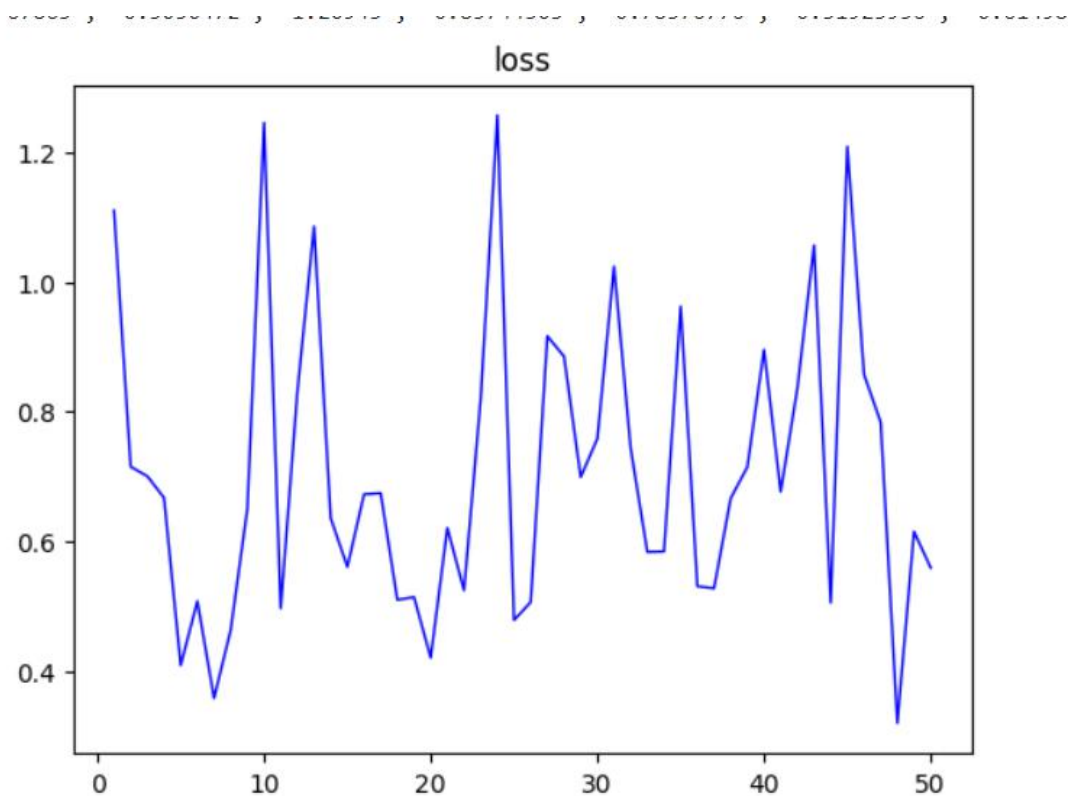
最终运行是在华为云上运行的,可以看到我们的模型从测试集上随机取 9 个图像,进行预测,最终的结果是全部预测正确了。(注意标签为蓝色的时候说明预测正确,下面展示一下带有预测错误情况的红色标签)

```
[33]: visualize_model('best.ckpt', ds_eval)
```



上面第七张图像预测错误,输出标签的颜色则变为红色。

③ 损失值变化曲线



我们也试着将 loss 的变化曲线绘制出来,需要注意的是因为我们开始读取的是已经预训练后的模型, 所以我们的上面的 loss 损失图并不是收敛向下的

5、遇到错误

① **TypeError: Image data of dtype object cannot be converted to float**

解析: 这个错误事实上一直没有解决, 出现场景是我在对数据集调用 `creat_dict_iterator` 迭代器, 接着调用 `plt.imshow()` 时出现的报错, 上网查阅后大多数人说是路径错误, 这个肯定是不符合我的情况的, 同时我们对想要显示的数据进行了多次检查后, 感觉没什么错误, 但遇到这个报错就很烦。最后绕了点远路, 换另一种方式将图片输出。

② **The folder cifar-10-batches-bin does not exist or is not a directory or permission denied !**

解析: 在华为云上运行时, 出现这个错误, 这个错误是说找不到数据文件夹, 但是明明路径什么的都是对的, 然后无意间发现需要在每次重新运行前将数据集所在的文件夹进行同步。即选中数据集文件夹后点击 **Sync OBS**, 错误消失。(华为云旧版, 当用新版环境时并没有发现这种错误)

③ **参数类型**


```

def __init__(self, eval_function, eval_param_dict, a=0, eval_start_epoch=1, interval=1, save_best_ckpt=True,
             ckpt_directory=".", best_ckpt_name="best.ckpt", metrics_name="acc"):
    super(EvalCallBack, self).__init__()
    self.eval_param_dict = eval_param_dict
    self.eval_function = eval_function
    self.eval_start_epoch = eval_start_epoch
    if interval < 1:
        raise ValueError("interval should >= 1.")
    self.interval = interval
    self.save_best_ckpt = save_best_ckpt
    self.best_res = 0
    self.best_epoch = 0
    if not os.path.isdir(ckpt_directory):
        os.makedirs(ckpt_directory)
    self.best_ckpt_path = os.path.join(ckpt_directory, best_ckpt_name)
    self.metrics_name = metrics_name

```

注意上面初始化参数列表中的 `a=0`，这个是我新加的，并没有什么实际的用途，最初是没有的，但是会一直报错，然后检查后发现会将 `eval_start_epoch` 的类型变为 `dict`，实际上我本意是 `int` 型，经过多次检查也没发现哪里出了问题，灵机一动在参数列表中加上一个 `a=0`，然后发现 `eval_start_epoch` 类型变为了 `int` 型。错误解决。

6、实验总结

整体实验看着是简单的，但整体做下来还是有不少收获的。这个实验的主要的目的是熟悉 `mindspore` 库的使用，直接上手肯定是有许多函数是不熟悉的，所以很多时候是在看着 `mindspore` 官网上的教程，所幸，官网上教程都会给上代码示例。华为云也是真的好用，训练速度贼快。

附上代码链接：<https://github.com/sysu19351002/Mindspore-LeNet>