实验5. 基于MindSpore和LeNet5的手写字符图像识别

实验目的

1. 使用MindSpore实现Lab4的三层神经网络
2. 了解卷积神经网络LeNet5的网络结构
3. 体验LeNet5在MindSpore平台下的构建

实验数据

MNIST 是 Yann LeCun 收集创建的手写数字识别数据集，训练集有 60,000 张图片，测试集有 10,000 张图片。数据集链接为：http://yann.lecun.com/exdb/mnist/。数据集下载解压后有4个二进制文件：

```
MNIST
    train
        train-images-idx3-ubyte: 训练集图片
        train-labels-idx1-ubyte: 训练集标签
    test
        t10k-images-idx3-ubyte:  测试集图片
        t10k-labels-idx1-ubyte:  测试集标签
```

实验环境安装

本实验使用深度学习框架MindSpore。具体版本结合自身CPU、GPU、操作系统、Python语言版本等情况选择。（在本机上安装，可选2.2.14）

【MindSpore安装网址】：https://www.mindspore.cn/install

在该网页上提供了下载链接生成，相关界面如下：

安装方式2：

命令行安装方式【清华镜像源】

```
pip install mindspore -i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装成功后，可使用如下命令确认MindSpore是否能工作

```
python -c "import mindspore;mindspore.set_context(device_target='CPU');mindspore.run_check()"
```

🥇 输出结果：

MindSpore version: 2.5.0

The result of multiplication calculation is correct, MindSpore has been installed on platform [CPU] successfully!

【Mindspore的官方教程】：https://www.mindspore.cn/tutorials/zh-CN/r2.3.1/index.html

【MindSpore的官方API文档】：https://www.mindspore.cn/docs/zh-CN/r2.3.1/index.html

1. 准备数据

1.1 读取数据

```
import mindspore
from mindspore import ops
from mindspore import nn
from mindspore.dataset import vision, transforms
from mindspore.dataset import MnistDataset

# 加载MNIST数据集
train_dataset_original = MnistDataset('./MNIST/train')
test_dataset_original = MnistDataset('./MNIST/test')
print(train_dataset_original.get_col_names())
```

Cell的输出：

📌 ['image', 'label']

1.2 数据预处理

MindSpore的dataset使用数据处理流水线（Data Processing Pipeline），需指定map、batch、shuffle等操作。这里我们使用map对图像数据及标签进行变换处理，将输入的图像缩放为1/255，根据均值0.1307和标准差值0.3081进行归一化处理，然后将处理好的数据集打包为大小为64的batch。

vision.Rescale()：

基于给定的缩放和平移因子调整图像的像素大小。输出图像的像素大小为：output = image * rescale + shift。

- rescale (float) - 缩放因子。
- shift (float) - 平移因子。

经过vision.Rescale(1.0 / 255.0, 0)后，每一个像素值从0~255缩小为0~1。

vision.Normalize()：

根据均值和标准差对输入图像进行归一化。此处理将使用以下公式对输入图像进行归一化：output[channel] = (input[channel] - mean[channel]) / std[channel]，其中 channel 代表通道索引，channel >= 1。

- mean (sequence) - 图像每个通道的均值组成的列表或元组。
- std (sequence) - 图像每个通道的标准差组成的列表或元组。

vision.HWC2CHW()：

将输入图像的shape从 <H, W, C> 转换为 <C, H, W>。 如果输入图像的shape为 <H, W> ，图像将保持不变。C为channel(通道数，比如彩色图有R、G、B，3个通道，灰度图的通道数为1)，H、W为图片的高和宽的像素值。

```
def create_dataset(dataset, batch_size):
    image_transforms = [
        vision.Rescale(1.0 / 255.0, 0),
        vision.Normalize(mean=(0.1307,), std=(0.3081,)),
        vision.HWC2CHW()
    ]
    label_transform = transforms.TypeCast(mindspore.int32)

    dataset = dataset.map(image_transforms, 'image')
    dataset = dataset.map(label_transform, 'label')
    dataset = dataset.batch(batch_size)
    return dataset

train_dataset = create_dataset(train_dataset_original, 64)
test_dataset = create_dataset(test_dataset_original, 64)
```

1.3 可视化部分手写数字图片

```
import matplotlib.pyplot as plt

mnist_dataset = MnistDataset('./MNist/train', num_samples=100)

# 设置图像大小
plt.figure(figsize=(8,8))
i=1
# 打印100张子图
for dic in mnist_dataset.create_dict_iterator(output_numpy=True):
    plt.subplot(10,10,i)
    plt.imshow(dic['image'][:,:,0])
    plt.axis('off')
    i += 1
plt.show()
```

2. 利用Mindspore实现传统神经网络手写数字识别

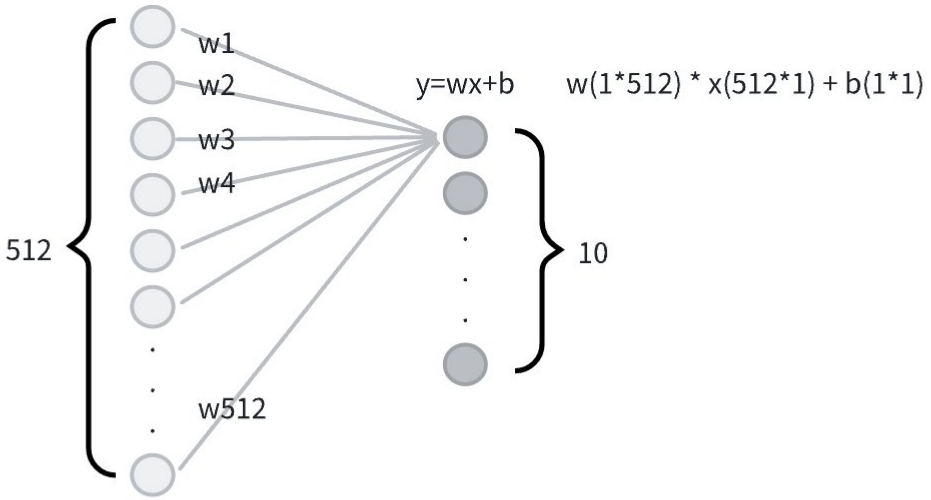
通过mindspore.nn.Dense()增加全连接层，以便学习更复杂的特征表示。比如nn.Dense(512, 512)，第一个512表示输入特征的维度，第二个512表示输出特征的维度。这意味着网络将512个输入特征转换成512个新的特征表示，这样做通常是为了增加网络的深度和表示能力。

2.1 传统神经网络的构建

```
# 网络构建
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.dense_relu_sequential = nn.SequentialCell(
            nn.Dense(28*28, 512),
            nn.ReLU(),
            nn.Dense(512, 512),
            nn.ReLU(),
            nn.Dense(512, 10)
        )

    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits

bpnn = Network()
print(bpnn)
```

nn.Dense()为全连接层，其使用权重和偏差对输入进行线性变换。

对于nn.Dense(512,10)的理解：



ReLU()是激活函数。激活函数是指在多层神经网络中，上层神经元的输出和下层神经元的输入存在一个函数关系，这个函数就是激活函数。引入激活函数的目的是为了增加神经网络的非线性拟合能力。

2.2 使用随机梯度下降SGD算法训练神经网络模型

在bpnn的基础上，构建Model(bpnn, loss_fn, optimizer, metrics)，之后调用mindspore.train.Model.train()调用SGD来训练神经网络。训练过程需多次迭代数据集，一次完整的迭代称为一轮（epoch）。

```
from mindspore.train import Model

# 模型训练
loss_fn = nn.CrossEntropyLoss()
optimizer = nn.SGD(bpnn.trainable_params(), 1e-2)

model = Model(bpnn, loss_fn=loss_fn, optimizer=optimizer, metrics={'acc', 'loss'})
model.train(epoch = 2, train_dataset = train_dataset)
```

2.3 网络模型的测试

```
metrics = model.eval(test_dataset, dataset_sink_mode=False)
print('Metrics:', metrics)
```

Cell的输出：

📌 Metrics: {'acc': 0.9276, 'loss': 0.2502654547429389}

输出可能略有不同，大致在输出范围内即可。

2.4 保存模型（检查点）

保存训练的模型，包括参数等。

```
# Save checkpoint
mindspore.save_checkpoint(bpnn, "bpnn.ckpt")
print("Saved Model to bpnn.ckpt")
```

也可以使用回调函数方式，加入检查点的自动保存

```
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig
config_ck = CheckpointConfig(save_checkpoint_steps=32, keep_checkpoint_max=10)
ckpoint_cb = ModelCheckpoint(prefix='resnet50', directory=None,
config=config_ck)
model.train(epoch_num, dataset, callbacks=ckpoint_cb)
```

如果准备将其生成一个以后可以直接用于预测的模型，可以使用export函数存成MindIR文件格式（IR：intermediate result），然后供给MindSporeLite直接用其预测。

```
from mindspore import export
model = YourModel()
inputs = YourInputData()  # 输入数据的定义
export(model, inputs, file_name="model.mindir", file_format="MINDIR")
```

2.5 加载模型并进行预测

加载保存的权重分为两步：

1. 重新实例化模型对象，构造模型。
2. 加载模型参数，并将其加载至模型上。

2.5.1 加载模型

```
# Instantiate a random initialized model
bpnn = Network()
```

```
# Load checkpoint and load parameter to model
params = mindspore.load_checkpoint("bpnn.ckpt")
mindspore.load_param_into_net(bpnn, params)
model = Model(bpnn)
```

2.5.2 用加载好的模型去预测

使用test数据中的一个batch的数据进行预测

```
for img, label in test_dataset:
    rslt = model.predict(img)
    print(rslt.argmax(1))
    print(label)
    break
```

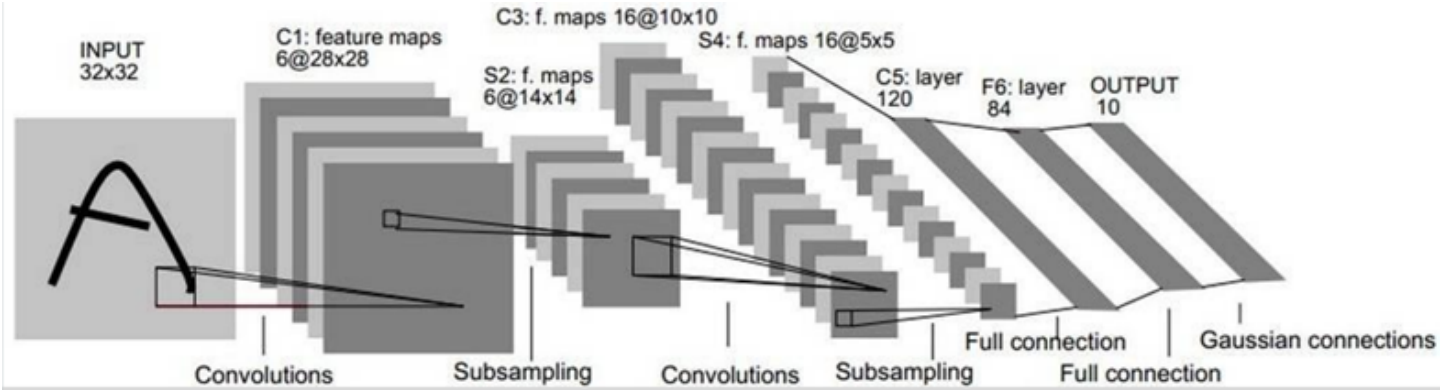
3. 使用MindSpore实现卷积神经网络LeNet5

3.1 创建数据集

```
def create_dataset_v2(dataset, batch_size):
    image_transforms = [
        vision.Resize((32, 32)),
        vision.Rescale(rescale=1/(255*0.3081), shift=-0.1307/0.3081),
        vision.HWC2CHW()
    ]
    label_transform = transforms.TypeCast(mindspore.int32)

    dataset = dataset.map(image_transforms, 'image')
    dataset = dataset.map(label_transform, 'label')
    dataset = dataset.batch(batch_size)
    # dataset = dataset.shuffle(buffer_size=64).batch(batch_size, drop_remainder=True)
    return dataset

train_dataset_original = MnistDataset('./MNIST/train')

train_dataset = create_dataset_v2(train_dataset_original, 32)
```

3.2 创建LeNet5网络

下图是LeNet5的网络结构，以手写字符为例给出它的具体结构，包括2个卷积层、2个池化层、3个全连接层和1个输出层。其中，原图的C5应改为F5，输出层也是全连接层，最后将输出值在通过softmax进行标准化。



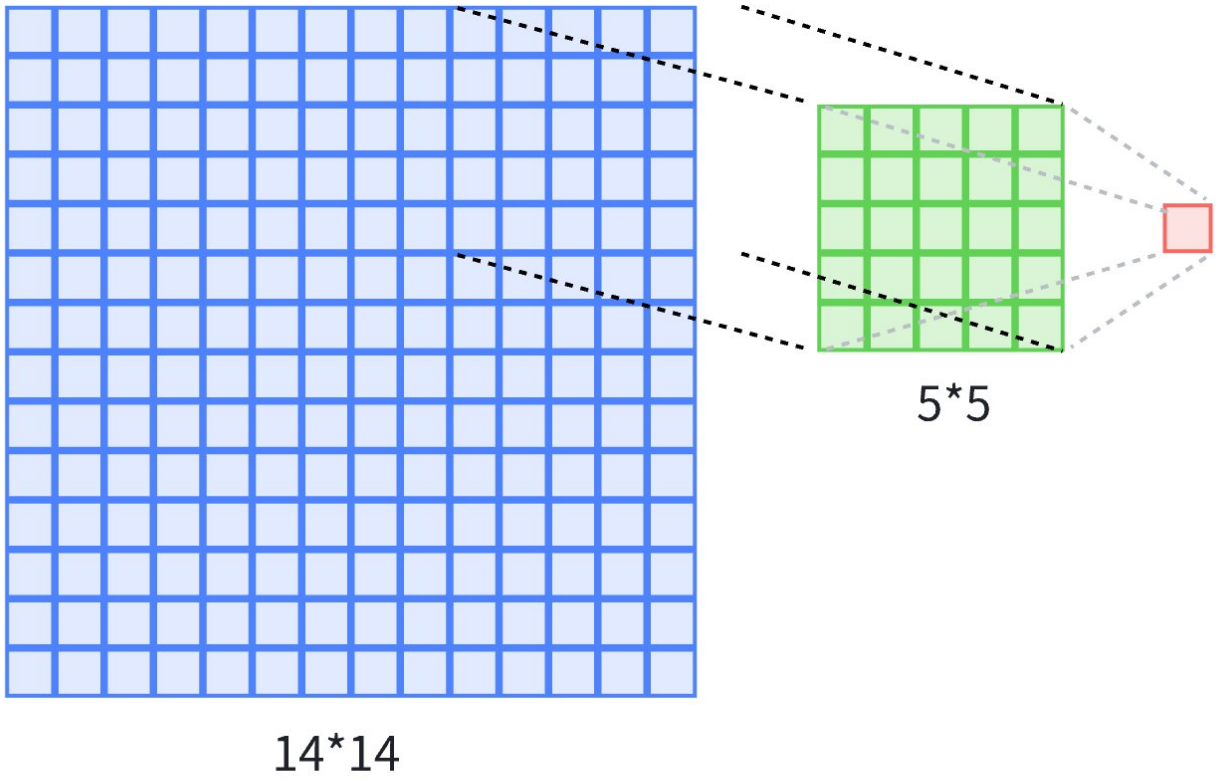
	输入	卷积、池化、神经元	输出	训练参数
输入层*			32*32	0
卷积层1	32*32	6个 5*5卷积核 步长为1	6*28*28（32-5+0）/1+1	1*（5*5）*6 +6=156
池化层1	6*28*28	2*2 步长为2	6*14*14	0
卷积层2	6*14*14	16个 5*5卷积核 步长为1	16*10*10 （14-5+0）/1+1	6*（5*5）*16 +16=2416
池化层2	16*10*10	2*2 步长为2	16*5*5	0
全连接层1	16*5*5	120个 5*5卷积核 步长为1	120*1*1（5-5+0）/1+1	16*（5*5）*120+120=48120
全连接层2	120		84	120*84+84=10164
输出层	84		10	84*10+10=850

卷积过程理解：

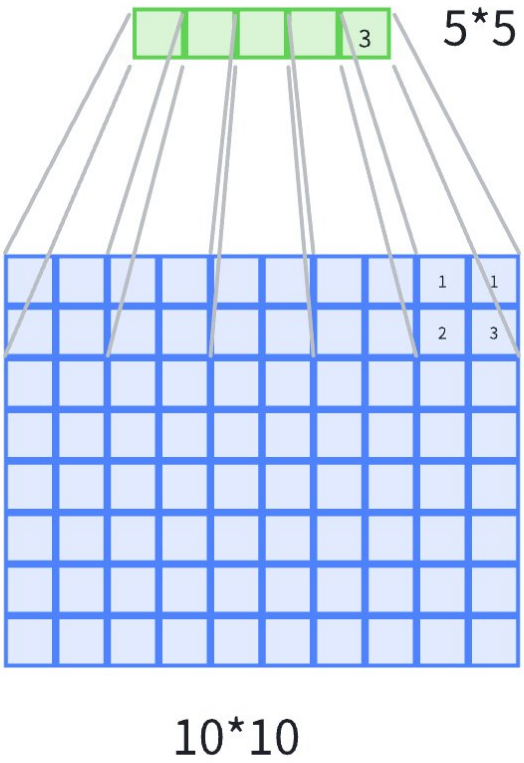
输出长度 =（输入长度+2*填充-卷积核尺寸）/步长+1

例如：卷积层2

输入尺寸为14*14，卷积核为5*5，填充为0，步长为1，所以输出尺寸为（14+2*0-5）/1+1=10。



池化过程理解：



对应代码如下：

```
1    # 根据LeNet5的网络结构创建模型
```

```
# 卷积的函数为nn.Conv2d(in_channel, out_channel, kernel_size, pad_mode='valid')
# 池化层nn.MaxPool2d(kernel_size, stride)
# 全连接层nn.Dense()
class LeNet5(nn.Cell):
    '''
    code here
    '''
```

3.3 使用动量法（Momentum）训练LeNet5模型

使用与传统神经网络类似的训练方式（和2.2节的代码比较一下），编写代码如下：

```
from mindspore.train.callback import LossMonitor

net = LeNet5()
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.01, momentum=0.9)
loss_cb = LossMonitor(per_print_times=train_dataset.get_dataset_size())

model = Model(network = net, loss_fn=loss, optimizer=optim, metrics={'acc', 'loss'})

model.train(epoch=3, train_dataset=train_dataset, callbacks=[loss_cb])
```

Cell输出：

📌 epoch: 1 step: 1875, loss is 0.18654417991638184

epoch: 2 step: 1875, loss is 0.010615820996463299

epoch: 3 step: 1875, loss is 0.06024821102619171

3.4 LeNet5网络的测试

```
test_dataset_original = MnistDataset('./MNIST/test')
test_dataset = create_dataset_v2(test_dataset_original, 32)
metrics = model.eval(test_dataset)
print('Metrics:', metrics)
```

Cell输出：

📌 Metrics: {'acc': 0.9853, 'loss': 0.04310403323785131}

ACC在98%左右即可，不用过分和参考对齐