目录

[1 MNIST手写体识别实验 3](#_Toc67814383)

[1.1 实验介绍 3](#_Toc67814384)

[1.2 实验准备 3](#_Toc67814385)

[1.3 实验详细设计与实现 3](#_Toc67814386)

[1.3.1 数据准备 3](#_Toc67814387)

[1.3.2 实验步骤 4](#_Toc67814388)

[1.4 开放题 8](#_Toc67814389)

# MNIST手写体识别实验

## 实验介绍

本例子会实现一个简单的图片分类的功能，整体流程如下：

1、处理需要的数据集，这里使用了MNIST数据集。

1. 定义一个网络，这里我们使用LeNet网络。
2. 定义损失函数和优化器。
3. 加载数据集并进行训练，训练完成后，查看结果及保存模型文件。
4. 加载保存的模型，进行推理。

验证模型，加载测试数据集和训练后的模型，验证结果精度。

## 实验准备

在动手进行实践之前，确保，你已经正确安装了MindSpore。如果没有，可以通过MindSpore官网安装页面：<https://www.mindspore.cn/install/> ，将MindSpore安装在你的电脑当中。

同时希望你拥有Python编码基础和概率、矩阵等基础数学知识。

## 实验详细设计与实现

### 数据准备

我们示例中用到的MNIST数据集是由10类28\*28的灰度图片组成，训练数据集包含60000张图片，测试数据集包含10000张图片。

MNIST数据集下载页面：http://yann.lecun.com/exdb/mnist/。页面提供4个数据集下载链接，其中前2个文件是训练数据需要，后2个文件是测试结果需要。

将数据集下载并解压到本地路径下，这里将数据集解压分别存放到工作区的./MNIST\_Data/train、./MNIST\_Data/test路径下。

目录结构如下：

└─MNIST\_Data

├─ test

│ t10k-images.idx3-ubyte

│ t10k-labels.idx1-ubyte

│

└─ train

train-images.idx3-ubyte

train-labels.idx1-ubyte

为了方便样例使用，我们在样例脚本中添加了自动下载数据集的功能。

### 实验步骤

1. 导入Python库&模块并配置运行信息

在使用前，导入需要的Python库。

目前使用到os库，为方便理解，其他需要的库，我们在具体使用到时再说明。

详细的MindSpore的模块说明，可以在MindSpore API页面中搜索查询。

可以通过context.set\_context来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入context模块，配置运行需要的信息。

import os

import mindspore as ms

import mindspore.context as context

#transforms.c\_transforms用于通用型数据增强，vision.c\_transforms用于图像类数据增强

import mindspore.dataset.transforms.c\_transforms as C

import mindspore.dataset.vision.c\_transforms as CV

#nn模块用于定义网络，model模块用于编译模型，callback模块用于设定监督指标

from mindspore import nn

from mindspore.train import Model

from mindspore.train.callback import LossMonitor

#设定运行模式为图模式，运行硬件为昇腾芯片

context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend') # Ascend, CPU, GPU

在样例中我们配置样例运行使用图模式。根据实际情况配置硬件信息，譬如代码运行在Ascend AI处理器上，则device\_target选择Ascend，代码运行在CPU、GPU同理。详细参数说明，请参见context.set\_context接口说明。

1. 数据处理

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。

定义数据集及数据操作

我们定义一个函数create\_dataset来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

1. 定义数据集。

2. 定义进行数据增强和处理所需要的一些参数。

3. 根据参数，生成对应的数据增强操作。

4. 使用map映射函数，将数据操作应用到数据集。

5. 对生成的数据集进行处理。

#根据数据集存储地址，生成数据集

def create\_dataset(data\_dir, training=True, batch\_size=32, resize=(32, 32),

rescale=1/(255\*0.3081), shift=-0.1307/0.3081, buffer\_size=64):

#生成训练集和测试集的路径

data\_train = os.path.join(data\_dir, 'train') # train set

data\_test = os.path.join(data\_dir, 'test') # test set

#利用MnistDataset方法读取mnist数据集，如果training是True则读取训练集

ds = ms.dataset.MnistDataset(data\_train if training else data\_test)

#map方法是非常有效的方法，可以整体对数据集进行处理，resize改变数据形状，rescale进行归一化，HWC2CHW改变图像通道

ds = ds.map(input\_columns=["image"], operations=[CV.Resize(resize), CV.Rescale(rescale, shift), CV.HWC2CHW()])

#利用map方法改变数据集标签的数据类型

ds = ds.map(input\_columns=["label"], operations=C.TypeCast(ms.int32))

# shuffle是打乱操作，同时设定了batchsize的大小，并将最后不足一个batch的数据抛弃

ds = ds.shuffle(buffer\_size=buffer\_size).batch(batch\_size, drop\_remainder=True)

return ds

其中，

batch\_size：每组包含的数据个数，现设置每组包含32个数据。

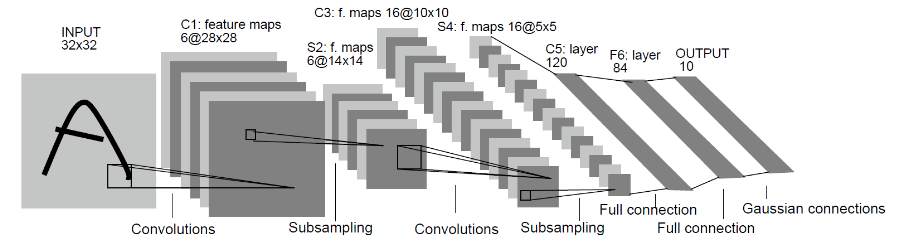
先进行修改图片尺寸，归一化，修改图像频道数等工作，再修改标签的数据类型。最后进行shuffle操作，同时设定batch\_size，设置drop\_remainder为True，则数据集中不足最后一个batch的数据会被抛弃。

MindSpore支持进行多种数据处理和增强的操作，各种操作往往组合使用，具体可以参考数据处理与数据增强章节。

1. 定义网络

我们选择相对简单的LeNet网络。LeNet网络不包括输入层的情况下，共有7层：2个卷积层、2个下采样层（池化层）、3个全连接层。每层都包含不同数量的训练参数，如下图所示：

LeNet-5



更多的LeNet网络的介绍不在此赘述，希望详细了解LeNet网络，可以查询http://yann.lecun.com/exdb/lenet/。

使用MindSpore定义神经网络需要继承mindspore.nn.cell.Cell。Cell是所有神经网络（Conv2d等）的基类。

神经网络的各层需要预先在\_\_init\_\_方法中定义，然后通过定义construct方法来完成神经网络的前向构造。按照LeNet的网络结构，定义网络各层如下：

#定义模型结构，MindSpore中的模型时通过construct定义模型结构，在\_\_init\_\_中初始化各层的对象

class LeNet5(nn.Cell):

def \_\_init\_\_(self):

super(LeNet5, self).\_\_init\_\_()

#定义卷积层，ReLU激活函数，平坦层和全连接层

#conv2d的输入通道为1维，输出为6维，卷积核尺寸为5\*5，步长为1，不适用padding

self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad\_mode='valid')

self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad\_mode='valid')

self.relu = nn.ReLU()

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(400, 120)

self.fc2 = nn.Dense(120, 84)

self.fc3 = nn.Dense(84, 10)

#构建Lenet5架构，x代表网络的输入

def construct(self, x):

x = self.relu(self.conv1(x))

x = self.pool(x)

x = self.relu(self.conv2(x))

x = self.pool(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.fc2(x)

x = self.fc3(x)

return x

1. 定义损失函数及优化器

在进行定义之前，先简单介绍损失函数及优化器的概念。

损失函数：又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失函数的值。定义一个好的损失函数，可以有效提高模型的性能。

优化器：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。

定义损失函数

MindSpore支持的损失函数有SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss等。这里使用SoftmaxCrossEntropyWithLogits损失函数。

MindSpore提供了callback机制，可以在训练过程中执行自定义逻辑，这里使用框架提供的ModelCheckpoint为例。 ModelCheckpoint可以保存网络模型和参数，以便进行后续的fine-tuning（微调）操作。

# 构建训练、验证函数进行模型训练和验证，提供数据路径，设定学习率，epoch数量

def train(data\_dir, lr=0.01, momentum=0.9, num\_epochs=3):

#调用函数，读取训练集

ds\_train = create\_dataset(data\_dir)

#调用函数，读取验证集

ds\_eval = create\_dataset(data\_dir, training=False)

#构建网络

net = LeNet5()

#设定loss函数

loss = nn.loss.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

#设定优化器

opt = nn.Momentum(net.trainable\_params(), lr, momentum)

#设定损失监控

loss\_cb = LossMonitor(per\_print\_times=ds\_train.get\_dataset\_size())

#编译形成模型

model = Model(net, loss, opt, metrics={'acc', 'loss'})

# 训练网络，dataset\_sink\_mode为on\_device模式

model.train(num\_epochs, ds\_train, callbacks=[loss\_cb], dataset\_sink\_mode=False)

#用验证机评估网络表现

metrics = model.eval(ds\_eval, dataset\_sink\_mode=False)

#输出相关指标

print('Metrics:', metrics)

1. 开始训练及验证过程

#main函数负责调用之前定义的函数，完成整个训练验证过程

if \_\_name\_\_ == "\_\_main\_\_":

#argsparse是python的命令行解析的标准模块，可以通过命令行传入参数

import argparse

parser = argparse.ArgumentParser()

#设定训练数据路径

parser.add\_argument('--data\_url', required=False, default='./FCN/MNIST/', help='Location of data.')

parser.add\_argument('--train\_url', required=False, default=None, help='Location of training outputs.')

args, unknown = parser.parse\_known\_args()

#判断路径是否为obs路径，如果是，从obs路径下载数据

if args.data\_url.startswith('s3'):

import moxing

# WAY1: copy dataset from your own OBS bucket to container/cache.

# moxing.file.copy\_parallel(src\_url=args.data\_url, dst\_url='MNIST/')

# WAY2: copy dataset from other's OBS bucket, which has been set public read or public read&write.

moxing.file.copy\_parallel(src\_url="s3://share-course/dataset/MNIST/", dst\_url='MNIST/')

data\_path = 'MNIST/'

else:

data\_path = os.path.abspath(args.data\_url)

#调用train函数，训练并验证模型

train(data\_path)

训练过程中会打印loss值，类似下图。loss值会波动，但总体来说loss值会逐步减小，精度逐步提高。每个人运行的loss值有一定随机性，不一定完全相同。 训练过程中loss打印示例如下：

...

epoch: 1 step: 1875, loss is 2.2767615

epoch: 2 step: 1875, loss is 0.109801136

epoch: 3 step: 1875, loss is 0.016787775

Metrics: {'acc': 0.9755608974358975, 'loss': 0.07498854234551963}

...

## 开放题

请使用MindSpore 框架实现MNIST手写体识别，并和本章实验进行对比，描述MindSpore框架的优点。