

**人工智能实验报告**

题 目 基于Mindspore框架与ModelArts 平台的MNIST手写体识别实验

专 业 人工智能

学　　 号

姓 名

# 一 背景简介/问题描述

**1.基于Mindspore框架的模型本地训练及预测**

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。

**2.基于Modelarts平台和Tensorflow框架的模型训练及部署**

使用PyTorch1.8实现手写数字图像识别，示例采用的数据集为MNIST官方数据集。通过学习本案例，可以了解如何在ModelArts平台上训练作业、部署推理模型并预测的完整流程。

# 二 算法介绍

# 2.1有关LeNet网络的介绍

LeNet网络的结构包含了两个卷积层（Convolutional layers）和两个池化层（Pooling layers），以及一个全连接层（Fully Connected layers）。传统的LeNet结构如下图所示：

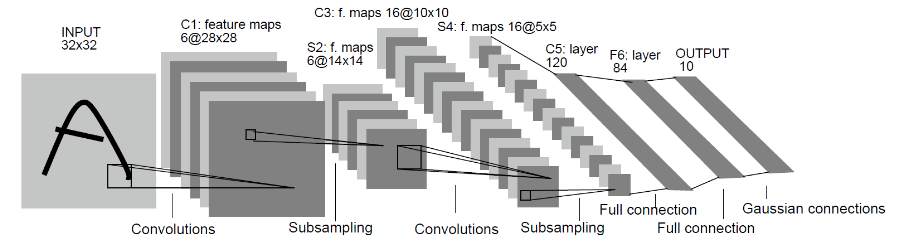


图1 LeNet网络模型

其中，输入是28\*28大小的手写数字图像，通过不断的卷积和池化操作，逐步提取更抽象的特征。最终通过全连接层将抽取的特征输出为所属的数字类别。具体来讲，LeNet网络的每一个Convolutional layer使用的都是同样的卷积核大小，即5x5，然后紧接着跟随一个2x2的池化层。

LeNet网络在处理手写数字的识别中，表现出色，但是它在处理更大尺寸的图像上可能不太适用。然而，LeNet的一些设计思路对后来的卷积神经网络设计产生了很大的影响，比如使用卷积层和池化层来逐步提取特征等。

## 2.2有关Modelarts平台和Tensorflow框架的介绍

ModelArts是华为云推出的一款人工智能开发平台，提供了TensorFlow、PyTorch、Caffe、MXNet等多种人工智能框架的全栈式、可持续集成及开发、部署、运维一体化的开发环境，能够快速实现一站式AI开发。同时ModelArts也支持多种AI开发的应用场景，包括图像识别、自然语言处理、机器学习、深度学习等。

TensorFlow是Google开源的一款人工智能框架，它可以用于构建从简单的线性回归模型到复杂的神经网络模型的各种机器学习模型，具有灵活性和可扩展性。TensorFlow包含了一系列工具和库，提供了各种功能，包括图像和语音识别、自然语言处理、语音合成等方面的任务。同时，TensorFlow还提供了多种语言的API，包括Python、C++、Java、Go等，使得开发者可以使用自己熟悉的语言进行开发。

# 三 算法实现

## 3.1 训练代码：

import os

import ast

import argparse

from src.config import mnist\_cfg as cfg

from src.dataset import create\_dataset

from src.lenet import LeNet5

import mindspore.nn as nn

from mindspore import context

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor

from mindspore.train import Model

from mindspore.nn.metrics import Accuracy

from mindspore.common import set\_seed

set\_seed(1)

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

parser = argparse.ArgumentParser(description='MindSpore Lenet Example')

# 设备设置

parser.add\_argument('--device\_target', type=str, default='CPU', choices=['Ascend', 'GPU', 'CPU'],

help='device where the code will be implemented (default: Ascend)')

parser.add\_argument('--data\_path', type=str, default="./MNIST\_Data",

help='path where the dataset is saved')

parser.add\_argument('--ckpt\_path', type=str, default="./ckpt", help='if is test, must provide\

path where the trained ckpt file')

parser.add\_argument('--dataset\_sink\_mode', type=ast.literal\_eval, default=True,

help='dataset\_sink\_mode is False or True')

args = parser.parse\_args()

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

ds\_train = create\_dataset(os.path.join(args.data\_path, "train"),

cfg.batch\_size)

network = LeNet5(cfg.num\_classes)

#设定loss函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

#设定优化器

net\_opt = nn.Momentum(network.trainable\_params(), cfg.lr, cfg.momentum)

time\_cb = TimeMonitor(data\_size=ds\_train.get\_dataset\_size())

config\_ck = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps,

keep\_checkpoint\_max=cfg.keep\_checkpoint\_max)

ckpoint\_cb = ModelCheckpoint(prefix="checkpoint\_lenet", directory=args.ckpt\_path, config=config\_ck)

#编译形成模型

model = Model(network, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()})

print("============== Starting Training ==============")

# 训练网络 train.py

model.train(cfg['epoch\_size'], ds\_train, callbacks=[time\_cb, ckpoint\_cb, LossMonitor()],

dataset\_sink\_mode=args.dataset\_sink\_mode)

## 3.2 训练数据代码：

import mindspore.dataset as ds

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

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

from mindspore.dataset.vision import Inter

from mindspore.common import dtype as mstype

def create\_dataset(data\_path, batch\_size=32, repeat\_size=1,

num\_parallel\_workers=1):

"""

create dataset for train or test

"""

# define dataset

mnist\_ds = ds.MnistDataset(data\_path)

resize\_height, resize\_width = 32, 32

rescale = 1.0 / 255.0

shift = 0.0

rescale\_nml = 1 / 0.3081

shift\_nml = -1 \* 0.1307 / 0.3081

# define map operations

resize\_op = CV.Resize((resize\_height, resize\_width), interpolation=Inter.LINEAR) # Bilinear mode

rescale\_nml\_op = CV.Rescale(rescale\_nml, shift\_nml)

rescale\_op = CV.Rescale(rescale, shift)

hwc2chw\_op = CV.HWC2CHW()

type\_cast\_op = C.TypeCast(mstype.int32)

# apply map operations on images

mnist\_ds = mnist\_ds.map(operations=type\_cast\_op, input\_columns="label", num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(operations=resize\_op, input\_columns="image", num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(operations=rescale\_op, input\_columns="image", num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(operations=rescale\_nml\_op, input\_columns="image", num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(operations=hwc2chw\_op, input\_columns="image", num\_parallel\_workers=num\_parallel\_workers)

# apply DatasetOps

buffer\_size = 10000

mnist\_ds = mnist\_ds.shuffle(buffer\_size=buffer\_size) # 10000 as in LeNet train script

mnist\_ds = mnist\_ds.batch(batch\_size, drop\_remainder=True)

mnist\_ds = mnist\_ds.repeat(repeat\_size)

return mnist\_ds

## 3.3 预测代码：

import os

import ast

import argparse

import mindspore.nn as nn

from mindspore import context

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore.train import Model

from mindspore.nn.metrics import Accuracy

from src.dataset import create\_dataset

from src.config import mnist\_cfg as cfg

from src.lenet import LeNet5

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

parser = argparse.ArgumentParser(description='MindSpore Lenet Example')

# 设备设置

parser.add\_argument('--device\_target', type=str, default='CPU', choices=['Ascend', 'GPU', 'CPU'],

help='device where the code will be implemented (default: Ascend)')

parser.add\_argument('--data\_path', type=str, default="./MNIST\_Data",

help='path where the dataset is saved')

parser.add\_argument('--ckpt\_path', type=str, default="./ckpt/checkpoint\_lenet-10\_1875.ckpt", help='if mode is test, must provide\

path where the trained ckpt file')

parser.add\_argument('--dataset\_sink\_mode', type=ast.literal\_eval,

default=False, help='dataset\_sink\_mode is False or True')

args = parser.parse\_args()

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

network = LeNet5(cfg.num\_classes)

#设定loss函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

repeat\_size = cfg.epoch\_size

#设定优化器

net\_opt = nn.Momentum(network.trainable\_params(), cfg.lr, cfg.momentum)

#编译形成模型

model = Model(network, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()})

print("============== Starting Testing ==============")

param\_dict = load\_checkpoint(args.ckpt\_path)

load\_param\_into\_net(network, param\_dict)

ds\_eval = create\_dataset(os.path.join(args.data\_path, "test"),

cfg.batch\_size,

1)

acc = model.eval(ds\_eval, dataset\_sink\_mode=args.dataset\_sink\_mode)

print("============== {} ==============".format(acc))

## 实验步骤：

**步骤一.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。

parser.add\_argument('--device\_target', type=str, default=" Ascend", choices=['Ascend', 'GPU', 'CPU'], help='device where the code will be implemented (default: Ascend)')

在样例中我们配置样例运行使用图模式。再train.py和eval.py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。使用本地和华为云处理器完成模型的训练。

**步骤二.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。步骤二的以下内容里，mindspore 提供了更多数据处理的方法，供大家以后使用mindspore时参考：

/\*--------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。--------\*/

**步骤三.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，

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, num\_class=10, num\_channel=1):

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(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, weight\_init=Normal(0.02))

self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))

self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))

self.relu = nn.ReLU()

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

#构建Lenet5架构，x代表网络的输入

def construct(self, x):

x = self.max\_pool2d(self.relu(self.conv1(x)))

x = self.max\_pool2d(self.relu(self.conv2(x)))

x = self.flatten(x)

x = self.relu(self.fc1(x))

x = self.relu(self.fc2(x))

x = self.fc3(x)

return x

**步骤四.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。

#设定loss函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

#设定优化器

net\_opt = nn.Momentum(network.trainable\_params(), cfg.lr, cfg.momentum)

#编译形成模型

model = Model(network, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()})

# 训练网络 train.py

model.train(cfg['epoch\_size'], ds\_train, callbacks=[time\_cb, ckpoint\_cb, LossMonitor()], dataset\_sink\_mode=args.dataset\_sink\_mode)

**步骤五.开始训练及验证过程：**

利用train.py和eval.py完成模型的训练及验证过程。

训练过程中会打印loss值。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。

# 四 讨论及结论

## 4.1 第一部分实验结果如下：

可见loss值大体上逐渐变小，本次实验使用开源的MNIST数据集，旨在训练LeNet模型对手写数字图像进行分类。经过训练，LeNet模型在测试集上取得了0.987的准确率，表现非常出色，证明了LeNet模型对于手写数字图像分类是有效的。

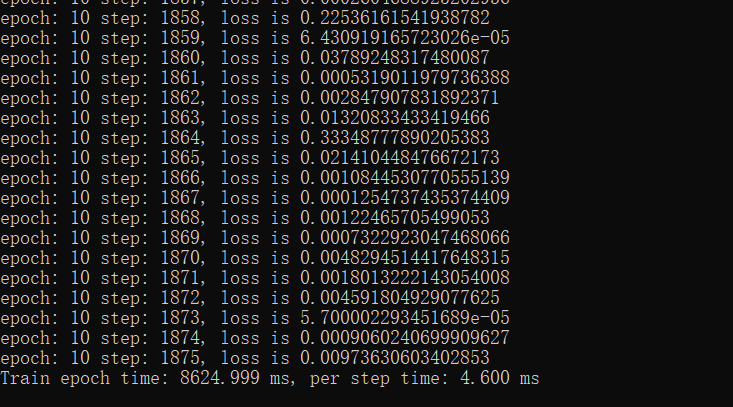


图2 运行train.py的部分结果截图

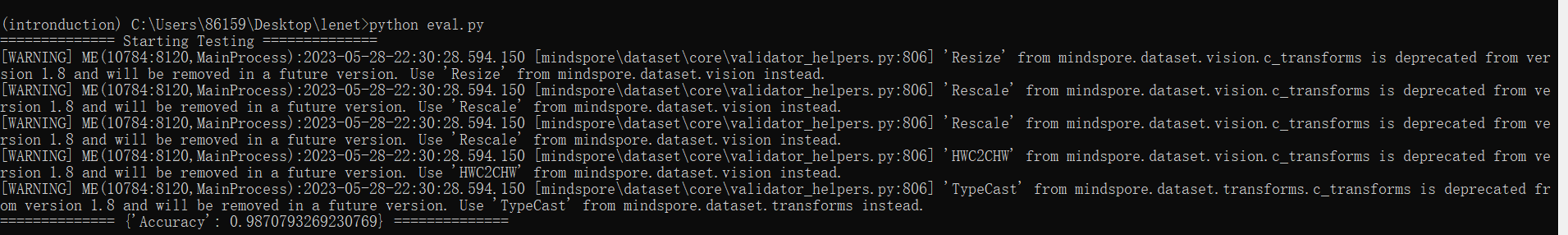


图3 运行eval.py的结果截图

## 4.2第二部分实验结果部分截图如下：

从下面实验结果和损失和准确率的变化曲线来看，模型在训练集和测试集上都表现良好，最终测试集上的分类准确率为 99.16%。



图4 创建的AI应用



图5 创建的在线服务

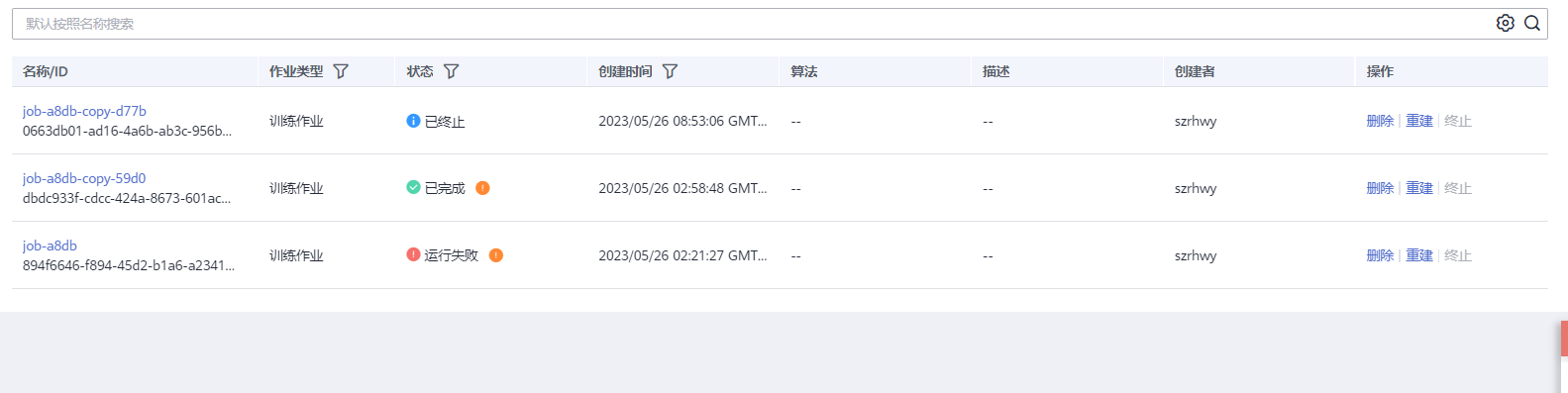


图6 创建训练作业

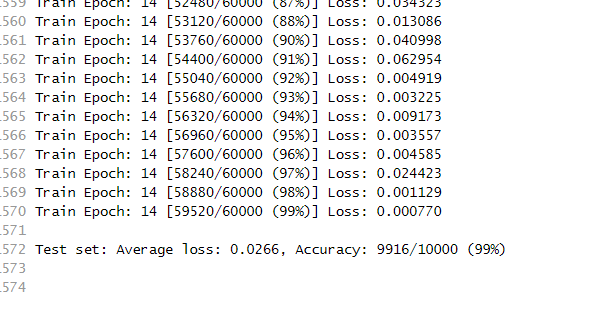


图7 训练作业结果截图

**参考文献**

[1]<https://support.huaweicloud.com/help-novice.html>

[2]<https://zhuanlan.zhihu.com/p/365665291>

[3]<https://zhuanlan.zhihu.com/p/397955168>