浙江大学
ZHEJIANG UNIVERSITY

数据分析与算法设计
实验报告

实验名称 基于 MindSpore 的手写体数字识别 (CNN)

姓　名 _____

学　号 _____

实验日期 2025.05.15

指导老师 李旻

Contents

1 实验目的

(1). 学习 ModelArts 云环境搭建方法。

(2). 以手写体识别案例为例，基于深度学习框架 MindSpore 熟悉 AI 开发流程。

(3). 了解深度卷积网络在图像分类中的应用。

(4). 掌握 Python 在人工智能开发中的常用操作。

2 开发及软件环境

开发环境：ModelArts 云平台

软件环境：MindSpore 2.2

3 实验内容和原理

实验要求使用 MindSpore 框架实现手写体数字识别，使用卷积神经网络（CNN）对 MNIST 数据集进行训练和测试。

以下对 CNN 的基本原理进行简要介绍:

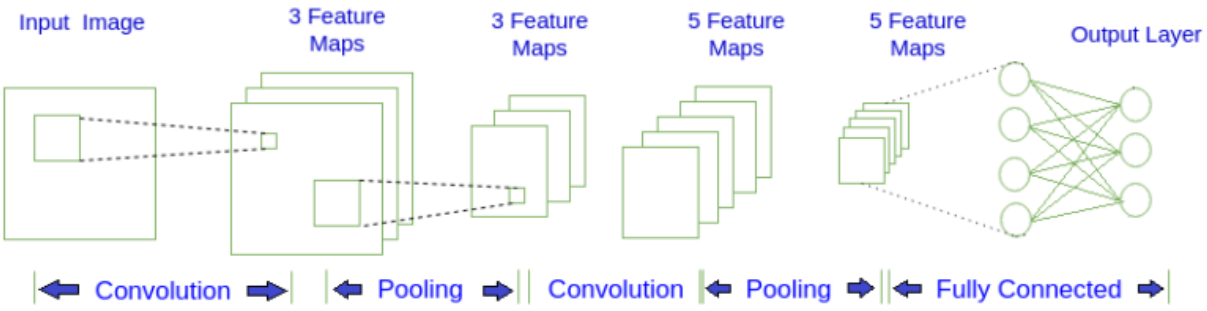

Figure 1: CNN 基本结构图

一个常见的 CNN 结构通常包括以下几个部分:

- 卷积层（Convolutional Layer）：通过卷积操作提取输入数据的局部特征。卷积核在输入数据上滑动，计算局部区域的加权和，生成特征图。可以通过调整 stride（步长）和 padding（填充）来控制特征图的尺寸。
- 激活函数（Activation Function）：通常使用 ReLU（Rectified Linear Unit）激活函数，对卷积层输出进行非线性变换。

- 池化层（Pooling Layer）：通过下采样操作减少特征图的尺寸，降低计算复杂度，同时保留重要特征。常用的池化方法有最大池化和平均池化。
- 全连接层（Fully Connected Layer）：经过若干层的卷积, 池化操作后, 将得到的特征图依次按行展开, 连接成向量, 输入全连接网络将前面提取的特征进行整合，输出最终的分类结果。

4 实验步骤与结果分析

本实验使用华为云 ModelArts 平台进行开发，开发环境搭建步骤在此不再赘述，可参考华为云官方文档进行操作。

4.1 使用 LeNet5 网络

(1). 数据集准备

MNIST 数据集可以从网络中广泛获取到, 这里可以通过示例代码从华为云 MindSpore 托管的资源库中下载, 也可以自己上传到同一文件目录下.

Listing 1: 数据集下载

```
!pip install download

%env no_proxy='a.test.com,127.0.0.1,2.2.2.2'
from download import download
# 下载MNIST数据集
url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/" \
      "notebook/datasets/MNIST_Data.zip"
path = download(url, "./", kind="zip", replace=True)
```

请注意, 需要将 pip install download 前加入!, 这是因为在 jupyter notebook 中, 需要使用! 来执行 shell 命令。这与给定的代码存在差异。

MNIST 数据集结构由训练集和测试集组成，每个数据集包含图像和标签。图像是 28x28 的灰度图像，标签是对应的数字（0-9）。数据集的目录结构如下：

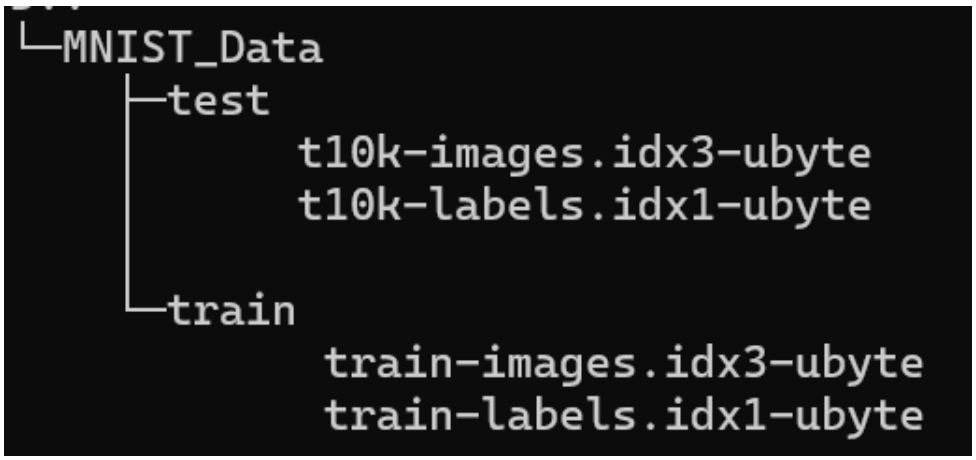

Figure 2: MNIST 数据集目录结构

(2). 导入库和 MindSpore 模块

这里设定运行模式为图模式，运行硬件平台为 Ascend。MindSpore 支持多种硬件平台，包括 Ascend、CPU 和 GPU。图模式是 MindSpore 的默认运行模式，适用于大多数深度学习任务。

Listing 2: 导入库和 MindSpore 模块

```
import numpy as np
import matplotlib.pyplot as plt
import mindspore as ms
from mindspore import nn
from mindspore.dataset import vision,transforms,MnistDataset
from mindspore.dataset.vision import Inter
from mindspore.common.initializer import Normal
from mindspore.train import Model

#设定运行模式为图模式，运行硬件为Ascend
ms.set_context(mode=ms.GRAPH_MODE, device_target='Ascend') # Ascend, CPU, GPU
```

(3). 数据集加载和预处理

对 MNIST 数据集进行预处理的目的是将图像数据转换为适合神经网络输入的格式。这里首先分别通过 MnistDataset 接口读取的 MNIST 的训练集和测试集，并进行 shuffle 操作 (shuffle=True 表示对数据集进行随机打乱，用来 消除数据集中潜在的顺序偏差，从而提高模型的泛化能力)，然后修改图片尺寸，归一化，修改图像频道，修改标签的数据类型，最后设定 batch_size。

Listing 3: 数据集加载和预处理

```
# 定义预处理操作的流程，整体对数据集进行处理
def datapipe(path, batch_size=32):
    image_transform = [
        vision.Resize(size=32, interpolation=Inter.LINEAR), # Resize：以双线性插值方式调整图像尺寸大小
        vision.Rescale(1.0 / 255.0, 0.0),                   # Rescale: 缩放图像的像素值大小，将像素值统一除255，数据类型由unit8转为float32
        vision.Normalize(mean=(0.1307,),std=(0.3081,)), # Mormalize: 将像素值归一化
        vision.HWC2CHW(),                                   # HWC2CHW: 将张量格式从（height,width,channel）转换成(channel,height,width)
    ]
    label_transform = transforms.TypeCast(ms.int32)
    # 利用MnistDataset接口读取解压后的MNIST的训练集和测试集，并进行shuffle操作
    dataset = MnistDataset(path, shuffle=True)
    # 通过map方法对每张图片应用数据处理操作
    dataset = dataset.map(operations=image_transform, input_columns=["image"])
    # 将每个标签的数据类型转换为int32
    dataset = dataset.map(operations=label_transform, input_columns=["label"])
    # 对数据集进行分批处理；当最后一个批处理数据包含的数据条目小于batch_size时，drop_remainder表示是否将该批处理丢弃，不传递给下一个操作。默认值：False，不丢弃。
    dataset = dataset.batch(batch_size, drop_remainder=True)
    return dataset

dataset_train = datapipe("MNIST_Data/train")
dataset_eval = datapipe("MNIST_Data/test")
```

(4). 打印随机图片

这里打印随机的一张图片数据和对应的标签，以验证数据预处理是否正确。

Listing 4: 打印随机图片

```
data_next = dataset_train.create_dict_iterator(output_numpy=True).
    __next__()
print('Batch Size/通道数/图像长/宽: ', data_next['image'].shape)
print('图像的标签样式: ', data_next['label'])

plt.figure()
plt.imshow(data_next['image'][1,...].squeeze(), cmap="gray")
plt.grid(False)
plt.show()
```

打印结果如下所示:

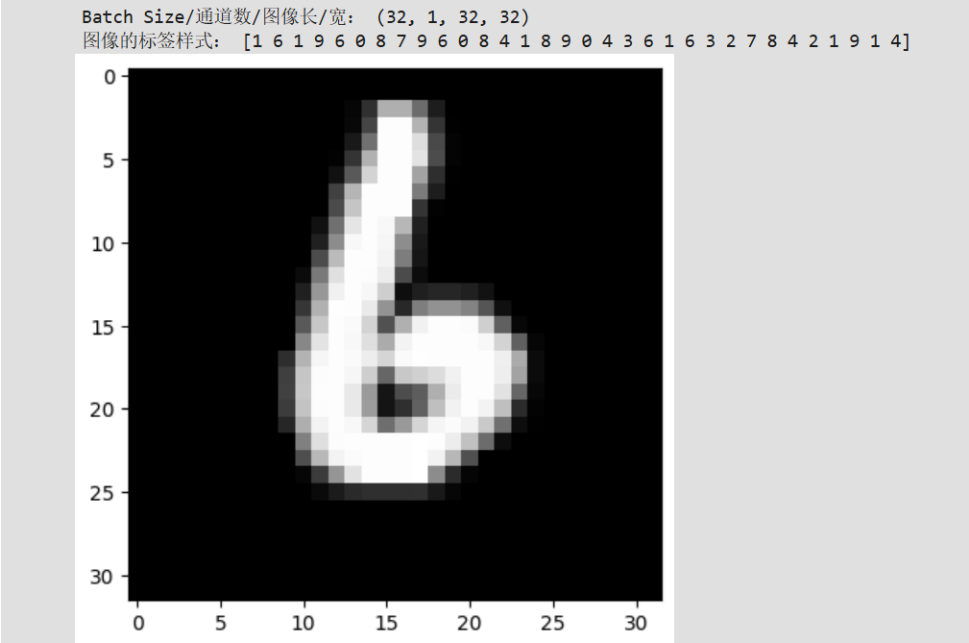

Figure 3: 任意数据内容

可以看到, 图像已被调整为 32x32 的大小，且通道数为 1(灰度图像),batch 的大小为 32, 批次中每个数据的标签为 0-9 之间的数字。

(5). 定义 LeNet5 网络结构

原始 LeNet5 模型激活函数为 sigmoid，池化层为平均池化层。本实验采用 ReLU 激活函数和最大池化层，以获得更好的效果。其他网络结构采用经典的 2 个卷积层和 3 个全连接层构建一个前向传播网络. 具体结构如下所示:

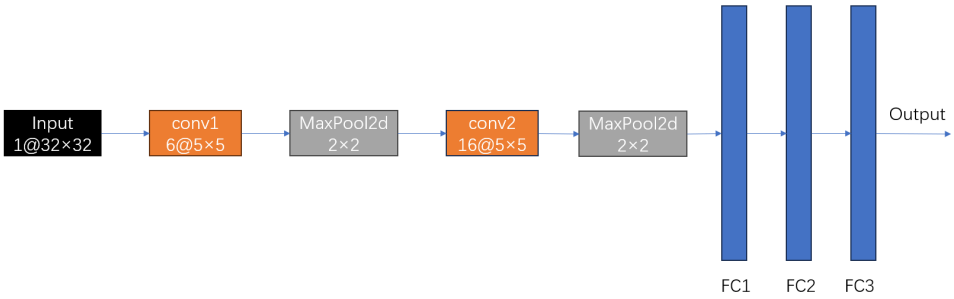

Figure 4: LeNet5 网络结构图

Listing 5: LeNet5 网络结构

```
class LeNet5(nn.Cell):
    """LeNet5"""

    def __init__(self, num_classes=10, num_channel=1):
        super(LeNet5, self).__init__()
        # 卷积层，输入的通道数为num_channel,输出的通道数为6,卷积核大小为5*5
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        # 卷积层，输入的通道数为6，输出的通道数为16,卷积核大小为5*5
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        # ReLU激活函数
        self.relu = nn.ReLU()
        # 池化层
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        # 多维数组展平为一维数组
        self.flatten = nn.Flatten()
        # 全连接层，输入个数为16*5*5，输出个数为120
        self.fc1 = nn.Dense(16 * 5 * 5, 120, weight_init=Normal(0.02))
        # 全连接层，输入个数为120，输出个数为84
        self.fc2 = nn.Dense(120, 84, weight_init=Normal(0.02))
        # 全连接层，输入个数为84，分类的个数为num_class
        self.fc3 = nn.Dense(84, num_classes, weight_init=Normal(0.02))

    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.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

(6). 模型训练

在进行正式的训练之前，我们需要对模型进行实例化，定义优化器和损失函数. 我们这里使用了 SoftmaxCrossEntropyWithLogits 损失函数也即交叉熵损失函数，优化器使用 Momentum 动量优化器。通过引入动量来加速收敛并减少振荡。定义用于训练的 train_loop 函数和用于测试的 test_loop 函数。原始给定的 epoch 数为 3, 这里我们将其改为 10, 以获得更好的效果。训练过程中每 100 个 batch 打印一次 loss 值。最后加入了绘制测试集平均准确率和平均损失的图像, 以更直观的方式展示模型的训练效果。

Listing 6: 模型训练

```
    # 对LeNet5进行实例化
network = LeNet5()

# 定义损失函数
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='
    mean')
# 定义优化器，通过model.trainable_params()方法获得模型的可训练参数，
    并传入学习率超参来初始化优化器
net_opt = nn.Momentum(network.trainable_params(), learning_rate
    =0.01, momentum=0.9)

# 定义用于训练的train_loop函数。
def train_loop(model, dataset, loss_fn, optimizer):
    # 定义正向计算函数
    def forward_fn(data, label):
```

```
13          logits = model(data)
14          loss = loss_fn(logits, label)
15          return loss
16
17      # 定义微分函数，使用mindspore.value_and_grad获得微分函数grad_fn，
            输出loss和梯度。
18      # 由于是对模型参数求导，grad_position 配置为None，传入可训练参数。
19      grad_fn = ms.value_and_grad(forward_fn, None, optimizer.
            parameters)
20
21      # 定义 one-step training函数
22      def train_step(data, label):
23          loss, grads = grad_fn(data, label)
24          optimizer(grads)
25          return loss
26
27      size = dataset.get_dataset_size()
28      model.set_train()
29      for batch, (data, label) in enumerate(dataset.
            create_tuple_iterator()):
30          loss = train_step(data, label)
31
32          if batch % 100 == 0:
33              loss, current = loss.asnumpy(), batch
34              print(f"loss: {loss:>7f} [{current:>3d}/{size:>3d}]")
35
36  # 定义用于测试的test_loop函数。
37  def test_loop(model, dataset, loss_fn):
38      num_batches = dataset.get_dataset_size()
39      model.set_train(False)
40      total, test_loss, correct = 0, 0, 0
41      for data, label in dataset.create_tuple_iterator():
42          pred = model(data)
43          total += len(data)
44          test_loss += loss_fn(pred, label).asnumpy()
45          correct += (pred.argmax(1) == label).asnumpy().sum()
46      test_loss /= num_batches
47      correct /= total
```

```
48      print(f"Test: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {
            test_loss:>8f} \n")
49      return correct,test_loss
50
51  epochs = 10
52  test_accuracies = []
53  test_losses = []
54  for t in range(epochs):
55      print(f"Epoch {t+1}\n-------------------------------")
56      train_loop(network, dataset_train, net_loss, net_opt)
57      ms.save_checkpoint(network, "./save_direct.ckpt")
58      acc, loss_val = test_loop(network, dataset_eval, net_loss)
59      test_accuracies.append(acc)
60      test_losses.append(loss_val)
61  print("Done!")
62
63  plt.figure(figsize=(12, 5))
64  plt.subplot(1, 2, 1)
65  plt.plot(range(1, epochs+1), [x*100 for x in test_accuracies], 'b-o'
        , label='Accuracy')
66  plt.xlabel('Epoch'), plt.ylabel('Accuracy (%)'), plt.title('Test
        Accuracy')
67  plt.grid(True)
68
69  plt.subplot(1, 2, 2)
70  plt.plot(range(1, epochs+1), test_losses, 'r-s', label='Loss')
71  plt.xlabel('Epoch'), plt.ylabel('Loss'), plt.title('Test Loss')
72  plt.grid(True)
73
74  plt.tight_layout()
75  plt.show()
```

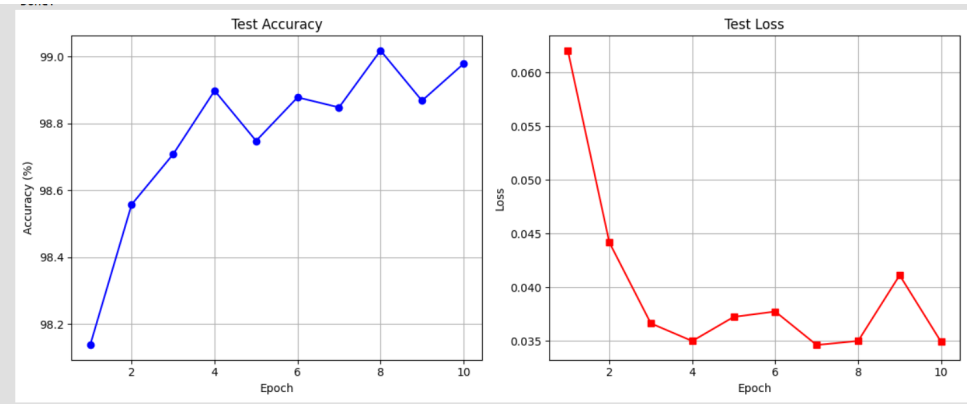

Figure 5: 测试集平均准确率和平均损失

从中可以看出, 随着训练的进行, 测试集的平均准确率逐渐提高到约 99%, 而平均损失逐渐降低到 0.035 左右, 表明模型收敛良好. 但我们观察到, 准确率和损失都在最后几个 epoch 中出现了明显波动, 我们将在 4.2 节中对 LeNet5 网络进行改进, 以获得更好的效果。

(7). 测试模型

Listing 7: 预测可视化

```
param_dict = ms.load_checkpoint("./save_direct.ckpt")
# 重新定义一个LeNet5神经网络
net = network
# 将参数加载到网络中
ms.load_param_into_net(net, param_dict)
model = Model(net)
data_test = dataset_eval.create_dict_iterator()
data = next(data_test)
images = data["image"].asnumpy()
labels = data["label"].asnumpy()

# 使用函数model.predict预测image对应分类
output = model.predict(ms.Tensor(data['image']))
pred = np.argmax(output.asnumpy(), axis=1)

plt.figure()
for i in range(1, 9):
    plt.subplot(2, 4, i)
    plt.imshow(images[i-1].squeeze(), cmap="gray")
```

```
20      plt.title(pred[i-1])
21  plt.show()
```

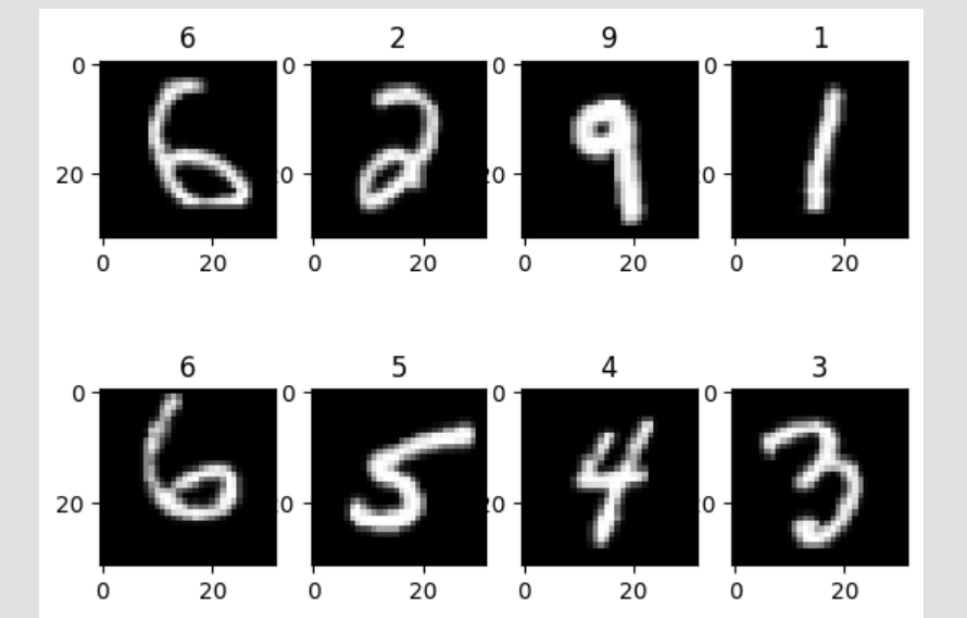

Figure 6: 手写体数字识别结果

4.2 拓展：对 LeNet5 网络进行改进

针对原始 LeNet5 网络的训练波动结果, 我进行了以下改进, 提出了 EnhancedLeNet 网络结构:

(1). 使用 HeNormal 代替 Normal 初始化, 这是由于 HeNormal 初始化在 ReLU 激活函数中表现更好，可以更好地抑制梯度消失或爆炸.

(2). 在卷积层后引入批归一化层（BatchNorm）, 其主要作用是通过对神经网络每一层的输入进行标准化处理，主要用来提升模型稳定性.

(3). 在全连接层后添加 Dropout 层, 这是一种常用的正则化技术以减少过拟合. 在训练过程中，Dropout 会以概率 p 将一部分神经元的输出设为 0，从而减少模型对特定神经元的依赖, 迫使网络学习更鲁棒的特征。

(4). 增加了通道数和卷积层数, 以提高模型的表达能力.

Figure 7: EnhancedLeNet 网络结构图

Listing 8: 改进的 EnhancedLeNet 网络结构

```
class EnhancedLeNet(nn.Cell):
    def __init__(self, num_classes=10, num_channel=1):
        super(EnhancedLeNet, self).__init__()
        self.conv1 = nn.Conv2d(num_channel, 16, 5, pad_mode='valid',
            weight_init=HeNormal(mode='fan_out'))
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, 5, pad_mode='valid', weight_init=
            HeNormal(mode='fan_out'))
        self.bn2 = nn.BatchNorm2d(32)
        self.relu = nn.ReLU()
        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(32*5*5, 256, weight_init=HeNormal(mode='
            fan_out'))
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Dense(256, 128, weight_init=HeNormal(mode='fan_out')
            )
        self.dropout2 = nn.Dropout(p=0.3)
        self.fc3 = nn.Dense(128, num_classes)

    def construct(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.max_pool(x)
        x = self.conv2(x)
```

```
        x = self.bn2(x)
        x = self.relu(x)
        x = self.max_pool(x)
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.dropout1(x)
        x = self.relu(self.fc2(x))
        x = self.dropout2(x)
        return self.fc3(x)
```

同时, 为协同改进性能, 我还使用 Adam 优化器代替 Momentum 优化器, 引入了自适应学习率并添加了权重衰减来进一步减少过拟合, 来获得更好的训练稳定性和收敛速度.

Listing 9: 优化器调整

```
network = EnhancedLeNet()

# 使用Adam优化器并添加权重衰减
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
net_opt = nn.Adam(network.trainable_params(), learning_rate=0.001, weight_decay=1e-4)
```

使用改进后的网络结构进行了测试, 结果如下图. 可以看出,EnhancedLeNet 网络的测试准确率最终可以达到 99.3% 左右, 平均损失降到 0.025, 且无震荡, 无反弹, 曲线平滑, 表明模型训练效果良好, 明显优于原始 LeNet5 网络.

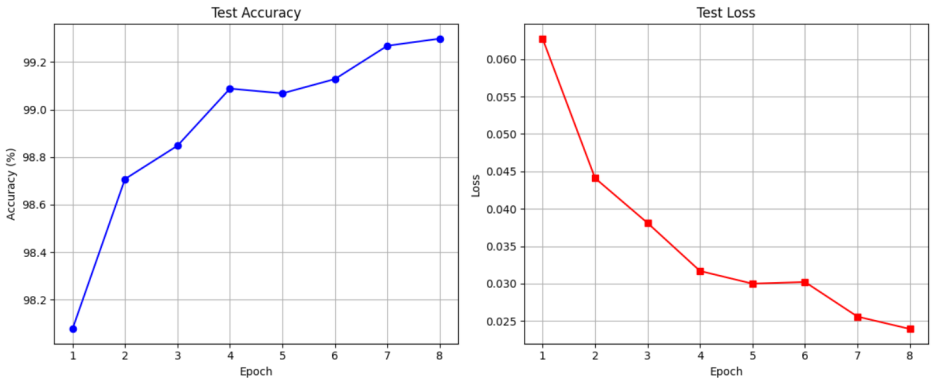

Figure 8: 改进网络结构后的测试集平均准确率和平均损失

改进后测试结果保持准确:

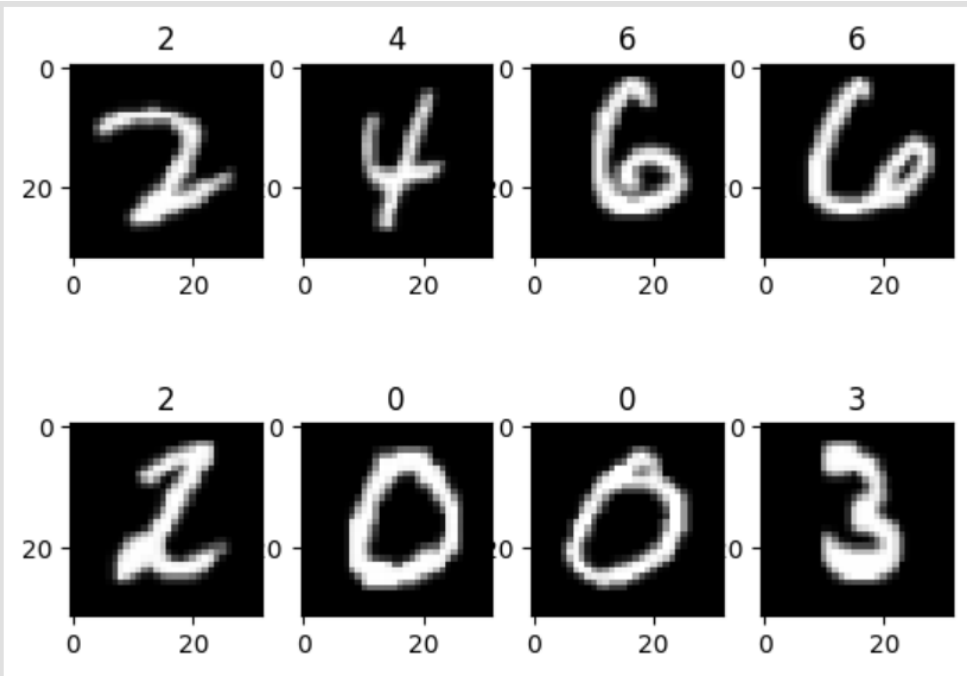

Figure 9: 改进后测试结果

5 心得体会

在这次实验中, 我首先按照手册进行了 ModelArts 云环境的搭建, 并在此基础上使用 MindSpore 框架完成了手写体数字识别的实验. 通过本次实验, 我对传统 LeNet5 的网络结构有了比较深刻的认知和理解，发现准确率和损失在训练过程中存在波动, 因此我立即查阅相关资料。并最终通过对 LeNet5 网络的改进, 提出了 EnhancedLeNet 网络结构, 获得了更好的训练效果，这对我探索和解决问题能力都有比较大的提升. 此外, 我还学习了如何使用 MindSpore 框架进行数据预处理、模型训练和测试等操作, 掌握了 Python 在人工智能开发中的常用操作. 总的来说, 本次实验让我对深度学习有了更深入的理解和实践经验.