广东工业大学计算机学院

《智能芯片原理与应用》

实验报告



|  |  |
| --- | --- |
| 课程名称： | 智能芯片原理与应用 |
| 实验名称： | 实验二：MindSpore 编程框架使用 |
| 指导教师： | 姬玉柱 |
| 学生姓名： | 陈婉瑜 |
| 学 号： | 3222004465 |
| 实验日期： | 2025-05-28 |
| 实验地点： | 工1-408 |
| 实验成绩： |  |

一、实验目的

**1.1关于本实验**

实验前理论课应该完成深度学习编程框架的讲解，学生已了解、基于 Tensorflow 或其他深度学习编程框架的基本原理。本实验介绍了Window/Linux 系统通过 Miniconda 安装 MindSpore、搭建 python 虚 拟环境，类比MindSpore、PyTorch 和 Tensorflow 的编程框架的语法区别。

**1.2实验目标：**

1. 熟悉 MindSpore 基本使用，包括tensor、数据加载及数据处理（选做）、模型搭建、模型训练与测试、模型保存与加载等，了解MindSpore 命令式编程，并完成相关代码。

2. 通过具体操作、熟悉相关类、方法等，并通过核心代码段、实现结果截图等记录实验结果。

**1.3软件介绍：MindSpore 深度学习编程框架**

昇思 MindSpore 是华为公司开发的一个全场景开源深度学习框架，旨在实现易开发、高效执行、全场景覆盖三大目标。用于支持深度学习、自然语言处理、图像处理等多个领域的应用开发，图 1 展示了MindSpore 深度学习框架的总体结构。其特点如下：

1. 全场景覆盖：MindSpore 支持多种设备、多种场景下的深度学习模型开发和发布，包括 CPU、GPU、Ascend 等处理器，同时也支持分布式和端到端联合训练。

2. 非透明自动微分：MindSpore 支持动态图，它可以自动微分，而且支持自动微分和数字信号处理等各方面技术。

3. 安全保障机制：MindSpore 针对最近出现的深度学习模型安全问题，加入了多种安全保障机制。

4. 速度快、易用性高：MindSpore 使用静态图作为开发模式，能够让用户更方便地进行模型优化，在保证速度的同时还能提高易用性。

MindSpore 的使用场景包括：

自然语言处理：MindSpore 能够帮助开发者快速构建和训练基于自然语言处理的模型，例如机器翻译、语音识别等。

图像处理：MindSpore 提供了丰富的算法支持，能够帮助开发者轻松构建和训练图像识别、图像分类等模型。

数据分析和预测：MindSpore 能够处理多维向量、矩阵、张量等数据，能够支持数据分析和预测等工作。

MindSpore 支持的语言包括 Python、C++等多种编程语言，在Python 中几乎可以实现全部功能，极大程度提高了开发效率。

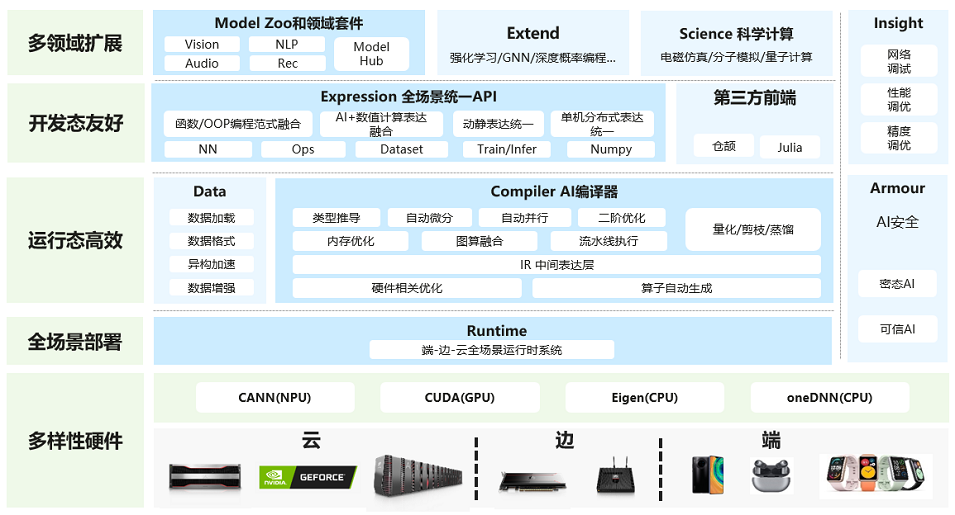


图 1 昇思 MindSpore 总体架构1

二、实验环境

**2.1硬件设备：**

版本 Windows 11

版本号 24H2

操作系统版本 26100.4061

设备名称 Yunyu

处理器 AMD Ryzen 5 6600HS Creator Edition 3.30 GHz

机带 RAM 16.0 GB (13.7 GB 可用)

设备 ID EA063112-4DA2-4364-BFC1-DA10B95B0380

产品 ID 00342-30614-55872-AAOEM

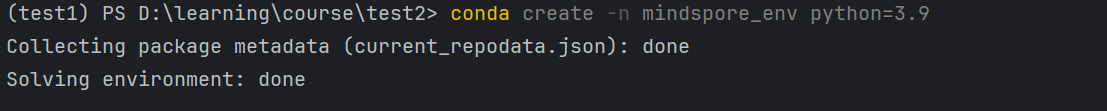
系统类型 64 位操作系统, 基于 x64 的处理器

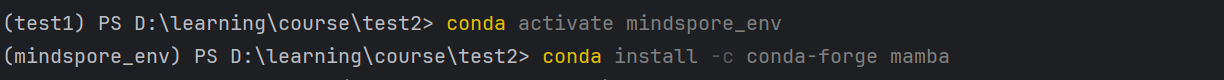
**2.2软件：**

PyCharm Community Edition 2023.3.3

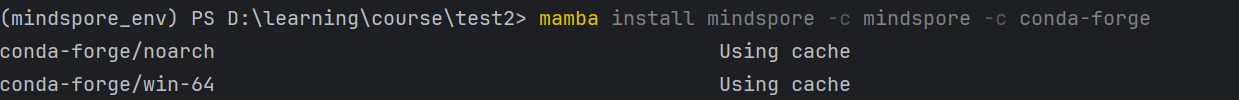
**2.3环境配置：**

（1）创建python3.9新环境并切换





（2）使用 Mamba 安装 MindSpore



三、实验内容与实验要求

**1 . MindSpore 的基本使用**

**3.1张量的定义与使用**

实验要求：

结合示例代码，可尝试改变示例代码相关内容与配置，运行结果，并截图记录。

引入 mindspore 库。若成功引入，通常无任何显示

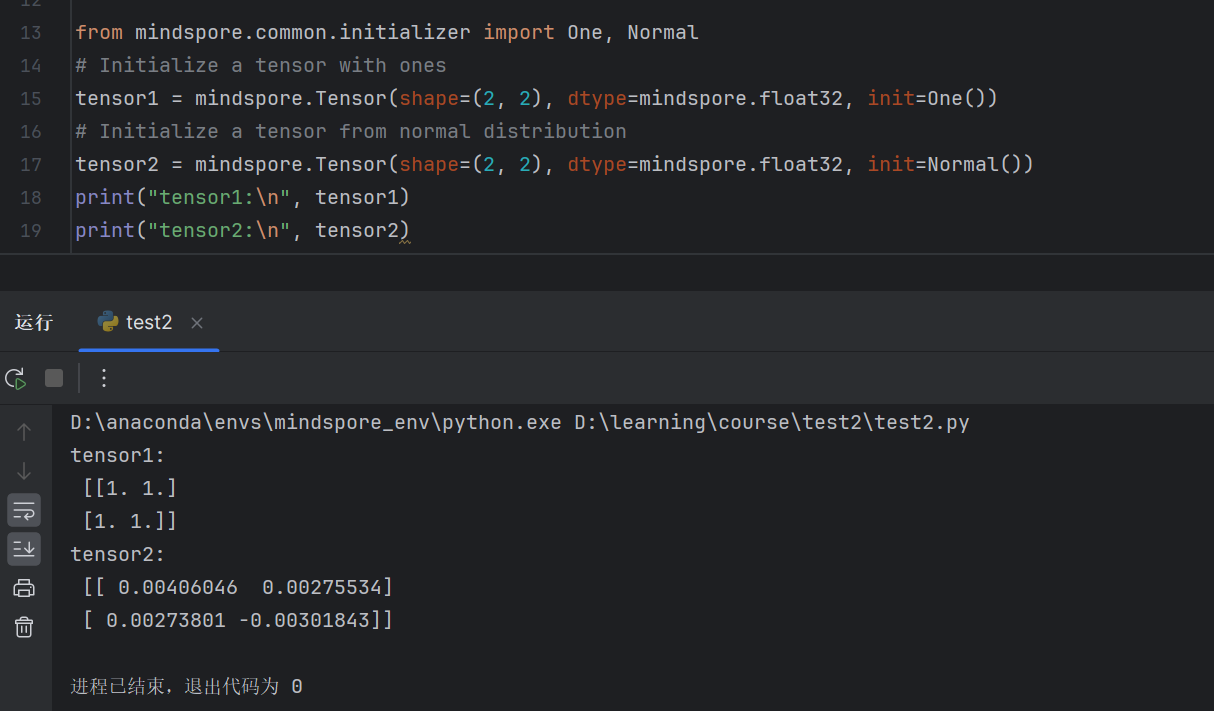
张量(Tensor)的概念对于理解深度学习框架中的数据处理和构成是非常重要的。通常，对于计算机视觉任务来说，接触最多的是图像类型数据，通常会出现四维张量，即为(N，H，W，C)维度的 tensor。 利用上述张量。再 Mindspore 中张量的使用与 Tensorflow 类似，主要分以下几个方面：

1. 创建张量：张量的创建方式有多种，构造张量时，支持传入`Tensor`、`float`、`int`、`bool`、`tuple`、`list`和`numpy.ndarray`类型。
2. **由数据直接生成：**
3. **由 Numpy 数字生成：**
4. **使用 init 初始化器构造张量：**

当使用 init 初始化器对张量进行初始化时，支持传入的参数有 init、 shape、dtype，其中：init: 支持传入 initializer 的子类。

shape: 支持传入 list、tuple、 int。

dtype: 支持传入 mindspore.dtype。



1. **继承另外一个 Tensor 的大小、类型，例如申请全零和全 1 张量时**



1. 查看张量属性

张量的属性包括形状、数据类型、转置张量、单个元素大小、占 用字节数量、维数、元素个数和每一维步长。可通过查看如下属性， 显示具体类型，例如：

形状（shape）：Tensor 的 shape，是一个 tuple。

数据类型（dtype）：Tensor 的 dtype，是 MindSpore 的一个数据类型。

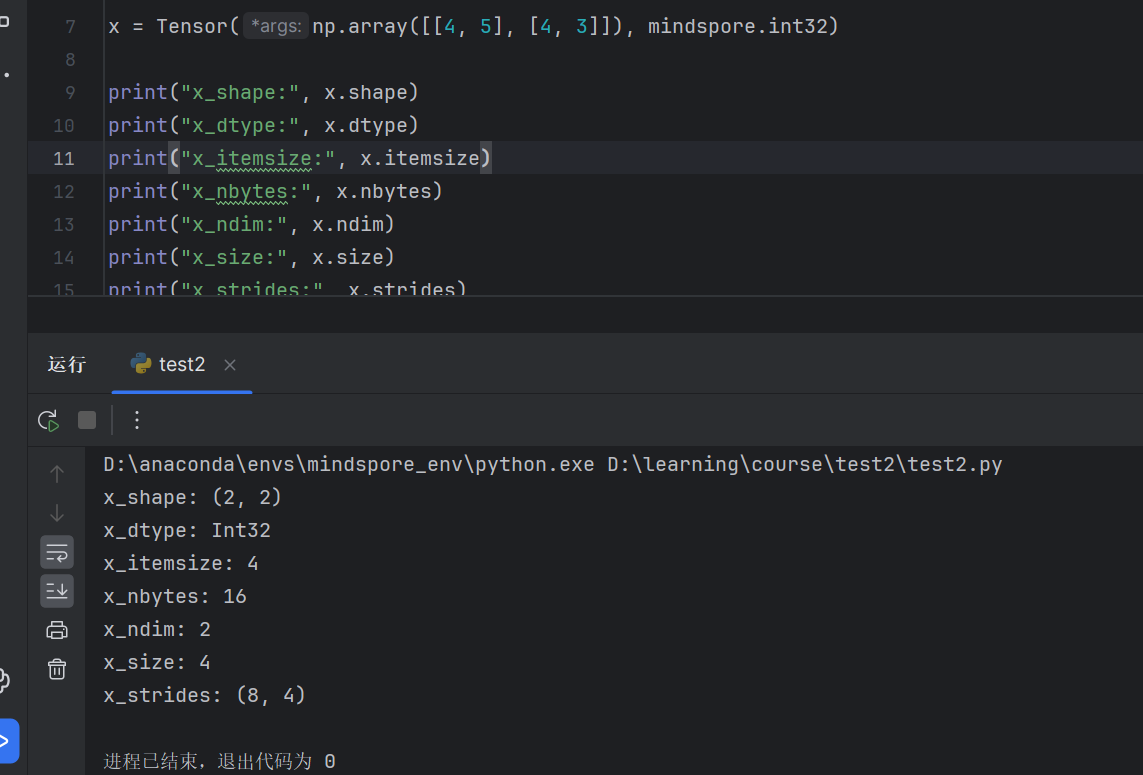
单个元素大小（itemsize）： Tensor 中每一个元素占用字节数，是一个整数。

占用字节数量（nbytes）： Tensor 占用的总字节数，是一个整数。

维数（ndim）： Tensor 的秩，也就是 len(tensor.shape)，是一个整数。

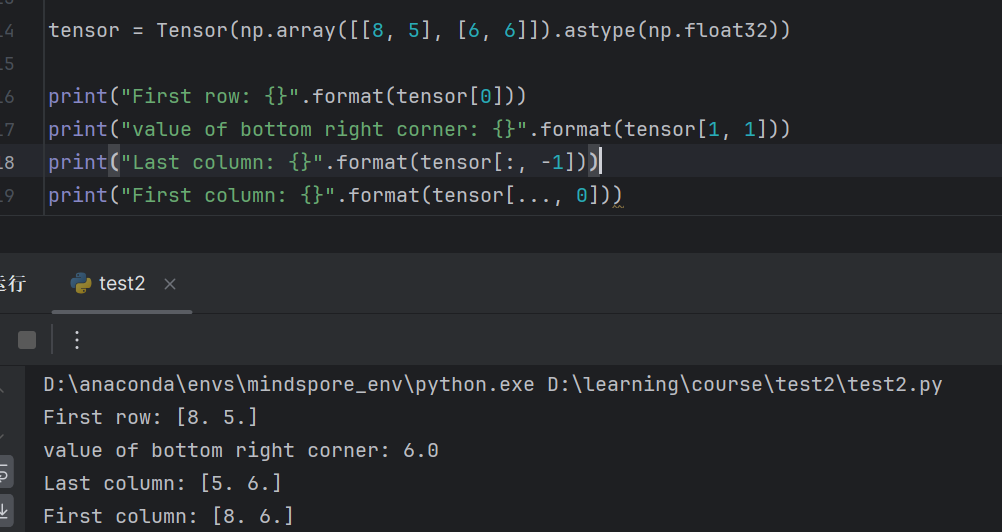
元素个数（size）： Tensor 中所有元素的个数，是一个整数。

每一维步长（strides）： Tensor 每一维所需要的字节数，是一个 tuple。



C. 张量如何索引

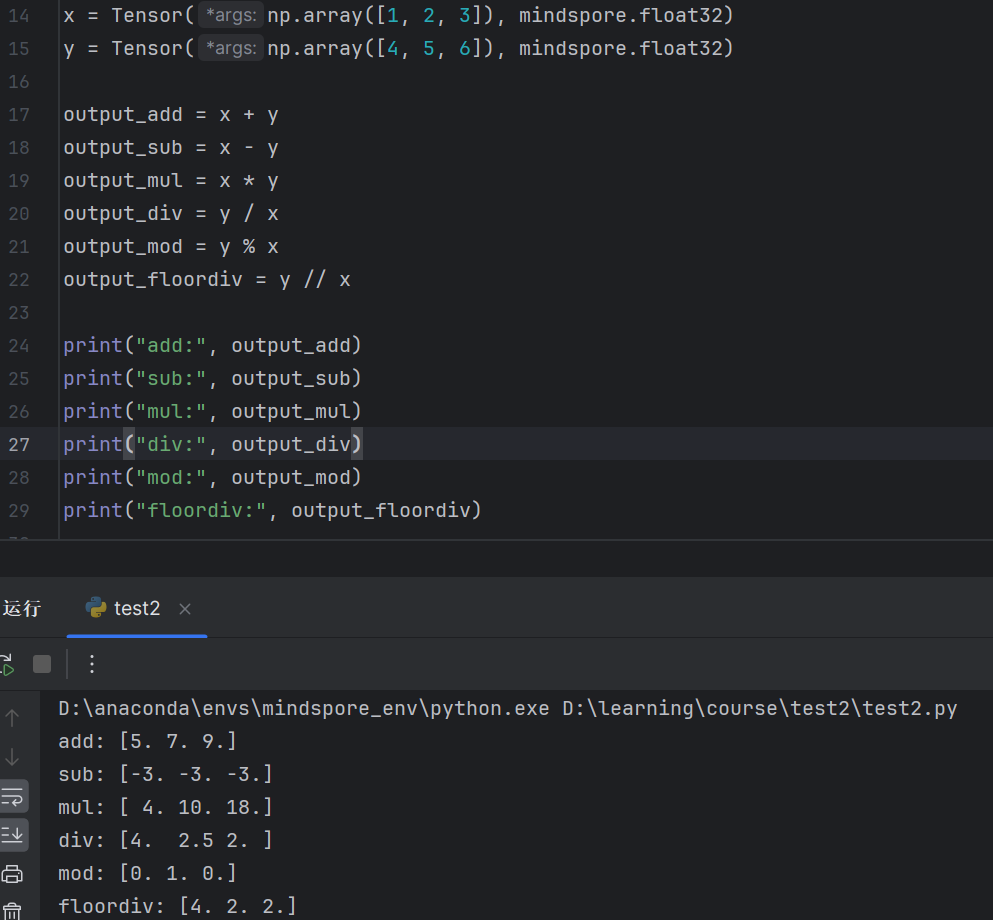
即如何获得具位置、范围 Tensor 中的数值。Tensor 索引与 Numpy 索引类似，索引从 0 开始编制，负索引表示按倒序编制，冒号:和 ... 用于对数据进行切片。



D. 张量运算

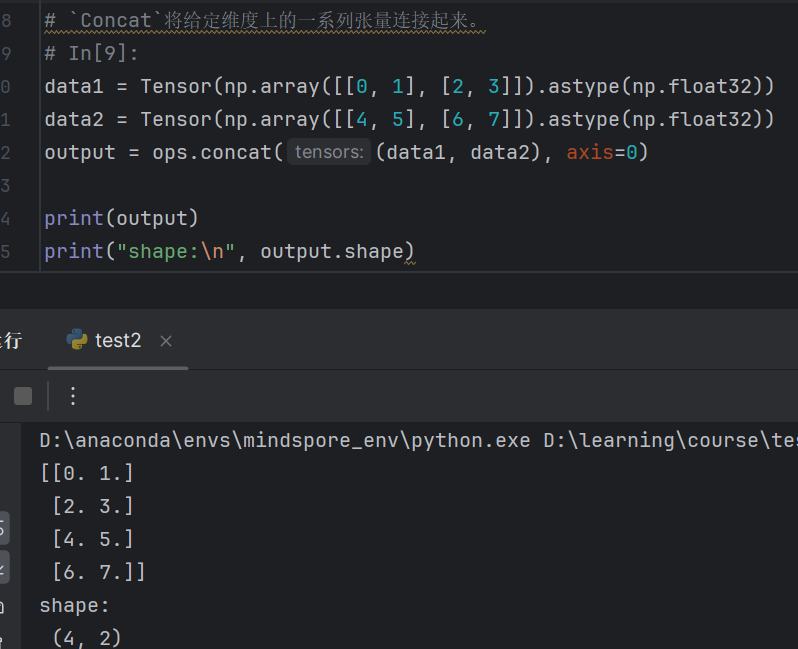
张量之间有很多运算，包括算术、线性代数、矩阵处理（转置、 标引、切片）、采样等，张量运算和 NumPy 的使用方式类似，下面介绍其中几种操作：

试查阅相关材料，实验完成转置、矩阵内积、外积运算等操作。

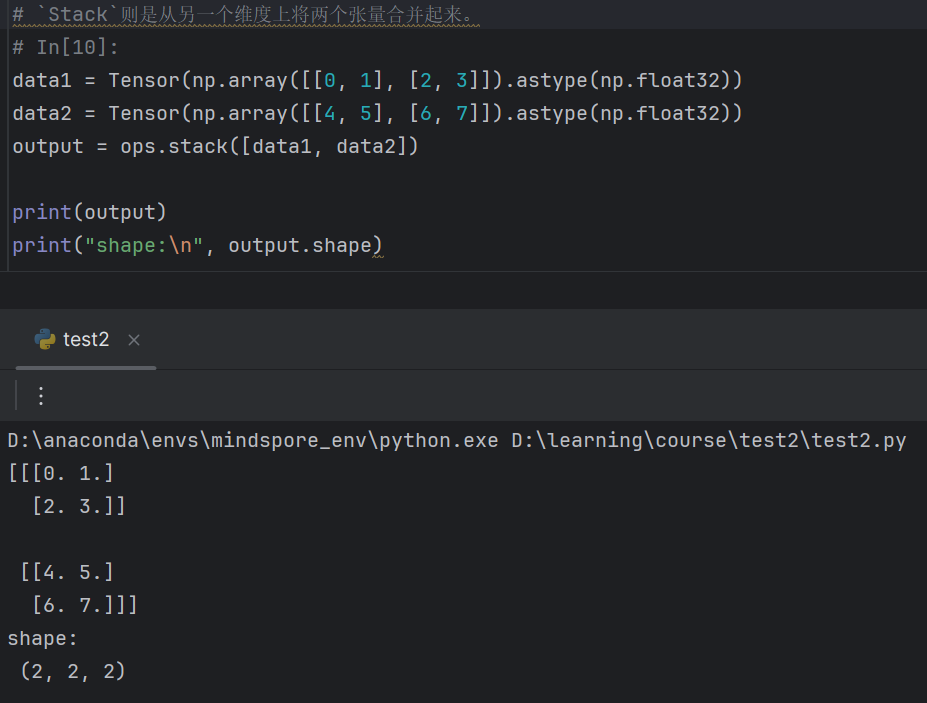


E. 其他张量操作

（1）Tensor 级联 concat：通过指定具体的维度，进行级联



（2）Tensor 的堆叠：



**3.2 MindSpore 网络构建**

实验要求：

结合示例代码，可尝试改变示例代码相关内容与配置，运行结果，并截图记录。

与 Tensorflow 类似，mindspore.nn 中提供了常见神经网络层的实 现，在 MindSpore 中，Cell 类是构建所有网络的基类，也是网络的基 本单元。一个神经网络模型表示为一个 Cell，它由不同的子 Cell 构 成。使用这样的嵌套结构，可以简单地使用面向对象编程的思维，对 神经网络结构进行构建和管理。构建前，需要引入 mindspore 的 nn。

1. 定义模型类：

a. 当定义神经网络时，可以继承 nn.Cell 类，在\_\_init\_\_方法中进行 子Cell 的实例化和状态管理，在 construct 方法中实现 Tensor 操作。

b. 构建完成后，实例化 Network 对象，并查看其结构。

c. 默认动态图机制，随便输入一个输入结果，测试下模型输出：

d. 多类分类，通过一个 nn.Softmax 层实例来获得预测概率。

e. 查看输出结果

（2）模型层 Layers、或 Op：

对于每层的属性及其计算结果，通过命令式编程，可以测试其模 型层的计算和设置方式。分解上节构造的神经网络模型中的每一层。 首先我们构造一个 shape 为(3, 28, 28)的随机数据（3 个 28x28 的图 像），依次通过每一个神经网络层来观察其效果

a. nn.Flatten: 实例化 nn.Flatten 层，将 28x28 的 2D 张量转换为 784 大小的连续数组

b. nn.Dense: 全连接层，其使用权重和偏差对输入进行线性变换。

c. nn.ReLU: 非线性的激活函数，帮助神经网络学习各种复杂的特征。

d. nn.SequentialCell: 是一个有序的 Cell 容器。输入 Tensor 将按照定义 的顺序通过所有 Cell。我们可以使用 SequentialCell 来快速组合构造 一个神经网络模型。(一种搭积木的形式)。

e. nn.Softmax: 最后使用 nn.Softmax 将神经网络最后一个全连接层返回 的 logits 的值缩放为[0, 1]，表示每个类别的预测概率。axis 指定的维度数值和为 1。

（3）模型参数

网络内部神经网络层具有权重参数和偏置参数（如 nn.Dense），这 些参数会在训练过程中不断进行优化，可通过 model.parameters\_and\_names() 来获取参数名及对应的参数详情。

**2. 模型训练**

实验要求：结合示例代码，可尝试改变示例代码相关内容与配置，运行结果，并截图记录。

1. **构造 MLP 进行模型训练和测试**

模型训练一般分为四个步骤：

1. 构建数据集。

2. 定义神经网络模型。

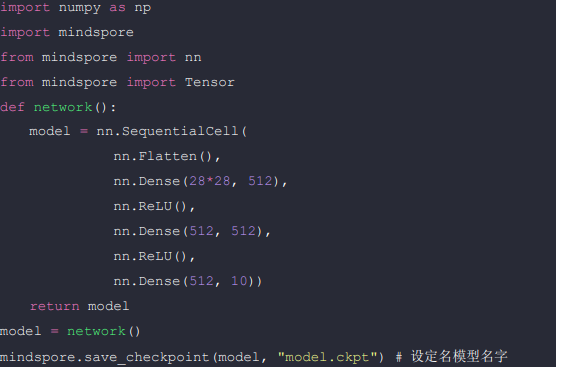
3. 定义超参、损失函数及优化器。

4. 输入数据集进行训练与评估。

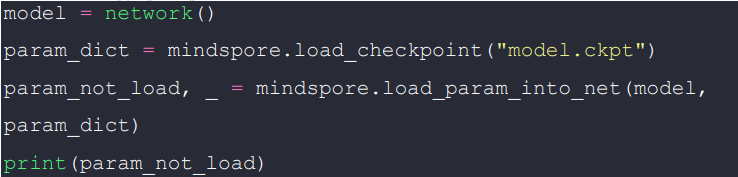
依然以 MLP 再 MNIST 数据集上训练为例，运行以下代码（mindspore\_train.py）

1. **模型存储与加载**

在训练网络模型的过程中，需要保存中间和最后的结果，用于微 调（fine-tune）和后续的模型推理与部署，本章节我们将介绍如何保 存与加载模型。保存模型使用 save\_checkpoint 接口，传入网络和指定的保存路径：



要加载模型权重，需要先创建相同模型的实例，然后使用 load\_checkpoint 和 load\_param\_into\_net 方法加载参数。



**实验 1**：结合训练测试代码，以及保存、加载模型的操作，在 mindspore\_train.py 中实现每一个 epoch，存一下模型，并且文件名跟 随 epoch 数量变化（查看 python 的字符串格式化代码，以及 if 判断语句等）。

**实验 2**：查阅相关材料（mindspore 的 nn.Conv2d 等模型层），在 mindspore\_train.py 中实现一个 CNN 模型，并在 MNIST 上训练和测 试，对比前后的精度结果。 实 验 3 （ 拓 展 可 选 做 ） ： 选 做 mindspore\_dataset.py 和 mindspore\_transforms.py 及 notebook 中的内容，运行或改变示例代码， 熟悉数据集加载方式和数据增强方法的使用，可加入实验报告中。 （save\_load 代码中 MindIR 模式在 PC 端不支持加载，仅限云和昇腾 系列计算设备上执行）

**实 验 3 （ 拓 展 可 选 做 ）** ： 选 做 mindspore\_dataset.py 和 mindspore\_transforms.py 及 notebook 中的内容，运行或改变示例代码， 熟悉数据集加载方式和数据增强方法的使用，可加入实验报告中。 （save\_load 代码中 MindIR 模式在 PC 端不支持加载，仅限云和昇腾 系列计算设备上执行）

四、实验过程与分析

**1 . MindSpore 的基本使用**

**3.1张量的定义与使用**

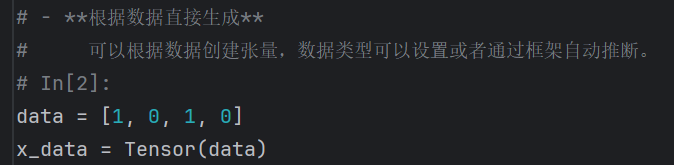
引入 mindspore 库。若成功引入，通常无任何显示



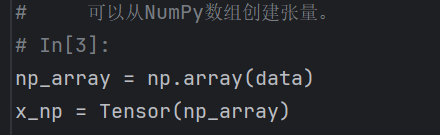
1. 创建张量：张量的创建方式有多种，构造张量时，支持传入`Tensor`、`float`、`int`、`bool`、`tuple`、`list`和`numpy.ndarray`类型。

**运行代码及结果如下：**

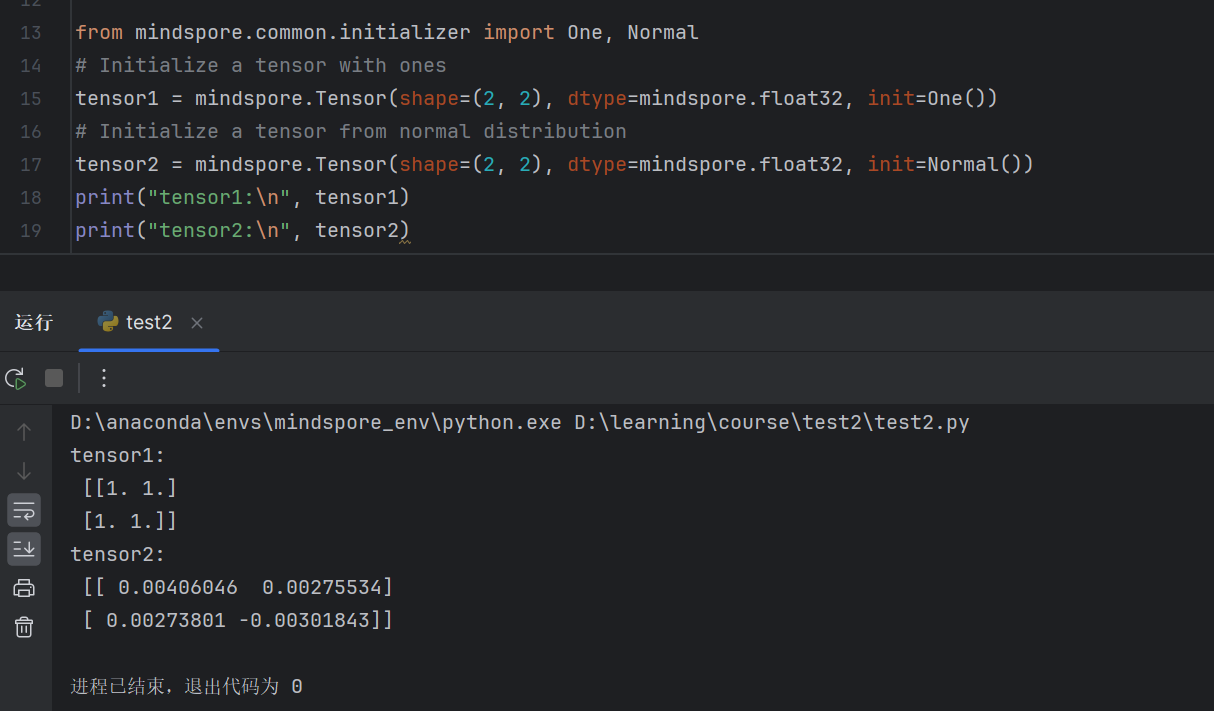
1. **由数据直接生成：**



1. **由 Numpy 数字生成：**



1. **使用 init 初始化器构造张量：**



1. **继承另外一个 Tensor 的大小、类型，例如申请全零和全 1 张量时**



1. 查看张量属性

张量的属性包括形状、数据类型、转置张量、单个元素大小、占 用字节数量、维数、元素个数和每一维步长。可通过查看如下属性， 显示具体类型，例如：

形状（shape）：Tensor 的 shape，是一个 tuple。

数据类型（dtype）：Tensor 的 dtype，是 MindSpore 的一个数据类型。

单个元素大小（itemsize）： Tensor 中每一个元素占用字节数，是一个整数。

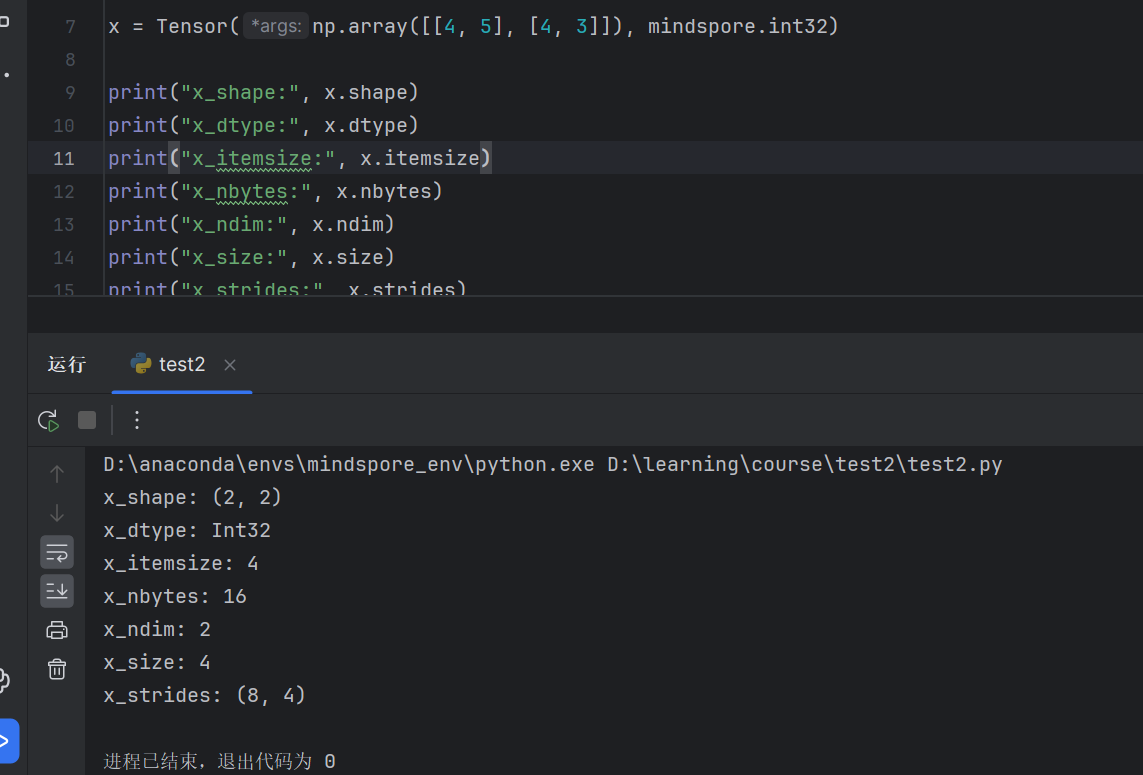
占用字节数量（nbytes）： Tensor 占用的总字节数，是一个整数。

维数（ndim）： Tensor 的秩，也就是 len(tensor.shape)，是一个整数。

元素个数（size）： Tensor 中所有元素的个数，是一个整数。

每一维步长（strides）： Tensor 每一维所需要的字节数，是一个 tuple。

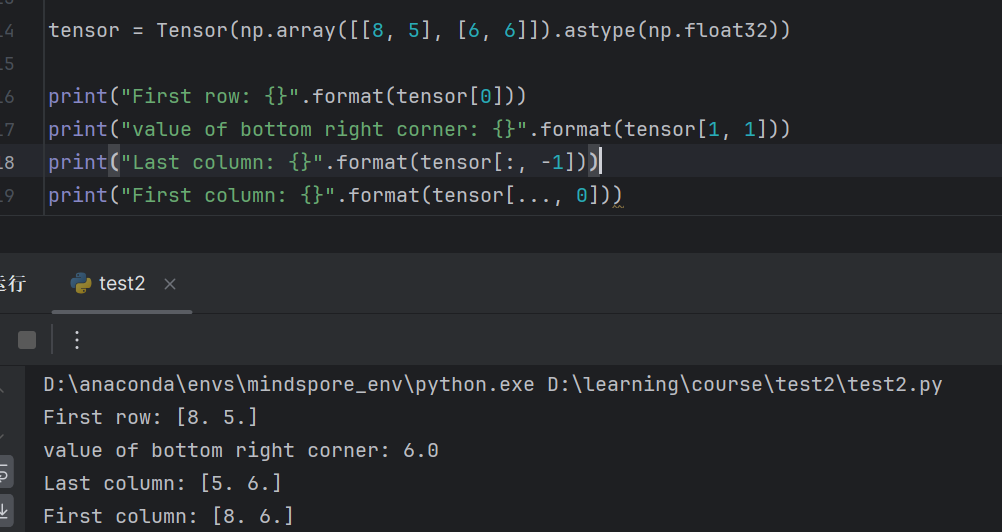
**运行代码及结果如下：**



1. 张量如何索引

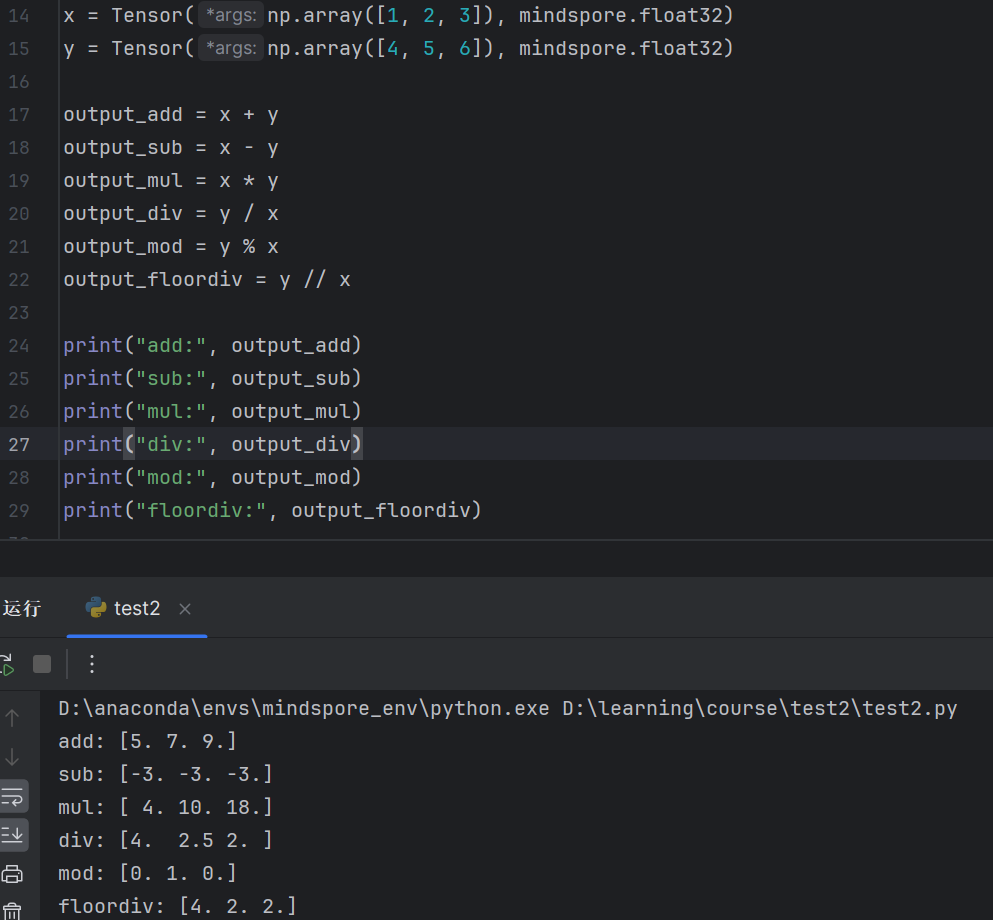
即如何获得具位置、范围 Tensor 中的数值。Tensor 索引与 Numpy 索引类似，索引从 0 开始编制，负索引表示按倒序编制，冒号:和 ... 用于对数据进行切片。

**运行代码及结果如下：**

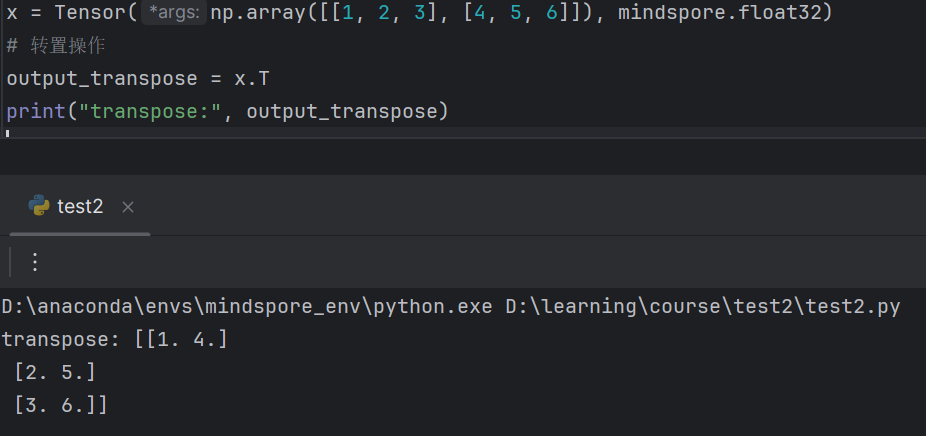


1. 张量运算

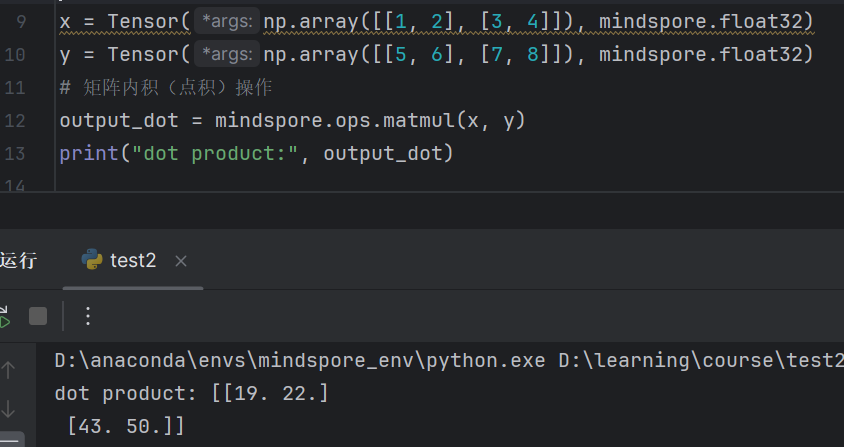
**运行代码及结果如下：**



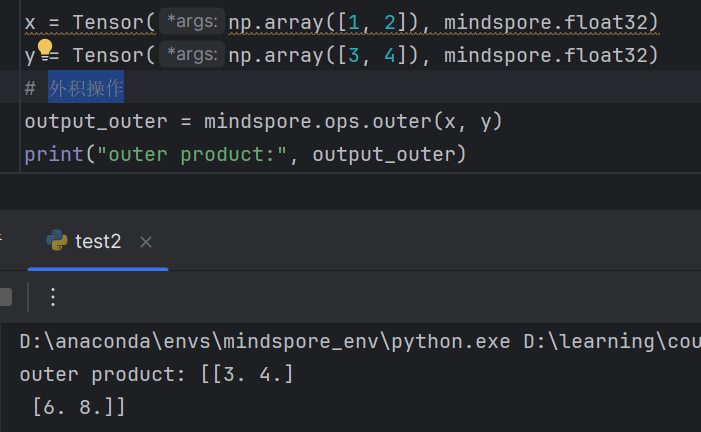
转置操作：



矩阵内积（点积）操作



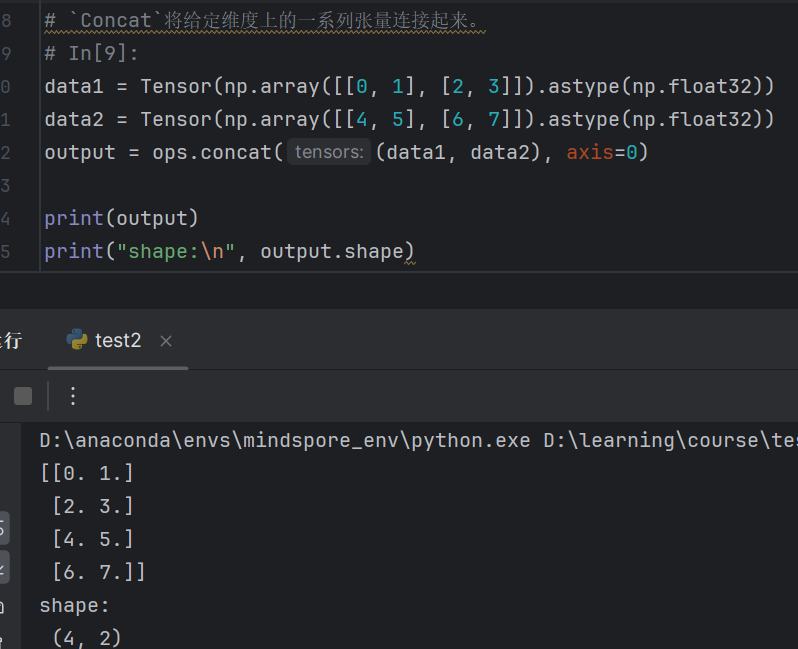
外积操作



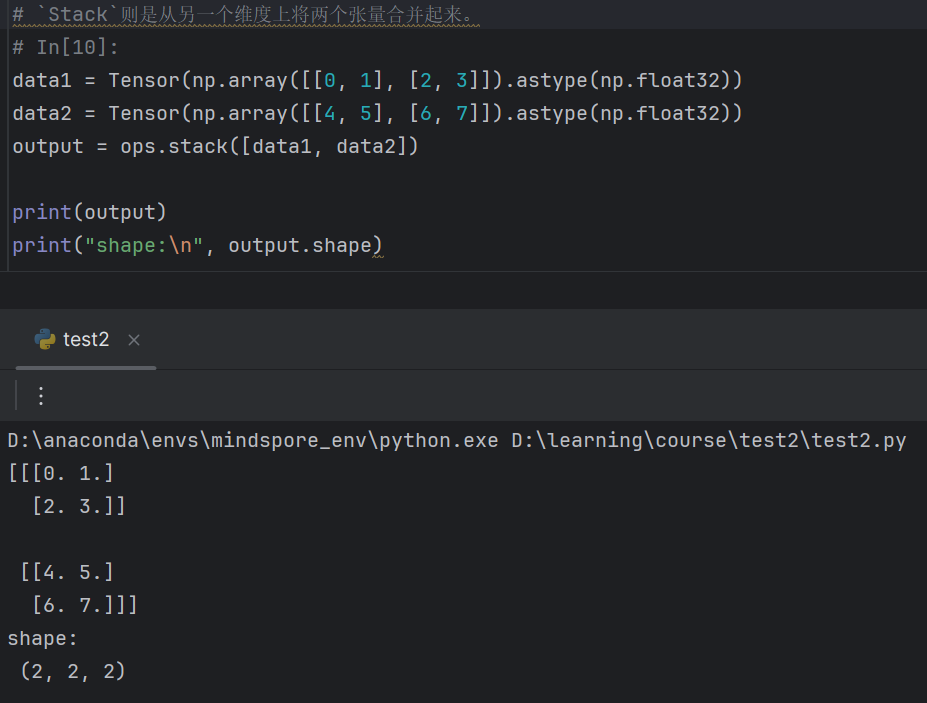
1. 其他张量操作

**运行代码及结果如下：**

（1）Tensor 级联 concat：通过指定具体的维度，进行级联



（2）Tensor 的堆叠：



**3.2 MindSpore 网络构建**

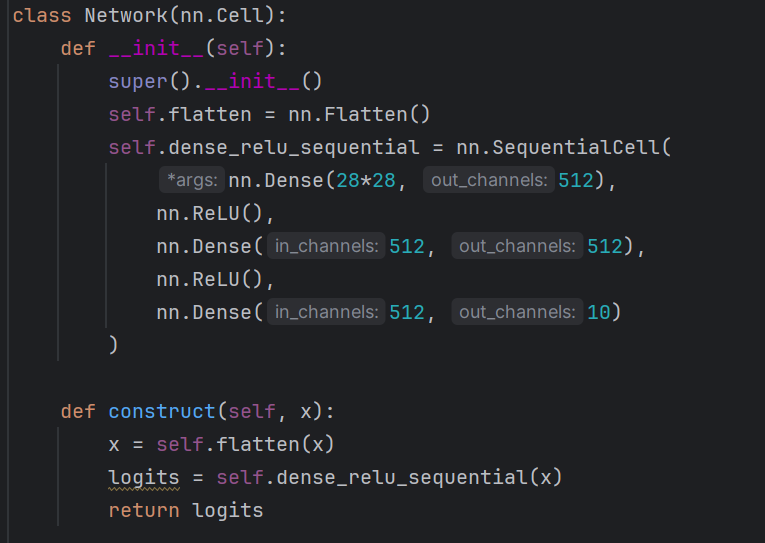
（1）定义模型类： 当定义神经网络时，可以继承 nn.Cell 类，在\_\_init\_\_方法中进行 子 Cell 的实例化和状态管理，在 construct 方法中实现 Tensor 操作。

构建完成后，实例化 Network 对象，并查看其结构。

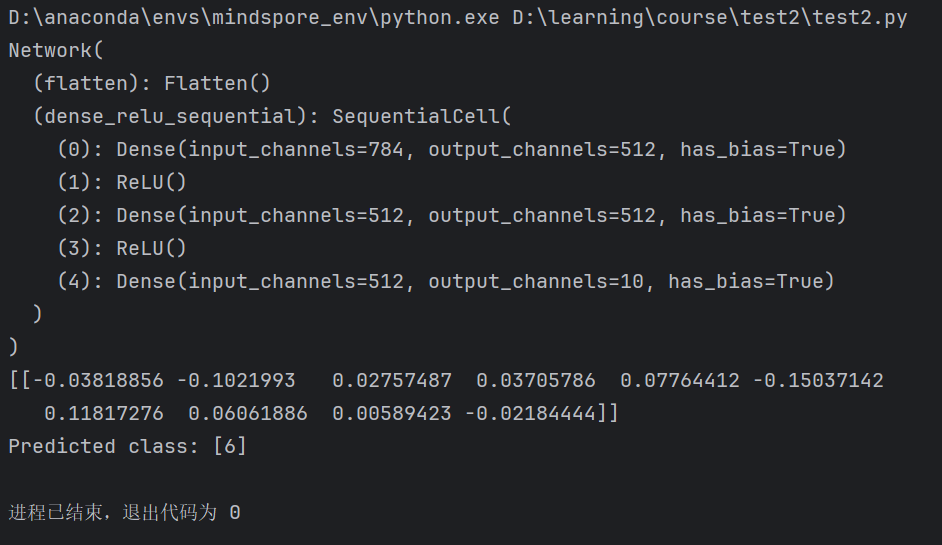
默认动态图机制，随便输入一个输入结果，测试下模型输出：

多类分类，通过一个 nn.Softmax 层实例来获得预测概率。

**运行代码及结果如下：**





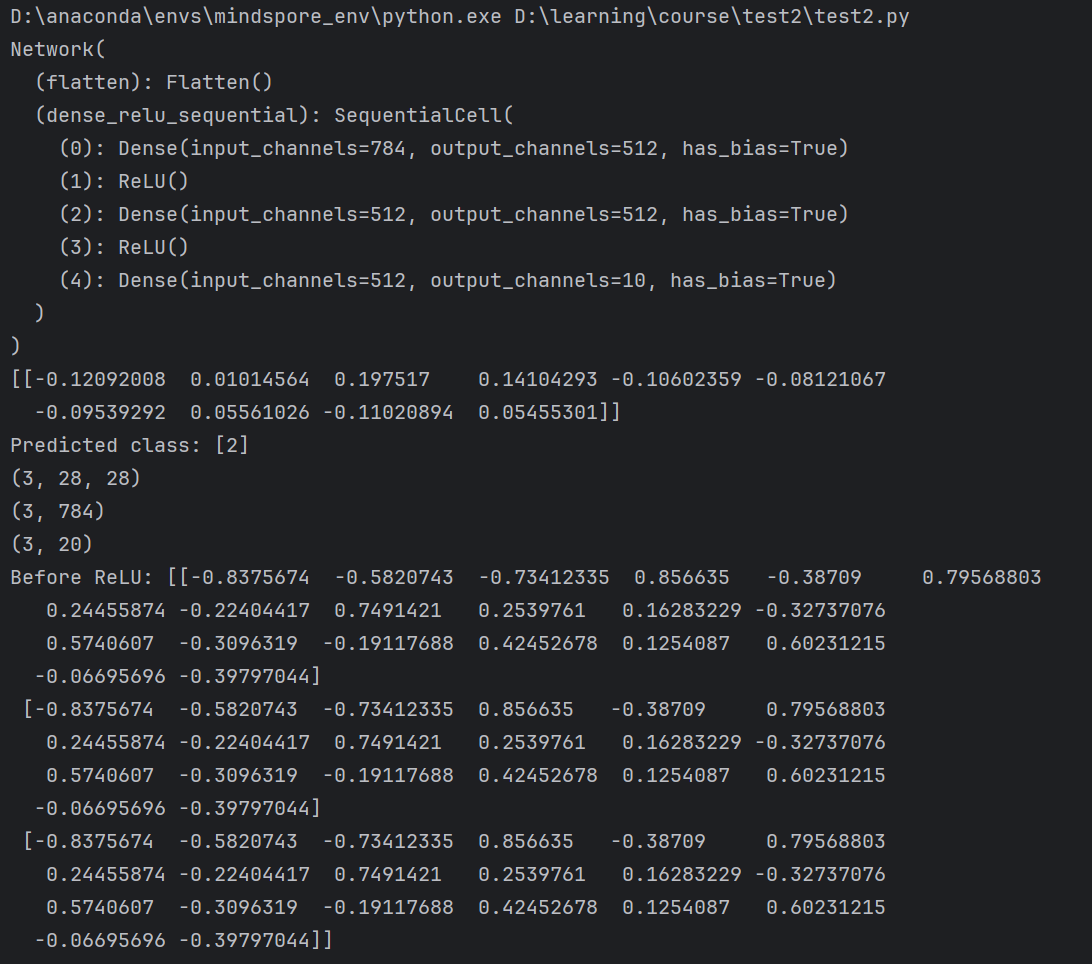


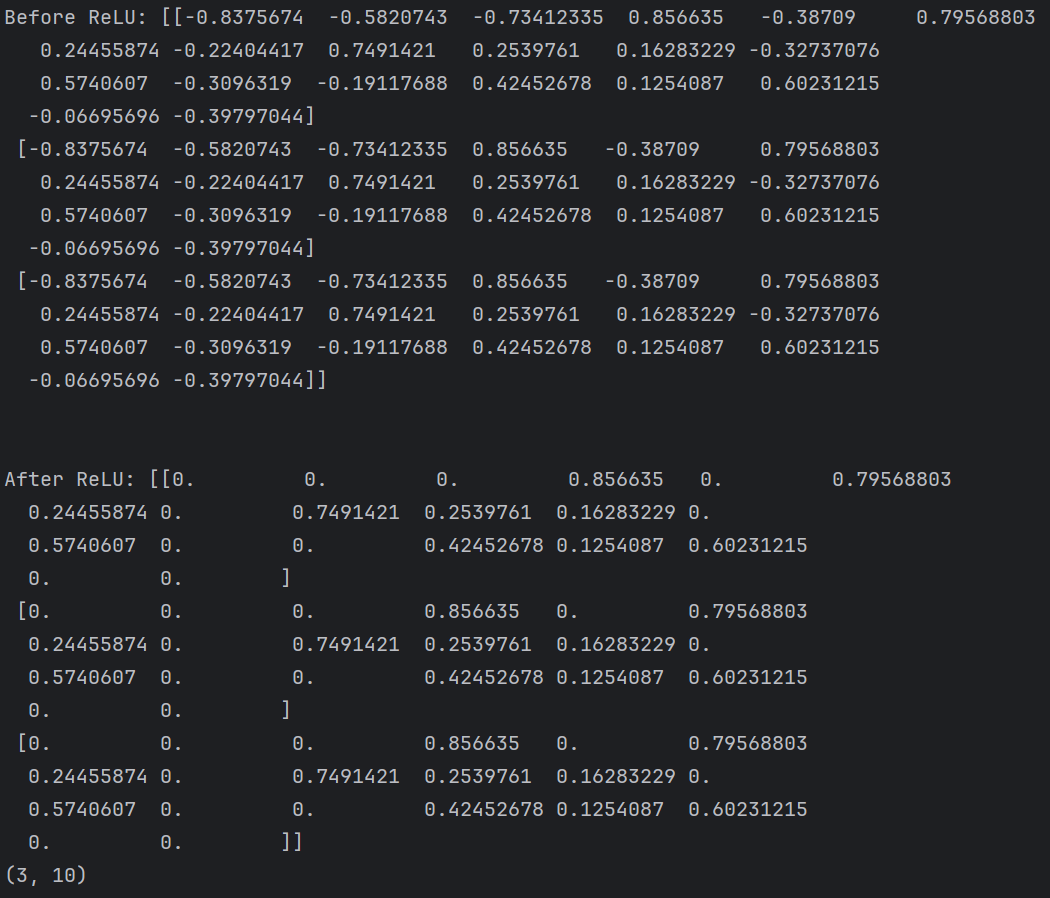
（2）模型层 Layers、或 Op：

对于每层的属性及其计算结果，通过命令式编程，可以测试其模 型层的计算和设置方式。分解上节构造的神经网络模型中的每一层。 首先我们构造一个 shape 为(3, 28, 28)的随机数据（3 个 28x28 的图 像），依次通过每一个神经网络层来观察其效果

**运行代码及结果如下：**



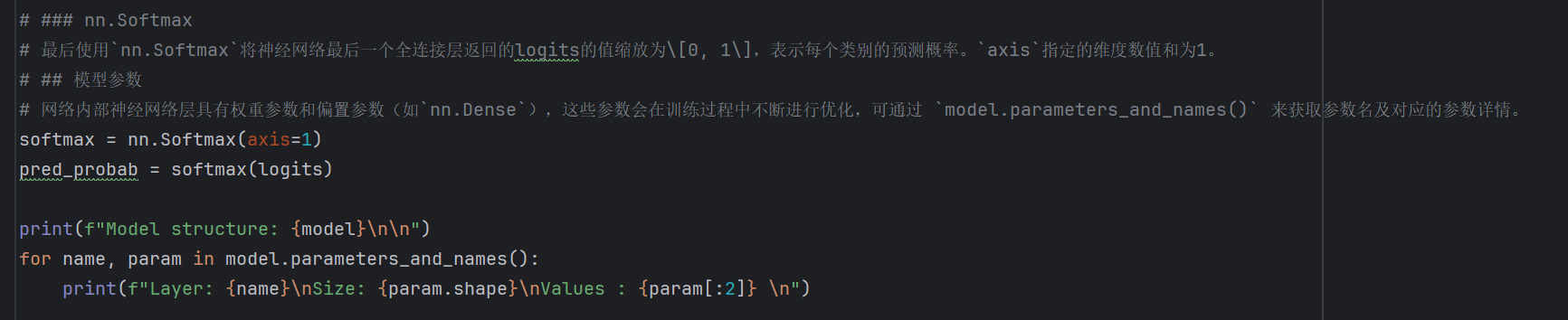


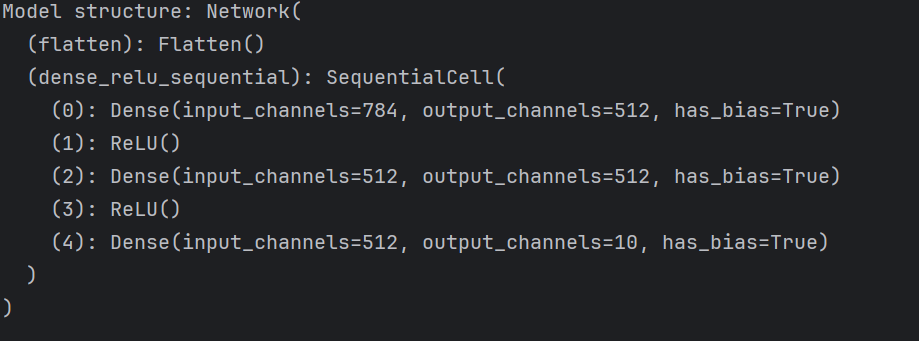


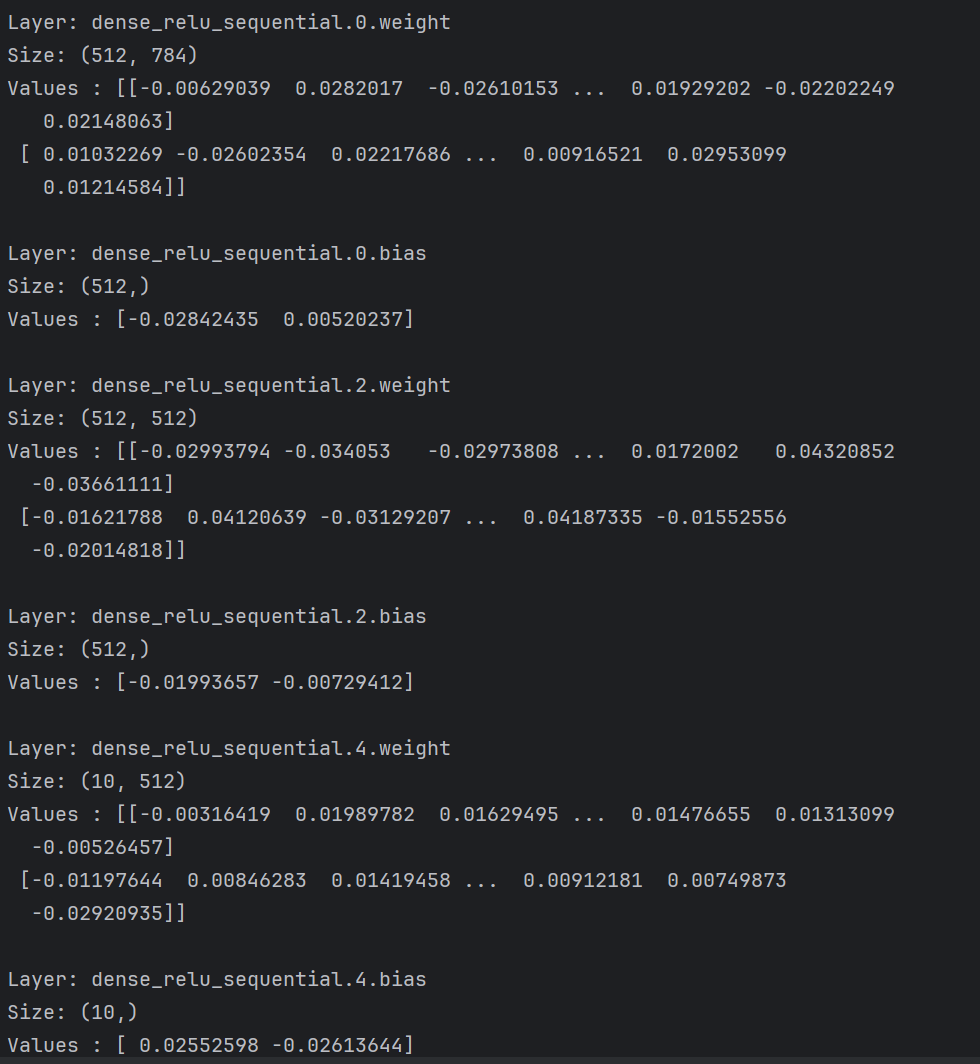
（3）模型参数

网络内部神经网络层具有权重参数和偏置参数（如 nn.Dense），这 些参数会在训练过程中不断进行优化，可通过 model.parameters\_and\_names() 来获取参数名及对应的参数详情。

**运行代码及结果如下：**





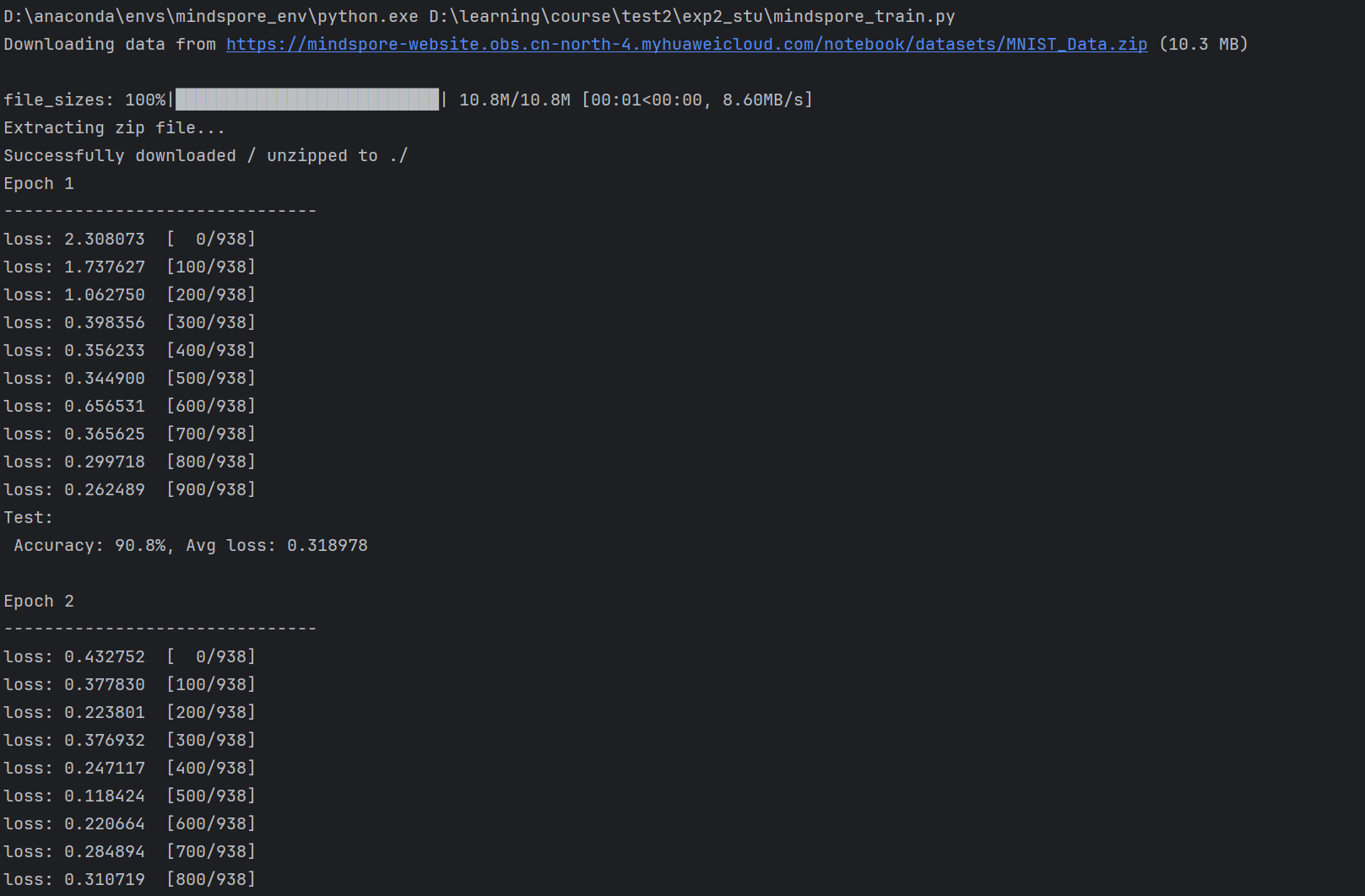


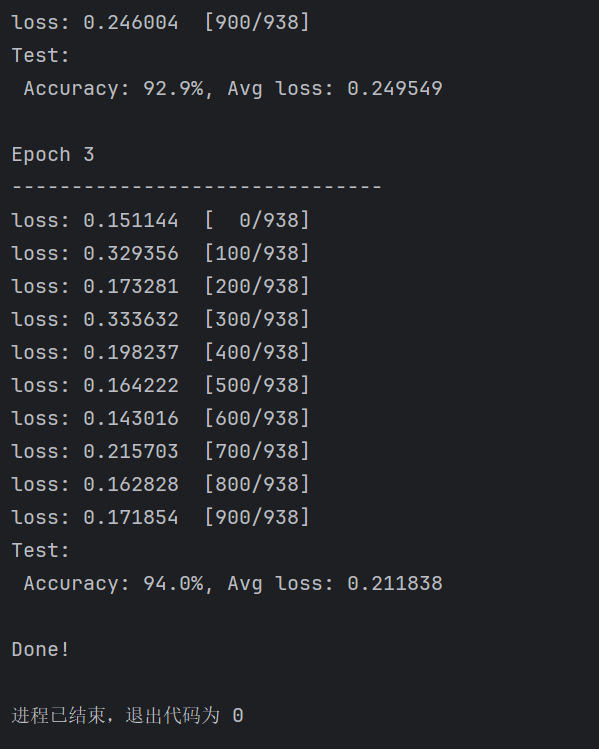
**2. 模型训练**

1. **构造 MLP 进行模型训练和测试**

以 MLP 再 MNIST 数据集上训练为例，运行以下代码（mindspore\_train.py）：

**运行结果如下：**





1. **模型存储与加载**

保存模型使用 save\_checkpoint 接口，传入网络和指定 的保存路径：

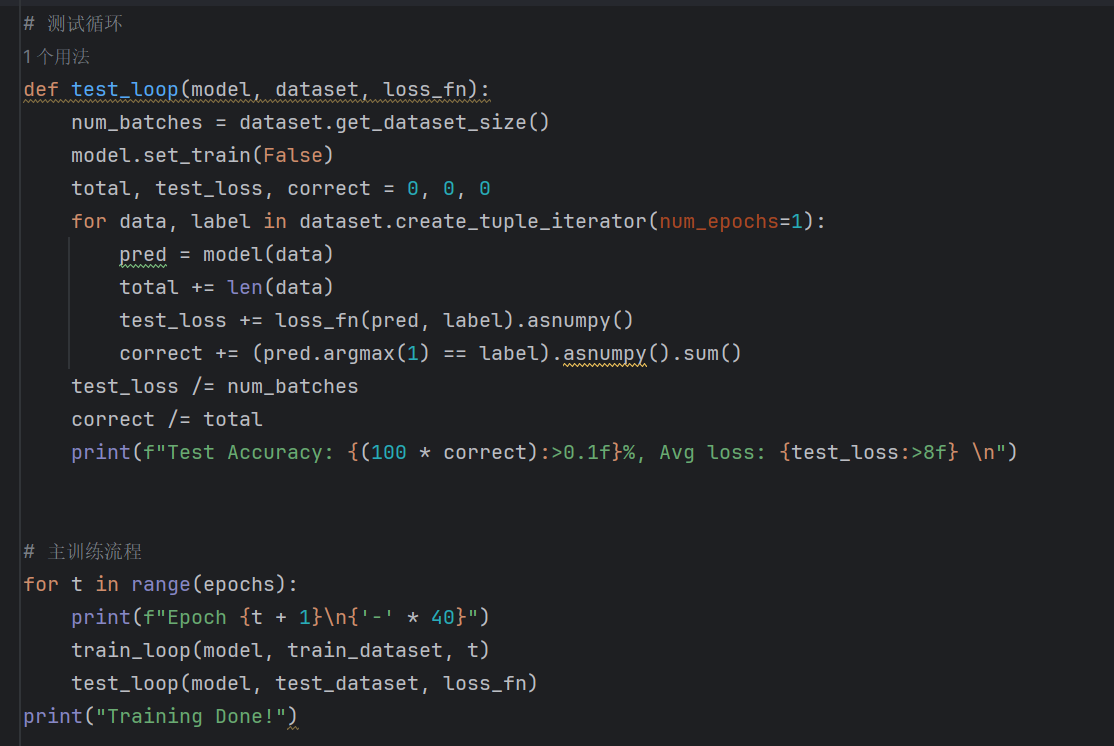
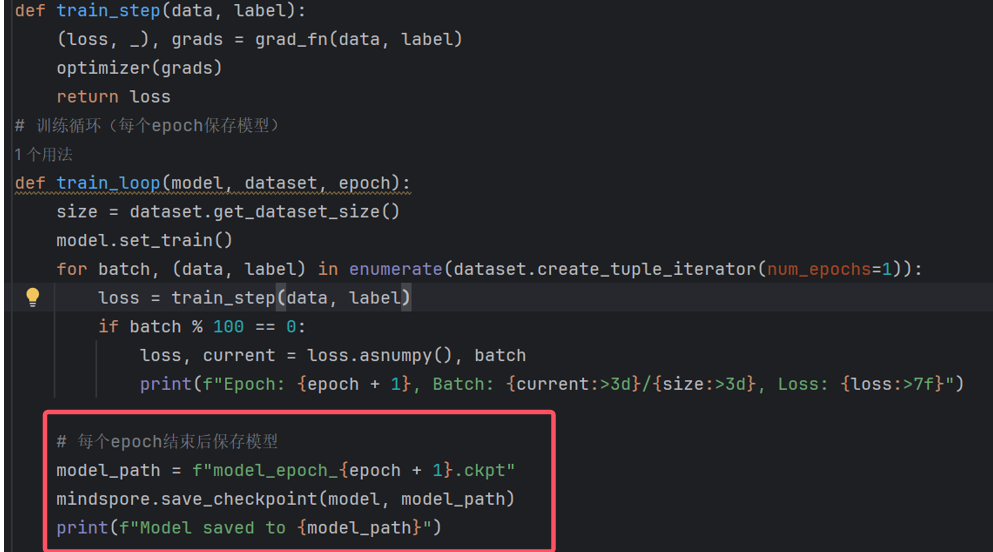
要加载模型权重，需要先创建相同模型的实例，然后使用 load\_checkpoint 和 load\_param\_into\_net 方法加载参数。

**实验 1：**结合训练测试代码，以及保存、加载模型的操作，在 mindspore\_train.py 中实现每一个 epoch，存一下模型，并且文件名跟 随 epoch 数量变化（查看 python 的字符串格式化代码，以及 if 判断 语句等）。

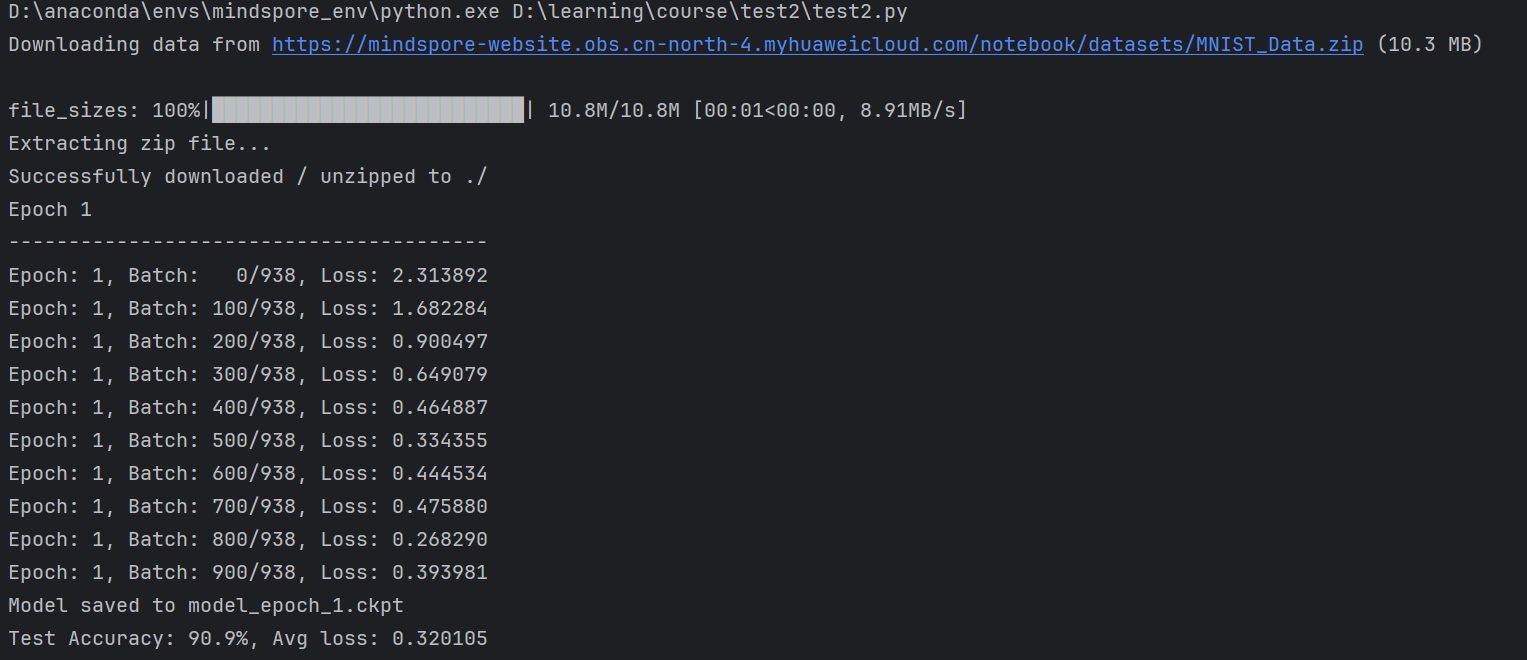
**具体实现代码：**

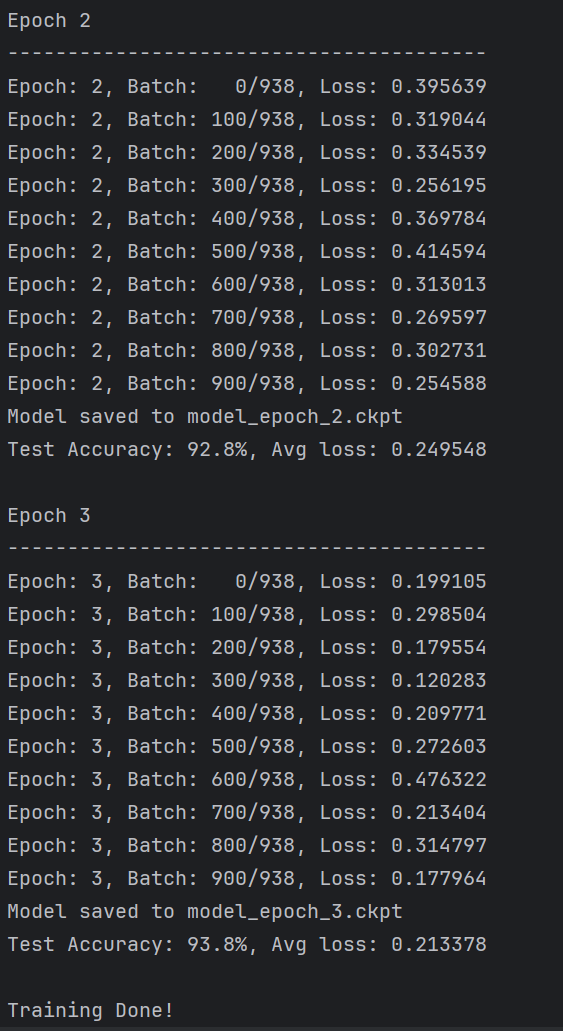


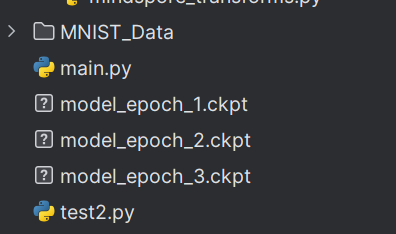




**运行结果：**





**模型保存文件：**

**模型保存实现：**

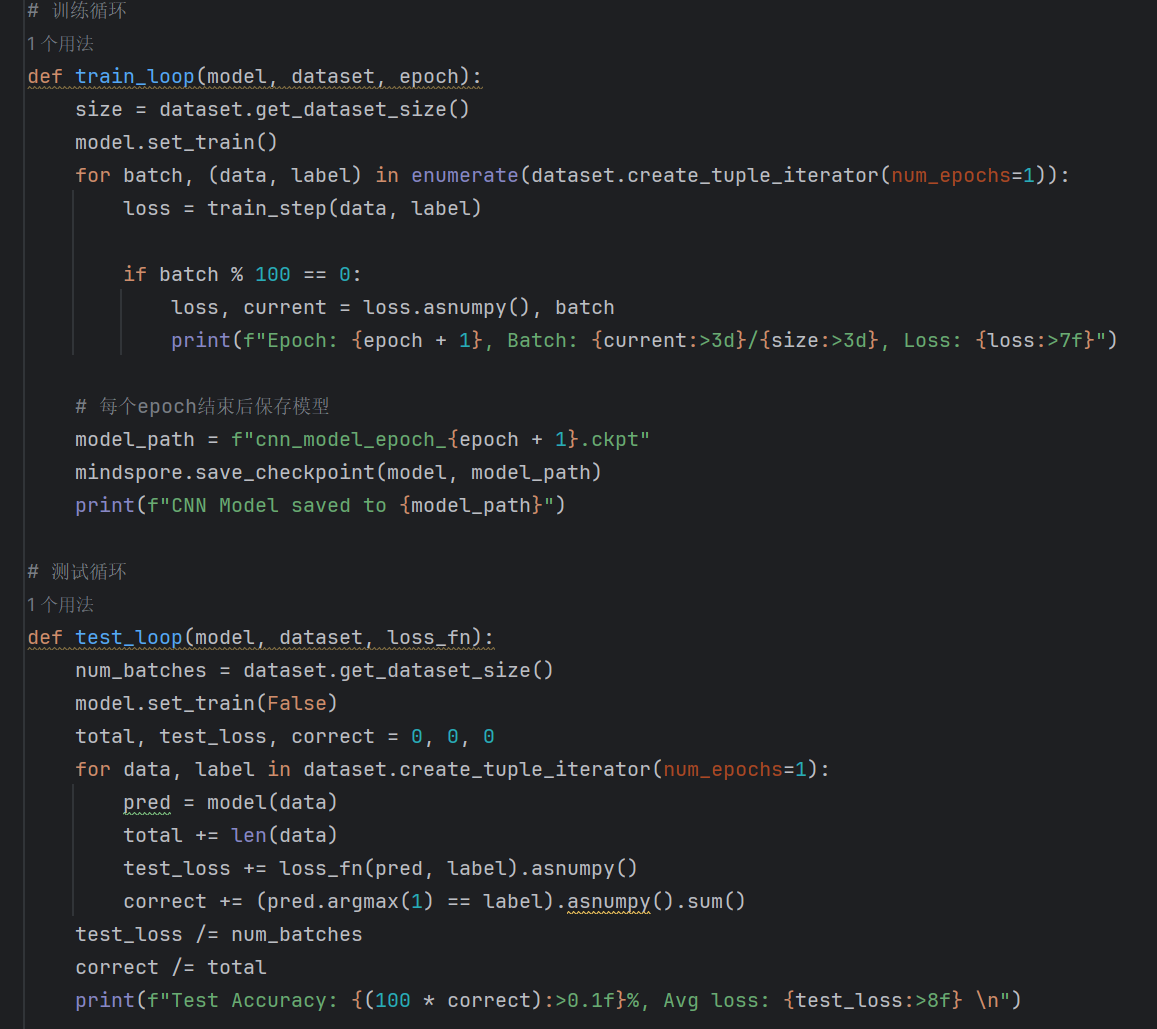
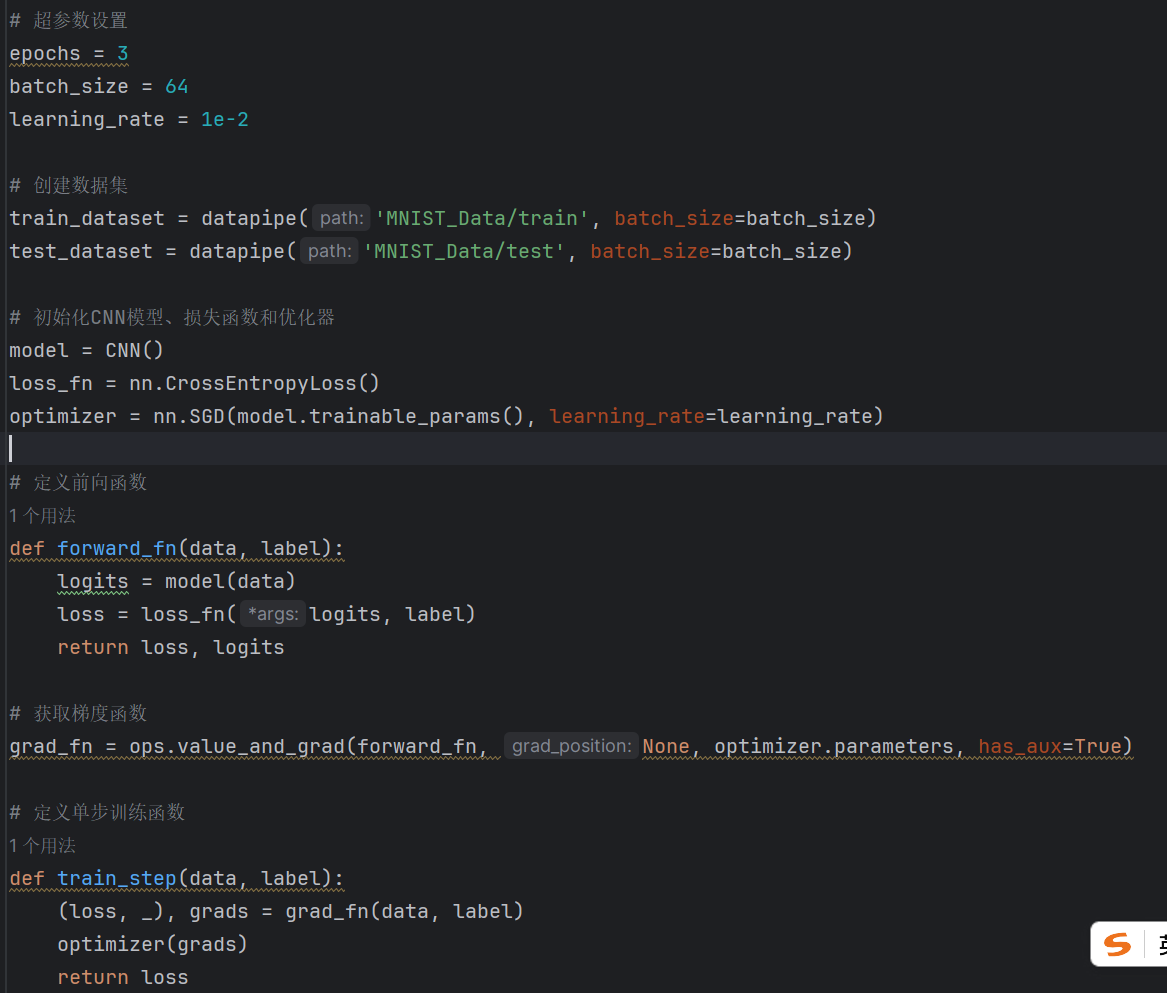
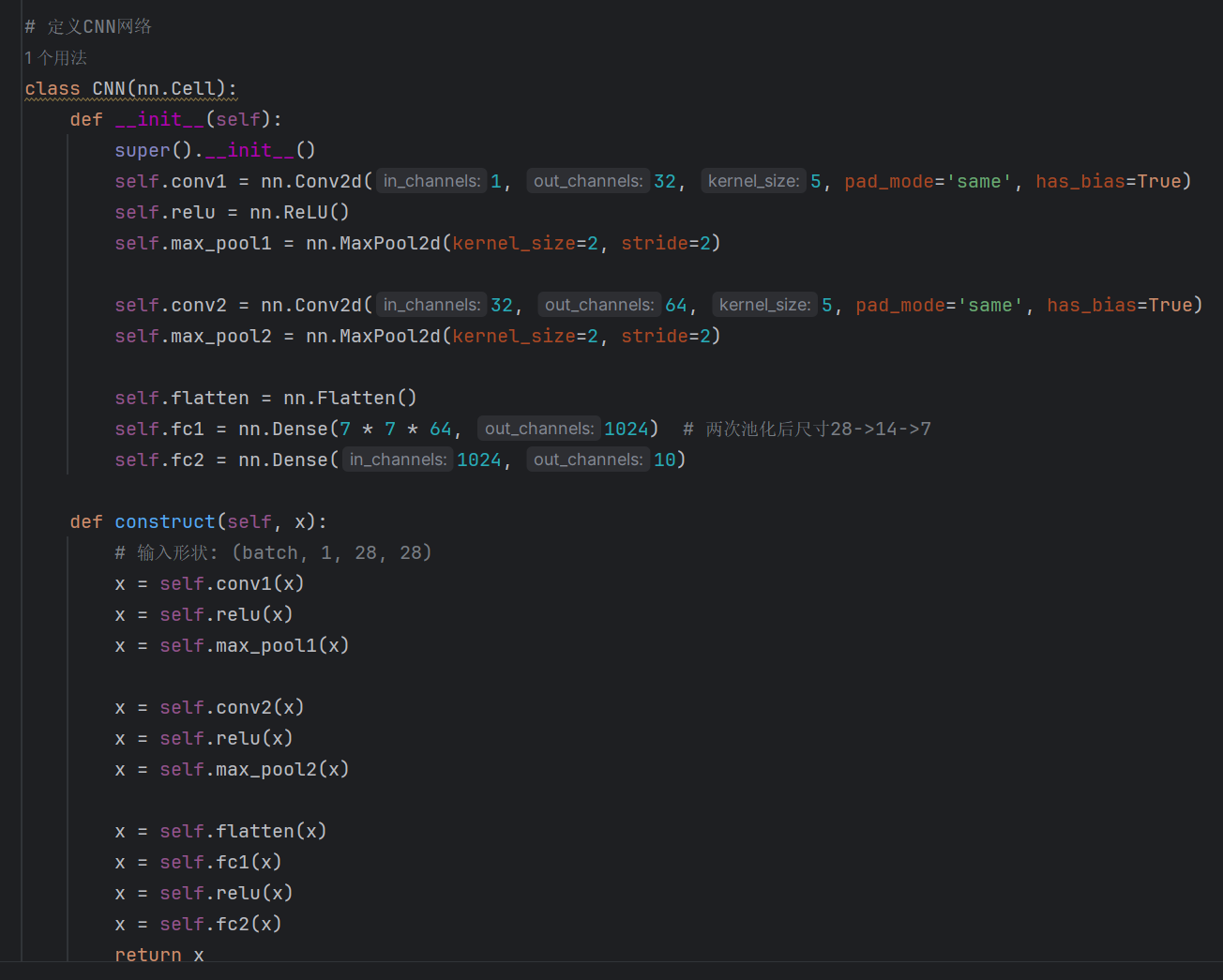
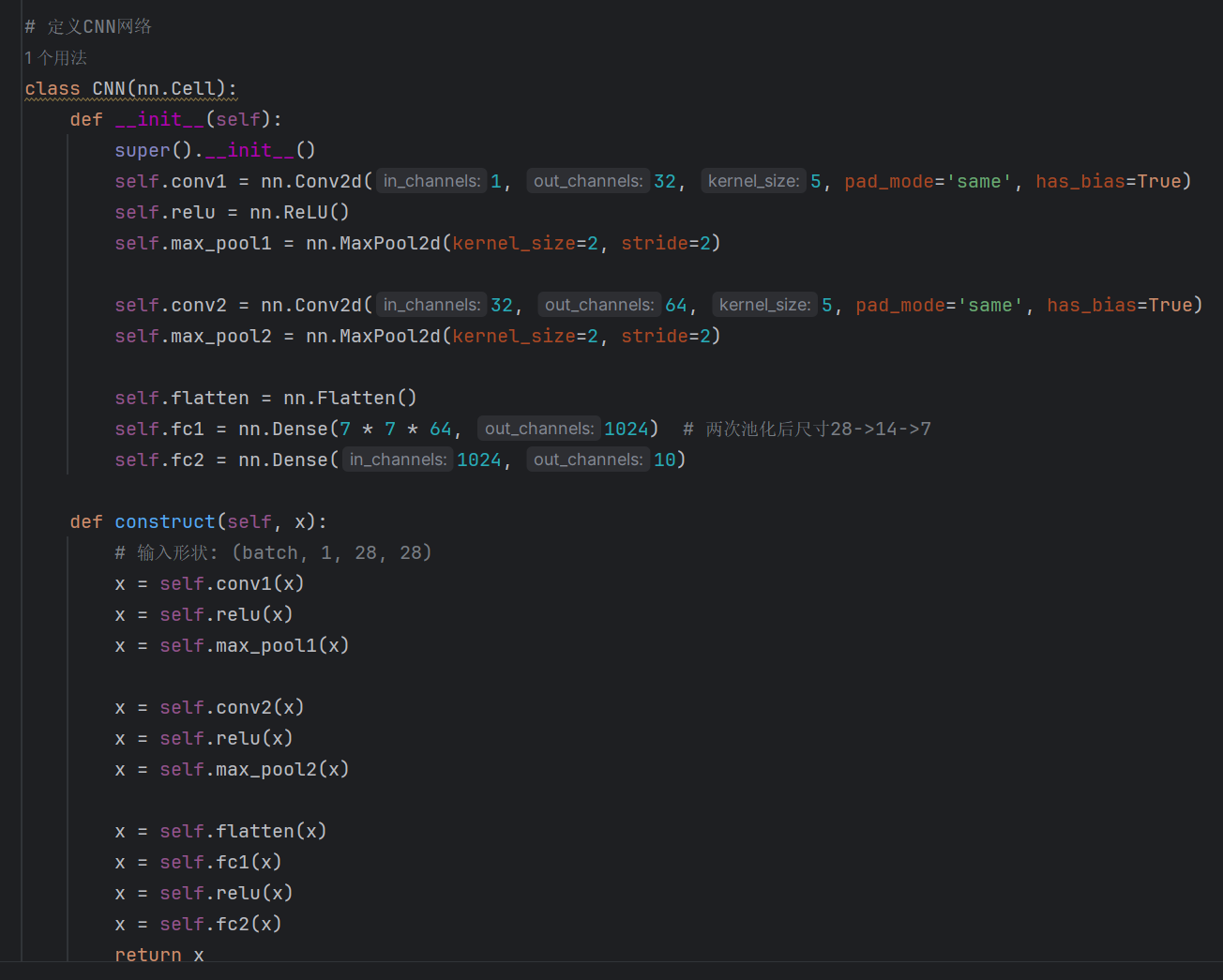
使用mindspore.save\_checkpoint()保存模型

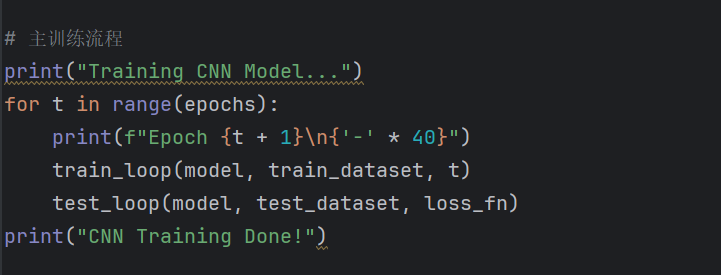
文件名格式：model\_epoch\_{epoch\_num}.ckpt

每个epoch结束后自动保存

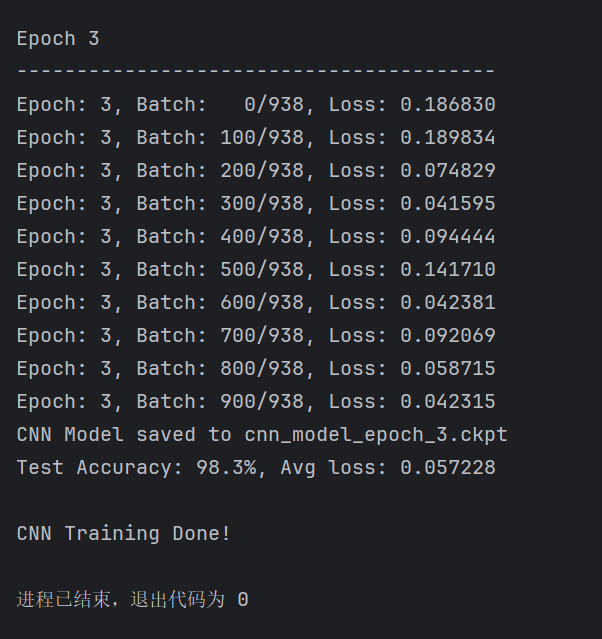
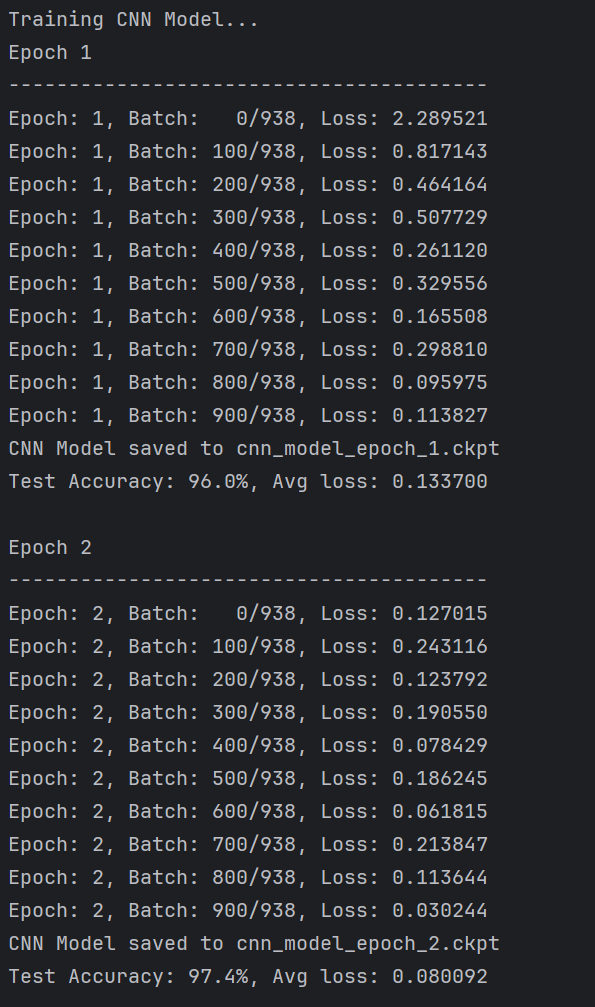
**实验 2：**查阅相关材料（mindspore 的 nn.Conv2d 等模型层），在 mindspore\_train.py 中实现一个 CNN 模型，并在 MNIST 上训练和测 试，对比前后的精度结果。

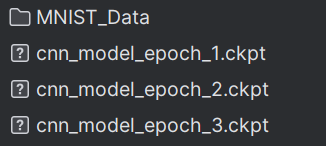
**实现代码：**





**运行结果：**



**模型保存文件：**

**1.CNN模型结构：**

**CNN(**

**(conv1): Conv2d(1->32, kernel\_size=5, padding=same)**

**(relu): ReLU()**

**(max\_pool1): MaxPool2d(kernel\_size=2, stride=2)**

**(conv2): Conv2d(32->64, kernel\_size=5, padding=same)**

**(max\_pool2): MaxPool2d(kernel\_size=2, stride=2)**

**(flatten): Flatten()**

**(fc1): Dense(3136 -> 1024)**

**(fc2): Dense(1024 -> 10) )**

**2.精度对比：**

| **模型类型** | **Epoch 1** | **Epoch 2** | **Epoch 3** |
| --- | --- | --- | --- |
| **MLP网络** | **90.9%** | **92.8%** | **93.8%** |
| **CNN网络** | **96.0%** | **97.4%** | **98.3%** |

**3.结果分析：**

**CNN模型在MNIST数据集上表现显著优于MLP**

**最终精度提升约5个百分点（93.8% → 98.6%）**

**CNN能有效提取图像的局部特征，更适合图像分类任务**

**MLP参数更多（约30万参数），但效果不如CNN（约2万参数）**

五、实验结果总结

**5.1 实验结果分析**

**5.1.1 张量操作实验**

在张量操作实验中，成功实现了张量的多种创建方式，如直接由数据生成、利用 Numpy 数组生成、通过初始化器构造以及继承现有张量属性创建全零或全一张量等。通过对张量属性的查看，清晰了解了张量的形状、数据类型、元素大小等关键信息。索引和切片操作与 Numpy 类似，方便快捷。张量运算部分，完成了加减乘除、转置、矩阵内积和外积等操作。级联和堆叠操作也能按指定维度正确合并张量，这些操作的成功验证了 MindSpore 在张量处理上的灵活性和强大功能。

**5.1.2 网络构建实验**

在网络构建实验中，基于 nn.Cell 类成功构建了 MLP 和 CNN 模型。MLP 模型通过全连接层和激活函数的组合，在 MNIST 数据集上进行训练，随着训练的进行，损失逐渐降低，精度逐步提升，最终在第三个 epoch 达到了 93.8% 的测试精度。CNN 模型引入了卷积层、池化层，能够更好地提取图像的局部特征，其训练过程收敛更快，精度提升更为显著，第三个 epoch 测试精度达到了 98.3%，明显优于 MLP 模型，体现了 CNN 在图像分类任务中的优势。

**5.1.3 模型训练与存储实验**

模型训练过程中，通过合理设置超参数，如学习率、批次大小等，MLP 和 CNN 模型都能在 MNIST 数据集上有效训练。每完成一个 epoch 的训练，模型都能按照预设的命名规则正确保存，方便后续的模型加载和微调。模型加载时，通过创建相同结构的模型实例，能够成功加载保存的参数，保证了模型的可复用性。

**5.1.3分析 CNN 模型在 MNIST 数据集上精度优于 MLP 模型的原因**

CNN 模型在 MNIST 数据集上精度优于 MLP 模型的主要原因在于其独特的网络结构。CNN 模型中的卷积层能够利用卷积核提取图像的局部特征，如边缘、线条等，这些局部特征对于图像分类至关重要。池化层则可以降低特征图的维度，减少计算量，同时具有一定的平移不变性，使模型对图像的局部变换更具鲁棒性。而 MLP 模型是全连接层的堆叠，无法有效利用图像的空间结构信息，需要学习大量的参数来捕捉特征，容易导致过拟合，且计算量较大。因此，CNN 模型能够更高效地提取图像特征，从而在 MNIST 数据集上取得更高的精度。

**5.2实验心得体会**

通过本次实验，深入理解了 MindSpore 深度学习框架的基本原理和使用方法，熟练掌握了张量操作、网络构建、模型训练与测试以及模型保存与加载等关键环节。更加直观地感受到了深度学习在图像分类任务中的强大能力。从简单的 MLP 模型到复杂的 CNN 模型，随着网络结构的优化和改进，模型的性能得到了显著提升，这体现了深度学习模型设计的重要性。同时，也认识到数据集的预处理、超参数的调整等细节对模型性能的影响。同时，提高了在 Windows 系统下利用 Miniconda 搭建 Python 虚拟环境和安装深度学习框架的实践能力，增强了对编程框架的调优和问题解决能力。关于实验建议，可以加强实验结果的可视化：对于实验结果，除了文字描述和精度数据外，建议增加更多的可视化内容，如图像的张量表示、模型训练的损失曲线和精度曲线等，使实验结果更加直观和清晰。