****

人工智能

大作业报告



**学院** **专业**

**任课教师**

**班级**

**学号**

**姓名**

**邮箱**

**手机** **完成日期** 2023-12-21

**成绩**

# 题目名称: 从手写数字识别入门深度学习

目录

[题目名称: 从手写数字识别入门深度学习 1](#_Toc154072278)

[一、介绍 1](#_Toc154072279)

[二、设计思路 1](#_Toc154072280)

[三、程序结果展示 4](#_Toc154072281)

[四、课程收获建议 5](#_Toc154072282)

[五、参考文献 5](#_Toc154072283)

[六、程序代码 5](#_Toc154072284)

一、介绍

在当今数字化的时代，计算机视觉技术的迅猛发展推动了许多令人振奋的应用，其中之一就是手写数字识别。手写数字识别是深度学习领域的经典问题之一，而Modified National Institute of Standards and Technology（MNIST）数据集则成为这一领域的基石。本报告将引导读者通过探索MNIST数据集，了解深度学习入门的关键概念和技术。

MNIST（Modified National Institute of Standards and Technology）是一个手写数字数据集，常用于机器学习和深度学习的图像识别任务。该数据集包含了大量的手写数字图像，每个图像都是28x28像素的灰度图像。这些图像涵盖了0到9的数字，每个数字有大约6000个样本。MNIST通常被用作入门级的图像分类问题，许多深度学习教程和模型评估都以MNIST为基础。研究人员和学生经常使用MNIST来验证新的机器学习算法和深度学习模型的性能。

入门深度学习我们也需要知道深度学习的概念：

深度学习是一种机器学习分支，其核心是构建和训练深层神经网络。在手写数字识别任务中，卷积神经网络（CNN）是一种常用的深度学习模型，特别适合处理图像数据。因为它具有以下特性：

局部连接：视觉具有很强的局部性，相邻数个点之间很可能构成一个完整的物体，距离越远的两个点之间的联系可能越弱，所以用神经网络做图像处理的时候，一般不需要做全连接，应该充分考虑领域信息，对局部做稠密连接即可。

权重共享：CNN使用卷积核（也称为滤波器或者卷积模板）做卷积处理，一张图片中不同的位置可以使用相同的卷积系数（即突触权重）。

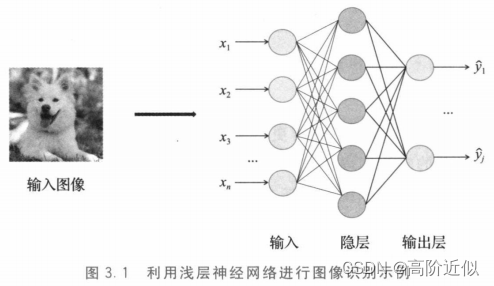


图1.1 CNN示例

本次这个实践中我们采用pytorch框架进行深度学习训练，pytorch框架作为一个深度学习框架有很多优点比如

易于理解和学习： PyTorch的API设计简洁明了，更贴近Python编程语言的风格，使得它相对于其他框架更容易学习和使用。这对于初学者来说是一个重要的优点。

丰富的文档和社区支持： PyTorch拥有丰富的官方文档和一个庞大的活跃社区。你可以轻松找到大量的教程、示例和资源，以帮助解决问题和加速开发。

自动微分： PyTorch提供了自动微分（Autograd）功能，能够自动计算梯度。这使得反向传播和梯度下降等优化算法的实现变得更加简单。

模块化设计： PyTorch的模块化设计使得构建复杂模型变得更容易，你可以方便地组合和重用各种组件。

这些优点让pytorch在深度学习社区中受到广泛欢迎，适用于各种深度学习任务，从学术研究到实际应用都得到了广泛应用。

二、设计思路

1. 准备数据

MNIST数据集是一个包含手写数字的数字图像数据集，包含60000张训练图像和10000张测试图像。数据集的每个图像都是28x28像素的灰度图像，表示一个数字。

要使用PyTorch框架处理MNIST数据集，首先需要下载数据集并将其解压缩到本地目录。然后，可以使用PyTorch提供的torchvision.datasets模块来加载数据集。

2. 构建模型

CNN模型由卷积层、池化层和全连接层组成。卷积层用于提取图像中的局部特征，池化层用于减少特征的维度，全连接层用于对特征进行分类。

MNIST数据集的图像尺寸较小，因此可以使用较小的卷积核和较小的池化窗口。卷积层的输出通道数可以根据需要进行调整。全连接层的输出节点数应等于数字类别的数量，即10。

3. 训练模型

模型训练的过程包括以下步骤：

将数据集分为训练集和测试集。

使用训练集来训练模型。

使用测试集来评估模型的性能。

训练模型可以使用PyTorch提供的torch.optim模块来实现。torch.optim模块提供了各种优化器，可以用于训练神经网络。

在训练模型时，需要设置以下超参数：

学习率：学习率是优化器在每次迭代中更新模型参数的幅度。

批大小：批大小是每次迭代中训练的样本数。

迭代次数：迭代次数是训练模型的次数。

4. 测试模型

测试模型的过程包括以下步骤：

使用测试集来评估模型的性能。

计算模型的准确率、召回率和F1分数。

测试模型可以使用PyTorch提供的torch.utils.data.DataLoader模块来实现。

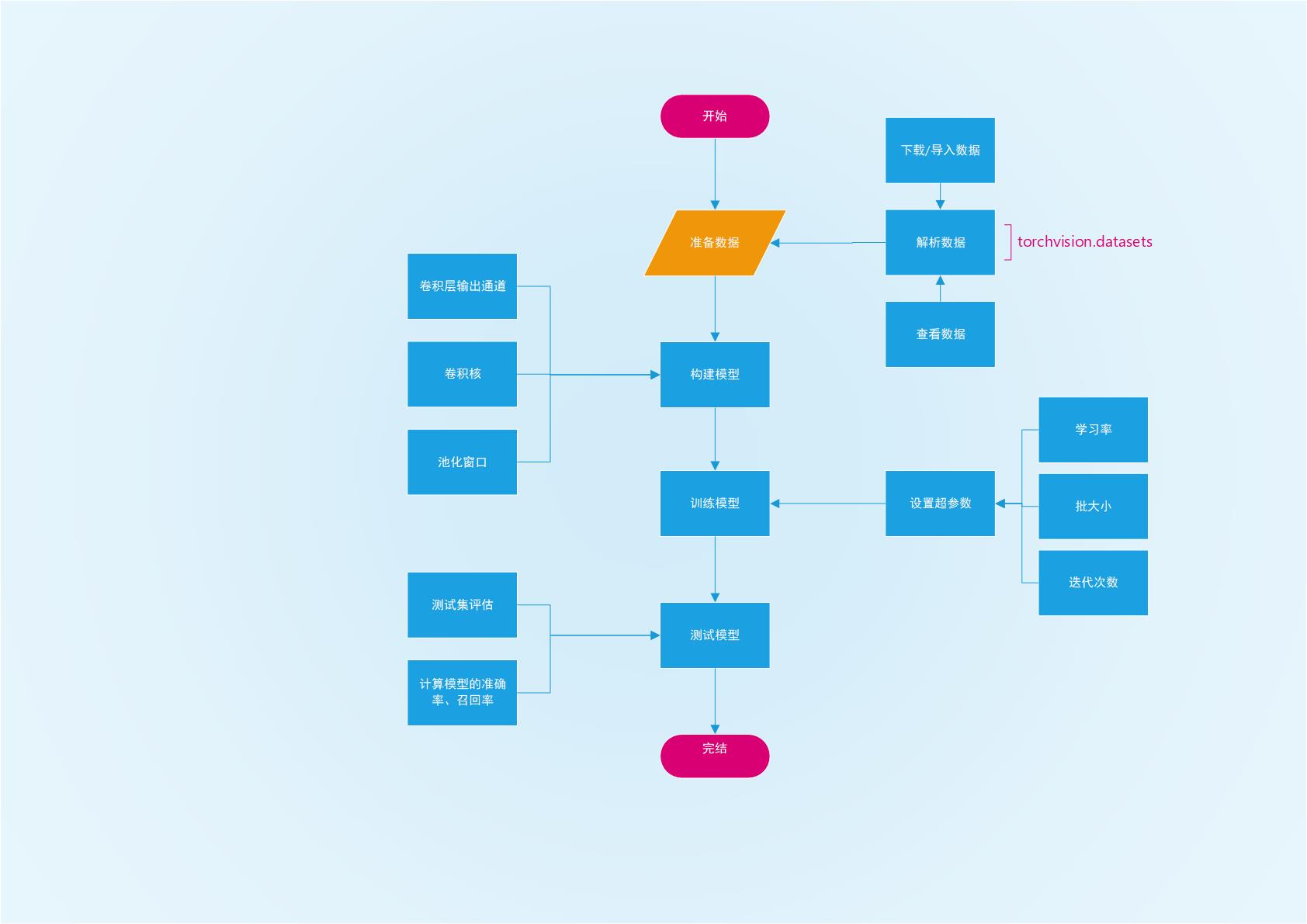


图2.1训练流程图

三、程序结果展示



图3.1查看数据集中数据（前五个）

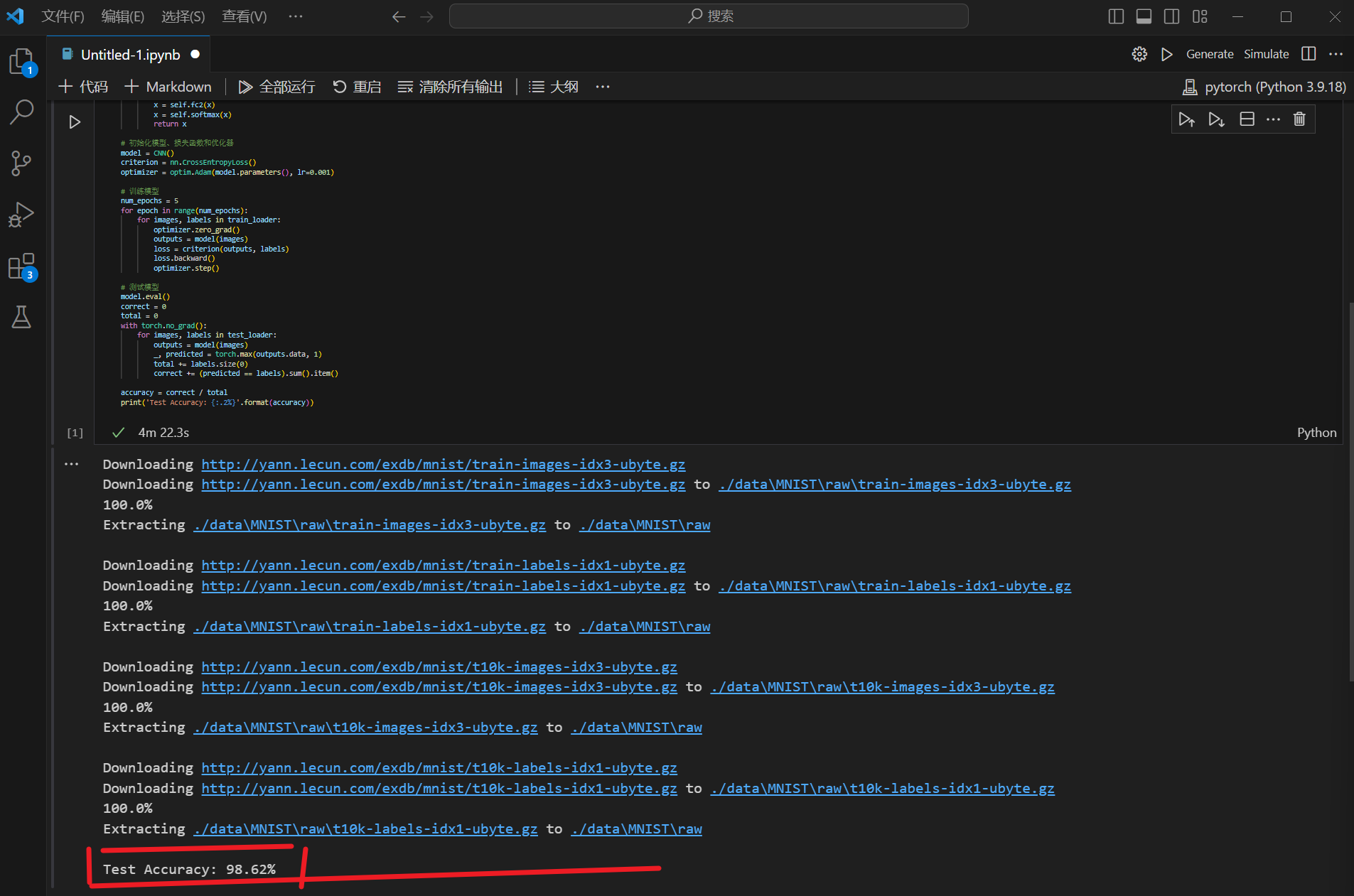


图3.2结果展示

测试集准确率98.62%基本实现了数字识别的准确性。

四、课程收获建议

通过上李老师的人工智能导论，我对人工智能领域有了一定的了解，从最初的机器学习回归，聚类，拟合等等算法，再到后面讲了所有的优化算法离不开 模拟退火 遗传 和粒子群算法，最后到了 后边讲解了深度学习网络等，把人工智能框架路径逐步搭建完成。

有一点对我启发很大，说理论很早就提出来了，但是由于计算机未发展，我们现在才去找十几年前的理论，放到现在模型上发现很实用，这对我们的以后生活和工作都有启发，我们学习追逐新事物新东西新知识的同时，也一定不能忘掉，之前提出的新理论新方法，只是受限于当时环境不能付诸实践的理论，站在巨人的肩膀上，从而看的更远。

五、参考文献

[1] 嵩天. 程序设计基础:Python语言[M]// 程序设计基础:PYTHON语言. 2014.

[2] [从手写数字识别入门深度学习丨MNIST数据集详解-CSDN博客](https://blog.csdn.net/OpenDataLab/article/details/125716623)

[3] [卷积神经网络 (CNN) 的介绍及其网络结构解释\_卷积神经网络cnn结构-CSDN博客](https://blog.csdn.net/weixin_45937291/article/details/130042672)

[4] chatGPT3.5

[5] Bard

六、程序代码

|  |
| --- |
| 查看数据集前五个数据 |
| import matplotlib.pyplot as plt  import numpy as np  # Get the first five samples from the training dataset  for i in range(5):      image, label = train\_dataset[i]        # Convert the image tensor to a NumPy array and unnormalize it      image = image.numpy().squeeze() / 2 + 0.5        # Plot the image      plt.subplot(1, 5, i + 1)      plt.imshow(image, cmap='gray')      plt.title(f'Label: {label}')      plt.axis('off')  plt.show() |

|  |
| --- |
| 使用pytorch直接可以实现下载训练 |
| import torch  import torch.nn as nn  import torch.optim as optim  from torch.utils.data import DataLoader  from torchvision import datasets, transforms  # 定义数据预处理  transform = transforms.Compose([      transforms.ToTensor(),      transforms.Normalize((0.5,), (0.5,))  ])  # 下载MNIST数据集  train\_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)  test\_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)  # 创建数据加载器  train\_loader = DataLoader(dataset=train\_dataset, batch\_size=64, shuffle=True)  test\_loader = DataLoader(dataset=test\_dataset, batch\_size=64, shuffle=False)  # 定义卷积神经网络模型  class CNN(nn.Module):      def \_\_init\_\_(self):          super(CNN, self).\_\_init\_\_()          self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, stride=1, padding=1)          self.relu1 = nn.ReLU()          self.pool1 = nn.MaxPool2d(kernel\_size=2, stride=2)          self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)          self.relu2 = nn.ReLU()          self.pool2 = nn.MaxPool2d(kernel\_size=2, stride=2)          self.flatten = nn.Flatten()          self.fc1 = nn.Linear(64 \* 7 \* 7, 128)          self.relu3 = nn.ReLU()          self.fc2 = nn.Linear(128, 10)          self.softmax = nn.Softmax(dim=1)      def forward(self, x):          x = self.conv1(x)          x = self.relu1(x)          x = self.pool1(x)          x = self.conv2(x)          x = self.relu2(x)          x = self.pool2(x)          x = self.flatten(x)          x = self.fc1(x)          x = self.relu3(x)          x = self.fc2(x)          x = self.softmax(x)          return x  # 初始化模型、损失函数和优化器  model = CNN()  criterion = nn.CrossEntropyLoss()  optimizer = optim.Adam(model.parameters(), lr=0.001)  # 训练模型  num\_epochs = 5  for epoch in range(num\_epochs):      for images, labels in train\_loader:          optimizer.zero\_grad()          outputs = model(images)          loss = criterion(outputs, labels)          loss.backward()          optimizer.step()  # 测试模型  model.eval()  correct = 0  total = 0  with torch.no\_grad():      for images, labels in test\_loader:          outputs = model(images)          \_, predicted = torch.max(outputs.data, 1)          total += labels.size(0)          correct += (predicted == labels).sum().item()  accuracy = correct / total  print('Test Accuracy: {:.2%}'.format(accuracy)) |