




用PyTorch实现 MNIST手写数字识别

组长：吴晨宇

组员：刘淳、刘毕宏、王民辉、房非

 指导老师：赵博

 汇报人：王民辉

目录

CONTENTS



需求分析

1. 识别手写数字 (0-9)
2. 实时识别并反馈结果



总体框架

1. 数据收集
2. 数据预处理
3. 模型选择
4. 模型训练
5. 模型评估
6. GUI应用



功能详细设计

- 1.data
- 2.dataset
- 3.mnist
- 4.pridect
- 5.evaluation
- 6.gui



测试验证

设计GUI界面
在窗口中手写数字
点击“识别”按钮查看结果
点击“清除”按钮清空画布



总结

01 需求分析



需求分析



1.基于PyTorch深度学习，实现手写数字识别
用MNIST数据集训练出一个CNN（卷积神经网络）模型，
让这个模型能够对手写数字图片（0-9）进行分类。

功能实现：数据预处理、模型训练、模型评估、结果预测。

2.创建GUI界面，用画布手写数字，实现实时的识别，反馈结果。

- 在窗口中手写数字，然后点击“识别”按钮查看结果。
- 点击“清除”按钮可以清空画布。

功能实现：GUI应用

MNIST数据集



MNIST 数据集：包含 70,000 张手写数字图像（60,000 张训练集，10,000 张测试集），每张图像大小为 28x28 像素，灰度图。

MNIST中所有样本都会将原本28*28的灰度图转换为长度为784的一维向量作为输入，其中每个元素分别对应了灰度图中的灰度值。MNIST使用一个长度为10的one-hot向量作为该样本所对应的标签，其中向量索引值对应了该样本以该索引为结果的预测概率。



CNN模型

卷积神经网络（CNN），包含卷积层，激活函数和池化层。

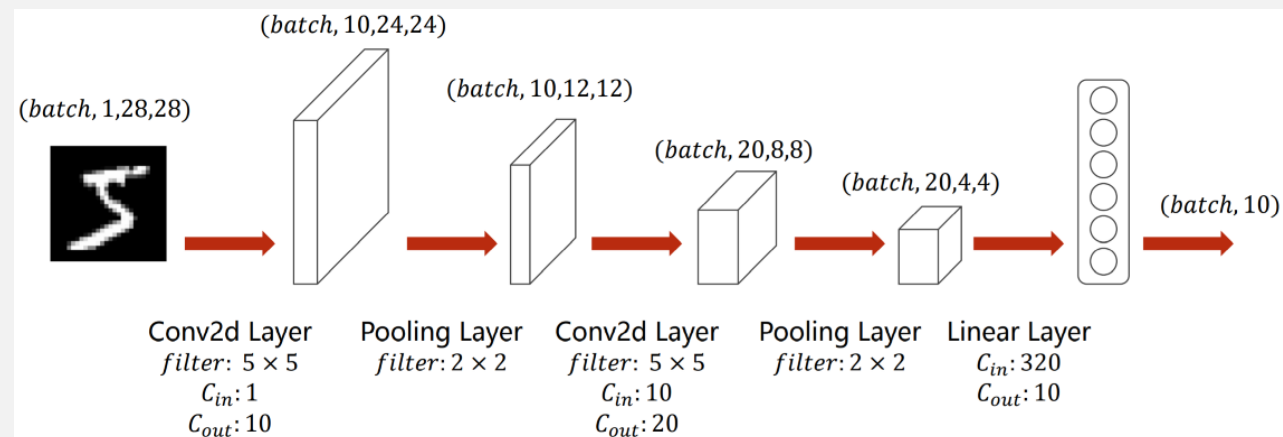
卷积层：用于提取图像特征。

激活函数：使用 ReLU 作为激活函数，引入非线性。

池化层：最大池化层，减少特征图尺寸，降低计算复杂度。

展平层：将多维特征展平为一维向量。

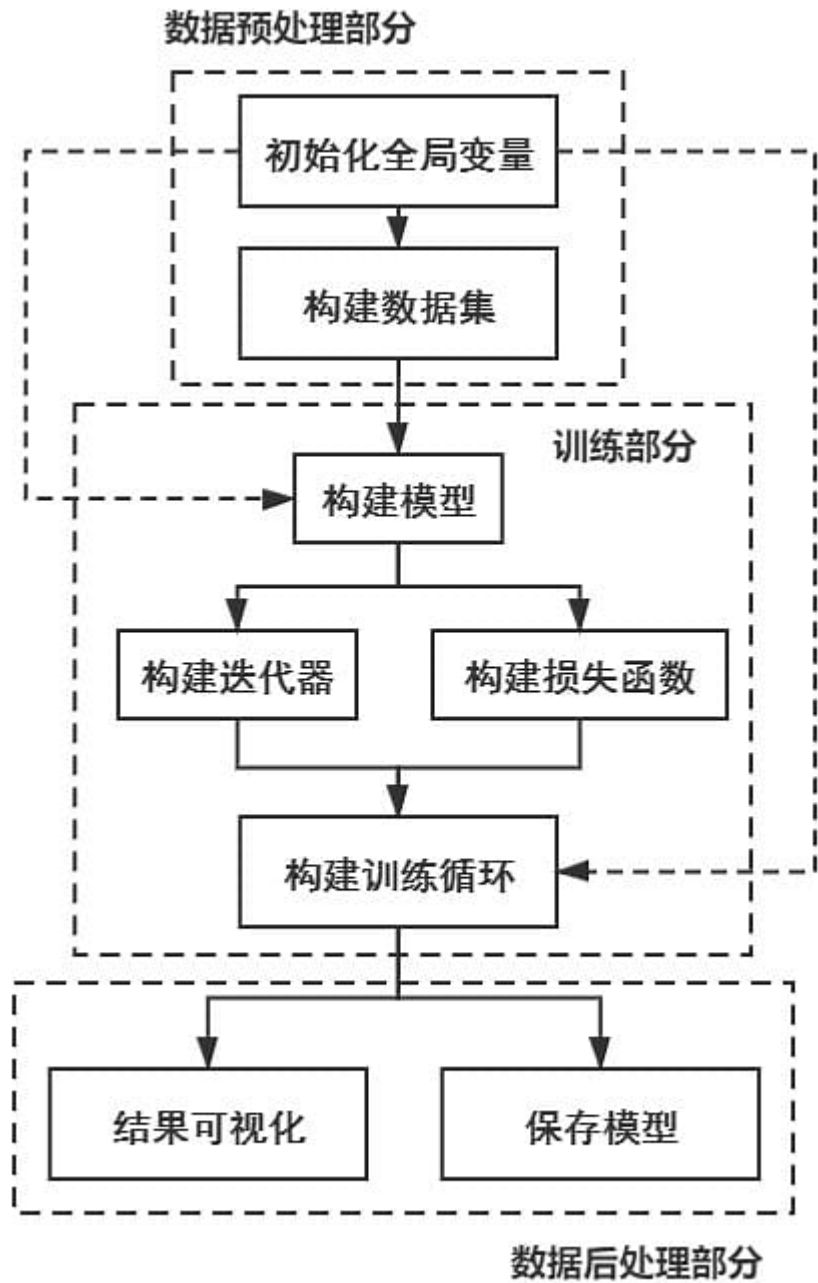
全连接层：输出 10 个类，分别对应数字 0-9。



02 总体框架



总体框架



训练过程

损失函数：交叉熵损失（Cross-Entropy Loss）。

优化器：Adam。

训练轮数：设置为 10 轮。

评估与可视化

准确率：在测试集上评估模型的分类准确率。

可视化：使用 Matplotlib 绘制训练过程中的损失和准确率曲线。

总体框架



1. 选择MNIST 数据集

train: 60,000 张图像, test: 10,000 张图像。

2. 数据预处理 (`dataset.py`)

加载数据: 使用库 (`PyTorch`) 加载数据集。

归一化处理: 将图像像素值缩放到 $[0, 1]$ 范围。

图像调整: 将图像调整为统一尺寸 (`28x28` 像素)。

3. 构建CNN模型

卷积层: 提取特征

池化层: 降低特征维度

全连接层: 进行分类

4. 构建迭代训练模型, 保存模型 (`model.pth`)。

5. 评估模型准确度(`evaluation.py`), 对手写结果进行预测 (`predict.py`)

6. GUI应用(`gui.py`)。












总体框架



handwritten_digit_recognition/

- |
- |— .idea/ # IDE 配置文件（可选）
- |— datasets/ # 存放数据集的文件夹
 - |— dataset.py # 数据集加载和处理脚本
- |— data/ # 数据存储目录（可选）
- |— evaluation.py # 模型评估脚本
- |— gui.py # 手写数字识别的GUI界面
- |— mnist.py # 模型训练和测试
- |— model.pth # 训练好的模型文件
- |— predict.py # 预测单张图片的脚本
- |— test_accuracy.png # 测试精度可视化图表
- |— test_loss.png # 测试损失可视化图表

名称

-  .idea
-  data
-  datasets
-  dataset.py
-  evaluation.py
-  gui.py
-  mnist.py
-  model.pth
-  predict.py
-  test_accuracy.png
-  test_loss.png

总体框架



开发环境: Python 3.9

深度学习框架: PyTorch

其他库:

NumPy: 用于数据处理。


Matplotlib: 用于可视化。


torchvision: 用于数据集加载和预处理。

Tkinter: 适合构建基本的 GUI 应用。


tqdm: 进度条的形式显示循环的进度。


名称


 .idea


 data


 datasets


 dataset.py


 evaluation.py


 gui.py

 mnist.py

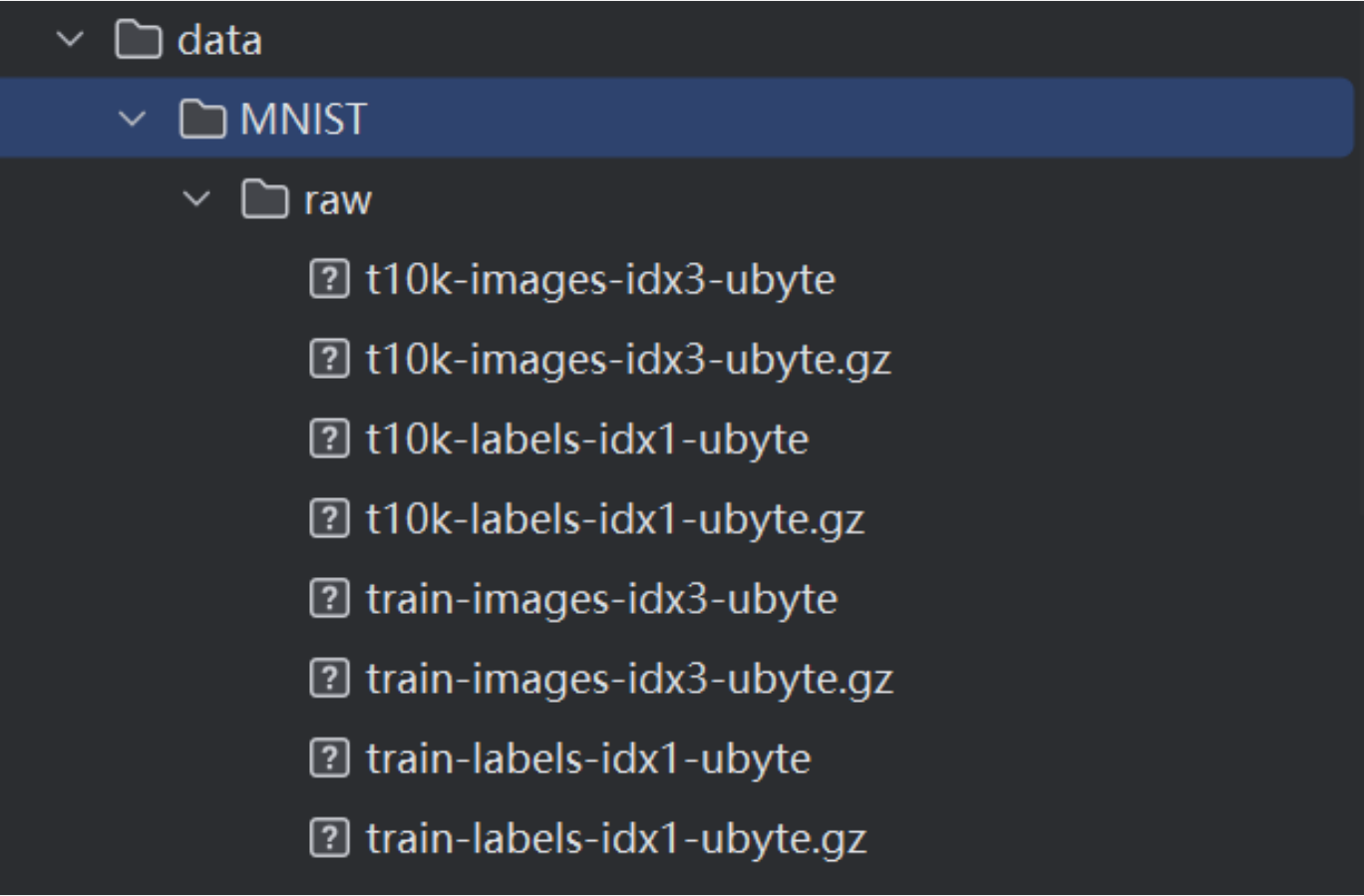
 model.pth

 predict.py

 test_accuracy.png

 test_loss.png

03 功能详细设计



训练集、训练集标签、测试集、测试集标签

文件名称	
train-images-idx3-ubyte.gz	55000张训练集、验证集
train-labels-idx1-ubyte.gz	训练集图片对应的标签
t10k-images-idx3-ubyte.gz	10000张测试集
t10k-labels-idx1-ubyte.gz	测试集图片对应的标签

功能详细设计

▼ datasets

数据预处理:

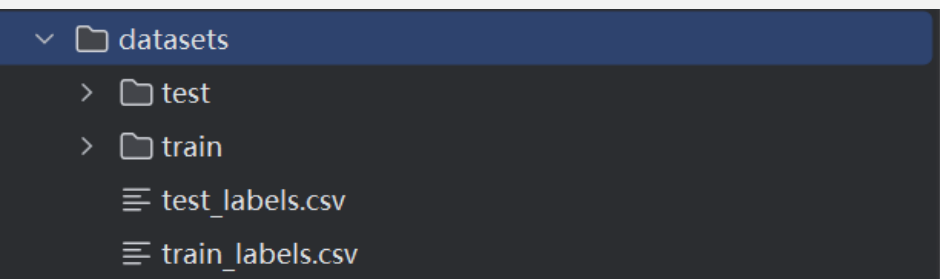
使用 `transforms.Compose` 将 Torch 张量转换为 PIL 图像。

数据集加载:

使用 `torchvision.datasets.MNIST` 加载训练和测试数据集, 并将其转换为张量。

保存数据集到磁盘:

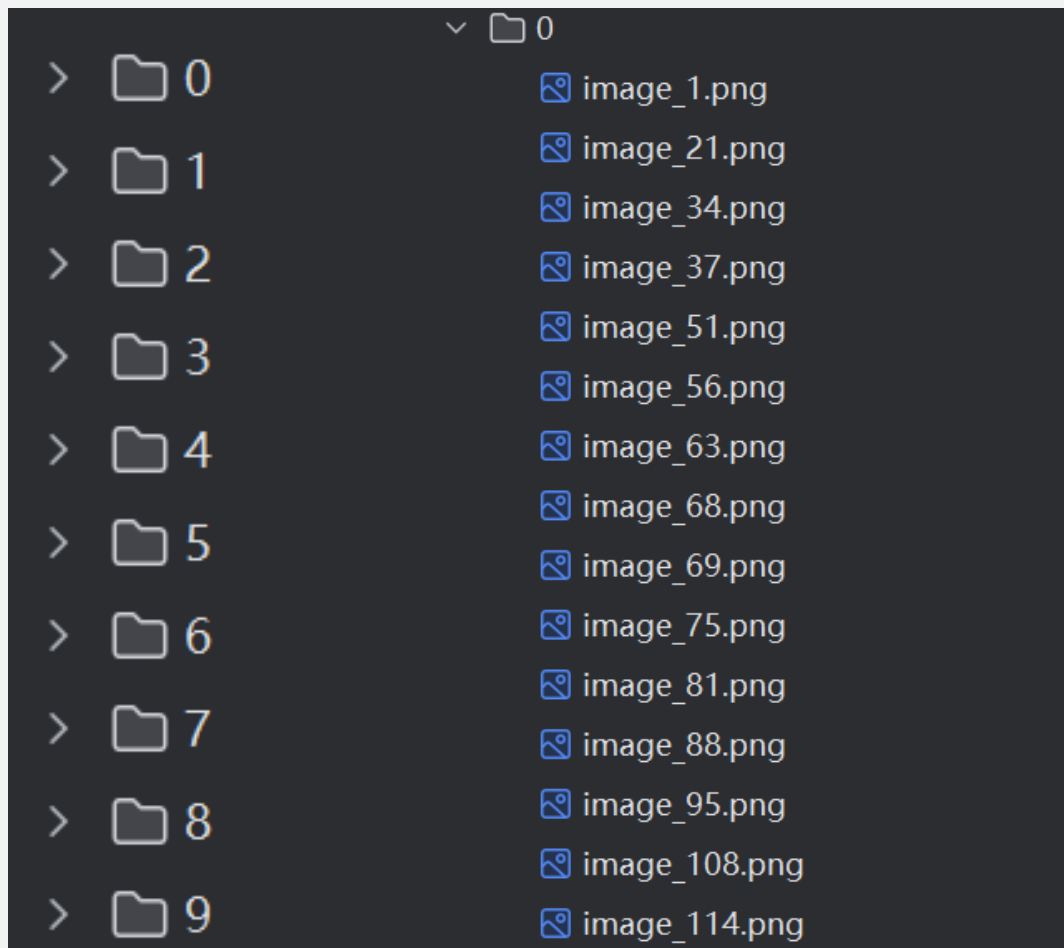
在 `save_dataset_to_disk` 函数中, 创建文件夹结构并保存图像文件到指定路径, 同时生成与图像对应的标签 CSV 文件。



```
1 > import ...
2
3 # 数据预处理
4 transform = transforms.Compose([
5     transforms.ToPILImage(), # 将 Tensor 转为 PIL 图像
6 ])
7
8 # 数据集加载
9 trainData = torchvision.datasets.MNIST('./data/', train=True, transform=torchvision.transforms.ToTensor(), download=True)
10 testData = torchvision.datasets.MNIST('./data/', train=False, transform=torchvision.transforms.ToTensor())
11
12 # 保存图片及标签到文件夹和 CSV 文件的函数
13 def save_dataset_to_disk(dataset, root_folder, sub_folder, csv_file_name): # 2用法
14     """
15     将 MNIST 数据集保存为图片文件, 并生成对应的标签 CSV 文件。
16     """
17     :param dataset: MNIST 数据集
18     :param root_folder: 根文件夹名称
19     :param sub_folder: 子文件夹名称 (train 或 test)
20     :param csv_file_name: 保存标签的 CSV 文件名
21     """
22     # 创建根文件夹
23     dataset_folder = os.path.join(root_folder, sub_folder)
24     os.makedirs(dataset_folder, exist_ok=True)
25
26     # 打开 CSV 文件进行写入
27     csv_path = os.path.join(root_folder, csv_file_name)
28     with open(csv_path, mode='w', newline='', encoding='utf-8') as csv_file:
29         writer = csv.writer(csv_file)
30         writer.writerow(['ImagePath', 'Label']) # 写入表头
31
32     # 遍历数据集
33     for idx, (img, label) in enumerate(dataset):
34         # 定义文件路径
35         img_folder = os.path.join(dataset_folder, str(label)) # 按标签分类到子文件夹
36         os.makedirs(img_folder, exist_ok=True)
37         img_path = os.path.join(img_folder, f'image_{idx}.png')
```

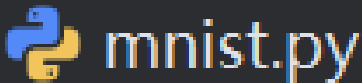
```
44 # 保存图片
45 img = transform(img) # 转换为 PIL 图像
46 img.save(img_path)
47
48 # 写入标签到 CSV 文件
49 writer.writerow([img_path, label])
50
51 # 可选: 打印进度
52 if idx % 1000 == 0:
53     print(f"Saved {idx}/{len(dataset)} images to {sub_folder}")
54
55 # 定义根文件夹
56 root_folder = './datasets'
57
58 # 保存训练数据集
59 save_dataset_to_disk(trainData, root_folder, sub_folder: 'train', csv_file_name: 'train_labels.csv')
60
61 # 保存测试数据集
62 save_dataset_to_disk(testData, root_folder, sub_folder: 'test', csv_file_name: 'test_labels.csv')
63
64 print("MNIST 数据集保存完成!")
```

功能详细设计



预处理后储存的图像文件（test和train）
包含数字0-9的图片

功能详细设计



（1）模型结构：

定义了一个卷积神经网络 **Net**，包含多层卷积和全连接层。

（2）数据预处理：

使用 **torchvision.transforms** 对 MNIST 数据集进行转换，包括转换为张量和归一化。

（3）训练与测试过程：

利用 **tqdm** 显示训练过程进度，计算并打印损失和准确率。

在每个 **epoch** 结束时，对测试集进行评估，计算平均损失和准确率。

（4）可视化：

使用 **matplotlib** 绘制测试损失和准确率随 **epoch** 变化的曲线，并保存为图像文件。

（5）模型保存：

将训练好的模型参数保存到文件 **model.pth** 中。

```
1 import sys
2
3 # 定义模型
4 class Net(torch.nn.Module):  # 网络
5     def __init__(self):
6         super(Net, self).__init__()
7         self.model = torch.nn.Sequential(
8             # 第一层卷积，输入通道数为1（灰度图），输出通道数为16，卷积核大小为3x3，步幅为1，边缘补充3像素
9             torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1),
10             torch.nn.ReLU(), # 激活函数 ReLU
11             torch.nn.MaxPool2d(kernel_size=2, stride=2), # 最大池化，池化窗口大小为2x2，步幅为2
12
13             # 第二层卷积，输入通道数为16，输出通道数为32，其他参数同上
14             torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
15             torch.nn.ReLU(),
16             torch.nn.MaxPool2d(kernel_size=2, stride=2),
17
18             # 第三层卷积，输入通道数为32，输出通道数为64，其他参数同上
19             torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
20             torch.nn.ReLU(),
21
22             # 展平，将多维特征图变为一维向量
23             torch.nn.Flatten(),
24             # 全连接层，输入特征数为7*7*64，输出特征数为128
25             torch.nn.Linear(in_features=7 * 7 * 64, out_features=128),
26             torch.nn.ReLU(),
27             # 最后一层全连接，输入特征数为128，输出为10（对应10个类别）
28             torch.nn.Linear(in_features=128, out_features=10)
29         )
30
31     def forward(self, input):
32         output = self.model(input) # 返回结果
33         return output
34
35 # 检查设备是否支持 GPU，如果支持则使用 CUDA，否则使用 CPU
36 device = "cuda:0" if torch.cuda.is_available() else "cpu"
37
38 # 定义数据预处理方法
39 transform = torchvision.transforms.Compose([
40     torchvision.transforms.ToTensor(), # 将图像转换为 Tensor 格式
41     torchvision.transforms.Normalize(mean=[0.5], std=[0.5]) # 对数据进行归一化，均值为0.5，标准差为0.5
42 ])
43
44 # 数据集与数据加载器
45 BATCH_SIZE = 256 # 每批次的数据量
46 EPOCHS = 10 # 训练轮数
47 trainData = torchvision.datasets.MNIST('./data/', train=True, transform=transform, download=True) # 加载训练数据集
48 testData = torchvision.datasets.MNIST('./data/', train=False, transform=transform) # 加载测试数据集
49 trainDataLoader = torch.utils.data.DataLoader(dataset=trainData, batch_size=BATCH_SIZE, shuffle=True) # 训练数据加载器，打乱数据
50 testDataLoader = torch.utils.data.DataLoader(dataset=testData, batch_size=BATCH_SIZE) # 测试数据加载器
51
52 # 初始化模型
53 net = Net().to(device) # 将模型加载到指定设备上
54 print(net) # 打印模型结构
55
56 # 定义损失函数和优化器
57 lossF = torch.nn.CrossEntropyLoss() # 使用交叉熵损失函数
58 learning_rate = 1e-3 # 学习率
59 optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate) # 优化器使用 Adam
60
61 # 训练与测试
62 history = {'Test Loss': [], 'Test Accuracy': []} # 保存每轮测试损失和准确率
63 for epoch in range(1, EPOCHS + 1):
64     processBar = tqdm(trainDataLoader, unit='step') # tqdm 进度条，用于显示训练进度
65     net.train(True) # 启用训练模式
66     for step, (train_imgs, labels) in enumerate(processBar):
67         train_imgs, labels = train_imgs.to(device), labels.to(device) # 将数据和标签加载到设备上
68
69         # 返回结果
70         outputs = net(train_imgs) # 获取模型输出
71         loss = lossF(outputs, labels) # 计算损失
72         predictions = torch.argmax(outputs, dim=1) # 获取预测值
73         accuracy = torch.sum(predictions == labels) / labels.shape[0] # 计算准确率
74
75     # 反向传播
76     optimizer.zero_grad() # 梯度清零
77     loss.backward() # 反向传播
78     optimizer.step() # 更新权重
79
80     processBar.set_description("[{}/{}] Loss: {:.4f}, Acc: {:.4f} %".format(epoch, EPOCHS, loss.item(), accuracy.item())) # 更新进度条信息
81
82 # 测试
83 correct, totalLoss = 0, 0 # 初始化测试数据
84 net.train(False) # 关闭训练模式，启用评估模式
85 with torch.no_grad(): # 不计算梯度
86     for test_imgs, labels in testDataLoader:
87         test_imgs, labels = test_imgs.to(device), labels.to(device) # 将测试数据加载到设备上
88         outputs = net(test_imgs) # 获取模型输出
89         loss = lossF(outputs, labels) # 计算损失
90         predictions = torch.argmax(outputs, dim=1) # 获取预测值
91
92         totalLoss += loss # 累积测试损失
93         correct += torch.sum(predictions == labels) # 累积正确预测数
94
95     testAccuracy = correct / len(testData) # 测试准确率，使用测试数据总数计算
96     testLoss = totalLoss / len(testDataLoader) # 平均测试损失
97     history['Test Loss'].append(testLoss.item()) # 保存测试损失
98     history['Test Accuracy'].append(testAccuracy.item()) # 保存测试准确率
99
100 print("[{}/{}] Test Loss: {:.4f}, Test Accuracy: {:.4f} %".format(epoch, EPOCHS, testLoss.item(), testAccuracy.item())) # 打印测试结果
101
102 # 保存模型
103 torch.save(net.state_dict(), './model.pth') # 保存模型参数
```

```
107 # 可视化
108 plt.plot(*args: history['Test Loss'], label='Test Loss') # 绘制测试损失曲线
109 plt.title('Test Loss vs Epoch') # 添加标题
110 plt.legend(loc='best') # 添加图例
111 plt.grid(True) # 添加网格
112 plt.xlabel('Epoch') # 横轴标签
113 plt.ylabel('Loss') # 纵轴标签
114 plt.savefig('./test_loss.png') # 保存图表为文件
115 plt.show() # 显示图表
116
117 plt.plot(*args: history['Test Accuracy'], color='red', label='Test Accuracy') # 绘制测试准确率曲线
118 plt.title('Test Accuracy vs Epoch') # 添加标题
119 plt.legend(loc='best') # 添加图例
120 plt.grid(True) # 添加网格
121 plt.xlabel('Epoch') # 横轴标签
122 plt.ylabel('Accuracy') # 纵轴标签
123 plt.savefig('./test_accuracy.png') # 保存图表为文件
124 plt.show() # 显示图表
125
126 # 保存模型
127 torch.save(net.state_dict(), './model.pth') # 保存模型参数
```


模型的设计思路

- （1）卷积层与池化层的组合：使用多个卷积层和池化层，目的是逐步提取图像中的特征，减少输入数据的维度并保留重要信息。
- （2）非线性激活：通过ReLU激活函数引入非线性，使得模型能够学习到复杂的特征。
- （3）全连接层：将卷积提取到的特征映射到最终的类别输出，适合处理多分类问题（如 MNIST 的 10 类数字）。

（1）卷积层：

in_channels: 输入特征图的通道数（对于 MNIST 为 1）。

out_channels: 输出特征图的通道数，每个卷积层提取的特征数量。

kernel_size: 卷积核的大小（通常为 3x3）。

stride: 卷积步幅，控制卷积核移动的速度（默认值为 1）。

padding: 在输入边缘填充0，以控制输出特征图的大小。

（2）激活函数：

使用 ReLU（Rectified Linear Unit）激活函数，引入非线性特征，帮助模型捕捉复杂模式。

（3）池化层：

`torch.nn.MaxPool2d(kernel_size, stride)`

下采样: 降低特征图的空间维度

最大池化: 选择池化窗口内的最大值

常用设置: 2x2 池化窗口，步幅为2

（4）展平层：

将多维特征图展平为一维向量，以便于传递给全连接层。

（5）全连接层：

将提取的特征进行分类，最终输出 10 个类别（对应数字 0-9）。

功能详细设计



预测文件

模型结构：

定义了一个卷积神经网络 **Net** 用于手写数字识别，具有多个卷积层和全连接层。

预测函数：

predict_image 函数用于加载单张图片，并使用训练好的模型进行预测。

包含了对输入图片和模型路径的检查，确保在执行预测时文件存在。

数据预处理：

通过 **transforms.Compose** 进行数据预处理，确保输入图像符合模型要求的格式。

```
1 import os
2
3 # 定义模型结构
4 class Net(torch.nn.Module): 2 用法
5     def __init__(self):
6         super(Net, self).__init__()
7         self.model = torch.nn.Sequential(
8             torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1),
9             torch.nn.ReLU(),
10            torch.nn.MaxPool2d(kernel_size=2, stride=2),
11
12            torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
13            torch.nn.ReLU(),
14            torch.nn.MaxPool2d(kernel_size=2, stride=2),
15
16            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
17            torch.nn.ReLU(),
18
19            torch.nn.Flatten(),
20            torch.nn.Linear(in_features=7 * 7 * 64, out_features=128),
21            torch.nn.ReLU(),
22            torch.nn.Linear(in_features=128, out_features=10) # 输出为 10 个类别
23        )
24
25    def forward(self, input):
26        output = self.model(input)
27        return output
28
29 # 定义预测函数
30 def predict_image(image_path, model_path): 1 个用法
31     """
32     对单张图片进行预测。
33
34     :param image_path: 图片的路径
35     :param model_path: 训练好的模型文件的路径
36     :return: 预测的标签
37     """
38     # 检查文件路径是否存在
39     if not os.path.exists(image_path):
40         raise FileNotFoundError(f"图片文件 {image_path} 不存在!")
41     if not os.path.exists(model_path):
42         raise FileNotFoundError(f"模型文件 {model_path} 不存在!")
43
44     # 加载训练好的模型
45     model = Net() # 使用与训练时相同的模型结构
46     model.load_state_dict(torch.load(model_path))
47     model.eval() # 设置为评估模式
48
49     # 数据预处理
50     transform = transforms.Compose([
51         transforms.Grayscale(num_output_channels=1), # 确保图片为单通道灰度图
52         transforms.Resize((28, 28)), # 调整为 MNIST 输入大小
53         transforms.ToTensor(),
54         transforms.Normalize(mean=[0.5], std=[0.5]) # 与训练时的标准化一致
55     ])
56
57     # 加载图片并应用预处理
58     img = Image.open(image_path)
59     img = transform(img)
60     img = img.unsqueeze(0) # 增加一个批次维度，形状变为 [1, 1, 28, 28]
61
62     # 预测
63     with torch.no_grad():
64         outputs = model(img)
65         predicted_label = torch.argmax(outputs, dim=1).item() # 获取预测的标签
66
67     return predicted_label
```

72

示例用法

73 ▶

if __name__ == "__main__":

74

输入图片路径和模型路径

75

image_path = 'datasets\\test\\8\\image_177.png'

76

model_path = 'model.pth'

77

78

try:

79

调用预测函数

80

result = predict_image(image_path, model_path)

81

print(f"预测结果: {result}")

82


except Exception as e:

83



print(f"发生错误: {e}")

功能详细设计

 evaluation.py

模型评估脚本，使用 Net 类定义了 CNN 模型。

evaluate_model 函数负责加载模型，评估其在测试集上的表现，并计算各项指标（准确率、精确率、召回率和 F1 分数）。

通过 torch.no_grad() 来禁用梯度计算，以提高评估效率。

主程序：

设置设备（CPU 或 GPU）。

加载 MNIST 测试集。

调用评估函数并打印结果。

```
1 # 定义模型结构
2
3 # 定义模型结构
4 class Net(torch.nn.Module): 2 用法
5     def __init__(self):
6         super(Net, self).__init__()
7         self.model = torch.nn.Sequential(
8             torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1),
9             torch.nn.ReLU(),
10            torch.nn.MaxPool2d(kernel_size=2, stride=2),
11            torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
12            torch.nn.ReLU(),
13            torch.nn.MaxPool2d(kernel_size=2, stride=2),
14            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
15            torch.nn.ReLU(),
16            torch.nn.Flatten(),
17            torch.nn.Linear(in_features=7 * 7 * 64, out_features=128),
18            torch.nn.ReLU(),
19            torch.nn.Linear(in_features=128, out_features=10) # 输出为 10 个类别
20        )
21
22    def forward(self, input):
23        output = self.model(input)
24        return output
25
26
27 # 定义评估函数
28 def evaluate_model(model_path, test_loader, device): 1 用法
29     """
30     评估模型的性能。计算 Accuracy, Precision, Recall 和 F1 Score.
31
32     :param model_path: 训练好的模型文件路径
33     :param test_loader: 测试数据集的 DataLoader
34     :param device: 设备 'cpu' 或 'cuda'
35     """
36
37     # 检查模型路径是否存在
38     if not os.path.exists(model_path):
39         raise FileNotFoundError(f"模型文件 {model_path} 不存在！")
40
41     # 加载模型
42     model = Net().to(device) # 将模型移动到设备上
43     model.load_state_dict(torch.load(model_path, map_location=device)) # 加载模型时指定设备
44     model.eval() # 设置为评估模式
45
46     # 存储预测值和真实值
47     y_true = []
48     y_pred = []
49
50     with torch.no_grad():
51         for images, labels in test_loader:
52             images, labels = images.to(device), labels.to(device) # 将数据移动到设备上
53             outputs = model(images)
54             predictions = torch.argmax(outputs, dim=1) # 获取预测结果
55
56             y_true.extend(labels.cpu().numpy())
57             y_pred.extend(predictions.cpu().numpy())
58
59     # 计算指标
60     accuracy = accuracy_score(y_true, y_pred)
61     #precision = precision_score(y_true, y_pred, average='weighted')
62     #recall = recall_score(y_true, y_pred, average='weighted')
63     #f1 = f1_score(y_true, y_pred, average='weighted')
64
65     print(f"Accuracy: {accuracy:.4f}")
66     print(f"Precision: {precision:.4f}")
67     print(f"Recall: {recall:.4f}")
68     print(f"F1 Score: {f1:.4f}")
69
70     return accuracy, precision, recall, f1
```

```
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
# 主程序
if __name__ == "__main__":
    # 设备设置
    device = "cuda:0" if torch.cuda.is_available() else "cpu"

    # 数据预处理
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5], std=[0.5]) # 与训练时一致
    ])

    # 加载测试集
    test_dataset = datasets.MNIST('./data/', train=False, transform=transform)
    test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=256)

    # 模型路径
    model_path = 'model.pth'

    try:
        # 调用评估函数
        evaluate_model(model_path, test_loader, device)
    except Exception as e:
        print(f"发生错误: {e}")
```

功能详细设计



```
1 > import ...
7
8
9 # 定义模型结构，与训练时一致
10 class Net(nn.Module): 2用法
11     def __init__(self):
12         super(Net, self).__init__()
13         self.model = nn.Sequential(
14             nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1),
15             nn.ReLU(),
16             nn.MaxPool2d(kernel_size=2, stride=2),
17             nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
18             nn.ReLU(),
19             nn.MaxPool2d(kernel_size=2, stride=2),
20             nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
21             nn.ReLU(),
22             nn.Flatten(),
23             nn.Linear(in_features=7 * 7 * 64, out_features=128),
24             nn.ReLU(),
25             nn.Linear(in_features=128, out_features=10)
26         )
27
28     def forward(self, input):
29         return self.model(input)
30
31
32 # 加载训练好的模型
33 class HandwrittenDigitRecognizer: 1个用法
34     def __init__(self, model_path):
35         self.model = Net() # 使用与训练时相同的模型
36         self.model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu'))) # 加载模型
37         self.model.eval() # 切换到评估模式
38         self.device = "cuda:0" if torch.cuda.is_available() else "cpu"
39         self.model.to(self.device)
40
41     def predict(self, image): 1个用法
42         transform = transforms.Compose([
43             transforms.Resize((28, 28)), # 将图像调整为28x28
44             transforms.Grayscale(num_output_channels=1), # 确保是单通道
45             transforms.ToTensor(), # 转换为 Tensor
46             transforms.Normalize((0.5,), (0.5,)) # 归一化
47         ])
48
49         image = transform(image).unsqueeze(0).to(self.device) # 添加批次维度
50         with torch.no_grad():
51             output = self.model(image)
52             _, predicted = torch.max(output.data, 1)
53             return predicted.item()
54
```

GUI界面

- (1) 定义模型结构，与训练时一致
- (2) 加载训练好的模型model.pth，进行预测
- (3) 图形用户界面 (App 类)

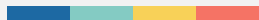
使用 Tkinter 创建了一个简单的 GUI，包括画布、按钮等组件。

```
56 # 创建 GUI 界面
57 class App(tk.Tk): 1个用法
58     def __init__(self):
59         super().__init__()
60         self.title("手写数字识别")
61         self.geometry("400x400")
62
63         self.canvas = tk.Canvas(self, bg="white", width=280, height=280)
64         self.canvas.pack(pady=20)
65
66         self.button_clear = tk.Button(self, text="清除", command=self.clear_canvas)
67         self.button_clear.pack(side=tk.LEFT, padx=10)
68
69         self.button_predict = tk.Button(self, text="识别", command=self.predict_digit)
70         self.button_predict.pack(side=tk.RIGHT, padx=10)
71
72         self.image = Image.new('L', (280, 280), color=255) # 创建白色背景图像
73         self.draw = ImageDraw.Draw(self.image)
74
75         self.canvas.bind("<B1-Motion>", self.paint)
76
77     def clear_canvas(self): 1个用法
78         self.canvas.delete("all")
79         self.image = Image.new('L', (280, 280), color=255) # 清空图像
80         self.draw = ImageDraw.Draw(self.image)
81
82     def paint(self, event): 1个用法
83         x, y = event.x, event.y
84         self.canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill="black", outline="black")
85         self.draw.ellipse((x - 5, y - 5, x + 5, y + 5), fill="black", outline="black")
86
87     def predict_digit(self): 1个用法
88         recognizer = HandwrittenDigitRecognizer('model.pth') # 确保路径正确
89         digit = recognizer.predict(self.image)
90         messagebox.showinfo(title="识别结果", message=f"识别的数字是: {digit}")
91
92
93 if __name__ == "__main__":
94     app = App()
```

04 测试验证



测试验证

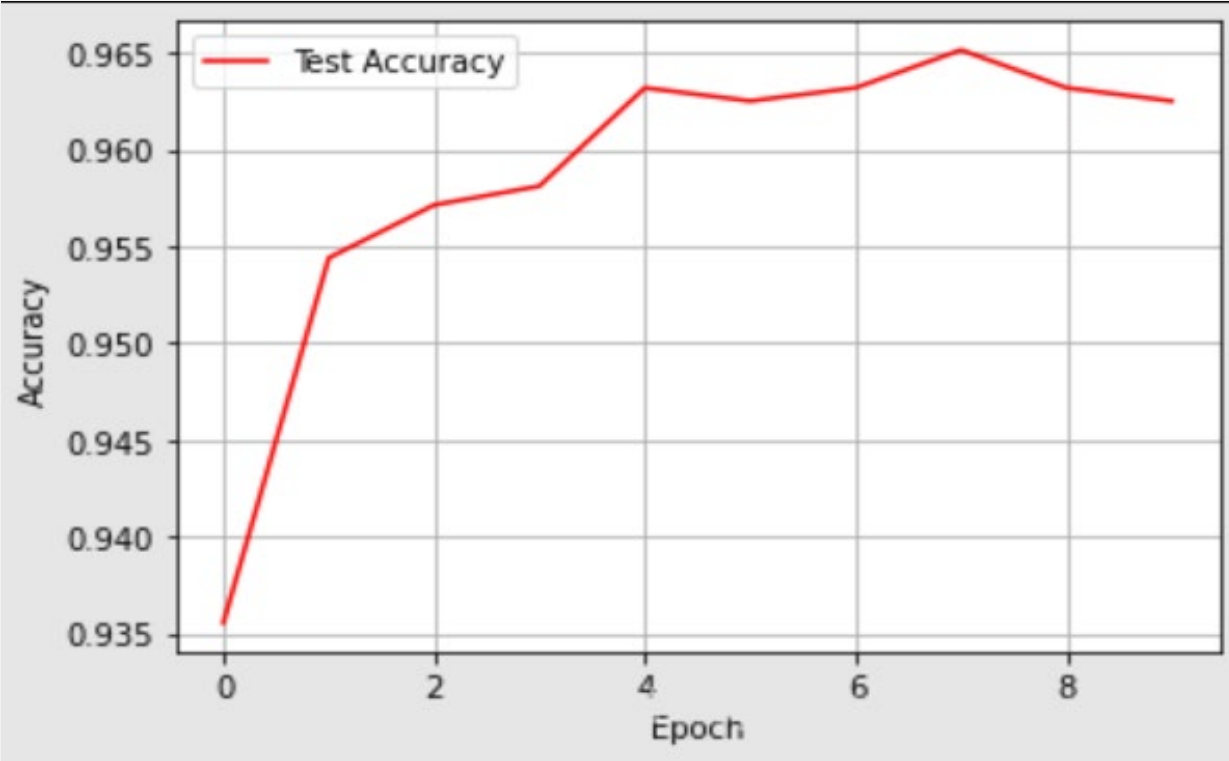
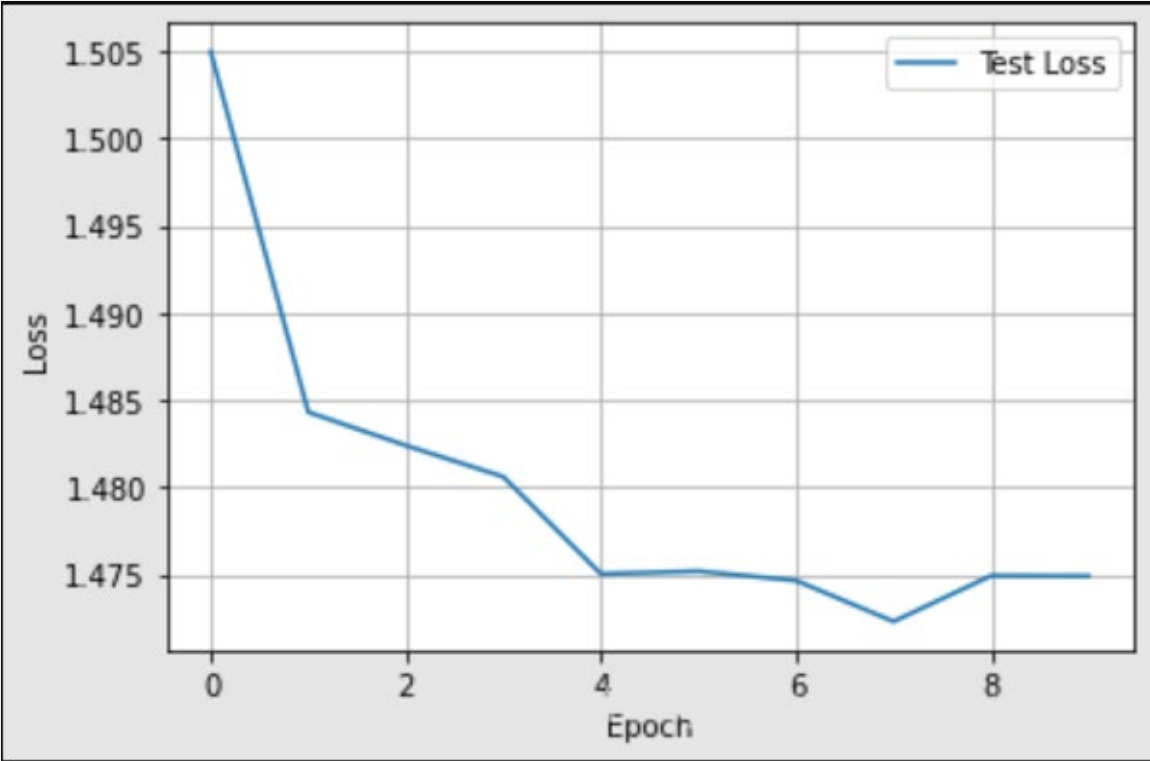


```
C:\Users\CHENYU\anaconda3\envs\new_env\python.exe C:\Users\CHENYU\Desktop\mnist\mnist.py
Net(
  (model): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=3136, out_features=128, bias=True)
    (10): ReLU()
    (11): Linear(in_features=128, out_features=10, bias=True)
  )
)
[1/10] Loss: 0.0497, Acc: 0.9792: 100%|██████████| 235/235 [00:35<00:00, 6.71step/s]
0%|          | 0/235 [00:00<?, ?step/s][Epoch 1/10] Test Loss: 0.0944, Test Accuracy: 0.9700
[2/10] Loss: 0.0422, Acc: 0.9688: 100%|██████████| 235/235 [00:24<00:00, 9.71step/s]
0%|          | 0/235 [00:00<?, ?step/s][Epoch 2/10] Test Loss: 0.0448, Test Accuracy: 0.9853
[3/10] Loss: 0.0170, Acc: 1.0000: 100%|██████████| 235/235 [00:23<00:00, 9.92step/s]
[Epoch 3/10] Test Loss: 0.0342, Test Accuracy: 0.9884
[4/10] Loss: 0.0112, Acc: 1.0000: 100%|██████████| 235/235 [00:24<00:00, 9.75step/s]
[Epoch 4/10] Test Loss: 0.0340, Test Accuracy: 0.9880
[5/10] Loss: 0.0028, Acc: 1.0000: 100%|██████████| 235/235 [00:22<00:00, 10.33step/s]
[Epoch 5/10] Test Loss: 0.0273, Test Accuracy: 0.9905
[6/10] Loss: 0.0145, Acc: 0.9896: 100%|██████████| 235/235 [00:21<00:00, 10.87step/s]
0%|          | 0/235 [00:00<?, ?step/s][Epoch 6/10] Test Loss: 0.0279, Test Accuracy: 0.9906
[7/10] Loss: 0.0018, Acc: 1.0000: 100%|██████████| 235/235 [00:19<00:00, 11.92step/s]
0%|          | 0/235 [00:00<?, ?step/s][Epoch 7/10] Test Loss: 0.0291, Test Accuracy: 0.9898
[8/10] Loss: 0.0221, Acc: 0.9792: 100%|██████████| 235/235 [00:21<00:00, 10.92step/s]
[Epoch 8/10] Test Loss: 0.0241, Test Accuracy: 0.9917
[9/10] Loss: 0.1091, Acc: 0.9896: 100%|██████████| 235/235 [00:23<00:00, 10.17step/s]
[Epoch 9/10] Test Loss: 0.0344, Test Accuracy: 0.9900
[10/10] Loss: 0.0170, Acc: 1.0000: 100%|██████████| 235/235 [00:22<00:00, 10.39step/s]
[Epoch 10/10] Test Loss: 0.0326, Test Accuracy: 0.9897
```

进程已结束，退出代码为 0

准确率与损失率测试
记录每一个Epoch的
Loss与Accuracy

测试验证



绘制出测试过程的损失率与准确率曲线

测试验证



```
C:\Users\CHENYU\anaconda3\envs\dl\python.exe D:\python\mnist\predict.py
```

预测结果： 8

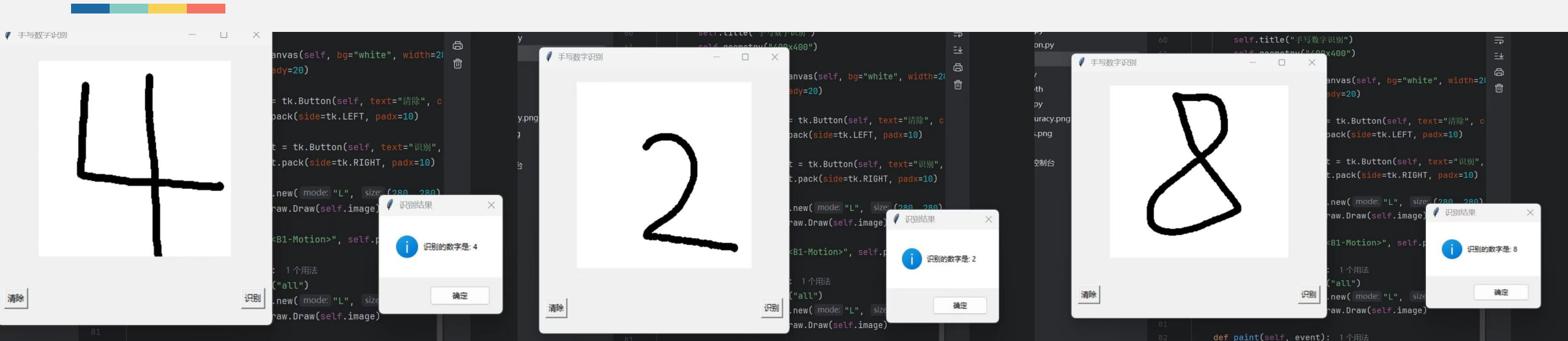
对test中数字8的图片进行预测，结果符合

```
C:\Users\CHENYU\anaconda3\envs\dl\python.exe D:\python\mnist\evaluation.py
```

Accuracy: 0.9928

评估模型准确度，达到0.9928

测试验证



GUI界面及识别结果

05 总结



总结



1.项目成果

- (1) 基于Pytorch深度学习，利用CNN模型对MNIST数据集进行训练。
- (2) 在测试集上达到了预期的准确率。
- (3) 完成了手写数字识别GUI系统的设计与实现。

2.未来工作

- (1) 进一步优化模型性能。
- (2) 利用其他数据集或者其他模型进行训练。
- (3) 扩展功能，如多语言支持、手写字符识别等。

3.进一步优化

- (1)数据增强：增加不同的旋转、缩放等变换，提高模型的泛化能力。
- (2)超参数调优：通过网格搜索等方法优化模型超参数。
- (3)更深的网络：尝试不同的 CNN 架构，如 ResNet 或 VGG。



敬 请 批 评 指 正

T A H N K Y O U F O R W A T C H I N G