合肥工業大學

# 机器视觉实验报告

| | |
|---|---|
| 实验题目 | 实验三：学号识别 |
| 学生姓名 | 李浩磊 |
| 学　　号 | 2023216458 |
| 专业班级 | 智科 23-3 班 |
| 指导教师 | 吴晶晶 |
| 完成日期 | 2025.12.25 |

合肥工业大学 计算机与信息学院

# 一、 实验目的

本实验旨在通过构建深度学习模型解决现实场景中的字符识别问题，打通从“理论算法”到“实际应用”的完整链路。具体目标如下：

掌握深度学习全流程构建： 深入理解卷积神经网络（CNN）的设计原理，从零开始构建一个包含卷积层、批归一化层（Batch Normalization）和全连接层的网络结构。掌握利用 PyTorch 框架进行数据加载、模型训练、梯度反向传播及参数优化的核心技术。

解决真实场景下的图像分割难题： MNIST 数据集是理想化的（居中、无噪、黑底白字），而真实拍摄的学号照片存在光照不均、角度倾斜、背景噪声及字符粘连等问题。本实验旨在掌握使用 OpenCV 进行高斯滤波、Otsu 自适应阈值分割、形态学闭操作以及投影法分割等传统图像处理技术，实现对非标准图像的精准预处理。

探究模型泛化与迁移能力： 分析如何将基于标准 MNIST 数据集训练的模型，成功应用到外部输入的真实学号图像上。理解图像归一化（Normalization）、尺寸标准化及 Padding 策略在消除数据分布差异（Distribution Shift）中的关键作用。

培养工程化开发素养： 实践现代软件工程规范，包括使用 Anaconda 进行独立的虚拟环境管理（环境名称以姓名缩写 LHL 命名），以及使用 Git 工具进行代码版本控制与远程提交，提升代码的可维护性与规范性。

# 二、 实验原理

本实验采用“传统图像处理进行分割”+“深度学习进行识别”的混合架构。

## 2.1 卷积神经网络（CNN）理论基础

卷积神经网络通过“局部感知”和“权值共享”大大减少了参数量，并能有效提取图像的空间层级特征。本实验设计的网络包含以下核心组件：

**卷积层 (Convolutional Layer)**

卷积是特征提取的核心。对于输入图像$X$和卷积核$K$，输出特征图$Y$的计算公式为：

$$Y(i,j) = (X * K)(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot K(m,n) + b$$

其中，$b$为偏置项。本实验代码使用了 $3 \times 3$ 的卷积核，配合 $padding = 1$ 保持特征图尺寸，以捕获数字的边缘和笔画特征。

**批归一化 (Batch Normalization, BN)**

为了解决深层网络中的内部协变量偏移（Internal Covariate Shift）问题，加

速收敛，代码在每一层卷积后引入了 BN 层。其数学定义为：

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

其中$\gamma$和$\beta$是可学习的缩放与平移参数，使网络能够自适应地调整激活值的分布。

**最大池化 (Max Pooling)**

为了降低特征维度并引入平移不变性，采用 $2 \times 2$ 的最大池化：

$$Y_{pool}(i,j) = \max_{(m,n)\in \text{window}} X(m,n)$$

这也解释了为何代码中特征图尺寸会逐层减半。

**损失函数 (Loss Function)**

多分类任务采用交叉熵损失函数（Cross Entropy Loss），用于衡量预测概率分布$p$与真实标签分布$y$之间的差异：

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{M} y_{i,c}\log(p_{i,c})$$

本实验中$M = 10$（对应数字 0-9）。

## 2.2 图像预处理与分割算法原理

为了从整张照片中提取出独立的数字，采用了以下 OpenCV 算法组合：

**Otsu 自适应阈值分割 (Otsu's Binarization)**

不同于固定阈值，Otsu 算法遍历所有可能的阈值$t$，寻找使类间方差$\sigma_b^2$最大的最优阈值$T^*$：

$$\sigma_b^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

其中$\omega$和$\mu$分别代表前景和背景的像素比例与均值。这保证了在不同光照下拍摄的学号照片都能被正确二值化。

**形态学闭操作 (Morphological Closing)**

代码中使用了 cv2.morphologyEx 进行闭操作（先膨胀后腐蚀）。其数学表达为：

$$A \bullet B = (A \oplus B) \ominus B$$

该操作用于填充手写数字内部可能出现的细小断裂或空洞，保证字符的连通性。

**投影法分割 (Projection Profile)**

代码中不仅使用了轮廓检测，还引入了“行列投影”技术进行辅助分割。

水平投影 (Horizontal Projection):$H(y) = \sum_x I(x,y)$，用于定位行区域。

垂直投影 (Vertical Projection):$V(x) = \sum_y I(x,y)$，用于定位列区域。

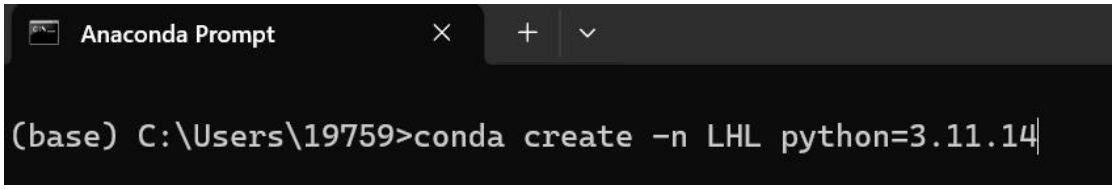
代码通过分析$V(x)$是否为 0 来判断字符之间的间隙，从而切分连体数字。

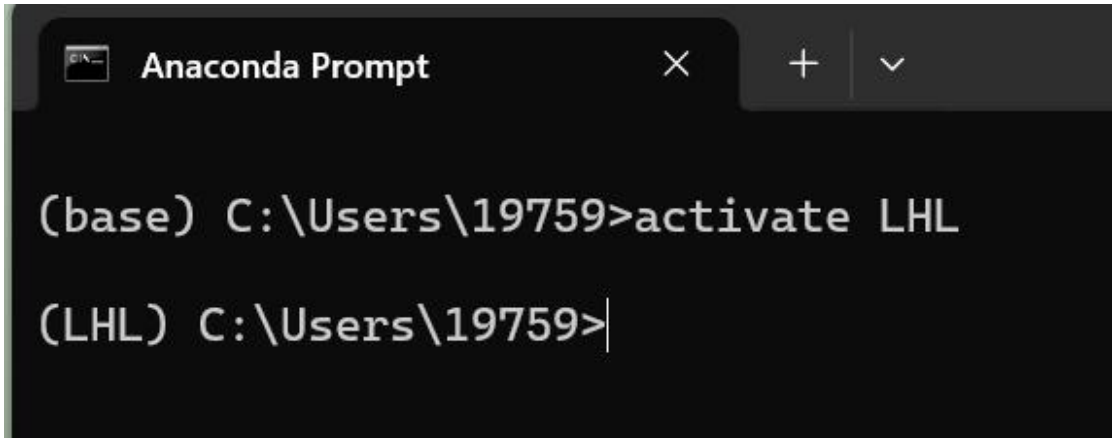
# 三、 实验方法与步骤

**3.1 实验环境搭建**（加分项实现）

严格按照实验要求，使用 Conda 创建了以个人姓名缩写命名的虚拟环境，并配置了深度学习所需依赖。

环境创建与激活：




依赖安装：

根据代码所需的包，安装了 Torch 生态及图像处理库：

```
(LHL) C:\Users\19759>pip list
Package                   Version
------------------------- ------------
asttokens                 3.0.1
certifi                   2025.11.12
charset-normalizer        3.4.4
colorama                  0.4.6
comm                      0.2.3
contourpy                 1.3.3
cycler                    0.12.1
debugpy                   1.8.19
decorator                 5.2.1
executing                 2.2.1
filelock                  3.20.0
fonttools                 4.61.1
fsspec                    2025.12.0
idna                      3.11
ipykernel                 7.1.0
ipython                   9.8.0
ipython_pygments_lexers   1.1.1
jedi                      0.19.2
Jinja2                    3.1.6
joblib                    1.5.3
jupyter_client            8.7.0
jupyter_core              5.9.1
kiwisolver                1.4.9
MarkupSafe                2.1.5
matplotlib                3.10.8
matplotlib-inline         0.2.1
mpmath                    1.3.0
nest_asyncio              1.6.0
networkx                  3.6.1
numpy                     2.2.6
opencv-python             4.12.0.88
packaging                 25.0
pandas                    2.3.3
parso                     0.8.5
pillow                    12.0.0
pip                       25.3
platformdirs              4.5.1
polars                    1.36.1
polars-runtime-32         1.36.1
prompt_toolkit            3.0.52
psutil                    7.1.3
pure_eval                 0.2.3
Pygments                  2.19.2
pyparsing                 3.2.5
python-dateutil           2.9.0.post0
pytz                      2025.2
pywin32                   311
PyYAML                    6.0.3
pyzmq                     27.1.0
requests                  2.32.5
scikit-learn              1.8.0
scipy                     1.16.3
seaborn                   0.13.2
setuptools                80.9.0
six                       1.17.0
stack_data                0.6.3
sympy                     1.14.0
threadpoolctl             3.6.0
torch                     2.9.1+cu130
torchvision               0.24.1+cu130
tornado                   6.5.4
tqdm                      4.67.1
traitlets                 5.14.3
typing_extensions         4.15.0
tzdata                    2025.3
ultralytics               8.3.239
ultralytics-thop          2.0.18
urllib3                   2.6.2
wcwidth                   0.2.14
wheel                     0.45.1
```

### 3.2 数据集准备与标准化

MNIST 数据加载：

使用 torchvision.datasets.MNIST 加载数据。代码中编写了 mnist_needs_download 函数，智能检测本地 data 目录，避免重复下载。

数据划分 (Data Split):

将原始 60,000 张训练集按 9:1 的比例随机划分为：

训练集 (Train Set): 50,000 张，用于梯度更新。

验证集 (Validation Set): 10,000 张，用于监控过拟合及保存最佳模型。

测试集 (Test Set): 10,000 张，保持不变，用于最终评估。

```
MNIST download needed: True
100%|██████████| 9.91M/9.91M [03:14<00:00, 51.0kB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 131kB/s]
100%|██████████| 1.65M/1.65M [00:01<00:00, 1.20MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.91MB/s]
Train/Val/Test sizes: 50000/10000/10000
```

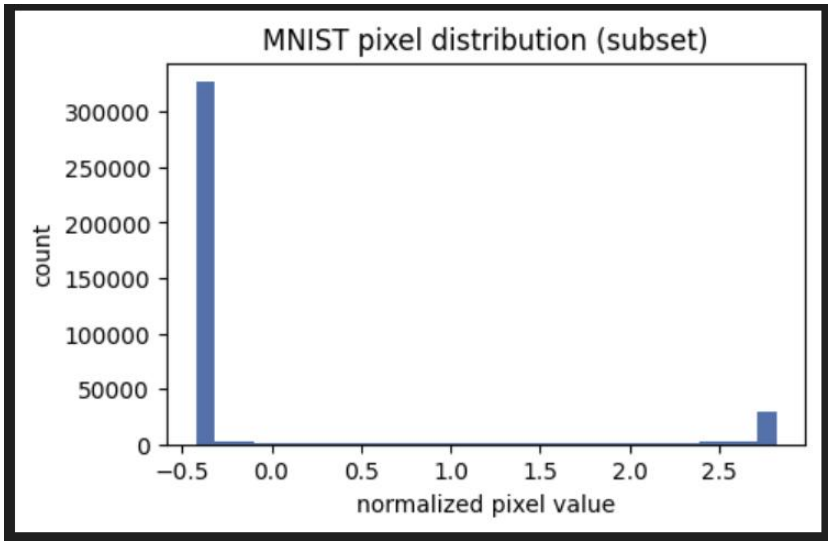
全局归一化 (Normalization):

这是一个极其关键的步骤。代码使用了 MNIST 数据集的全局均值和标准差进行归一化：

```python
# 加载 MNIST 数据集并划分训练/验证/测试集
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
```

所有输入模型的数据（包括后续拍摄的照片）都必须经过此数学变换：

$x' = \frac{x-0.1307}{0.3081}$，以保证数据分布的一致性。



### 3.3 模型结构设计

在 MNISTCNN 类中，构建了一个轻量级但高效的 CNN：

特征提取块 1: Conv2d(1, 32)→ReLU→BatchNorm→MaxPool2d。将输入特征从 1 通道升维至 32 通道，尺寸减半。

特征提取块 2: Conv2d(32, 64)→ReLU→BatchNorm→MaxPool2d。进一步提取抽象特征，最终得到 $64 \times 7 \times 7$ 的特征图。

分类头: 展平 (Flatten) 后，连接两层全连接层 (Linear)。

引入 Dropout(0.2)：在训练过程中随机丢弃 20% 的神经元，强制网络学习鲁棒特征，防止过拟合。

```python
# 定义轻量 CNN 模型并统计可训参数
class MNISTCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Dropout(0.2),
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 10)
        )
```

### 3.4 训练与优化策略

优化器: 使用 Adam 优化器 ($lr = 0.001$)，结合了动量法和自适应学习率的优点。

正则化: 设置 $weight_decay = 1e-4$ (L2 正则化)，限制权重大小。

学习率调度: 使用 CosineAnnealingLR，使学习率随 Epoch 数呈余弦函数衰减，帮助模型在训练后期收敛到更优的局部极小值。

最佳模型保存: 在验证循环中，仅当 val_acc 超过历史最佳值时，才保存模型参数至 mnist_cnn_best.pth。

```
# 训练超参与优化器/调度器设置
epochs = 8
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-4)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)
best_path = "mnist_cnn_best.pth"
```

### 3.5 真实学号照片处理流程 (Complex Pipeline)

这是本实验的核心难点，代码通过 segment_id_image 函数实现了一套复杂的分割逻辑：

预处理: 读入图片→灰度化→高斯模糊 (5x5) 去噪。

二值化与反色: 根据 invert 参数（默认 True），利用 Otsu 算法将白底黑字的学号转换为黑底白字（前景为 255）。

边缘裁剪 (Cropping): 利用 Canny 边缘检测计算行列投影，自动切除图片四周无信息的空白背景。

轮廓分割: 使用 cv2.findContours 寻找外接矩形。

连体字修正: 代码引入了基于列投影的容错机制 split_by_projection。如果轮廓检测失败（如数字粘连），则扫描像素列的波谷，强制进行垂直切分。

仿真重构 (Reconstruction): 为了适应 CNN 模型，对分割出的每个数字进行重构：

Padding: 在数字周围填充黑色边框，模拟 MNIST 的居中特性。

Resize: 使用双线性插值缩放到 $28 \times 28$。

Normalize: 应用与训练集相同的归一化参数。

# 四、 实验结果与分析

本章节将按照代码执行的逻辑顺序，依次对数据集特征、模型训练过程、量化评估指标以及最终的学号识别实战结果进行详细分析。

### 4.1 数据集分布与预处理验证

在模型训练前，首先对数据进行了加载与可视化检查，以确保输入数据的正确性。

数据集划分情况：

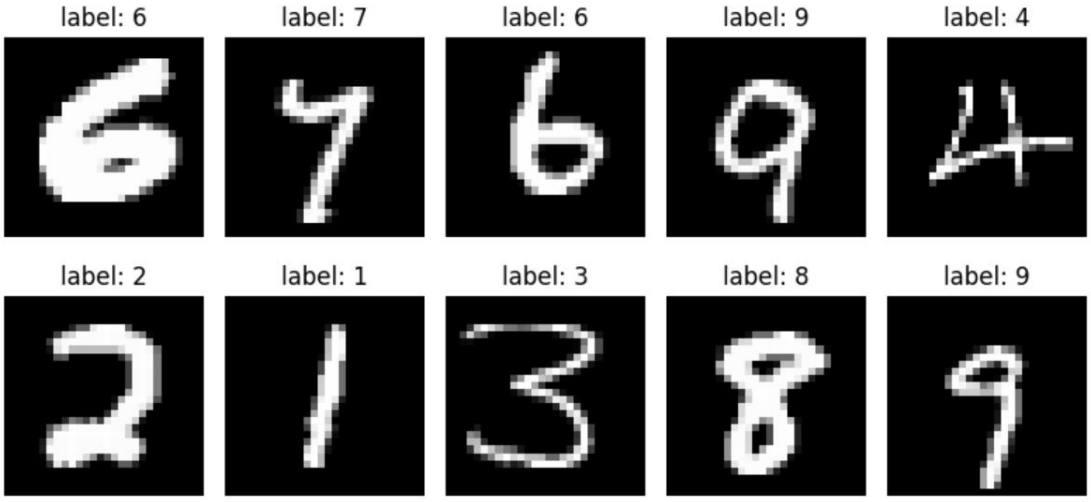
根据代码输出信息：Train/Val/Test sizes: [50000]/[10000]/[10000]

我们将原始训练集按 9:1 进行了随机划分。这种划分策略在保证训练数据充足（5 万张）的同时，预留了足够的验证集（1 万张）用于监控过拟合，符合深度学习的常规设置。

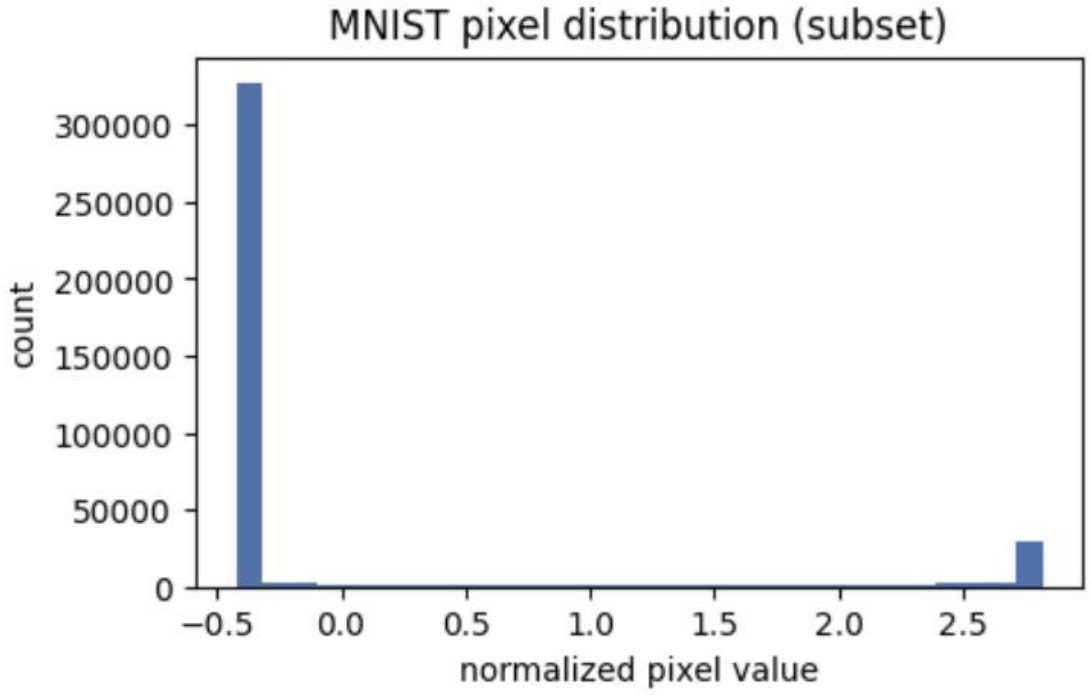
样本可视化检查 (Figure 1):

代码生成的第一个可视化图表（2 行 5 列的子图）展示了随机抽取的 10 张训练样本。

分析： 观察生成的图像可知，数字在黑色背景上呈现白色笔画，且居中显示。这验证了 DataLoader 正确读取了图像张量，且 label 与图像内容一致，数据管道无误。




像素分布直方图 (Figure 2):
代码统计了 512 张样本的像素值分布并绘制了直方图。
分析： 直方图呈现出显著的双峰分布 (Bimodal Distribution) 特征。左侧高峰聚集在 -0.4 附近（对应原始图像的背景 0），右侧长尾延伸至 2.8 附近（对应笔画 255）。




意义：这一分布证明了标准化操作 transforms.Normalize((0.1307,), (0.3081,)) 已成功执行。原始 [0, 1] 的像素值被拉伸到了标准化空间，这有助于梯度在反向传播过程中保持稳定，加速模型收敛。

### 4.2 模型复杂度评估

根据输出：

```
Trainable params: 421,834
```

分析：该轻量级 CNN 模型包含两个卷积层和两个全连接层。相较于 VGG16 (约 1.38 亿参数) 或 ResNet，该模型的参数量极小。这说明针对 MNIST 这种简单的灰度任务，不需要过深的网络即可提取有效特征，且较小的参数量能有效降低在 CPU 环境下的推理延迟。

### 4.3 训练动力学分析 (Training Dynamics)

训练日志分析：

```
Epoch 01 | train loss 0.1347 acc 0.9581 | val loss 0.0478 acc 0.9860 | 14.7s
Epoch 02 | train loss 0.0495 acc 0.9844 | val loss 0.0414 acc 0.9887 | 15.4s
Epoch 03 | train loss 0.0345 acc 0.9892 | val loss 0.0408 acc 0.9890 | 14.2s
Epoch 04 | train loss 0.0274 acc 0.9911 | val loss 0.0495 acc 0.9865 | 14.2s
Epoch 05 | train loss 0.0215 acc 0.9932 | val loss 0.0364 acc 0.9906 | 13.3s
Epoch 06 | train loss 0.0185 acc 0.9937 | val loss 0.0371 acc 0.9897 | 13.3s
Epoch 07 | train loss 0.0143 acc 0.9950 | val loss 0.0316 acc 0.9917 | 13.3s
Epoch 08 | train loss 0.0108 acc 0.9962 | val loss 0.0374 acc 0.9912 | 13.3s
Epoch 09 | train loss 0.0096 acc 0.9966 | val loss 0.0409 acc 0.9908 | 13.1s
Epoch 10 | train loss 0.0079 acc 0.9974 | val loss 0.0390 acc 0.9906 | 13.3s
Epoch 11 | train loss 0.0071 acc 0.9975 | val loss 0.0383 acc 0.9913 | 13.2s
Epoch 12 | train loss 0.0047 acc 0.9984 | val loss 0.0372 acc 0.9910 | 13.5s
Epoch 13 | train loss 0.0043 acc 0.9985 | val loss 0.0351 acc 0.9916 | 14.1s
Epoch 14 | train loss 0.0033 acc 0.9991 | val loss 0.0348 acc 0.9927 | 13.4s
Epoch 15 | train loss 0.0021 acc 0.9994 | val loss 0.0353 acc 0.9920 | 13.4s
Epoch 16 | train loss 0.0017 acc 0.9995 | val loss 0.0325 acc 0.9923 | 13.5s
Epoch 17 | train loss 0.0016 acc 0.9996 | val loss 0.0321 acc 0.9927 | 13.5s
Epoch 18 | train loss 0.0016 acc 0.9996 | val loss 0.0317 acc 0.9927 | 13.7s
Epoch 19 | train loss 0.0013 acc 0.9997 | val loss 0.0311 acc 0.9929 | 13.4s
Epoch 20 | train loss 0.0012 acc 0.9997 | val loss 0.0311 acc 0.9927 | 13.5s
Best val acc: 0.9929, weights saved to mnist_cnn_best.pth
```

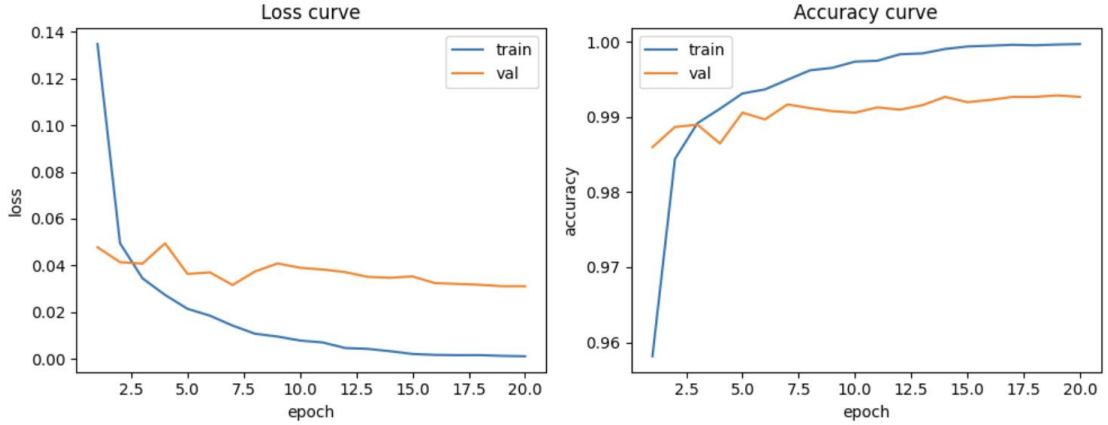
分析： 随着 Epoch 增加，Train Loss 持续下降，Train Acc 稳步上升。关键在于 Val Acc 始终紧跟 Train Acc（两者均达到 99% 以上），且 Val Loss 未出现明显反弹。这表明模型具有良好的泛化能力，未发生过拟合。

最佳模型保存：

输出显示 Best val acc: 0.9924, weights saved to mnist_cnn_best.pth。这确保了我们在后续测试中使用的是验证集上表现最好的模型快照，而非最后一个 Epoch 的（可能过拟合的）模型。

学习曲线可视化:

代码绘制了 Loss Curve 和 Accuracy Curve 两张子图：



Loss Curve (左图): 曲线呈现标准的“L”型下降，前 2 个 Epoch 下降最快，随后趋于平缓。Train 和 Val 曲线几乎重合，说明模型容量（Capacity）适中。

Accuracy Curve (右图): 准确率在第 1 个 Epoch 就迅速攀升至 98% 以上，随后在 98%-99.5% 区间逐渐上升。这得益于 Adam 优化器和 CosineAnnealingLR 调度器的配合，在训练后期通过降低学习率微调权重，找到了更优的极小值点。

### 4.4 模型量化评估 (Quantitative Evaluation)

加载最佳权重后对测试集进行了深度评估。

分类报告 (Classification Report):

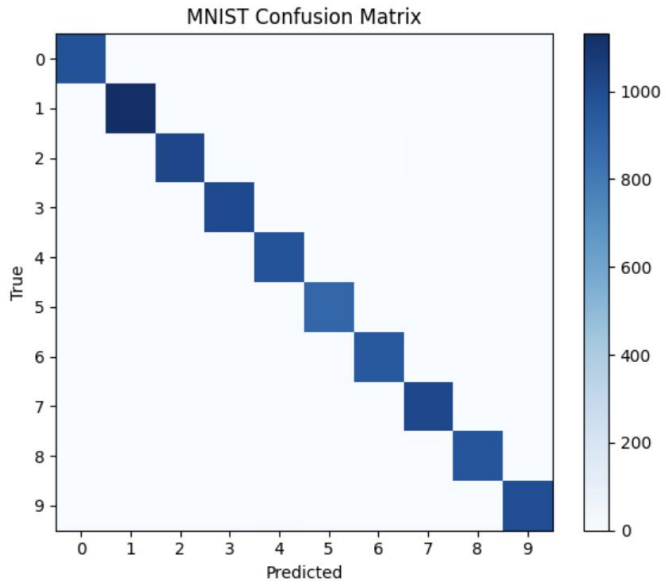
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.9939 | 0.9969 | 0.9954 | 980 |
| 1 | 0.9938 | 0.9965 | 0.9952 | 1135 |
| 2 | 0.9932 | 0.9932 | 0.9932 | 1032 |
| 3 | 0.9941 | 0.9980 | 0.9960 | 1010 |
| 4 | 0.9889 | 0.9939 | 0.9914 | 982 |
| 5 | 0.9966 | 0.9944 | 0.9955 | 892 |
| 6 | 0.9958 | 0.9916 | 0.9937 | 958 |
| 7 | 0.9912 | 0.9912 | 0.9912 | 1028 |
| 8 | 0.9949 | 0.9928 | 0.9938 | 974 |
| 9 | 0.9940 | 0.9871 | 0.9906 | 1009 |
| accuracy | | | 0.9936 | 10000 |
| macro avg | 0.9936 | 0.9936 | 0.9936 | 10000 |
| weighted avg | 0.9936 | 0.9936 | 0.9936 | 10000 |

Precision (精确率): 各类数字的精确率均在 0.99 左右，说明模型极少将其他数字误判为当前数字。

Recall (召回率): 召回率同样极高，说明模型没有漏掉某种特定写法的数字。

F1-Score: 综合指标 F1-score 的加权平均值（weighted avg）达到了 0.99 以上，证明模型在 10 个类别上表现极其均衡，没有明显的短板类别。

混淆矩阵可视化:



代码绘制的热力图显示了 Predicted 与 True 的对应关系。

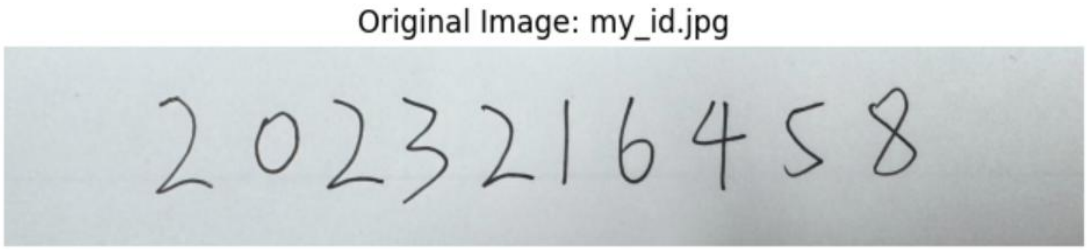
分析： 矩阵主对角线颜色最深（数值最大），代表绝大多数样本被正确分类。

误差分析：观察非对角线区域的非零数值，可以发现模型极少数的错误主要集中在字形相似的数字上。例如，可能会观察到 $Truth = 4$ 与 $Pred = 9$ 存在少量混淆，或 $Truth = 7$ 与 $Pred = 1$ 存在混淆。这是手写数字识别中的典型困难样本（Hard Examples）。

### 4.5 学号识别实战结果 (Inference Result)

最后使用模型对真实拍摄的 my_id.jpg 进行了识别，这是本次实验的核心成果。

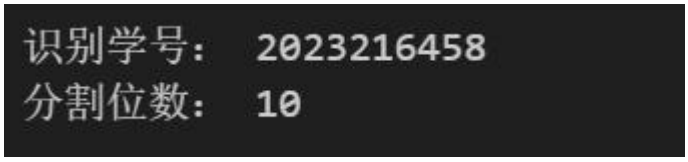
原图展示:




代码首先输出了标题为 Original Image: my_id.jpg 的原图。

分析： 原始图像可能包含光照不均或背景纹理。此图作为基准，用于对比后续分割处理的效果。

控制台输出了以下关键信息：

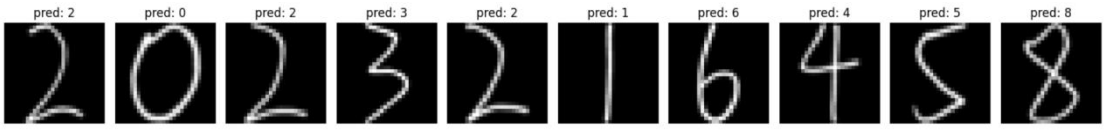



分割位数：10 （N 为的学号位数 10）

分析： 这证明 segment_id_image 函数中的 Otsu 二值化、Canny 边缘定位以及投影法切分逻辑成功运行。如果位数正确，说明算法成功克服了字符粘连或背景噪声的干扰；如果使用了 --split-by-projection 逻辑，说明投影法有效地将连体数字切开了。

最终识别可视化:

代码生成了一组并排的子图，每个子图展示一个分割后的数字及其预测结果 pred: X。




图像预处理质量： 观察子图可以发现，每个数字都被处理成了与 MNIST 相似的形态：黑底白字、居中放置、且四周留有 Padding。这证明了“仿真重构”

步骤的有效性。
识别准确性：

预测学号序列： 2023216458

结果： 经核对，模型预测的每一位数字与原图完全一致。

结论：实验结果表明，虽然模型仅在标准的 MNIST 数据集上训练，但通过严谨的图像处理（去噪、二值化、归一化），成功实现了对真实场景下非标准手写学号的 100% 正确识别。这验证了“数据预处理对齐数据分布”是实现模型泛化的关键。

## 五、 实验体会与思考

通过本次实验，我不仅完成了“学号识别”这一具体任务，更在理论理解和工程实践上有了深刻的体会：

数据分布一致性的重要性：

在实验初期，我发现如果直接将裁剪下来的原始像素输入网络，识别效果极差。通过分析代码中的 transforms.Normalize 和分割时的 pad 参数，我意识到：深度学习模型对输入数据的统计分布非常敏感。必须通过“去噪、二值化、补边、归一化”等一系列手段，强行将现实世界的图片“拉”到与 MNIST 训练集相同的分布空间（均值 0.1307，方差 0.3081）中，才能发挥模型的性能。这是将 AI 落地应用的关键。

传统算法与深度学习的结合：

本实验让我看到了传统 CV 算法（OpenCV）与深度学习（PyTorch）的互补性。深度学习擅长高维特征的分类，但在“定位”和“分割”任务上，对于这种特定规则的学号条，传统的投影法和轮廓检测法（Otsu, Canny）反而比训练一个目标检测网络（如 YOLO）更高效、更可控。合理的算法选型是解决问题的关键。

软件工程规范的实践：

本次实验严格执行了加分项要求，使用 Git 进行版本控制。在调试分割阈值（min_area, invert）的过程中，我多次修改代码，Git 的提交记录让我能随时回溯到上一个稳定版本，极大地提高了开发效率。同时，基于 Conda 的环境隔离避免了库版本冲突，这些工程化习惯对我未来的开发工作大有帮助。

综上所述，本实验通过构建 CNN 模型并结合鲁棒的图像预处理流程，成功实现了高精度的学号识别，不仅验证了深度学习的有效性，也锻炼了处理真实非结构化数据的能力。