

合肥工业大学

2025 年《机器视觉》实验



实验内容: 学号识别

姓 名: 折浩宇

学 号: 2023217896

完成时间: 2025/12/27

一、实验目的

1. 掌握手写数字识别的基本方法与完整流程：
包括数据集准备、模型训练、测试评估以及在实际图像中的应用。
2. 基于 MNIST 数据集分别构建并训练两种识别模型：
传统机器学习方法 SVM 与深度学习方法 CNN，理解两者的建模思路与差异。
3. 设计并实现“学号照片识别”流程：
对学号图片进行预处理与数字分割，将单个数字归一化后输入模型识别，最终按顺序拼接输出学号结果。
4. 对比 SVM 与 CNN：
在测试集准确率、训练耗时以及真实学号图片识别效果上的表现，分析影响识别效果的主要因素并总结改进方向。

二、实验原理

2.1 SVM 实现多分类原理

多分类策略：化整为零

SVM 本身是一个二元分类器，即只能区分两个类别。要将其应用于多分类任务（如本实验的手写数字识别），必须采用组合策略。

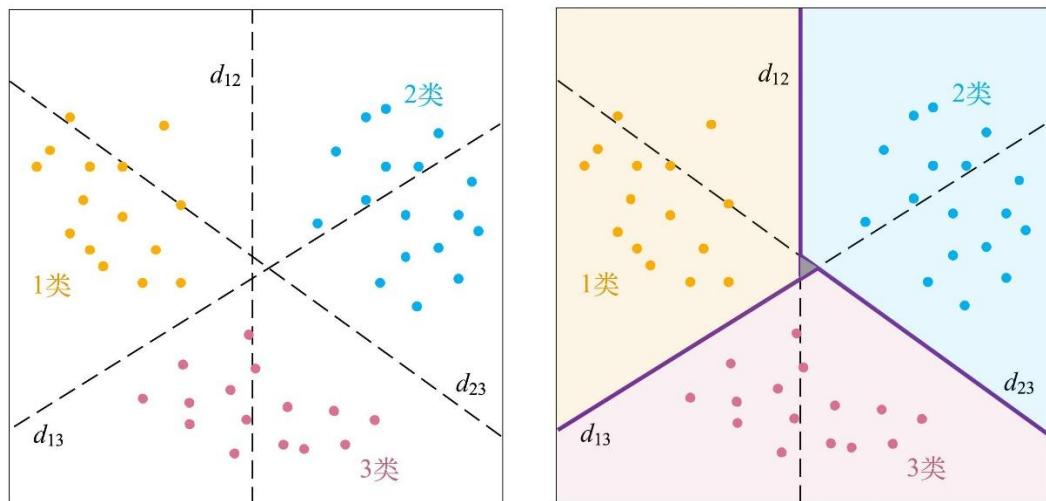
➤ 一对一 (One-vs-One)：

策略：这是一种“循环赛”策略。如果一个任务有 K 个类别，那么就为每一对类别组合都训练一个专门的二元分类器。总共需要训练 $K * (K - 1) / 2$ 个模型。

训练：例如，对于数字 0-9 ($K=10$)，需要训练 45 个分类器 (0 vs 1, 0 vs 2, ..., 8 vs 9)。每个分类器只用对应两个类别的数据进行训练。

预测：当一个新样本需要预测时，它会被输入到所有 45 个分类器中。每个分类器都会进行一次“投票”。例如，“0 vs 1”分类器如果认为样本是“1”，则类别“1”获得一票。最终，得票最多的那个类别就是最终的预测结果。

特点：虽然模型数量多，但每个模型训练的数据量小，速度较快，且在处理类别不平衡问题时表现稳健。Scikit-learn 库中的 SVC 分类器默认采用此策略。

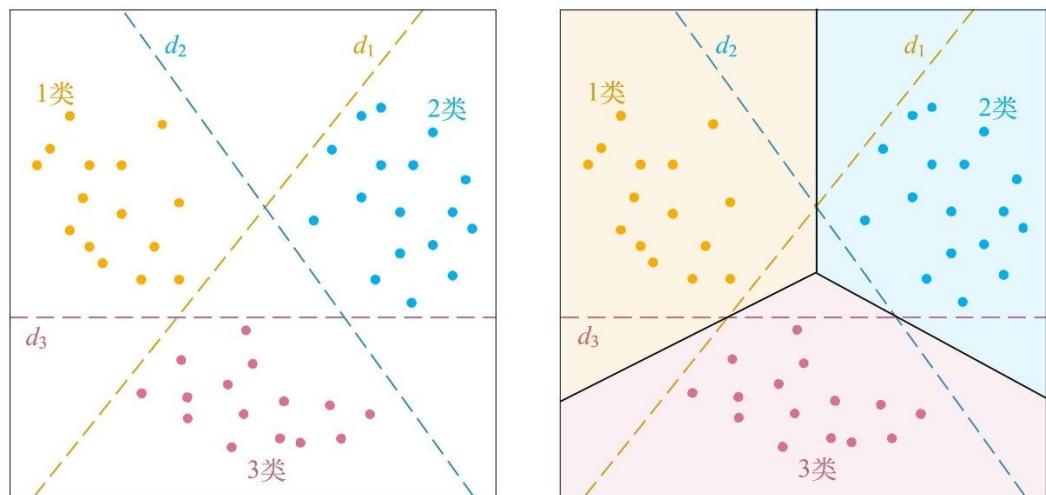


➤ 一对多 (One-vs-Rest):

策略: 这是一种“选拔赛”策略。对于 K 个类别，只训练 K 个分类器。

训练: 每个分类器 i 的任务是区分“是类别 i ”还是“不是类别 i ”（即其余所有 $K-1$ 个类别）。

预测: 新样本会被输入到所有 K 个分类器中，每个分类器都会输出一个判断其为“本类别”的置信度分数。最终，置信度分数最高的那个分类器所代表的类别就是预测结果。

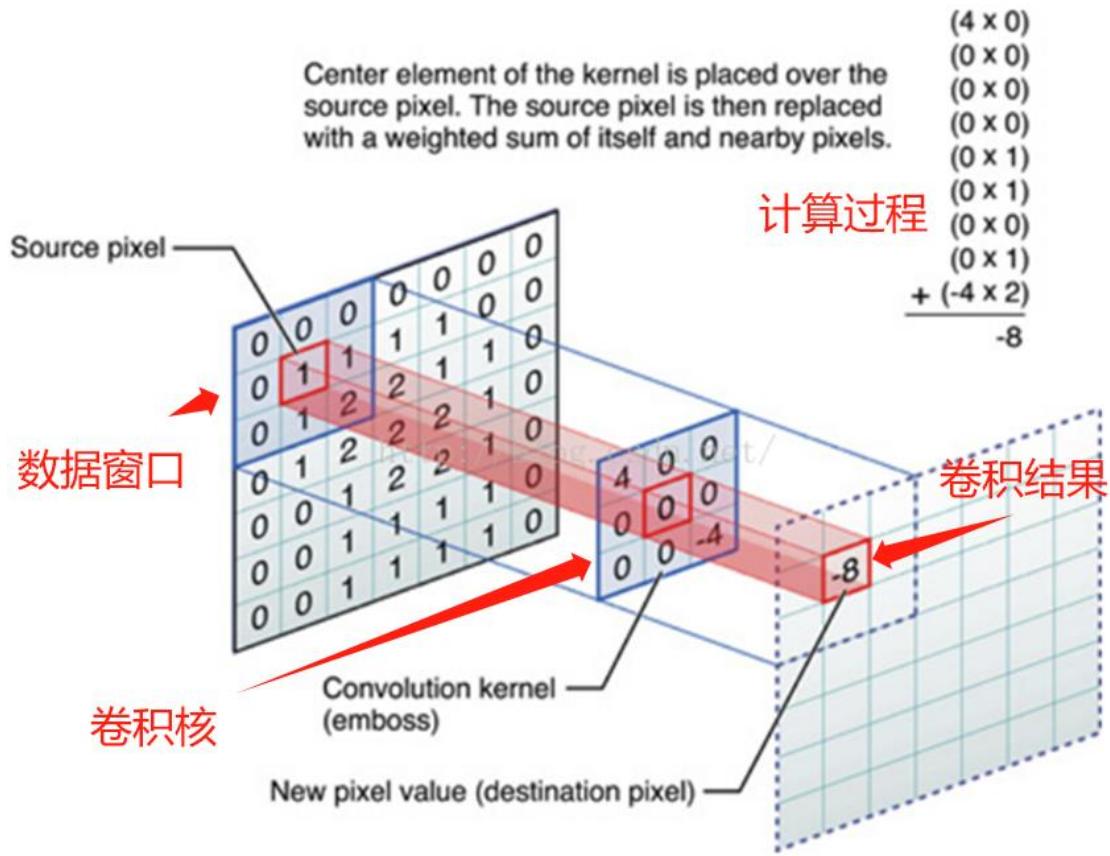


2.2 卷积神经网络原理

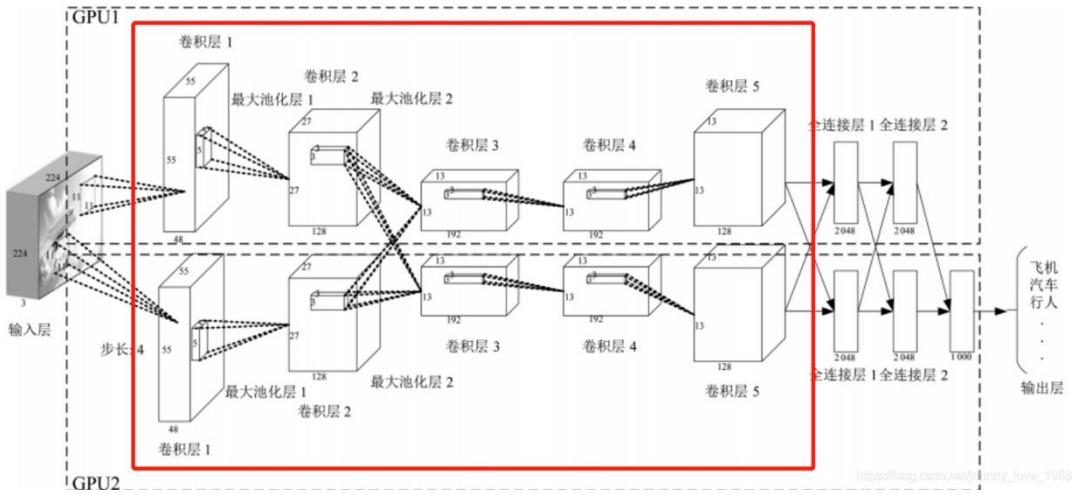
2.2.1 卷积

卷积操作就是用一个可移动的小窗口来提取图像中的特征，这个小窗口包含了一组特定的权重，通过与图像的不同位置进行卷积操作，网络能够学习并捕捉

到不同特征的信息。



2.2.2 卷积神经网络



1 输入层

输入层接收原始图像数据。图像通常由三个颜色通道（红、绿、蓝）组成，形成一个二维矩阵，表示像素的强度值。

2 卷积和激活

卷积层将输入图像与卷积核进行卷积操作。然后，通过应用激活函数（如

ReLU) 来引入非线性。这一步使网络能够学习复杂的特征。

3 池化层

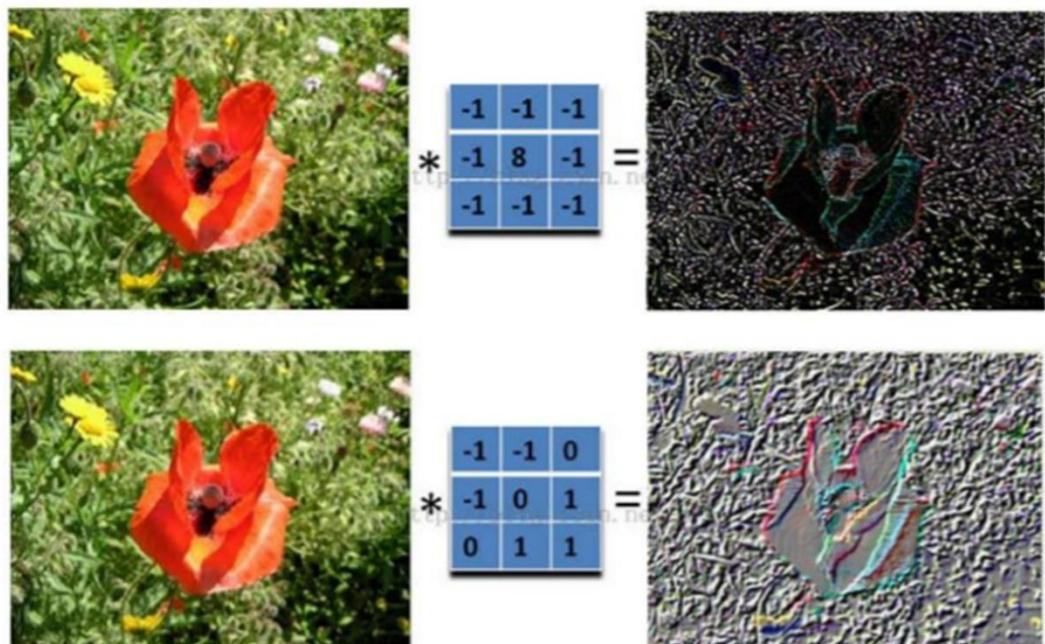
池化层通过减小特征图的大小来减少计算复杂性。它通过选择池化窗口内的最大值或平均值来实现。这有助于提取最重要的特征。

4 多层堆叠

CNN 通常由多个卷积和池化层的堆叠组成，以逐渐提取更高级别的特征。深层次的特征可以表示更复杂的模式。

5 全连接和输出

最后，全连接层将提取的特征映射转化为网络的最终输出。这可以是一个分类标签、回归值或其他任务的结果。



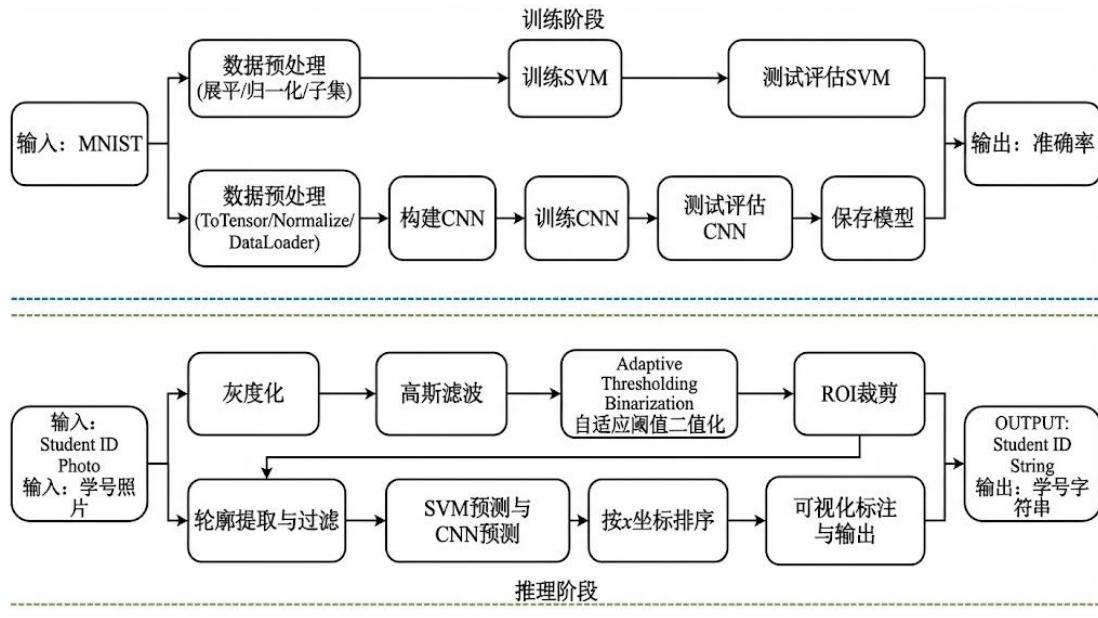
图片经过卷积后的样子

三、实验流程与解析

3.1 整体流程

本实验首先导入 PyTorch、torchvision、sklearn 与 OpenCV 等库并配置运行设备（优先使用 GPU），随后下载并加载 MNIST 训练集与测试集；针对 SVM 分支，将 28×28 灰度图展平为 784 维向量并归一化到 $[0,1]$ ，选取部分训练样本训练 RBF 核 SVM，并在测试集上计算准确率与分类报告；针对 CNN 分支，使用 ToTensor 与 Normalize 对数据标准化，构建包含两层卷积+池化、Dropout

与全连接层的网络，采用交叉熵损失与 Adam 优化器训练若干轮并在测试集评估准确率，最后保存模型参数。完成训练后，将学号照片读入并进行灰度化、高斯滤波去噪与自适应阈值二值化，再通过轮廓提取定位每个数字区域，对每个 ROI 按比例缩放并居中填充到 28×28 画布以匹配 MNIST 输入分布，随后分别输入 SVM 与 CNN 得到逐位预测结果，并按数字框的 x 坐标从左到右排序拼接生成最终学号，同时在原图上绘制预测框与标签实现可视化对比输出。



实验流程图

3.2 关键步骤解释

3.2.1: MNIST 数据加载与双分支数据准备 (SVM / CNN)

获取 MNIST 训练集与测试集，并分别构造成适配 SVM 与 CNN 的输入格式。

- 使用 `torchvision.datasets.MNIST` 下载并加载训练集（60000）与测试集（10000）。
- **SVM 分支：**
 1. 从 `dataset.data` 取出形状为 $(N, 28, 28)$ 的原始灰度图数组；
 2. 将每张图像展平为 784 维向量： `reshape(N, 784)`；
 3. 像素归一化： $X = X / 255.0$ ，将像素从 $[0, 255]$ 统一到 $[0, 1]$ 。
- **CNN 分支：**
 1. 使用 `transforms.ToTensor()` 将图像转为张量并缩放到 $[0, 1]$ ；

2. 使用 `transforms.Normalize((0.1307,), (0.3081,))` 进行标准化;
3. 用 `DataLoader` 按 batch (如 64) 构造训练/测试迭代器。

SVM 只能接受二维“特征向量”输入，因此必须展平；CNN 需要保留空间结构 ($1 \times 28 \times 28$)，并通过标准化让输入分布稳定、训练更易收敛。

3.2.2: 训练 SVM (RBF 核) 并在 MNIST 测试集评估

用传统机器学习模型 SVM 完成 MNIST 十分类训练与评估，作为对比基线。

- 训练样本选择：为控制训练耗时，从训练集中取前 10000 张作为训练子集。
- 模型配置：`SVC(C=10, kernel='rbf', gamma='scale')`。
- 训练：调用 `fit(X_train_svm, y_train_svm)`，同时记录训练起止时间得到耗时。
- 测试：对完整测试集 `X_test_flat` 做 `predict` 得到 `y_pred`;
- 评估：计算准确率 `accuracy_score(y_test, y_pred)`，并输出 `classification_report` (precision/recall/F1)。

RBF 核能处理非线性分类边界；由于 SVC 在大规模数据上训练成本高，使用子集训练能在可接受时间内完成对比实验，同时保留较强的分类能力。

3.2.3: 构建并训练 CNN (卷积网络) 以及测试集评估与保存

训练 CNN 作为深度学习方法的对比模型，并保存权重用于后续学号照片推理。

- 网络结构要点：两层卷积 + 两次 2×2 最大池化 + Dropout + 全连接输出 10 类。尺寸变化为：
 $1 \times 28 \times 28 \rightarrow 32 \times 14 \times 14 \rightarrow 64 \times 7 \times 7 \rightarrow \text{Flatten}(64 \times 7 \times 7) \rightarrow 128 \rightarrow 10$ 。
- 训练配置：损失函数 `CrossEntropyLoss`；优化器 `Adam(lr=0.001)`；训练轮数 `epochs=5`。
- 训练过程：每个 batch 执行 `zero_grad` → `forward` → `loss` → `backward` → `step`，每个 epoch 输出一次完成提示并记录总耗时。
- 测试评估：`model.eval() + torch.no_grad()`，对测试集逐 batch 前向推理，统计预测正确数/总数得到测试准确率。
- 保存模型：`torch.save(model.state_dict(), "mnist_cnn.pt")`。

CNN 能自动学习图像局部特征（边缘/笔画/结构），通常比手工展平特征更

鲁棒；保存权重后可直接加载用于真实学号图片识别，保证训练与推理一致。

3.2.4：训练过程中的关键训练策略与可复现性设置

在训练阶段明确训练策略（epoch/batch/优化器/损失）并说明可复现性与训练效率控制方法，保证实验结果可信、可复现。

1. **设备选择：** 使用 `torch.device("cuda" if torch.cuda.is_available() else "cpu")`，将模型与数据移动到同一设备上运行；GPU 可显著加速 CNN 训练。
2. **训练超参数设定：**
 - batch 大小: `batch_size=64` (兼顾梯度稳定与训练速度);
 - 学习率: `lr=0.001` (Adam 常用稳定起点);
 - epoch: `epochs=5` (在训练时间与效果之间折中)。
3. **损失函数与优化器：**
 - 多分类任务采用 `CrossEntropyLoss` (内部包含 softmax + NLL，适配 10 类分类);
 - 使用 Adam 自适应更新，减少手动调参成本。
4. **训练循环的标准步骤** (每个 batch 都严格遵循):
 - `optimizer.zero_grad()` 清空上一步梯度;
 - `output = model(data)` 前向传播得到 logits;
 - `loss = criterion(output, target)` 计算损失;
 - `loss.backward()` 反向传播计算梯度;
 - `optimizer.step()` 更新参数。
5. **训练时间记录：** 记录训练起止时间，得到训练耗时，用于与 SVM 训练耗时对比。

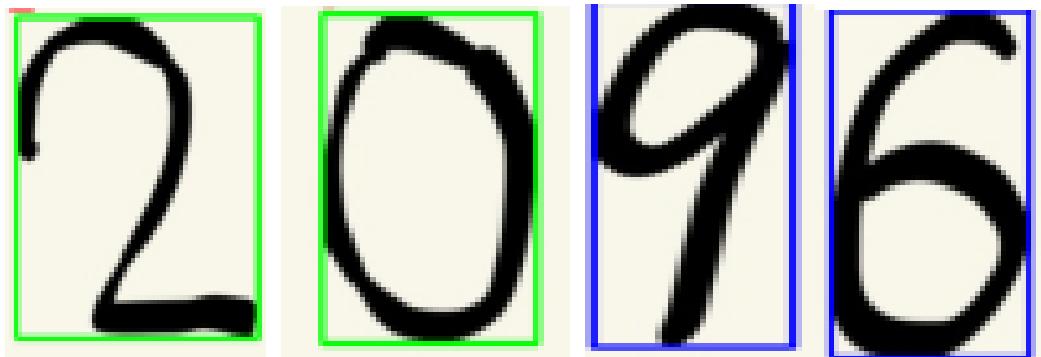
3.2.5：轮廓提取→ROI 裁剪→归一化到 28×28 (核心)

从整张学号图中分割出每个数字 ROI，并将其转换成与 MNIST 相同的输入风格 (28×28、比例合适、居中)。

1. **轮廓提取：** 在二值图上 `findContours(RETR_EXTERNAL)` 仅取外轮廓，减少内部噪声；

2. 对每个轮廓取外接矩形 (x,y,w,h) , 使用阈值 $w \geq 5$ 且 $h \geq 15$ 过滤小噪声;
3. ROI 裁剪: $roi = thresh[y:y+h, x:x+w]$;
4. 等比例缩放到“最大边≈20”:
 - 计算 $scale = 20.0 / \max(h_{roi}, w_{roi})$;
 - $roi_resized = \text{resize}(roi, (\text{int}(w_{roi}*scale), \text{int}(h_{roi}*scale)))$;
5. 居中贴到 28×28 画布:
 - 创建全零画布 $\text{canvas}(28,28)$;
 - 计算偏移: $x_offset = (28 - w_{resized})//2$, $y_offset = (28 - h_{resized})//2$;
 - 将 $roi_resized$ 贴到 canvas 对应位置。

模型训练使用的是 MNIST, MNIST 的数字具有固定画幅 28×28 、数字占比接近 20 像素范围且大多居中。真实照片分割出的 ROI 尺寸、比例和位置都不一致，若直接输入会造成“输入分布偏移”，显著降低识别率。通过“最大边缩放到 $20 + 28 \times 28$ 居中填充”，可以最大程度让真实 ROI 的外观统计特性接近 MNIST，从而提升 SVM/CNN 的可用性与准确率。



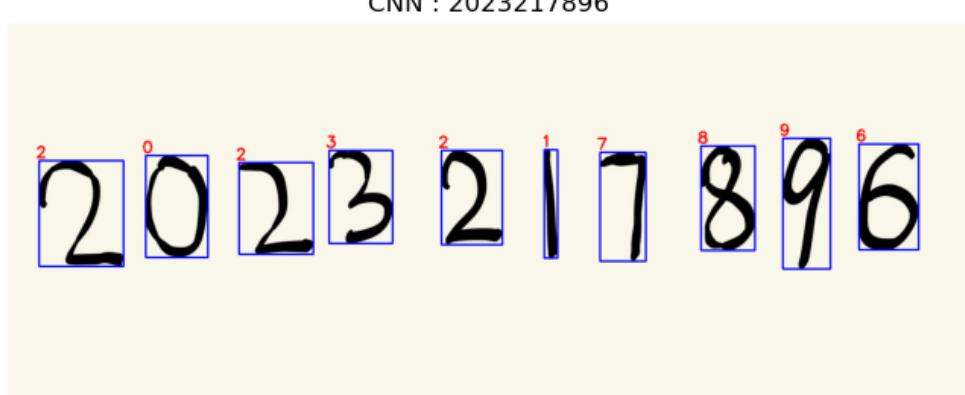
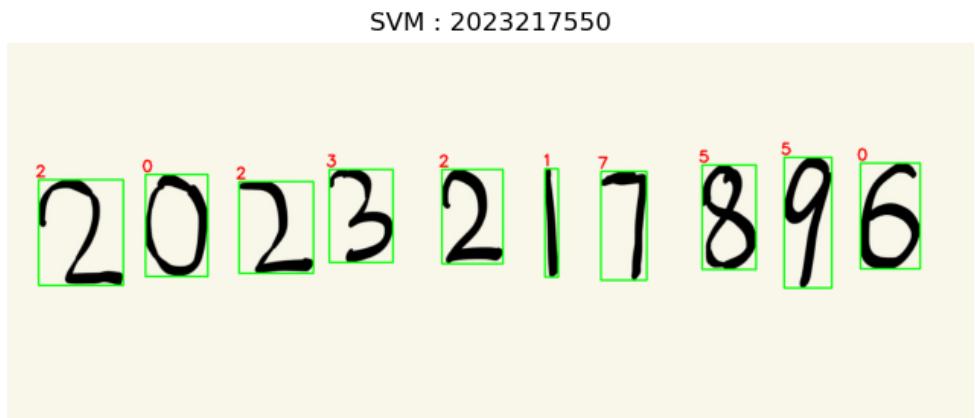
经过 ROI 裁剪后的图片

3.2.6: 双模型预测、按位置排序、拼接学号与可视化对比

对每个数字 ROI 分别用 SVM 与 CNN 预测，并按从左到右顺序拼接为最终学号字符串，同时可视化检查分割与识别是否正确。

- **SVM 预测:** 将 canvas 展平为 784 并归一化: $\text{canvas.reshape}(1,784)/255.0$, 调用 svm_model.predict 得到数字; 保存为 (pred, x) 。
- **CNN 预测:** 对 canvas 做与训练一致的 $\text{ToTensor} + \text{Normalize}$ ，并 $\text{unsqueeze}(0)$ 增加 batch 维; $\text{cnn_model.eval()} + \text{no_grad}()$ 前向，取 argmax 得到数字; 同样保存 (pred, x) 。

- **顺序保证（关键）：**对预测列表按 x 坐标升序排序（轮廓返回顺序不可靠），再将数字依次拼接成字符串得到 student_id_svm 与 student_id_cnn。
- **可视化：**在原图上绘制矩形框并标注预测值（SVM/CNN 不同颜色），并并排展示，便于定位错误来源（分割错还是分类错）。



四、我对模型的改进

4. 1 数字黏连（两个数字贴在一起/轮廓连成一块）

我的解决方案：

A. 预处理层面：让“连接处断开”

1. 强更强的二值化与去噪

- 试试：自适应阈值参数调小/调大 (blockSize、C)，或改用 Otsu (光照较均匀时更稳)。
- 加一步中值滤波 (medianBlur) 对椒盐噪声更友好。

2. 形态学“开运算”断开细连接

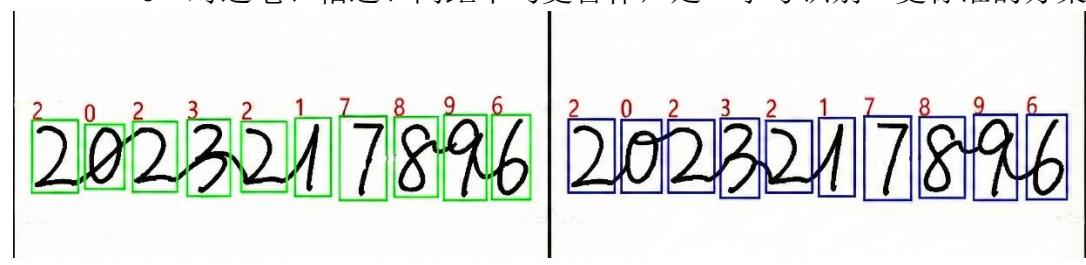
- 思路：黏连往往靠很细的桥连接 (笔画或墨迹)，用 erode → dilate 能断开连接。
- 核建议：cv2.getStructuringElement(MORPH_RECT, (2, 2)) 或 (3, 3)，迭代 1 次起步。

- 注意：核太大/迭代太多会把“1”这类细笔画腐蚀没。
3. 距离变换 + 分水岭（适合黏连严重）
- 对二值前景做 distance transform，找到两个数字的“峰”，再 watershed 分割成两个连通块。
 - 这是处理“粘连目标分割”的经典方法，报告里写会很加分。
- B. 分割层面：把一个大 ROI 切成两个 ROI
- 垂直投影切割（最适合数字水平排列的学号）
 - 对黏连 ROI 做列方向投影：每一列统计白像素数。
 - 找到投影的“低谷”作为切割线（通常是两个数字之间的空隙或最窄连接处）。
 - 轮廓内部分析：找凹陷点切割
 - 用 convexity defects（凸包缺陷）找轮廓凹陷位置，凹陷常对应两个数字的分界处。

4. 2 连笔/连写（数字之间有拖尾或一笔带过，导致分割不干净）

我的解决方案：

- A. 把“拖尾/连笔线”削弱或去掉
- 细化/骨架化 + 去分支（结构化处理）
 - 先做 skeleton（骨架化），再检测细长分支（拖尾通常是细长、端点明显的分支），剪掉长度小于阈值的分支。
 - 方向性腐蚀（专门打断“横向连笔”）
 - 连笔在学号中常是横向把相邻数字连起来。
 - 用“横向结构元”的腐蚀能优先削弱横向细线：
 - 例如核：(5, 1) 或 (7, 1) 做一次 erode，再适当 dilate 恢复主体。
 - 连通域过滤：按形状/面积/长宽比去掉拖尾碎片
 - 连笔线经开运算后可能变成小碎片，用面积阈值、长宽比阈值丢弃。
 - 适合“拖尾较细、易被处理成小连通域”的情况。
- B. 从“分割-识别”升级为“检测/序列识别”（根治思路）
- 用检测模型先定位每个数字（不靠轮廓）
 - 连笔时轮廓会粘连，传统 findContours 很容易失败。
 - 更稳做法：训练/使用数字检测器（例如 YOLO 小模型）输出每个数字框，再分类。
 - 端到端序列模型（CRNN/Transformer + CTC）
 - 不做字符分割，直接输入整行学号图，输出数字序列。
 - 对连笔、粘连、间距不均更鲁棒，是“学号识别”更标准的方案。



经过改进的识别效果展示

五、心得体会

做完这次实验之后，我感觉收获最大的不是“跑出来多少准确率”，而是终于把“从训练到真实图片识别”这条链路真正走通了一遍。刚开始我以为只要在 MNIST 上把模型训练好，拿去识别学号图片就会很顺利，**但实际做下来发现：MNIST 上准确率高不代表真实场景就能直接用，真实图片里各种问题（光照、背景、笔画粗细、数字挤在一起）都会让识别效果大幅波动。**

在对比 SVM 和 CNN 的过程中，我更直观地理解了两种方法的差别。SVM 需要把 28×28 展平成 784 维，再做归一化，这种做法对输入形式特别敏感：只要数字位置偏一点、笔画粗一点，或者分割出来的 ROI 大小不一致，识别就容易出错。而 CNN 因为有卷积和池化，会自动学一些“笔画结构”的特征，整体上更稳一些，但也不是随便丢进去就行，训练和推理时的预处理要保持一致，比如 Normalize 的参数要一样、推理要用 eval/no_grad，否则结果也会飘。

我觉得这次实验里最关键、最值得写清楚的一步就是 **ROI 归一化到 28×28** 。一开始我没太在意，后来发现这一步几乎决定了“能不能识别”。因为模型是在 MNIST 这种“数字居中、大小相对固定”的数据上训练的，**但从学号照片里切出来的数字大小和位置都不一样，如果直接塞给模型就会出现很明显的分布不一致**。我的做法是把 ROI 先等比例缩放，让最大边接近 20，再居中贴到 28×28 画布里，这样看起来就更像 MNIST，识别效果也明显更稳定。

另外一个我一开始容易忽略的点是“多位数字的顺序”。轮廓提取出来的顺序并不一定是从左到右的，所以必须按每个框的 x 坐标排序后再拼接学号，不然就算每一位都识别对了，拼出来的字符串也可能是错的。把结果画框标注出来也很有用，因为一旦识别错，我能很快判断到底是“分割错了”（框就不对）还是“分类错了”（框对但数字预测错）。

在真实学号图片上，**我感觉最难的就是“数字黏连”和“连笔”。数字一旦挤得太紧或者写得连在一起，轮廓就会把两位数字当成一个整体，后面再怎么分类也没用，因为输入就已经错了**。针对黏连，我觉得更有效的思路是先在预处理里想办法把连接处断开，比如用形态学开运算或调整阈值；如果还是不行，就对大 ROI 再做一次切割（比如用垂直投影找分割线）。对于连笔拖尾，我理解就是数字之间有细线连接，比较适合用“方向性腐蚀”之类的方法把细线削弱掉，保

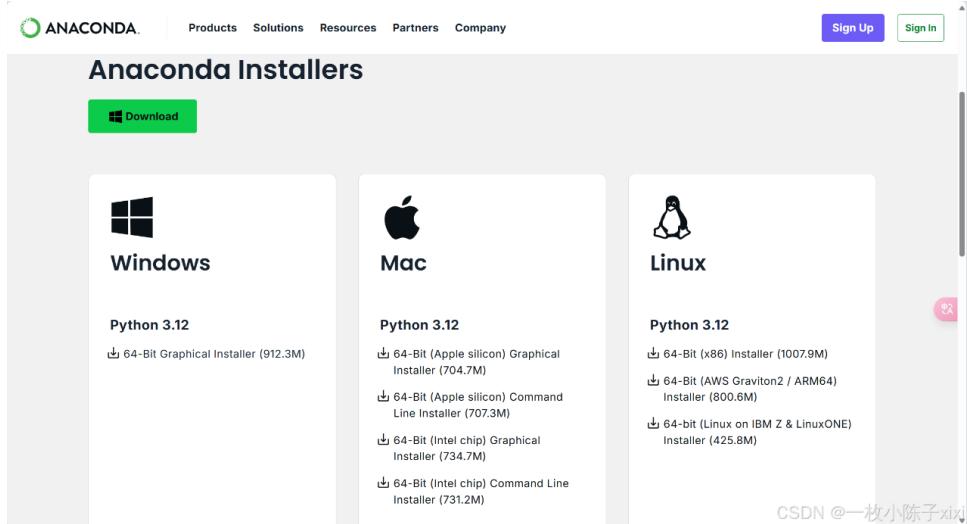
留主体笔画。

总体来说，这次实验让我明白：一个能用的识别系统，不是只看模型，而是每一步都要稳定——从预处理、分割、ROI 归一化到排序拼接，任何一步出问题都会把最终结果带偏。后续如果继续优化，我会优先把分割这块做得更鲁棒（黏连/连笔的处理更自动化），同时也考虑加入一些数据增强或者用少量真实学号图片做微调，让模型更贴近真实书写情况。

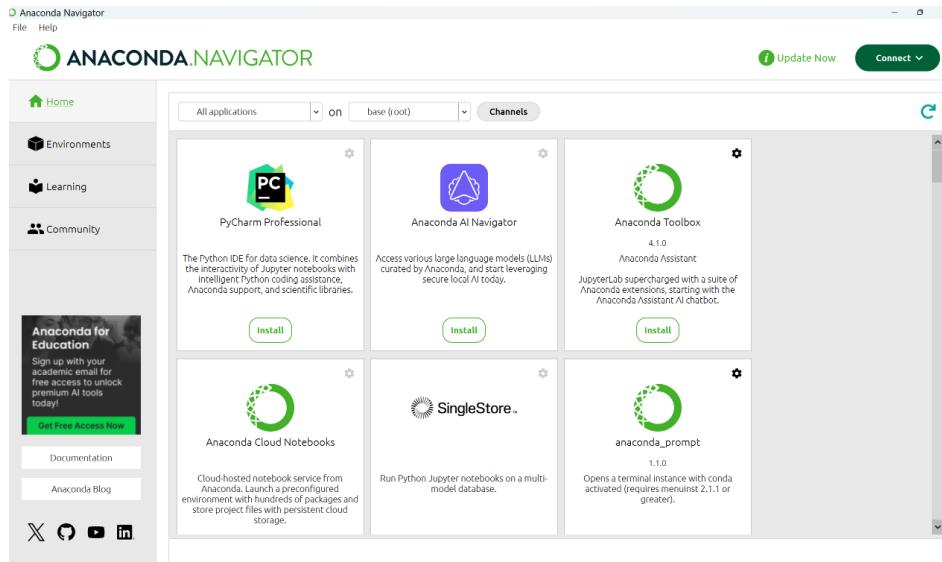
环境配置过程。

anaconda 安装与使用

1、下载安装包



2、打开 Anaconda Navigator



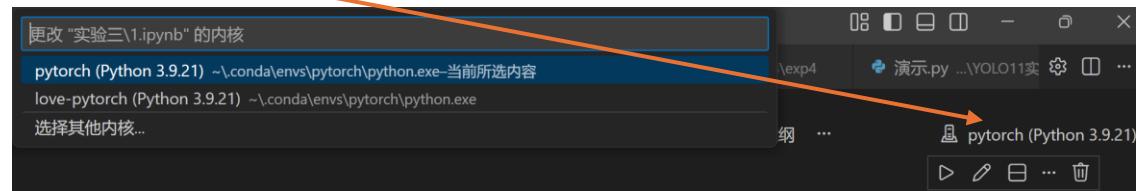
3、创建环境

The screenshot shows the Anaconda Navigator application. On the left, there's a sidebar with environment names: 'base (root)', 'jqxx', and 'pytorch' (which is selected and highlighted with a green border). Below the sidebar are buttons for 'Create', 'Clone', 'Import', 'Backup', and 'Remove'. The main area is titled 'Channels' and shows a list of packages. A search bar labeled 'Search Packages' is located at the top right of this area. An orange arrow points from the text above to this search bar.

Name	Description	Version
_py-xgboost-mutex	Scalable, portable and distributed gradient boosting (gbdt, gbrt or gbm) library, for python, r, java, scala, c++ and more, runs on single machine, hadoop, spark, link and dataflow	2.0
anyio	High level compatibility layer for multiple asynchronous event loop implementations on python	4.9.0
aom	Alliance for open media video codec	3.12.1
argon2-cffi	The secure argon2 password hashing algorithm.	23.1.0
argon2-cffi-bindings	Low-level python ffi bindings for argon2	21.2.0
arrow	Better dates & times for python	1.3.0
asttokens	The asttokens module annotates python abstract syntax trees (asts) with the positions of tokens and text in the source code that generated them.	3.0.0
async-lru	Simple lru_cache for asyncio	2.0.5
attrs	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	25.3.0
autograd	Efficiently computes derivatives of numpy code.	1.7.0
autograd-gamma	Autograd compatible approximations to the derivatives of the gamma-family of functions.	0.5.0
babel	Utilities to internationalize and localize python applications	2.17.0

这里我创建的是名字为 pytorch 的环境，并且可以在右上角搜索要下载的包，如果需要下载什么包可以自由下。

4、打开 vscode 并且切换环境



5、可以在 jupyter 里敲代码了