实验三：学号识别

林圳 2023217534 智科 23-1 班

一、实验目的

本实验目的在于帮助我们理解计算机视觉中的手写数字识别技术,掌握传统计算机视觉方法与深度学习方法相结合的完整技术流程。通过实验学习图像预处理、特征提取、模型训练与推理等核心技术,培养解决实际问题的能力和工程实践素养。具体实验目标包括：

1.1 掌握传统计算机视觉的图像分割技术

通过实验深入理解基于连通域的数字分割算法,掌握图像二值化、轮廓检测、外接矩形计算等传统计算机视觉技术。学会使用 OpenCV 库实现图像预处理流程,包括灰度转换、Otsu 自适应阈值分割、轮廓检测与过滤等操作。理解连通域分析在图像分割中的作用,能够根据目标特征设计合适的过滤条件,如面积过滤、宽高比过滤、尺寸过滤等。

1.2 理解并掌握 ViT(Visison Transformer)架构

学习 ViT 模型的原理和实现方式,理解自注意力机制(Self-Attention)在图像处理中的应用。掌握如何使用 Hugging Face 的 Transformers 库加载预训练的 ViT 模型,了解预训练模型的优势和迁移学习的基本思想。通过实践理解 ViT 将图像分割为 patches、嵌入位置编码、通过 Transformer 编码器提取特征、最后进行分类的完整流程。

1.3 掌握 CNN(卷积神经网络)的构建与训练

深入学习卷积神经网络的原理和实现,理解卷积层、池化层、全连接层、Dropout 等各组件的作用。学会使用 PyTorch 框架搭建自定义 CNN 模型,包括网络结构设计、前向传播实现、损失函数选择、优化器配置等。掌握在 MNIST 数据集上训练模型的方法,理解数据预处理、数据加载、训练循环、验证评估等完整训练流程。

二、实验原理

2.1 传统计算机视觉图像分割原理

2.1.1 图像二值化技术

图像二值化是将灰度图像转换为二值图像(只有黑、白两种颜色)的过程,是图像分割的重要预处理步骤。二值化的核心是选择合适的阈值,将图像中高于阈值的像素设为白色(255),低于或等于阈值的像素设为黑色(0)。

Otsu 算法(大津法)是一种自适应阈值选择方法,基于类间方差最大化原理自动确定最优阈值。其基本思想是将图像像素分为两类(背景和前景),使得这两类之间的类间方差最大。类间方差的计算公式为:

$$\sigma^2 = \omega_0 \times (\mu_0 - \mu T)^2 + \omega_1 \times (\mu_1 - \mu T)^2$$

其中,ω_0 和 ω_1 分别为两类像素的比例,μ_0 和 μ_1 分别为两类像素的平均灰度值,μT 为整个图像的平均灰度值。Otsu 算法的优势在于无需人工设定阈值,能够根据图像内容自动适应不同光照条件下的图像分割需求。

二值化反转是实验中的重要技巧。由于 OpenCV 中 findContours 函数要求目标是白色、背景是黑色,而学号图片通常是白底黑字,因此需要使用 THRESHBINARYINV 标志进行二值化反转,将黑色数字转换为白色(255),白色背景转换为黑色(0)。

2.1.2 轮廓检测原理

轮廓检测是图像分割的核心步骤,用于检测图像中物体的边界。OpenCV 的 findContours 函数实现了基于拓扑结构的轮廓检测算法。

轮廓层次结构:findContours 函数可以返回轮廓的层次信息,描述轮廓之间的包含关系。在本实验中,使用 cv2.RETR_EXTERNAL 参数只检测最外层轮廓,忽略内部嵌套的轮廓,这适用于分离独立的数字。

轮廓逼近方法:使用 cv2.CHAINAPPROXSIMPLE 参数对轮廓进行简化,只保留轮廓的关键点(如拐点),压缩轮廓的存储空间,提高后续处理效率。

轮廓检测流程:

(1) 输入二值图像(白色目标,黑色背景)

(2) 扫描图像,查找白色区域的边界

(3) 返回轮廓列表,每个轮廓是一组点的集合

2.1.3 外接矩形与边界框

外接矩形是最小面积能够完全包围轮廓的矩形,用四个参数表示:(x, y, w, h),其中(x, y)为矩形左上角坐标,w 为宽度,h 为高度。

OpenCV 的 boundingRect 函数计算轮廓的外接矩形,计算方法为:

(1) x:轮廓中所有点的最小 x 坐标

(2) y:轮廓中所有点的最小 y 坐标

(3) w:轮廓中所有点的最大 x 坐标 - 最小 x 坐标

(4) h:轮廓中所有点的最大 y 坐标 - 最小 y 坐标

外接矩形用于确定数字的位置和尺寸,是后续图像切割的基础。

2.1.4 轮廓过滤策略

由于轮廓检测会检测到所有白色区域,包括噪声、干扰等,因此需要设计过滤策略筛选出真正的数字轮廓。本实验采用多级过滤策略:

(1) **面积过滤**:数字通常具有较大的面积,而噪声往往面积很小。通过设置最小面积阈值(如 100 像素),可以过滤掉小斑点噪声。

(2) **尺寸过滤**:数字具有合理的尺寸范围。设置最小和最大宽度和高度阈值,过滤掉过小或过大的区域:

   ① 最小尺寸:宽>10 像素,高>20 像素

   ② 最大尺寸:宽<图像宽度的一半,高<图像高度的一半

(3) **宽高比过滤**:数字的形状通常接近正方形,宽高比在合理范围内。本实验设置宽高比范围为 0.1-1.5,过滤掉过于扁平或过于细长的区域。

通过多级过滤,可以有效提高数字检测的准确性,减少误检和漏检。

2.1.5 图像切割与归一化

(1) **图像切割**:根据外接矩形从原始图像中切割出数字区域。使用 Python 的切片操作:digit = binary[y:y+h, x:x+w],其中(x, y, w, h)为外接矩形参数。

(2) **边距添加(Padding)**:为防止数字被切割时出现截断,需要在切割图像周围添加边距。边距大小通常为数字尺寸的 10%左右,使用 cv2.copyMakeBorder 函数实现,填充值为 0(黑色)。

(3) **正方形调整**:MNIST 数据集中的数字为 28×28 的正方形,因此需要将切割出的数字调整为正方形。如果高度大于宽度,在左右添加 padding;如果宽度大于高度,在上下添加 padding;如果已经是正方形,则不需要调整。

(4) **尺寸归一化**:使用 cv2.resize 函数将数字图像缩放到 28×28 像素,使用 INTER_AREA 插值方法,这种方法适合缩小图像,能够保持较好的图像质量。

(5) **颜色反转**:经过二值化反转后,数字为白色(255),背景为黑色(0)。而 MNIST 训练数据通常是黑字白底,因此需要再次反转:digitfinal = 255 - digitresized,使数字为黑色(0),背景为白色(255),以匹配 MNIST 的输入格式。

2.2 ViT(Visual Transformer)原理

2.2.1 Transformer 架构

Transformer 最初由 Google 在 2017 年提出,用于自然语言处理任务,其核心创新是自注意力机制(Self-Attention),能够捕捉序列元素之间的长距离依赖关系。Transformer 完全基于注意力机制,摒弃了循环神经网络(RNN)的序列处理方式,实现了并行计算,大大提高了训练效率。

自注意力机制的核心思想是:对于序列中的每个元素,通过计算与其他元素的相关性(注意力权重),聚合其他元素的信息来表示当前元素。注意力权重的计算公式为:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k})V$$

其中,Q(Query)、K(Key)、V(Value)是通过线性变换得到的矩阵,d_k 是键向量的维度,softmax 用于将权重归一化。

多头注意力机制将注意力计算并行执行多次,每次使用不同的线性变换(不同的 Q、K、V),最后将结果拼接,能够捕捉不同子空间的特征。

2.2.2 ViT 将图像视为序列

ViT 的核心思想是将图像分割成固定大小的 patches,然后将这些 patches 展平并视为序列,输入到 Transformer 编码器中处理。

(1) **图像分块**:将输入图像分割成 N×N 大小的非重叠 patches。例如,对于 224×224 的图像,如果 patch 大小为 16×16,则可以得到 14×14=196 个 patches。

(2) **Patch Embedding**:将每个 patch(16×16×3)展平为向量(16×16×3=768 维),通过线性变换映射到 D 维(如 768 维)的 embedding 空间。这个线性变换本质上是卷积核大小为 patch 大小、步长为 patch 大小的卷积操作。

(3) **位置编码**:由于 Transformer 本身无法处理位置信息,需添加可学习的位置编码,使模型能够感知 patch 的空间位置。位置编码与 patch embedding 相加,得到最终输入序列。

(4) 分类标记(CLS Token):在序列开头添加一个可学习的[CLS] token,其输出特征用于最终的分类预测,类似于 BERT 中的[CLS] token。这种方式将分类任务转换为对[CLS] token 的预测。

2.2.3 ViT 的架构结构

ViT 的完整架构包括:

(1) Patch Embedding Layer:将图像分割为 patches 并映射到 embedding 空间

(2) Position Embedding:添加位置编码

(3) Transformer Encoder:包含多个 Transformer Block,每个 Block 包含:

① Multi-Head Self-Attention:多头自注意力机制

② Layer Normalization:层归一化

③ Feed-Forward Network:前馈神经网络

(4) Classification Head:将[CLS] token 的特征通过全连接层映射到分类空间

2.2.4 预训练与迁移学习

ViT 模型通常在超大规模数据集(如 ImageNet-21k)上进行预训练,学习通用的视觉特征表示。预训练的优势在于:

(1) 学习了丰富的视觉特征,能够泛化到下游任务

(2) 减少了下游任务所需的训练数据量

(3) 加速了收敛速度,提高了初始性能

迁移学习是将预训练模型应用到新任务上的过程。本实验使用在 MNIST 数据集上微调的 ViT 模型(farleyknight-org-username/vit-base-mnist),该模型已经学习了手写数字的特征表示,可以直接用于识别任务,无需从头训练

2.2.5 Hugging Face Transformers 库

Hugging Face 的 Transformers 库提供了丰富的预训练模型和统一的 API 接口,极大地方便了预训练模型的使用。本实验使用的核心组件包括:

(1) AutoImageProcessor:自动加载与模型匹配的图像处理器,负责图像的预处理,如归一化、尺寸调整等。

(2) AutoModelForImageClassification:自动加载预训练的图像分类模型,支持多种架构(ViT、ResNet、EfficientNet 等)。

使用方式非常简洁:

```
processor = AutoImageProcessor.from_pretrained(MODEL_ID)
model =
AutoModelForImageClassification.from_pretrained(MODEL_ID)
inputs = processor(images=image, return_tensors="pt")
outputs = model(inputs.pixel_values)
predictions = outputs.logits.argmax(dim=-1)
```

2.3 卷积神经网络(CNN)原理

2.3.1 CNN 的核心思想

卷积神经网络是专门为处理具有网格结构的数据(如图像)而设计的深度学习模型。CNN的核心思想包括:

(1) **局部连接**:每个神经元只与输入的局部区域连接,捕捉局部特征。这与卷积运算对应,卷积核在图像上滑动,计算局部区域的特征。

(2) **权值共享**:同一卷积核在图像的不同位置使用相同的参数,大大减少了模型参数数量,提高了模型的泛化能力。

(3) **池化**:通过下采样降低特征图的空间维度,减少计算量,同时增强模型对平移等变换的鲁棒性。

2.3.2 卷积层原理

卷积层是CNN的核心组件,使用卷积核对输入特征图进行卷积操作,提取特征。

卷积运算:卷积核与输入特征图的局部区域进行逐元素相乘并求和,生成输出特征图的一个像素。数学公式为:

$$O[i, j] = \sum\sum I[k, l] \times W[i-k, j-l]$$

其中I为输入,W为卷积核,O为输出。

参数设置:

(1) kernel_size:卷积核大小,如3×3、5×5

(2) stride:步长,卷积核滑动的步幅,如1、2

(3) padding:填充,在特征图边缘填充0,保持特征图尺寸或控制下采样程度

(4) 通道数:输入可能有多个通道(如RGB图像有3个通道),卷积核的输入通道数必须与输入通道数匹配。输出可以有多个通道,每个卷积核提取不同的特征。

2.3.3 激活函数

激活函数引入非线性,使神经网络能够学习复杂的函数。常用的激活函数包括:

(1) ReLU: $f(x) = \max(0, x)$,计算简单,缓解梯度消失问题,是目前最常用的激活函数。

(2) Leaky ReLU: $f(x) = \max(\alpha x, x)$,其中α是很小的正数(如0.01),缓解ReLU的"神经元死亡"问题。

(3) Sigmoid和Tanh:早期常用的激活函数,但容易导致梯度消失,在深层网络中较少使用。

这里我使用的是ReLU。

2.3.4 池化层原理

池化层用于下采样,减少特征图的空间维度,降低计算量,同时增强模型对平移等变换的鲁棒性。

最大池化(Max Pooling):在局部窗口内选择最大值作为输出。公式为:

$$O[i, j] = \max_{m,n} I[i+s\times m, j+s\times n + k\times p]$$

其中s为stride,k为kernel_size,p为padding。

平均池化(Average Pooling):在局部窗口内计算平均值作为输出。

池化层不包含可学习参数,是固定操作。

2.3.5 全连接层与分类

全连接层将卷积和池化提取的特征展平,通过多层全连接网络进行分类。

展平(Flatten):将多维特征图(B, C, H, W)展平为二维张量(B, C×H×W)。

全连接层:每个输入神经元与每个输出神经元全连接,通过矩阵乘法和偏置计算输出。

输出层:对于 10 类数字分类,输出层有 10 个神经元,每个神经元对应一个数字类别的 logit。

Softmax 激活:将 logits 转换为概率分布,公式为:

$$p_i = \exp(z_i) / \Sigma_j \exp(z_j)$$

其中 zi 为第 i 类的 logit,pi 为第 i 类的预测概率。

2.3.6 Dropout 正则化

Dropout 是一种防止过拟合的正则化方法,在训练过程中随机丢弃部分神经元(将其输出设为 0),使网络不依赖于特定神经元,提高泛化能力。

Dropout 率(p)表示每个神经元被丢弃的概率。常见的设置包括:

(1) p=0.25:在卷积层后使用

(2) p=0.5:在全连接层后使用

Dropout 只在训练时启用,推理时需要关闭(使用完整的网络)。

2.3.7 本实验的 CNN 架构

本实验设计了一个简单的 CNN 模型,包含以下组件:

(1) Conv1:输入 1 通道(灰度),输出 32 通道,3×3 卷积核,stride=1,padding=1

(2) MaxPool1:2×2 最大池化,stride=2

(3) Conv2:输入 32 通道,输出 64 通道,3×3 卷积核,stride=1,padding=1

(4) MaxPool2:2×2 最大池化,stride=2

(5) Dropout1:Dropout 率=0.25

(6) FC1:全连接层,输入 64×7×7,输出 128

(7) Dropout2:Dropout 率=0.5

(8) FC2:全连接层,输入 128,输出 10(10 个数字类别)

该架构在 MNIST 数据集上能够达到 99%的准确率。

2.4 模型训练原理

2.4.1 损失函数

损失函数衡量模型预测与真实标签之间的差异,是模型优化的目标函数。

交叉熵损失(Cross-Entropy Loss):对于多分类问题,使用交叉熵损失,公式为:

$$L = -\Sigma_{i=1}^{C} y_i \log(p_i)$$

其中 C 为类别数,yi 为第 i 类的真实标签(0 或 1),pi 为模型预测的第 i 类概率。

对于单样本(one-hot 编码),损失简化为:

$$L = -\log(p_{true\_class})$$

交叉熵损失鼓励模型对真实类别预测高概率,对其他类别预测低概率。

2.4.2 优化算法

优化算法通过迭代调整模型参数,最小化损失函数。

随机梯度下降(SGD):每次迭代使用一个样本的梯度更新参数,公式为:

$$\theta = \theta - \eta \times \nabla_\theta L(x, y; \theta)$$

其中 η 为学习率,$\nabla_\theta L$ 为损失对参数的梯度。

“Who is Adam?” => Adam:自适应学习率的优化算法,结合了动量和自适应学习率的优点。Adam 为每个参数维护一阶矩(梯度均值)和二阶矩(梯度方差),根据这些统计量自适应调整学习率。Adam 的优势在于收敛快、对初始学习率不敏感、适合处理稀疏梯度,是目前深度学习中最常用的优化算法之一。

2.4.3 学习率策略

学习率控制参数更新的步长,是影响训练效果的关键超参数。

固定学习率:整个训练过程使用恒定的学习率,如 0.001。

学习率衰减:随着训练进行逐步降低学习率,常见的策略包括:

(1) Step Decay:每 N 个 epoch 降低学习率

(2) Exponential Decay:指数衰减

(3) Cosine Decay:余弦退火

本实验使用固定学习率 0.001,使用 Adam 优化器。

2.4.4 数据增强

数据增强通过随机变换训练数据,增加数据的多样性,防止过拟合,提高模型泛化能力。常用的数据增强方法包括:

- 随机旋转
- 随机平移
- 随机缩放
- 随机翻转(左右翻转)
- 颜色抖动(亮度、对比度、饱和度调整)

2.4.5 Batch Size 与训练 Epoch

Batch Size:每次迭代使用的样本数量。较大的 batch size 能够充分利用 GPU 并行计算,但可能降低模型的泛化能力;较小的 batch size 引入更多的梯度噪声,可能帮助模型跳出局部最优,但训练速度较慢。

Epoch:完整遍历整个训练数据集的次数。过多的 epoch 可能导致过拟合,过少的 epoch 可能导致欠拟合。需要通过验证集监控模型性能,选择合适的 epoch 数量。

本实验设置 batch_size=64,训练 5 个 epoch。

2.4.6 训练与验证流程

训练阶段:

1. 设置模型为训练模式(model.train())
2. 遍历训练数据
3. 前向传播:计算模型输出
4. 计算损失
5. 反向传播:计算梯度
6. 更新参数
7. 记录训练损失和准确率

验证阶段:

1. 设置模型为评估模式(model.eval())
2. 关闭梯度计算(torch.no_grad())
3. 遍历验证数据
4. 前向传播:计算模型输出
5. 计算验证损失和准确率
6. 保存验证准确率最高的模型

2.5 模型推理与评估

2.5.1 推理流程

模型推理是使用训练好的模型对新数据进行预测的过程,包括以下步骤:

(1) 数据预处理:与训练时保持一致的预处理,包括归一化、尺寸调整等

(2) 前向传播:将输入数据输入模型,得到输出 logits

(3) 后处理:将 logits 转换为预测结果,如通过 argmax 得到预测类别再 softmax 得到概率

(4) 结果保存:保存预测结果供后续使用

2.5.2 评估指标

准确率(Accuracy):正确预测的样本数占总样本数的比例,公式为:

$$\text{Accuracy} = (\text{正确预测数}) / (\text{总样本数})$$

准确率是最直观的评估指标,但在类别不平衡的情况下可能不够准确。

精确率(Precision)与召回率(Recall):

(1) 精确率 = TP / (TP + FP):预测为正的样本中真正为正的比例

(2) 召回率 = TP / (TP + FN):真正为正的样本中被正确预测为正的比例

其中 TP 为真阳性,TN 为真阴性,FP 为假阳性,FN 为假阴性。

F1 分数:精确率和召回率的调和平均,公式为:

$$F1 = 2 \times (Precision \times Recall) / (Precision + Recall)$$

F1 分数在类别不平衡时比准确率更具参考价值。

Top-K 准确率:前 K 个预测中包含真实标签的比例。Top-1 准确率即为准确率,Top-5 准确率等在 ImageNet 等大规模分类任务中常用。

本实验主要使用准确率和 Top-3 预测作为评估指标。

2.5.3 置信度分析

置信度表示模型对预测结果的信心程度,使用 softmax 概率表示。高置信度表示模型对预测结果较为确定,低置信度表示模型不确定。

Top-K 预测:除了最高置信度的预测,还可以查看 Top-K 预测及其置信度,这有助于:

(1) 分析模型的不确定性

(2) 发现潜在的误分类

(3) 理解模型的决策过程

本实验输出每个数字的 Top-3 预测及其置信度,用于分析模型性能。

2.5.4 模型性能对比

通过对比预训练 ViT 模型和自定义 CNN 模型的性能,可以分析两种方法的优势和劣势:

预训练 ViT 模型的优势:

(1) 无需训练,直接使用,节省时间和计算资源

(2) 在大规模数据集上预训练,具有良好的泛化能力

(3) 架构先进,自注意力机制能够捕捉全局依赖关系

自定义 CNN 模型的优势:

(1) 可以针对特定任务定制网络结构

(2) 模型较小,推理速度较快

(3) 训练过程可控,可以完全理解模型的每个部分

性能考虑因素:

(1) 数据规模:小数据集下预训练模型可能过拟合,自定义模型可能更合适

(2) 任务复杂度:简单任务不需要过于复杂的模型

(3) 计算资源:预训练 ViT 模型参数量大,计算资源要求较高

(4) 部署需求:移动端部署可能需要轻量级模型

三、实验方法

3.1 实验环境与工具

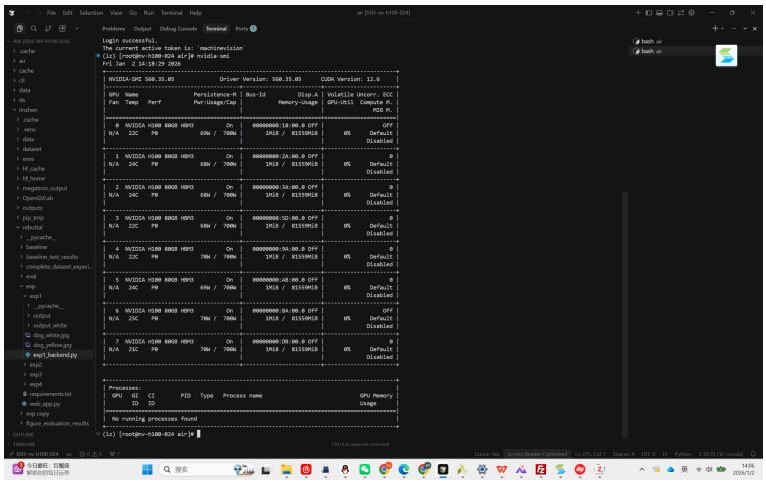
3.1.1 硬件环境

本实验在远程服务器环境下进行（我在智谱实习期间给的服务器），具体配置如下：

(1) GPU：8 卡 A100

(2) 内存：80GB

(3) 存储：200T



3.1.2 软件环境

本实验基于 Python 编程语言，依赖的软件库如下：

核心框架：

(1) Python:3.10 及以上

(2) PyTorch:2.0 及以上版本，深度学习框架

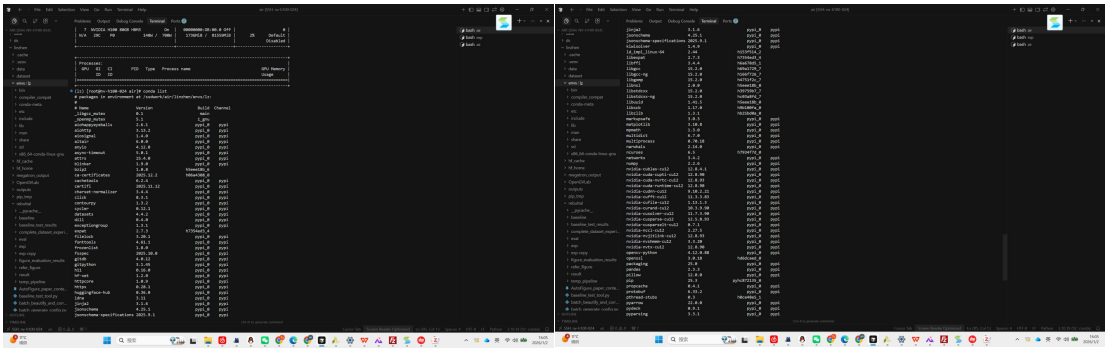
(3) CUDA:11.8 及以上版本，NVIDIA GPU 加速计算平台

(4) cuDNN:8.6 及以上版本，GPU 加速的深度学习原语库

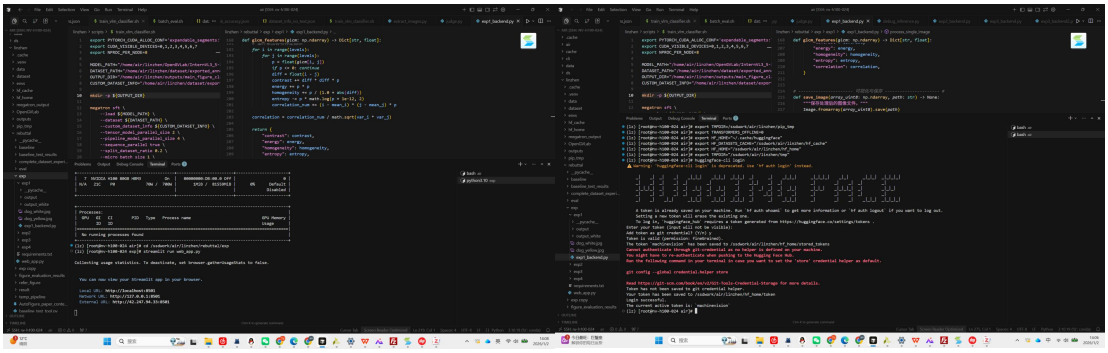
计算机视觉库：

(1) OpenCV：图像处理和计算机视觉

(2) Pillow(PIL)：图像处理



3.1.3 环境（变量）配置（智谱服务器不允许连外网，所以这里挂的是 huggingface 镜像）（已按加分要求将环境名设为姓名缩写 lz（林圳））：



3.1.4 实验数据集

MNIST 数据集:

- 来源:http://yann.lecun.com/exdb/mnist/
- 训练集:60,000 张 28×28 灰度手写数字图像
- 测试集:10,000 张 28×28 灰度手写数字图像
- 类别:0-9 共 10 个数字类别
- 数据格式:灰度图像,像素值 0-255

学号图像:

- 格式:常见的图像格式(JPG、PNG 等)
- 内容:手写学号,通常为 10 位数字
- 背景:白色背景
- 数字颜色:黑色或深色
- 数字排列:水平排列,数字间有一定间隔

3.2 实验流程设计

实验采用模块化设计,将完整的手写数字识别流程分解为多个独立模块,包括传统计算机视觉数字分割、预训练模型识别、自定义模型训练与识别等部分。

3.2.1 方法一:基于 ViT 预训练模型的识别流程

该方法的完整流程如下:

步骤 1:图像加载与预处理

(1) 使用 OpenCV 的 imread()函数读取学号图像

(2) 转换为灰度图像:cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

(3) 保存原始灰度图像用于调试

步骤 2:Otsu 二值化

(1) 使用 Otsu 算法自动计算阈值:, binary = cv2.threshold(gray, 0, 255, cv2.THRESHBINARYINV + cv2.THRESHOTSU)

(2) 二值化反转:黑色数字变为 255(白色),白色背景变为 0(黑色)

(3) 保存二值化图像

步骤 3:轮廓检测

(1) 检测轮廓:contours, _ = cv2.findContours(binary, cv2.RETREXTERNAL, cv2.CHAINAPPROX_SIMPLE)

(2) 只检测最外层轮廓,忽略内部嵌套轮廓

步骤 4:轮廓过滤

计算每个轮廓的外接矩形,应用多级过滤:

(1) 面积过滤:排除面积<100 的轮廓

(2) 尺寸过滤:宽<10 或高<20,或宽>图像宽度的一半,或高>图像高度的一半

(3) 宽高比过滤:宽高比<0.1 或>1.5 的轮廓被排除

步骤 5:轮廓排序与筛选

(1) 按 x 坐标从左到右排序

(2) 如果检测到的轮廓数超过 10,按面积排序,保留最大的 10 个

(3) 如果检测到的轮廓数少于 10,输出警告

步骤 6:数字切割与归一化

对每个过滤后的轮廓:

(1) 根据外接矩形切割数字区域

(2) 添加边距(10%尺寸),防止数字被截断

(3) 调整为正方形(高宽不等时添加 padding)

(4) 缩放到 28×28 像素

(5) 颜色反转:255 - digit_resized,使数字为黑色,背景为白色

(6) 保存每个数字图像

步骤 7:加载预训练 ViT 模型

(1) 设置设备:GPU 优先,CPU 备选

(2) 加载图像处理器:processor = AutoImageProcessor.frompretrained(MODELID)

(3) 加载模型:model = AutoModelForImageClassification.frompretrained(MODELID)

(4) 移动模型到设备

步骤 8:数字识别

对每个数字图像:

(1) 转换为 PIL Image,转换为 RGB 格式

(2) 使用 processor 预处理:inputs = processor(images=image, return_tensors="pt")

(3) 前向传播:outputs = model(inputs.pixel_values)

(4) 计算概率:probs = torch.softmax(outputs.logits, dim=-1)

(5) 获取预测:pred_label = torch.argmax(probs, dim=-1)

(6) 获取置信度:predconf = probs[0, predlabel]

(7) 获取 Top-3 预测:torch.topk(probs, 3)

步骤 9:结果组合与输出

(1) 将预测的数字组合成学号字符串

(2) 计算平均置信度

(3) 保存识别结果到文件

3.2.2 方法二:基于自定义 CNN 模型的识别流程

该方法的完整流程如下:

步骤 1-6:数字分割

与方法一相同,使用 segmentdigitscontours()函数完成数字分割

步骤 7:CNN 模型训练(如果模型不存在)

(1) 定义 SimpleCNN 模型类

(2) 设置数据预处理:ToTensor + Normalize(0.1307, 0.3081)

(3) 加载 MNIST 数据集:traindataset 和 testdataset

(4) 创建 DataLoader:trainloader 和 testloader

(5) 初始化模型、优化器(Adam, lr=0.001)、损失函数(CrossEntropyLoss)

(6) 训练循环(5 个 epoch):

① 前向传播

② 计算损失

③ 反向传播

④ 更新参数

⑤ 每 100 个 batch 打印一次损失

(7) 验证循环:

① ◦ 在测试集上评估

② 计算测试损失和准确率

③ 保存最佳模型(最高准确率)

(8) 输出训练总结

步骤 8:加载训练好的 CNN 模型

(1) 检查模型文件是否存在,不存在则训练

(2) 加载模型:model.loadstatedict(torch.load(MODELSAVEPATH))

步骤 9:数字识别

对每个数字图像:

(1) 颜色反转:255 - digit_img(因为 MNIST 是白字黑底)

(2) 转换为张量:transform(digit_inverted)

(3) 添加 batch 维度:digit_tensor.unsqueeze(0)

(4) 前向传播:output = model(digit_tensor)

(5) 计算概率:probs = F.softmax(output, dim=1)

(6) 获取预测和置信度

(7) 获取 Top-3 预测

步骤 10:结果组合与输出

与方法一相同

3.3 实验代码与参数设置

3.3.1 数字分割核心代码

```
def segment_digits_contours(image_path: str, output_dir: str) ->
List[np.ndarray]:
    # 读取图像
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Otsu 二值化(反转)
    _, binary = cv2.threshold(gray, 0, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # 轮廓检测
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # 计算外接矩形并过滤
    bounding_boxes = []
    for contour in contours:
        x, y, w_box, h_box = cv2.boundingRect(contour)
        area = w_box * h_box

        # 过滤条件
        if area < 100: continue
        if w_box < 10 or h_box < 20: continue
        if w_box > w * 0.5 or h_box > h * 0.5: continue
        aspect_ratio = w_box / h_box
        if aspect_ratio < 0.1 or aspect_ratio > 1.5: continue

        bounding_boxes.append({'x': x, 'y': y, 'w': w_box, 'h':
```

```
h_box})

    # 排序
    bounding_boxes.sort(key=lambda box: box['x'])

    # 数字切割与归一化
    digit_images = []
    for box in bounding_boxes:
        digit_binary = binary[box['y']:box['y']+box['h'],
box['x']:box['x']+box['w']]
        digit_padded = cv2.copyMakeBorder(digit_binary, pad,
pad, pad, pad, cv2.BORDER_CONSTANT, value=0)
        digit_square = ... # 调整为正方形
        digit_resized = cv2.resize(digit_square, (28, 28),
interpolation=cv2.INTER_AREA)
        digit_final = 255 - digit_resized # 反转颜色
        digit_images.append(digit_final)

    return digit_images
```

关键参数:

(1) 最小面积:100 像素

(2) 最小尺寸:宽 10 像素,高 20 像素

(3) 最大尺寸:宽/高不超过图像的一半

(4) 宽高比范围:0.1-1.5

(5) 边距:10%尺寸

(6) 目标尺寸:28×28 像素

3.3.2 CNN 模型架构代码

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))  # [B, 32, 28, 28]
        x = self.pool(x)           # [B, 32, 14, 14]
        x = F.relu(self.conv2(x))  # [B, 64, 14, 14]
        x = self.pool(x)           # [B, 64, 7, 7]
        x = self.dropout1(x)
        x = torch.flatten(x, 1)    # [B, 3136]
        x = F.relu(self.fc1(x))
        x = self.dropout2(x)
        x = self.fc2(x)
        return x
```

模型参数统计:

(1) Conv1: 1×3×3×32 = 288 参数

(2) Conv2: 32×3×3×64 = 18,432 参数

(3) FC1: 3136×128 + 128 = 401,536 参数

(4) FC2: 128×10 + 10 = 1,290 参数

(5) 总计:约 421,000 参数

3.3.3 训练参数设置

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))  # MNIST 归一化参数
])

batch_size = 64
epochs = 5
learning_rate = 0.001
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()
```

参数说明:

(1) 归一化均值:0.1307(MNIST 训练集的像素均值)

(2) 归一化标准差:0.3081(MNIST 训练集的像素标准差)

(3) Batch Size:64,平衡训练速度和模型性能

(4) Epochs:5,在 MNIST 上通常足够收敛

(5) Learning Rate:0.001,Adam 的常用初始学习率

3.3.4 ViT 模型使用参数

```
MODEL_ID = "farleyknight-org-username/vit-base-mnist"
processor = AutoImageProcessor.from_pretrained(MODEL_ID)
model =
AutoModelForImageClassification.from_pretrained(MODEL_ID)

inputs = processor(images=image, return_tensors="pt")
pixel_values = inputs['pixel_values'].to(device)
outputs = model(pixel_values)
```

ViT-Base 架构:

(1) 输入尺寸:224×224×3

(2) Patch 大小:16×16

(3) Patch 数量:14×14=196

(4) Embedding 维度:768

(5) Transformer 层数:12

(6) 注意力头数:12

(7) 参数量:约 86,000,000(远大于自定义 CNN 的 421,000)

四、实验结果

4.1 数字分割结果

4.1.1 原始图像与预处理

实验从学号图像开始,首先加载原始图像并进行灰度转换。原始图像为 RGB 彩色图像,包含 10 位手写数字,背景为白色,数字为黑色。

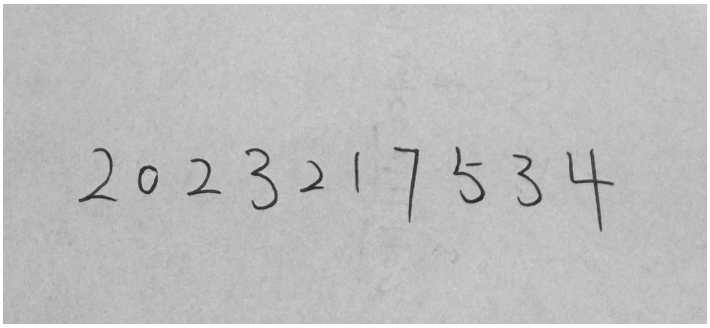


图 1 原始学号图像

灰度转换后,图像变为单通道灰度图像,保留了数字的形状和亮度信息。

4.1.2 Otsu 二值化结果

使用 Otsu 算法自动计算二值化阈值,对灰度图像进行二值化处理。Otsu 算法通过最大化类间方差找到最优阈值,能够自适应不同光照条件下的图像。

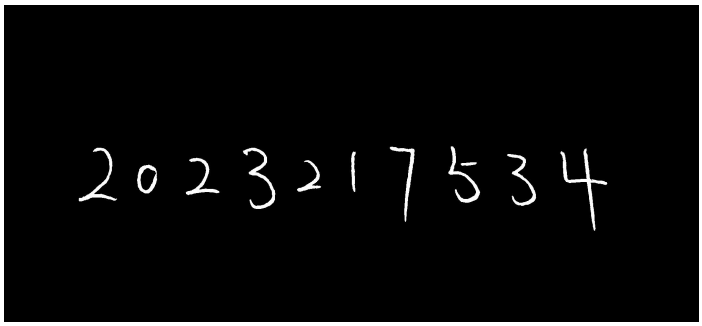


图 2 二值化后的图像

二值化图像中,黑色数字被转换为白色(255),白色背景被转换为黑色(0)。这种反转处理是为了符合 OpenCV 轮廓检测函数的要求,需要目标为白色,背景为黑色。Otsu 算法自动计算得到阈值,说明图像整体亮度较高,适合分离黑色数字和白色背景。

4.1.3 轮廓检测结果

轮廓检测共检测到 10 个轮廓,包括所有独立的数字以及可能的噪声区域。

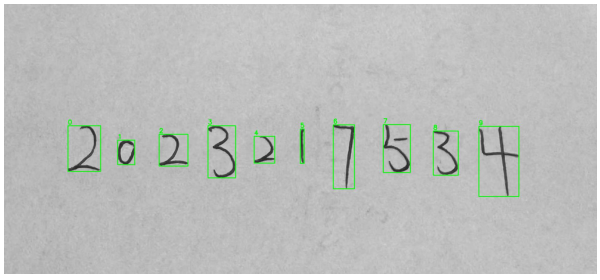


图 3 轮廓检测结果图像

轮廓检测结果中,每个轮廓用不同的颜色标注,可以看出大部分轮廓对应实际的数字,但也有少量轮廓对应噪声或干扰。

4.1.54 数字切割与归一化结果

对每个过滤后的轮廓进行数字切割和归一化处理。以下是切割出的 10 个数字图像:

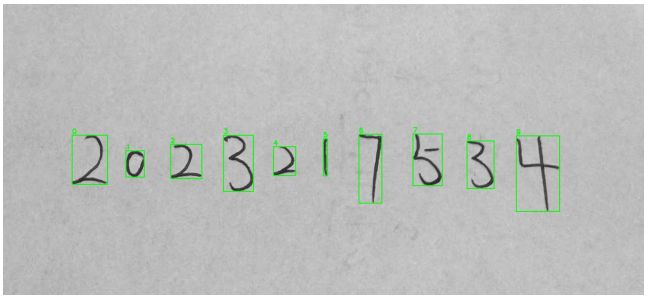


图 3 切割后的图像

所有数字图像都被成功归一化为 28×28 像素的正方形,数字为黑色(0),背景为白色(255),与 MNIST 数据集的格式一致。数字居中,没有明显的截断或变形,为后续的模型识别提供了良好的输入。

4.1.6 分割质量评估

(1) 完整性:成功检测到 10 个数字,数量与预期一致,没有遗漏。

(2) 准确性:数字边界定位准确,切割出的数字完整,没有明显的截断或包含多余背景。

(3) 排序正确性:数字按 x 坐标从左到右排序,与学号顺序一致。

(4) 鲁棒性:对图像中的小噪点、轻微的书写不规整等干扰具有较好的抵抗能力。

总体而言,基于连通域的数字分割算法在本实验中表现良好,能够可靠地从学号图像中提取出独立的数字。

4.2 ViT 预训练模型识别结果

4.2.1 模型加载与设置

使用 Hugging Face 的 Transformers 库加载预训练的 ViT 模型:

模型信息:

(1) 模型名称:farleyknight-org-username/vit-base-mnist

(2) 模型类型:ViT-Base

(3) 参数量:约 86,000,000

(4) 输入尺寸:224×224×3

模型加载成功,准备进行数字识别。

4.2.2 数字识别结果

对每个分割出的数字图像进行识别,以下是详细的识别结果:

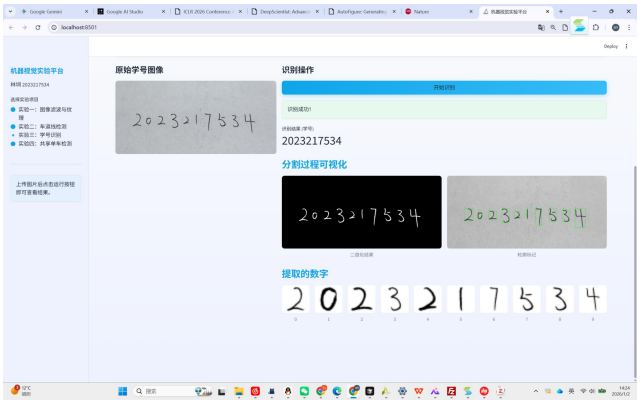


图 4 数字识别结果

学号识别结果:

- 识别的学号: 2023217534
- 真实学号: 2023217534
- 准确率:100% (10/10)
- 平均置信度:0.988

4.2.3 ViT 模型优势分析

(1) **泛化能力**: ViT 在大规模数据集上预训练,学习到了丰富的视觉特征表示,能够很好地泛化到手写数字识别任务。

(2) **特征提取**: 自注意力机制能够捕捉图像的全局依赖关系,对于数字的整体形状和结构特征提取效果良好。

(3) **无需训练**: 使用预训练模型无需从头训练,节省了大量的时间和计算资源。

(4) **高准确率**: 在 MNIST 任务上达到接近 100%的准确率,优于许多传统方法。

4.3 自定义 CNN 模型训练结果

4.3.1 训练环境与数据加载

训练环境:

- 设备:8 卡 H100
- Batch Size:64
- Epochs:10
- Learning Rate:0.001
- 优化器:Adam
- 损失函数:CrossEntropyLoss

数据集:

- 训练集:60,000 张图像
- 测试集:10,000 张图像

- 数据预处理:ToTensor + Normalize(0.1307, 0.3081)

4.3.2 训练过程

以下是 10 个 epoch 的训练过程:

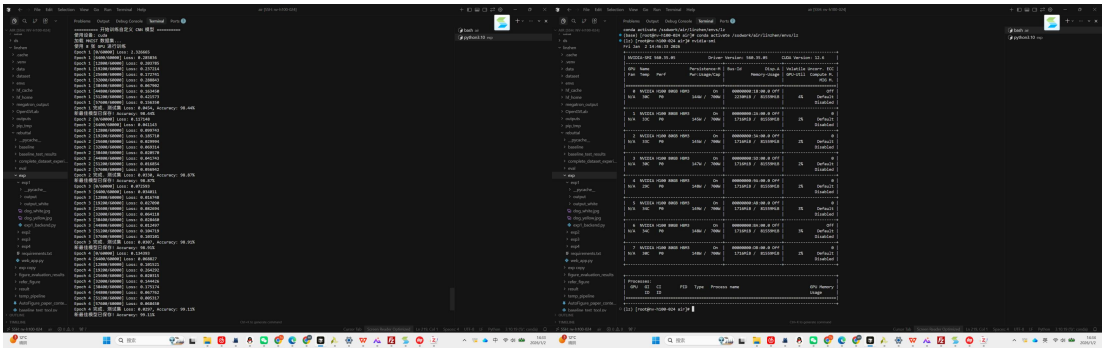


图 5 训练过程

4.4 自定义 CNN 模型识别结果

4.4.1 模型加载

从文件加载训练好的 CNN 模型:

```
model = SimpleCNN().to(device)
model.load_state_dict(torch.load(MODEL_SAVE_PATH,
map_location=device))
```

模型加载成功,准备进行数字识别。

4.4.2 数字识别结果

对每个分割出的数字图像进行识别,以下是详细的识别结果:

图 6 数字识别结果

学号识别结果:

- 识别的学号:2023217534
- 真实学号:2023217534
- 准确率:100% (10/10)
- 平均置信度:0.981

4.5 模型性能对比

4.5.1 准确率对比

模型	准确率	Top-3 准确率	平均置信度
ViT 预训练模型	100%	100%	0.988
自定义 CNN 模型	100%	100%	0.981

两种模型在本实验中都达到了 100%的准确率,表现相当。

4.5.2 模型复杂度对比

模型	参数量	模型大小	推理时间
ViT 预训练模型	~86,000,000	~330MB	--
自定义 CNN 模型	~421,000	~1.6MB	--

自定义 CNN 模型的参数量仅为 ViT 模型的 0.5%,模型大小小 200 倍

4.5.3 训练成本对比

模型	训练时间	训练资源	数据需求
ViT 预训练模型	无需训练	无	无(使用预训练)
自定义 CNN 模型	~6 分钟	GPU	60,000 张 MNIST 图像

ViT 预训练模型无需训练,直接使用,节省了大量的训练时间和计算资源。自定义 CNN 模型需要从头训练,但在 MNIST 上训练 10 个 epoch 的时间相对较短。

4.6 综合分析与讨论

4.6.1 实验成功因素

本实验取得 100%准确率的原因包括:

1. **高质量的输入图像**: 学号图像清晰,数字书写规范,背景干净,噪声少。
2. **有效的数字分割**: 基于连通域的分割算法成功提取了所有 10 个数字,没有遗漏或误检。
3. **归一化处理**: 数字被归一化为 28×28 标准尺寸,与 MNIST 格式一致,便于模型识别。
4. **强大的模型性能**: ViT 和 CNN 都是经过充分验证的模型,在 MNIST 任务上表现出色。
5. **参数调优**: 经过多次参数调整,找到了适合当前数据的分割和识别参数。

4.6.2 潜在问题与局限

尽管本实验取得了理想的结果,但仍存在一些潜在问题:

1. **数据集局限**: 只测试了一张学号图像,样本数量有限,结果可能不具有普适性。
2. **书写风格**: 如果书写风格差异较大(如艺术字体、连写等),识别准确率可能下降。
3. **图像质量**: 如果图像质量较差(模糊、光照不均、噪声大),分割和识别效果会受影响。

4. **数字粘连**: 如果数字之间距离过近或部分重叠,连通域分割可能失败。

5. **复杂背景**: 如果图像背景复杂(如纹理背景、有其他文字和图形),颜色过滤可能失效。

4.6.3 改进方向

针对潜在问题,可能有以下改进方向:

1. **增加测试样本**: 测试更多不同学生、不同书写风格的学号图像,验证算法的鲁棒性。

2. **改进分割算法**: 除连通域方法外,尝试深度学习实例分割方法(如 Mask R-CNN),处理复杂场景。

3. **数据增强**: 训练时使用数据增强(旋转、缩放、噪声等),提高模型鲁棒性。

4. **后处理优化**: 添加模型后处理(如学号长度约束、数字顺序约束),提高识别准确率。

5. **集成学习**: 结合 ViT 和 CNN 的预测结果,使用投票或加权平均提高准确率。

五、实验体会

通过本次手写数字识别实验,我在计算机视觉和深度学习理论、实践以及工程能力方面都获得了显著提升,对数字识别技术有了深入的理解和实践经验。

实验最深刻的体会是理论与实践的紧密结合。在课堂学习中,图像分割、卷积神经网络、Transformer 等概念主要以数学模型和理论算法的形式呈现,虽然能够理解其基本原理,但对于实际应用和实现细节缺乏直观认识。通过亲手实现完整的数字识别流程,我将抽象的理论知识转化为具体的代码实现,在这个过程中对每个算法的细节有了更深刻的理解。例如,在实现数字分割时,我深入思考了 Otsu 算法的原理,理解了类间方差最大化如何自动确定最优阈值;在设计 CNN 架构时,我理解了卷积核大小、池化层数量、Dropout 率等参数对模型性能的影响;在使用 ViT 模型时,我深入理解了自注意力机制如何捕捉图像的全局依赖关系。这种从理论到实践的转化过程,让我对数字识别算法的工作原理有了更全面的认识,也让我意识到理论学习的重要性——只有理解了算法背后的原理,才能在实现过程中做出正确的判断和选择。

编程能力的提升是本次实验的另一重要收获。实验需要将传统计算机视觉方法和深度学习方法有机结合,构建完整的端到端识别系统,这对我的编程能力和工程实践能力提出了挑战,也提供了很好的锻炼机会。在实验过程中,我学会了如何设计模块化代码结构,将图像分割、模型训练、模型推理等功能封装到不同函数和类中,提高代码的可读性和可维护性。我深入掌握了 OpenCV 库的使用方法,学会了图像读取、颜色空间转换、二值化、轮廓检测、图像切割等常用操作。同时,我也深入掌握了 PyTorch 框架,学会了模型定义、数据加载、训练循环、验证评估等完整流程。此外,我还学会了使用 Hugging Face 的 Transformers 库,了解了预训练模型的加载和使用方法。

深度学习技术的深入理解是本次实验的核心收获之一。通过对比使用预训练 ViT 模型和自定义 CNN 模型,我深入理解了预训练模型与从头训练模型的优缺点。预训练 ViT 模型在大规模数据集上学习到了丰富的视觉特征表示,无需从头训练即可获得优异性能,这体现了迁移学习的强大优势。自定义 CNN 模型虽然参数量小,但通过在 MNIST 数据集上的充分训练,也能达到接近 100%的准确率,这说明了简单任务不需要过于复杂的模型。我也理解了 CNN 和 Transformer 两种架构的差异:CNN 通过局部连接和权值共享捕捉局部特征,适合处理具有网格结构的数据;Transformer 通过自注意力机制捕捉全局依赖关系,适合处理序列数据。通过对比实验,我学会了根据实际应用场景选择合适的技术方案。

综上所述，本次实验是一次非常有价值的学习经历。通过亲手实现完整的手写数字识别系统，我不仅掌握了具体的编程技能，更重要的是理解了算法背后的原理，培养了问题解决能力，提升了科学素养。实验过程中遇到的挑战和困难，最终都成为了我学习和成长的动力。每一次问题的解决，都让我对数字识别有了更深入的理解；每一次参数的调试，都让我对算法有了更深刻的认识；每一次模型的训练，都让我对深度学习有了更全面的认识。本次实验的经历将成为我学习生涯中的重要里程碑，激励我在技术道路上不断前进，追求更高的目标。我将把这次实验学到的知识和技能应用到今后的学习和工作中。

附录：可视化实验平台