第 8 周：手写简答题识别

能看懂学生写的答案吗?

北京石油化工学院
人工智能研究院
王文通

北京石油化工学院
BEIJING INSTITUTE OF PETROCHEMICAL TECHNOLOGY

北京石油化工学院
人工智能研究院

2025-2026 学年

课程概览

本周内容：

- 手写识别概述
- PaddleOCR 手写识别
- TrOCR 高精度识别
- 图像预处理优化
- 思考题与作业

学期项目：AI 阅卷助手

1. 图像采集与预处理
2. 答题卡定位（Timing Marks）
3. 填涂检测与识别
4. 手写文字 OCR（本周重点）
5. 成绩统计与输出

本周时间分配（160 分钟 = 3 学时）

第 1 学时（45 分钟）：

00:15 手写识别概述（15min）

00:35 PaddleOCR 手写识别（20min）

00:45 讨论与答疑（10min）

第 2 学时（45 分钟）：

01:15 TrOCR 高精度识别（30min）

01:30 图像预处理优化（15min）

第 3 学时（70 分钟）：

01:30-02:00 代码实战：手写识别（30min）

02:00-02:20 参数调优实践（20min）

02:20-02:35 互动测验（15min）

02:35-02:45 总结与作业（10min）

时间控制提示

如果进度落后，建议跳过”挑战任务”

预备知识（课前 5 分钟视频）

视频 1：神经网络基础（2 分钟）

神经元模型：

- 模拟人脑神经元
- 输入 → 加权和 → 激活 → 输出
- 可学习的权重和偏置

神经网络层次：

- 输入层：接收数据
- 隐藏层：特征提取
- 输出层：产生结果

视频 2：卷积运算直观理解（3 分钟）

什么是卷积?

- 滑动窗口提取特征
- 卷积核（滤波器）
- 特征图（Feature Map）

卷积在手写识别中的作用：

- 提取笔画特征
- 捕获局部结构
- 平移不变性

常见卷积操作：

- Valid 卷积：尺寸减小
- Same 卷积：保持尺寸

本周分组策略

分组原则：

- 每 4 人为一组
- 确保不同专业背景混合
- 建议包含：理工科、文科、无编程基础、有编程基础

角色分工：

角色	职责	适合
组长	统筹协调、进度管理	组织能力强的
算法实现者	实现 OCR 代码、模型调用	有编程基础的
参数调优者	调整识别参数、优化效果	细心负责的
测试者	收集测试用例、报告问题	细心负责的

本周协作任务

本周在智能阅卷系统中的位置

已完成模块：

1. 图像预处理（第 3 周）
2. 版面分析（第 4 周）
3. 选择题识别（第 5 周）
4. 判断题识别（第 6 周）
5. 印刷文字识别（第 7 周）

本周目标：

- 识别手写简答答案
- 对比 PaddleOCR vs TrOCR
- 优化预处理参数

智能阅卷流程：

从”印刷文字”到”手写文字”的跨越

手写识别在智能阅卷中的作用

多屏协同设计

本课程采用多屏协同教学方式：

主屏（左侧）：理论讲解

- PPT 幻灯片
- 概念和原理讲解
- 图像和代码展示
- 互动测验

侧屏（右侧）：实时演示

- OCR 代码实时演示
- 手写识别效果展示
- 参数调整实时反馈
- 调试过程展示

移动设备互动

使用手机参与互动测验（问卷星）

手写识别的挑战

印刷文字：

- 字体统一
- 字形稳定
- 间距规范

手写文字：

- 每人不同
- 形状变化大
- 间距随意

主要难点：

1. 字形差异大
2. 连笔问题
3. 倾斜旋转
4. 图像质量

脱机识别 vs 联机识别

特性	脱机识别 (Offline HWR)	联机识别 (Online HWR)
输入方式	静态图像（扫描/拍照）	动态笔画序列（触摸屏/数位板）
可用信息	仅图像像素	笔画轨迹、速度、压力、时间
难度	更难（信息损失）	相对容易
应用场景	文档数字化、阅卷系统	手写输入、签名验证
典型方法	CNN+RNN+CTC	LSTM、Transformer

课程重点

本课程重点讲解**脱机手写识别**，用于智能阅卷系统中的简答题识别。

手写识别在智能阅卷中的重要性

传统阅卷痛点：

- 人工阅卷效率低
- 主观评分不一致
- 教师工作负担重
- 成绩统计耗时

AI 阅卷优势：

- 秒级识别手写答案
- 客观题自动评分
- 简答题辅助批改
- 自动成绩统计

手写识别的关键作用

1. **答案区域定位**
   - 检测简答题答题框
   - 分割手写文字区域
2. **手写文字识别**
   - 识别学生手写答案
   - 转换为可编辑文本
3. **语义理解辅助评分**
   - 提取关键词
   - 匹配标准答案

手写文字与印刷文字的区别

书写风格差异：

- **印刷文字**：统一字体，字形规范
- **手写文字**：因人而异，风格多样
- 连笔、省略笔画现象普遍

字形变化与连笔问题：

- 同一字的不同写法
- 笔画顺序变化
- 连笔导致字形模糊
- 草书难以辨认

笔画粗细与倾斜：

- **印刷**：笔画宽度一致
- **手写**：笔尖压力变化导致粗细不均
- 整体或局部倾斜
- 基线不水平

字间距与行间距变化：

- 印刷文字间距固定
- 手写间距随意
- 字重叠或分离
- 行间距不均匀

手写简答题识别的难点

1. **自由书写，无固定格式**
   - 答案长度不固定
   - 书写位置不固定
   - 答案组织方式多样
   - 编号、序号使用不一致

2. **混合中英文与公式**
   - 中英混杂
   - 数学公式、符号
   - 特殊字符

3. **涂改与补充痕迹**
   - 划掉的文字
   - 修改痕迹
   - 补充内容
   - 箭头、连线等标记

4. **书写质量差异大**
   - 字迹清晰度不同
   - 笔画完整性差异
   - 字体大小不统一
   - 书写压力不同

5. **背景干扰（格线、印刷文字）**

手写识别技术发展

阶段	技术	代表模型/方法	准确率
传统方法 (1960s-2000s)	特征工程 + 统计分类器	HMM、SVM、KNN	60-70%
CNN 时代 (2012-2015)	卷积神经网络	LeNet, AlexNet, VGG	80-85%
RNN+Attention (2015-2018)	循环神经网络 + 注意力	LSTM+Attention	90-92%
Transformer (2018-至今)	自注意力机制	**TrOCR, Donut, PARSeq**	**95%+**

关键里程碑：

- 1998: LeNet-5 手写数字识别
- 2012: AlexNet 开启深度学习时代

当前最佳：TrOCR（Microsoft）

- 架构：ViT (图像编码器) + GPT-2 文本解码器

主流手写识别工具对比

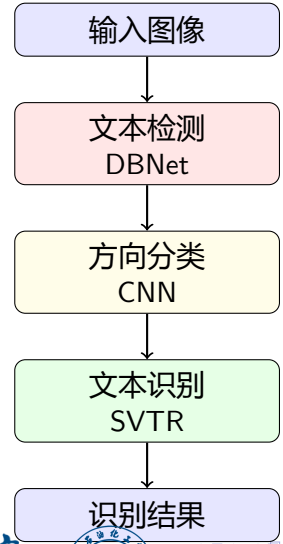
工具	核心技术	优点	缺点	适用场景
TrOCR	ViT + GPT-2 Transformer	准确率最高(95%+)，支持手写印刷混合	速度慢，GPU 需求高	高精度 OCR
PaddleOCR	DB + CRNN + CTC	速度快，中文支持好，轻量级	手写准确率略低(85-90%)	实时 OCR、移动端
Tesseract	LSTM + 传统 CV	免费开源，多语言支持	中文手写支持差	印刷体英文
EasyOCR	CRAFT + Seq2Seq	支持 80+ 语言，易用	中文手写效果一般	多语言通用

选择建议：

PaddleOCR 架构概述

PP-OCRv4 三大核心模块

1. **文本检测** (Text Detection)
   - DBNet 算法
   - 可微分二值化
2. **方向分类** (Orientation Classification)
   - 0/90/180/270 度分类
   - 轻量级 CNN
3. **文本识别** (Text Recognition)
   - SVTR_LCNet 架构
   - CTC 解码



检测模型：DBNet 原理

DBNet (Differentiable Binarization)

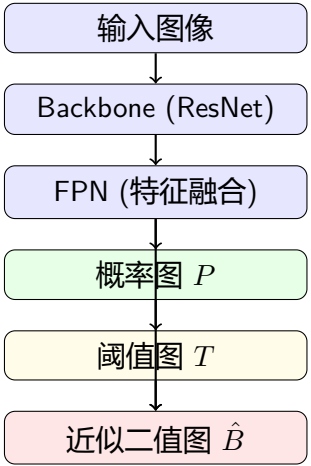
核心创新：可微分二值化

- 传统方法：二值化不可导，无法端到端训练
- DBNet：将二值化近似为可微分函数
- 梯度可直接传播到阈值预测

二值化公式：

$$B_{i,j} = \frac{1}{1 + e^{-k(P_{i,j} - T_{i,j})}}$$

其中：P 为概率图，T 为阈值图，k 为放大因子



识别模型：SVTR_LCNet 原理

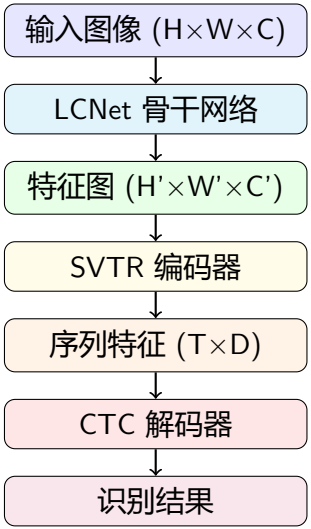
SVTR_LCNet 架构

设计目标：

- 中文场景文字识别
- 支持横排和竖排文本
- 平衡精度与速度

网络结构：

1. **LCNet 骨干网络**
   - 轻量级 CNN 架构
   - 深度可分离卷积
2. **SVTR 编码器**
   - 基于 Sub-Block 的 Transformer
   - 全局特征建模



PaddleOCR 手写配置

```python
from paddleocr import PaddleOCR

# TODO: 使用AI助手补全PaddleOCR初始化
# 提示词: "PaddleOCR手写识别配置参数说明"
ocr_handwrite = PaddleOCR(
    use_angle_cls=______,  # TODO: 是否启用方向分类（True/False）
    lang=______,           # TODO: 语言设置（'ch'/'en'/...）
    show_log=______        # TODO: 是否显示日志（True/False）
)

# TODO: 调用OCR识别方法
# 提示词: "PaddleOCR ocr()方法参数说明"
result = ocr_handwrite.ocr(
    ______,  # TODO: 输入图像路径或数组
    cls=______  # TODO: 是否启用方向分类（True/False）
```

手写识别参数调优

```
# TODO: 使用AI助手补全手写识别优化配置
# 提示词: "PaddleOCR手写识别参数调优指南"
ocr_optimized = PaddleOCR(
    use_angle_cls=______,  # TODO: 是否启用方向分类
    lang=______,           # TODO: 语言设置
    # TODO: 模型路径配置
    det_model_dir=______,  # TODO: 检测模型路径
    rec_model_dir=______,  # TODO: 识别模型路径
    cls_model_dir=______,  # TODO: 方向分类模型路径
    # TODO: 性能参数调优
    rec_batch_num=______,   # TODO: 批量识别数（整数）
    max_text_length=______, # TODO: 最大文本长度（整数）
    drop_score=______,      # TODO: 置信度阈值（0-1浮点数）
    show_log=______         # TODO: 是否显示日志
)
```

多行手写文字识别

```
import cv2
from paddleocr import PaddleOCR

# TODO: 使用AI助手补全图像读取代码
# 提示词: "OpenCV读取图像的常用方法"
image = cv2._____  # TODO: 读取图像文件（如'answer_sheet.jpg'）

# TODO: 初始化OCR
# 提示词: "PaddleOCR初始化参数说明"
ocr = PaddleOCR(
    use_angle_cls=_____,  # TODO: 是否启用方向分类
    lang=_____            # TODO: 语言设置
)

# TODO: 执行OCR识别
```

增强版手写识别（含错误处理）

```python
import os
import cv2
import numpy as np
from paddleocr import PaddleOCR
from typing import List, Dict, Optional, Union

class HandwritingOCR:
    """增强版手写OCR识别器"""

    def __init__(self,
                 use_gpu: bool = False,
                 det_model_dir: Optional[str] = None,
                 rec_model_dir: Optional[str] = None,
                 drop_score: float = 0.3):
        """
```

批量处理与结果可视化

```
    def batch_recognize(self,
                        image_paths: List[str],
                        save_results: bool = True,
                        output_dir: str = "./results") -> List[
                            Dict]:
        """
        批量识别手写文字

        Args:
            image_paths: 图像路径列表
            save_results: 是否保存可视化结果
            output_dir: 输出目录

        Returns:
            识别结果列表，每项包含
```

单张图像识别与可视化

```
    def recognize_single(self,
                         image_path: str,
                         save_vis: bool = True,
                         output_dir: str = "./results") -> Dict:
        """
        单张图像手写文字识别

        Args:
            image_path: 图像路径
            save_vis: 是否保存可视化结果
            output_dir: 输出目录

        Returns:
            识别结果字典
        """
```

结果可视化方法

```
    def _visualize_result(self,
                          image: np.ndarray,
                          boxes: List[List],
                          texts: List[str],
                          confidences: List[float],
                          box_color: Tuple[int, int, int] = (0,
                              255, 0),
                          text_color: Tuple[int, int, int] = (255,
                              0, 0)) -> np.ndarray:
        """
        可视化识别结果

        Args:
            image: 原始图像
            boxes: 边界框列表
```

TrOCR 简介

TrOCR： Transformer-based OCR

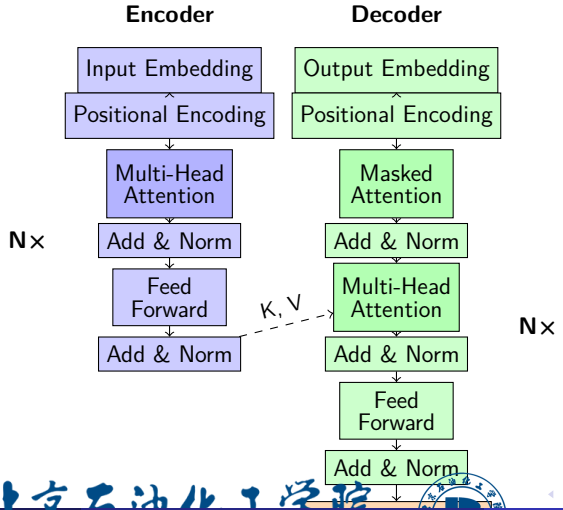
- 开发者：Microsoft
- 架构：ViT (图像编码器) + GPT-2 (文本解码器)
- 优势：端到端训练，准确率最高

模型：

- trocr-base-handwritten
- trocr-large-handwritten

Transformer 架构概述

Transformer 是一种完全基于注意力机制的序列转换模型



自注意力机制原理

自注意力（Self-Attention）：计算序列中每个位置与其他所有位置的关联强度

核心公式：

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

Query (Q)：查询向量
Key (K)：键向量
Value (V)：值向量

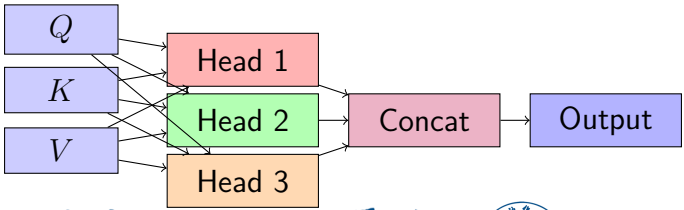
- QK^T：计算相似度
- $\sqrt{d_k}$：缩放因子
- softmax：归一化为概率
- 加权求和得到输出

多头注意力机制

多头注意力（Multi-Head Attention）：并行计算多组自注意力，捕获不同子空间的信息

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

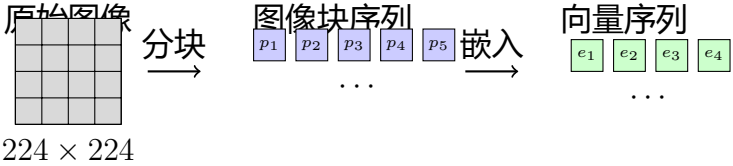
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Vision Transformer (ViT) 架构

ViT 将 Transformer 应用于图像分类，TrOCR 使用 ViT 作为图像编码器

核心思想：将图像分割成固定大小的块（Patches），将每个块视为一个" 词"



ViT 编码器结构详解

ViT 编码器的主要组件：

1. **图像分块（Patch Embedding）**

$$x_p = \text{Flatten}(x_p^{(H)} \times x_p^{(W)}) \cdot E$$

其中 $E \in \mathbb{R}^{(P^2 \cdot C) \times D}$ 是可学习的嵌入矩阵

2. **类别 Token（[CLS] Token）**

$$z_0 = [x_{\text{class}}; x_p^1 E; x_p^2 E; \cdots; x_p^N E] + E_{\text{pos}}$$

3. **Transformer 编码器层**

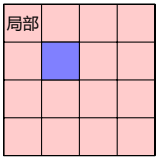
$$z'_\ell = \text{MSA}(\text{LN}(z_{\ell-1})) + z_{\ell-1}$$

$$z_\ell = \text{MLP}(\text{LN}(z'_\ell)) + z'_\ell$$

ViT 与 CNN 编码器对比

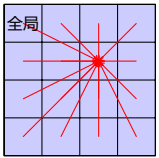
CNN 编码器

- 局部感受野
- 平移等变性
- 归纳偏置强
- 需要大量卷积层提取全局信息
- 计算效率高



ViT 编码器

- 全局感受野
- 自注意力机制
- 归纳偏置弱
- 直接建模全局关系
- 数据需求大



GPT-2 解码器架构

GPT-2 (Generative Pre-trained Transformer 2) 作为 TrOCR 的文本解码器

核心特点：

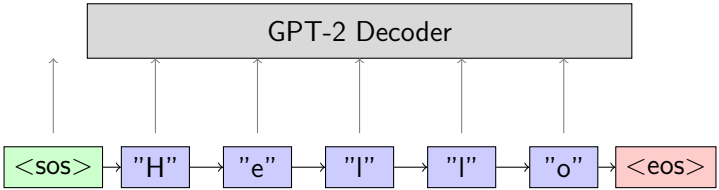
- **仅解码器结构：**使用 Masked Self-Attention，只能看到当前位置之前的信息
- **自回归生成：**逐个字符预测，每个新字符依赖于已生成的序列
- **位置编码：**使用可学习的位置嵌入

解码过程：

$$
\begin{aligned}
h_0 &= W_e \cdot x + W_p \\
h_l &= \text{TransformerBlock}(h_{l-1}), \quad l = 1, \ldots, n \\
P(y_i | y_{<i}) &= \text{softmax}(W_e^T h_n^{(i)})
\end{aligned}
$$

自回归生成与解码策略

自回归生成过程：



每次预测下一个最可能的字符

常用解码策略：

- **贪心解码（Greedy）：**每次选择概率最高的词
- **束搜索（Beam Search）：**保留 top-k 个候选序列
- **采样解码：**按概率分布随机采样，配合温度参数

TrOCR 预训练过程

TrOCR 采用两阶段预训练策略：

阶段一：图像编码器预训练（ViT）

- 数据集：ImageNet-21k
- 任务：图像分类
- 目标：学习通用的视觉特征表示

阶段二：端到端文本识别预训练

- 数据集：
  - 合成数据：Synthetically generated handwritten text (数百 GB)
  - 真实数据：IAM, CVL, RIMES 等手写数据集
- 目标：学习从图像到文本的映射

TrOCR 微调方法

在特定任务上微调 TrOCR 的常用策略：

1. 全模型微调（Full Fine-tuning）

- 解冻所有参数
- 学习率：$5e-6$ 到 $1e-5$
- 适合：数据充足的场景

2. 冻结编码器微调

- 冻结 ViT 编码器
- 只微调 GPT-2 解码器
- 适合：数据有限的场景

3. 增量学习

- 在新数据上继续预训练
- 保持原有知识
- 逐步适应新领域

关键超参数：

- Batch size: 8-32
- Epochs: 5-20
- Learning rate: $1e-6$ 到 $5e-5$
- Warmup steps: 100-500
- Weight decay: 0.01

微调数据集准备

准备用于微调 TrOCR 的数据集：

1. 数据格式

labels.txt
示 $\texttt{image}_0 01.jpg image_0 02.jpg World$ 例

2. 数据增强策略

- 随机旋转（-5° 到 +5°）
- 随机缩放（0.95x 到 1.05x）
- 高斯噪声添加
- 随机亮度/对比度调整

TrOCR 使用

```
# TODO: 使用AI助手补全TrOCR手写识别代码
# 提示词: "使用TrOCR进行手写文字识别的完整流程"
from transformers import TrOCRProcessor,
    VisionEncoderDecoderModel
from PIL import Image

# TODO: 加载TrOCR处理器和模型
# 提示词: "TrOCRProcessor和VisionEncoderDecoderModel的
    from_pretrained用法"
processor = TrOCRProcessor.from_pretrained('microsoft/__________'
    )
model = VisionEncoderDecoderModel.from_pretrained('microsoft/
    __________')

# TODO: 读取图像并进行预处理
```

扩充代码：设备自动切换与优化

```python
# TODO: 使用AI助手补全TrOCRRecognizer类的初始化代码
# 提示词："实现TrOCRRecognizer类的__init__方法，支持自动设备检测和模型加载"
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
from PIL import Image
import time

class TrOCRRecognizer:
    def __init__(self, model_name='microsoft/trocr-base-handwritten', device=None):
        """
        初始化TrOCR识别器
        :param model_name: 模型名称
        :param device: 指定设备 ('cuda', 'cpu', 或 None自动检测)
        """
        # TODO: 实现设备自动检测
        # 提示词："PyTorch自动检测GPU/CPU设备的代码"
        if device is None:
            self.device = torch.device('__________' if torch.cuda.__________() else '__________')
        else:
            self.device = torch.device(device)
        print(f"使用设备: {self.device}")

        # TODO: 加载模型和处理器
        # 提示词："TrOCRProcessor和VisionEncoderDecoderModel的from_pretrained加载"
        print("加载模型中...")
        self.processor = TrOCRProcessor.from_pretrained(__________)
        self.model = VisionEncoderDecoderModel.from_pretrained(__________)
        # TODO: 将模型移动到指定设备并设置为评估模式
        self.model.to(self.device)
```

扩充代码：批处理与置信度评估

```python
def recognize_batch(self, image_paths, batch_size=8):
    """
    批量图像识别，优化GPU利用率
    :param image_paths: 图像路径列表
    :param batch_size: 批处理大小
    :return: 识别结果列表 [(path, text, confidence), ...]
    """
    # TODO: 实现批量图像识别
    # 提示词："使用TrOCR进行批量图像识别的完整流程"
    results = []
    for i in range(0, len(image_paths), batch_size):
        # TODO: 获取当前批次的路径和图像
        batch_paths = image_paths[i:__________]
        batch_images = [Image.open(p).convert('__________') for p in batch_paths]
        # TODO: 批量处理图像
        pixel_values = self.processor(
            images=batch_images,
            return_tensors="__________",
            padding=True
        ).pixel_values.to(self.device)
        # TODO: 生成识别结果并获取分数
        outputs = self.model.generate(
            pixel_values,
            max_length=128,
            num_beams=4,
            output_scores=__________,
            return_dict_in_generate=__________,
            early_stopping=True
        )
```

扩充代码：完整错误处理与使用示例

```python
# TODO: 使用AI助手补全TrOCRRecognizer类的完整实现
# 提示词："设计一个支持错误处理和置信度评估的TrOCR识别类"
def safe_recognize(self, image_path):
    """
    带完整错误处理的识别方法
    :return: (success, result, error_message)
    """
    # TODO: 添加文件存在性检查
    # 提示词："Python检查文件是否存在的方法"
    import os
    if not os.__________(image_path):
        return False, None, f"文件不存在: {image_path}"

    # TODO: 添加文件格式验证
    # 提示词："检查文件扩展名的Python代码"
    valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp', '.tiff')
    if not image_path.lower().__________(valid_extensions):
        return False, None, f"不支持的文件格式"

    # TODO: 实现图像打开和尺寸检查
    # 提示词："PIL Image打开图像并获取尺寸"
    try:
        image = Image.open(image_path)
        width, height = image.size
    except Exception as e:
        return False, None, f"无法打开图像: {str(e)}"

    # TODO: 调用识别方法并返回结构化结果
    # 提示词："调用recognize_with_confidence并封装结果字典"
```

TrOCR 批量识别

```
# TODO: 使用AI助手补全TrOCR批量识别代码
# 提示词: "使用TrOCR进行批量手写文字识别的完整流程"
from transformers import TrOCRProcessor,
    VisionEncoderDecoderModel
from PIL import Image
import torch

# TODO: 加载TrOCR处理器和模型
# 提示词: "TrOCRProcessor和VisionEncoderDecoderModel的
    from_pretrained用法"
processor = TrOCRProcessor.from_pretrained('microsoft/__________'
    )
model = VisionEncoderDecoderModel.from_pretrained('microsoft/
    __________')
```

PaddleOCR vs TrOCR 对比

特性	PaddleOCR	TrOCR
准确率	85-90%	**95%+**
速度	**快**	慢
中文支持	**原生支持**	一般
部署难度	**简单**	复杂
GPU 需求	低	**高**

选择建议：

- 追求准确率 → TrOCR
- 追求速度/易用 → PaddleOCR

手写图像预处理

```python
def preprocess_handwriting(image):
    """手写图像预处理完整流程"""
    # TODO: 使用AI助手补全手写图像预处理代码
    # 提示词: "OpenCV手写图像预处理: 灰度化→去噪→对比度增强→二值
        化"

    # ========== 第一步: 转灰度 ==========
    # TODO: 使用cv2.cvtColor将图像转为灰度图
    # 提示词: "cv2.cvtColor BGR2GRAY转换"
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # ========== 第二步: 去噪 ==========
    # TODO: 使用fastNlMeansDenoising进行非局部均值去噪
    # 提示词: "cv2.fastNlMeansDenoising参数说明和推荐值"
    denoised = cv2.fastNlMeansDenoising(gray, None, 10, 7, 21)
```

文本行检测与分割

水平投影法

水平投影法原理

- 对二值图像按行求和
- 文字区域投影值高
- 空白行投影值接近 0
- 通过阈值分割出行位置

适用场景：

- 工整手写或印刷体
- 水平排列文本
- 行间距较明显

```
对每一行求 h_proj = np.sum(binary_img, axis = 1)
和可视化投 plt.figure(figsize=(10, 5)) plt.subplot(121)
        plt.imshow(binary_img, cmap =
        影'gray')plt.title('BinaryImage')
plt.subplot(122)
        plt.plot(h_proj, range(len(h_proj)))plt.title(影'HorizontalPro
return h_proj
def find_text_lines(h_proj, threshold = 100) : """        """TODO :
        AI                "                "
text_lines = []in_line = Falsestart = 0
for i, proj in
        enumerate(h_proj) : if proj > threshold and not in_line :
          start = i in_line = True elif proj <= threshold and in_line :
          end = i text_lines.append((start, end)) in_line = False
影处理最后一个字 if not
        in_line : text_lines.append((start, len(h_proj)))
return text_lines符
```

文本行检测与分割

连通域分析与深度学习方法

连通域分析法

步骤:

1. 对二值图像进行连通域标记
2. 计算每个连通域的外接矩形
3. 根据垂直位置进行聚类
4. 合并同一行的连通域

优点:

- 能处理倾斜文本
- 对行间距不均匀适应性好
- 可识别弯曲行

基于深度学习的方法

DBNet (Differentiable Binarization)

- 端到端文本检测
- 可检测任意形状文本
- 速度快、精度高

EAST

- 全卷积网络
- 直接预测文本框
- 支持多方向文本

适用场景:

- 复杂背景文档

单字/单词分割

垂直投影与改进方法

垂直投影分割原理

- 对文本行图像按列求和
- 字符区域投影值高
- 字符间隙投影值接近 0
- 通过阈值确定分割点

存在的问题：

- 粘连字符难以分割
- 汉字偏旁可能被分开
- 间距不均影响效果

```python
def vertical_projection_split(line_img,
                              threshold=50):
    """
    垂直投影法分割字符
    """
    # TODO: 使用AI助手补全垂直投影分割代码
    # 提示词："NumPy垂直投影：对每列求和找字符间隙"

    # 对每一列求和
    v_proj = np.sum(line_img, axis=0)

    # 找到投影小于阈值的列（间隙）
    gaps = v_proj < threshold

    # 找出间隙的起止位置
    char_regions = []
    in_gap = False
    start = 0

    for i, is_gap in enumerate(gaps):
        if is_gap and not in_gap:
            # 字符区域结束
            end = i
            char_regions.append((start, end))
            in_gap = True
        elif not is_gap and in_gap:
            # 字符区域开始
```

单字/单词分割

高级分割方法

最小生成树 (MST) 分割
算法步骤：

1. 提取所有连通域
2. 构建连通域之间的距离图
3. 使用 Kruskal 算法构建 MST
4. 切割权重最大的边
5. 得到分割结果

优点：

- 考虑全局信息
- 对粘连字符效果好

基于聚类的分割
K-means 聚类：

- 提取每个连通域特征
- 使用笔画宽度、宽高比等
- 聚类确定字符分组

过度分割与合并策略：
过度分割：

- 先进行细粒度分割
- 保证每个字符都被切开
- 可能产生过多片段

合并策略：

文字规范化

倾斜校正与尺寸归一化

倾斜校正算法

1. **基于投影的校正：**
   - 尝试不同角度旋转
   - 计算水平投影的方差
   - 选择方差最大的角度
2. **基于 Hough 变换：**
   - 检测文本行基线
   - 计算倾斜角度
   - 进行旋转校正
3. **基于主成分分析（PCA）：**

尺寸归一化

为什么要归一化：

- 统一输入尺寸
- 消除大小差异
- 提高模型泛化能力

常用方法：

1. **线性缩放：**
   - 直接缩放到固定尺寸
   - 保持宽高比或拉伸
2. **保持比例填充：**
   - 等比例缩放
   - 不足部分填充背景色

文字规范化

笔画宽度归一化与骨架提取

笔画宽度归一化

目的：

- 消除不同笔迹粗细差异
- 统一笔画宽度便于识别
- 提高模型泛化能力

实现方法：

1. **笔画宽度变换 (SWT)：**
   - 计算每个像素笔画宽度
   - 根据宽度进行归一化

骨架提取

什么是骨架：

- 保持字符拓扑结构的细线
- 笔画中心线
- 宽度为 1 像素的线条

常用算法：

1. **中轴变换 (Medial Axis Transform)**
2. **细化算法 (Thinning)：**
   - Zhang-Suen 算法
   - Guo-Hall 算法

完整预处理流程可视化

```
def visualize_preprocessing(image_path):
    """
    预处理全流程可视化（TODO脚手架版本）
    """
    # TODO: 使用AI助手补全可视化代码
    # 提示词: "matplotlib子图展示图像预处理各阶段结果"

    import matplotlib.pyplot as plt

    # TODO: 读取原图
    original = cv2.imread(_____)

    # 创建子图布局
    fig, axes = plt.subplots(2, 3, figsize=(15, 10))
```

预处理效果对比

预处理步骤	OCR 准确率	速度	推荐
无预处理	65%	快	
仅灰度化	72%	快	
灰度 + 去噪	80%	中等	
完整预处理	**90%+**	慢	

建议：

- 生产环境：使用完整预处理
- 快速测试：至少灰度 + 去噪
- 实时应用：平衡速度与精度

关键参数：

- CLAHE clipLimit: 2.0-4.0
- 去噪强度: h=10-20
- 二值化阈值: OTSU 自适应

课堂思考题

问题 1：手写识别挑战

- 为什么手写识别比印刷识别难？
- 连笔字如何处理？

问题 2：模型选择

- 什么时候用 PaddleOCR？什么时候用 TrOCR？
- 如何平衡准确率和速度？

课堂互动 Quiz - 手写识别技术选择

Quiz 1：单选题 - OCR 工具选择

场景：需要识别一份手写简答题试卷，要求准确率 90% 以上，服务器有 GPU。

最佳选择是?

A. 传统模板匹配方法
B. PaddleOCR + 预处理优化
C. 仅使用 TrOCR，不做预处理
D. 手工录入所有答案

正确答案：B

解析：*PaddleOCR* 对中文支持好，配合预处理可以达到 *90%+* 准确率。

课堂互动 Quiz - 对比分析

Quiz 2：多选题 - PaddleOCR vs TrOCR

关于 PaddleOCR 和 TrOCR 的比较，以下说法正确的是：

1. PaddleOCR 对中文手写体支持更好
2. TrOCR 更适合英文手写识别
3. PaddleOCR 部署更简单
4. TrOCR 不需要文本行检测
5. PaddleOCR 支持更多预处理选项

正确答案：(1)(2)(3)(5)

Quiz 3：排序题 - 预处理步骤

将以下预处理步骤按正确顺序排列：

A 去噪
B 二值化
C 灰度化
D 对比度增强

正确顺序：C → A → D → B

解释：

1. 先转灰度
2. 去噪避免增强噪声

课堂互动 Quiz - 实时投票

投票 1：你认为手写识别最大的挑战是？

A. 字体风格差异大（草书 vs 楷书）
B. 书写质量参差不齐
C. 中文同音字太多
D. 计算资源限制

请举手或使用在线投票工具选择你的答案

字体风格差异书写质量同音字问题

示例投票结果

参数调优实验挑战 - 对比实验设计

挑战 1：不同预处理参数对比

实验目标： 找到最优的预处理参数组合

实验设计：

实验组	去噪强度	CLAHE clip	二值化方式	OCR 准确率
A	无	-	简单阈值	?
B	h=10	2.0	OTSU	?
C	h=15	3.0	OTSU	?
D	h=20	4.0	自适应	?

任务： 在提供的测试图片上运行各组参数，记录 OCR 准确率，找出最优参数组合。

参数调优实验挑战 - 模型参数调优

挑战 2：PaddleOCR 参数调优

可调参数：

- `det_limit_side_len`: 检测尺寸限制
- `det_db_thresh`: DB 阈值
- `det_db_box_thresh`: 文本框阈值
- `rec_batch_num`: 识别批大小
- `drop_score`: 置信度过滤阈值

实验任务：

- 固定其他参数

挑战 3：TrOCR beam size 调优

什么是 beam search?

束搜索是一种解码策略，在每个时间步保留得分最高的 k 个候选序列，平衡搜索质量和效率。

beam size 对结果的影响：

beam size	准确率	速度
1 (贪婪)	较低	最快
5	中等	较快
10	较高	中等
20	最高	较慢

案例分析：识别失败诊断

案例 1：粘连字符识别失败

[手写”国国”粘连]

识别结果：”國”

正确结果：”国国”

失败原因分析：

1. **过度粘连**：两个字笔画交叉过多
2. **预处理不足**：二值化阈值不合适
3. **分割失败**：字符分割算法无法区分
4. **模型局限**：训练集中粘连样本少

改进方案：

- 优化预处理参数，增强对比度
- 使用笔画宽度变换 (SWT) 辅助分割
- 尝试过度分割 + 合并策略
- 对粘连样本进行数据增强训练

案例分析：识别失败诊断（续）

案例 2：潦草字迹识别失败

[潦草手写" 解决问题"]

识别: " 觖央问趣"

正确: " 解决问题"

原因分析：

- 连笔过多，笔画断裂
- 字形变形严重
- 笔画顺序不规律

改进方案：

案例 3：背景干扰识别失败

[带横线的答题纸]

识别: " 一答案一"

正确: " 答案"

原因分析：

- 横线与文字粘连
- 二值化后横线未去除
- 预处理未针对横线优化

改进方案：

学生讨论与分享

讨论题目

1. **识别失败案例分享**
   - 你在使用 OCR 时遇到过什么问题?
   - 是如何解决的?
   - 有什么经验可以分享?
2. **改进方案设计**
   - 针对刚才的案例，你有什么改进想法?
   - 如何在准确率和速度之间取得平衡?

分享规则

1. 每组派代表分享一个案例（2 分钟）
2. 其他组可以补充或提问

最佳实践投票与讨论

投票 1：不同场景下的工具选择

场景 1：处理 1000 份手写简答题试卷，要求准确率高，服务器有 GPU。

A PaddleOCR + 完整预处理
B TrOCR + 简单预处理
C 商业 API（百度/腾讯 OCR）
D 自己训练专用模型

推荐：A 或 D

大量数据时，专用模型效果最佳；
快速部署选 *PaddleOCR*。

投票 2：实际部署经验

问题：在生产环境中，你最常遇到的 OCR 问题是？

A 识别准确率不够高
B 推理速度太慢

经验分享：

常见问题讨论

Q1: 如何提高潦草字迹的识别率?

解决方案:

- 增强预处理（CLAHE 去噪）
- 使用骨架提取辅助
- 结合上下文语义校正
- 对潦草样本数据增强

Q3: 如何去除答题纸的横线干扰?

解决方案:

- 霍夫变换检测直线并擦除
- 形态学开运算去除横线
- 使用 in painting 修复
- 训练专门的去线模型

Q2: 识别速度太慢怎么办?

优化方案:

- 降低输入图像分辨率
- 使用更小的模型（Mobile 版）

Q4: 粘连字符如何分割?

解决方案:

- 垂直投影 + 动态阈值
- 最小生成树 (MST) 分割

课后作业

题目

实现简答题手写识别模块

要求：

1. 使用 PaddleOCR 或 TrOCR 识别手写文字
2. 实现图像预处理优化（至少包含灰度化、去噪、对比度增强、二值化）
3. 实现文本行分割（水平投影法或连通域分析）
4. 识别手写答案内容并分析识别准确率
5. 撰写实验报告，包含参数调优过程和结果分析

评分标准：

- 模型配置与调用：20 分
- 预处理实现：25 分
- 文本行分割：20 分
- 识别效果与分析：25 分

下节预告

第 9 周：系统架构与分组开发

故事问题：把所有模块组合起来

你将学会：

- 系统架构设计
- 模块集成方法
- 团队协作开发

课程设计开始！

谢谢！

遇到问题？使用 AI 调试助手

手写识别常见错误与 AI 调试策略：

- **PaddleOCR 安装失败**：
  - 提示 AI："paddlepaddle 安装报错：[粘贴错误信息]，提供替代方案"
  - AI 会建议：使用清华镜像源、检查 Python 版本
- **TrOCR 模型加载慢**：
  - 提示 AI："TrOCR 模型加载很慢，如何优化？"
  - AI 会建议：使用本地缓存、降低模型精度、使用 CPU 模式
- **识别结果为空**：
  - 提示 AI："TrOCR 返回空结果，检查图像路径 [路径]，代码如下：[代码]"
  - AI 会检查：图像格式、尺寸、模型选择
- **GPU 内存不足**：
  - 提示 AI："TrOCR 推理时 CUDA out of memory，如何优化？"
  - AI 会建议：降低 batch size、使用 CPU、使用小模型

AI 辅助调试三部曲

AI 辅助调试三部曲

1. **粘贴完整错误**：Traceback、警告信息、异常堆栈
2. **说明环境信息**：Python 版本、操作系统、GPU 型号、依赖包版本
3. **提供可复现代码**：最小化代码片段 + 数据样本描述

Prompt 模板（手写识别场景）：

```
错误信息 [粘贴完整Traceback]
的代码片 from paddleocr import PaddleOCR ocr =
PaddleOCR(use_angle_cls = True, lang = 段'ch')result = ocr.ocr('handwriting.jpg', cls = True)
段数据样本描 述手写图片，内容是学生手写的答"案解决问" 题，图片尺800x600寸，字迹清晰但有轻微连 笔
```

使用 AI 辅助学习 PaddleOCR 手写识别

推荐场景：

- 参数配置时：让 AI 解释各参数含义
- 识别结果解析时：让 AI 说明数据结构
- 预处理优化时：让 AI 提出改进建议
- 参数调优时：让 AI 设计实验方案

Cursor 快捷键：

快捷键	功能
Ctrl+K	原地编辑选中代码
Ctrl+L	打开 AI 对话侧边栏
Ctrl+I	AI 生成代码

手写识别专用 Prompt

差异化学习路径

三种角色，三种任务：

角色 0026 基础任务 (60 分) 0026 **进阶/挑战任务 (40 分)**
观察者 0026 跟着运行 PaddleOCR 手写识别代码，理解识别流程 0026 分析不同预处理方法（去噪强度、CLAHE

本周要点总结

手写识别概述

- 字形差异大、连笔问题
- 准确率 95%+ (TrOCR)
- 工具选择：PaddleOCR vs TrOCR

PaddleOCR

- 中文支持好
- 速度快
- 配置灵活

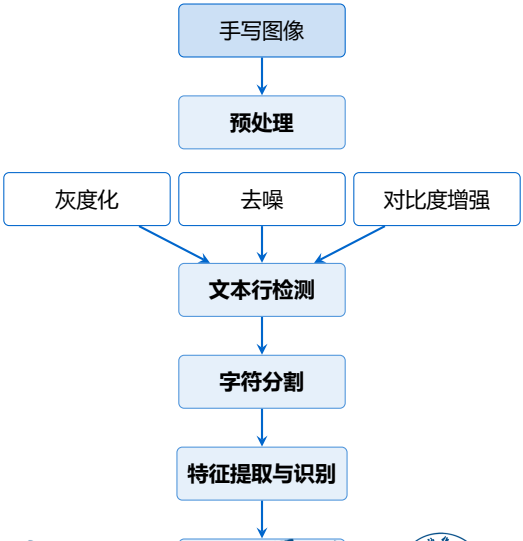
TrOCR

- 准确率最高
- Transformer 架构
- 适合高精度需求

预处理优化

- 灰度化 + 去噪
- 对比度增强
- 二值化

手写识别完整流程图



手写识别流程详解

1. 预处理阶段

- **灰度化**：降低计算复杂度
- **去噪**：高斯滤波、中值滤波
- **对比度增强**：直方图均衡化、CLAHE
- **二值化**：自适应阈值分割

2. 文本行检测

- **投影法**：水平投影分割行
- **连通域分析**：DBSCAN、MSER
- **深度学习方法**：EAST、CRAFT、PSENet

3. 字符分割与识别

- **字符分割**：垂直投影、Viterbi 算法
- **特征提取**：
  - 传统：HOG、SIFT、Gabor 特征
  - 深度学习：CNN 特征、Transformer 特征
- **分类识别**：SVM、CNN、Transformer

4. 后处理优化

- **语言模型校正**：N-gram、RNNLM
- **词典匹配**：模糊匹配、编辑距离
- **上下文校正**：BERT-based 纠错

传统方法 vs 深度学习方法对比

ccc

对比维度	传统方法
特征提取	人工设计特征：HOG、SIFT、Gabor 特征、投影特征等。需要领域专家知识，特征设计
模型复杂度	模型简单：SVM、KNN、HMM 等，参数量小，可解释性强。
数据需求	数据需求小：在少量样本上也能取得不错效果，适合数据稀缺场景。

传统方法 vs 深度学习方法对比（续）

ccc

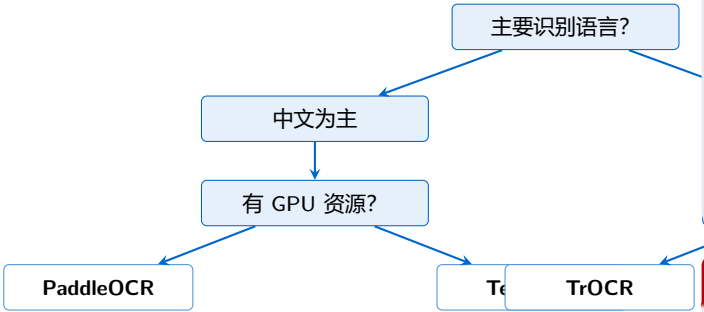
对比维度	传统方法	深度学习
准确率	准确率有限：手写汉字识别率通常在 70-85%，对复杂变形、连笔、噪声敏感。	准确率 TrOCR 型可达 9 对复杂场 棒性强，能力好。
推理速度	速度快：轻量级模型，CPU 上可实时处理，适合边缘设备部署。	速度较慢 要 GPU 大模型推 迟较高，通过量化 枝优化。
可解释性	可解释性强：特征工程清晰，决策过程透明，易于调试和分析错误。	可解释

主流 OCR 工具/框架对比

ccccc

对比维度	PaddleOCR	TrOCR	Tesseract	EasyOCR
开发团队	百度	Microsoft	Google	社区开源
支持语言	80+ 种，中文优秀	多语言，英文最优	100+ 种	80+ 种
手写识别	良好（需微调）	优秀（原生支持）	一般	一般
印刷体识别	优秀	优秀	良好	良好
准确率	85-90%	95%+	70-80%	75-85%
推理速度	快（GPU/CPU）	中等 (需 GPU)	快（CPU）	中等
模型大小	轻量-中等	大 (Transformer)	轻量	中等
部署难度	简单	中等	简单	简单
文档完善度	优秀	良好	良好	一般
社区活跃度	高（Star 40k+）	中等	高	中等

OCR 工具选择决策树



选择建议

中文场景：

- 有 GPU：PaddleOCR
- 无 GPU：Tesseract

英文场景：

- 高精度：TrOCR
- 快速部署：EasyOCR

提示

实际选择时还需考虑：

- 硬件资源限制
- 实时性要求
- 维护成本

常见问题与解决方案（一）

问题 1：识别准确率低

原因分析：

- 图像质量差：噪声、模糊、光照不均
- 预处理不足：未进行有效的去噪和增强
- 模型不匹配：选择的模型不适合特定场景
- 字体/书写风格差异：训练数据覆盖不足

解决方案：

1. 加强预处理：使用自适应阈值、CLAHE 增强对比度
2. 尝试多种去噪算法：高斯滤波、中值滤波、双边滤波
3. 更换或组合模型：如 PaddleOCR + TrOCR 结果融合
4. 针对性微调：使用领域数据对模型进行微调

常见问题与解决方案（二）

问题 2：处理速度慢

优化方案：

- **硬件加速：**使用 GPU 替代 CPU，批量处理
- **模型轻量化：**使用轻量级模型（如 PaddleOCR mobile 版）
- **推理优化：**ONNX 导出、TensorRT 加速、量化压缩
- **多线程并行：**利用多核 CPU 并行处理多图像
- **预处理优化：**降低输入图像分辨率（保持宽高比）

问题 3：内存不足

解决方案：

- **分批处理：**避免一次性加载所有图像
- **降低 batch size：**减少同时处理的样本数
- **模型精简：**使用更小的模型变体
- **清空缓存：**及时处理释放不再使用的变量
- **使用 16 位浮点：**FP16 混合精度训练/推理

问题 4：特殊字符识别困难

- 扩充训练数据覆盖特殊字符
- 使用专用字体数据集微调

常见问题解答

Q1：手写识别准确率能达到多少？

A：TrOCR 可以达到 95%+，PaddleOCR 约 85-90%，取决于预处理质量。

Q2：如何提高识别准确率？

A：

1. 优化图像预处理（去噪、增强对比度）
2. 使用更高精度的模型（TrOCR large）
3. 针对特定场景进行微调训练

Q3：连笔字如何处理？

A：TrOCR 对连笔字识别效果更好，因为 Transformer 能捕捉上下文关系。

作业提交要求

作业名称

week08_ 手写简答题识别.ipynb

提交内容：

1. **代码部分**（60%）
   - 模型配置与加载
   - 预处理函数实现
   - 手写文字识别
   - 结果可视化
2. **分析报告**（40%）
   - 识别准确率分析
   - 不同模型的对比
   - 遇到的问题及解决方案

评分标准细则

评分项	具体要求	分值
模型配置	正确加载 PaddleOCR 或 TrOCR 模型	25 分
预处理优化	实现灰度化、去噪、增强、二值化	25 分
识别效果	能正确识别手写文字，准确率高	30 分
分析报告	对比分析、问题总结、改进建议	20 分
总分		**100 分**

加分项

- 同时实现 PaddleOCR 和 TrOCR 并对比（+5 分）
- 实现批处理多张手写图片（+5 分）
- 可视化识别结果（标注识别区域）（+5 分）