# 线图提取

# 2.线图提取

线图提取具体流程如下：将考古报告的图片，交给YOLOv8对考古报告中的元素进行预测，以生成详细坐标。根据生成的坐标文件来运行脚本，使线图和图注序号可以自动匹配并裁剪。下面给出了线图提取中的必要步骤介绍，根据介绍部署运行即可。

## 2.1 YOLOv8确定线图坐标

由于考古学文档线图复杂多样，难以用普通OCR的方法做提取，所以尝试使用计算机视觉领域常用的目标检测模型YOLO做提取，并验证可行性。目前YOLO的最新版本是刚刚发布的v10，但是关于v10说明文档和教程比较少，不易于部署，采用了2023 年1月开源的v8版本。

GitHub：GitHub - ultralytics/ultralytics: NEW - YOLOv8 🚀 in PyTorch

官方文档：https://docs.ultralytics.com/zh

部署流程：https://divis.yuque.com/staff-tgoia2/nk5uhg/yggwrt4r4hupxalt

经过训练过的模型文件：

weights.rar

使用预测命令，更换训练好的模型和图片地址，对整本考古报告做预测。（该指令model需换成经过训练的模型文件位置，source后面填写图片文件夹地址）

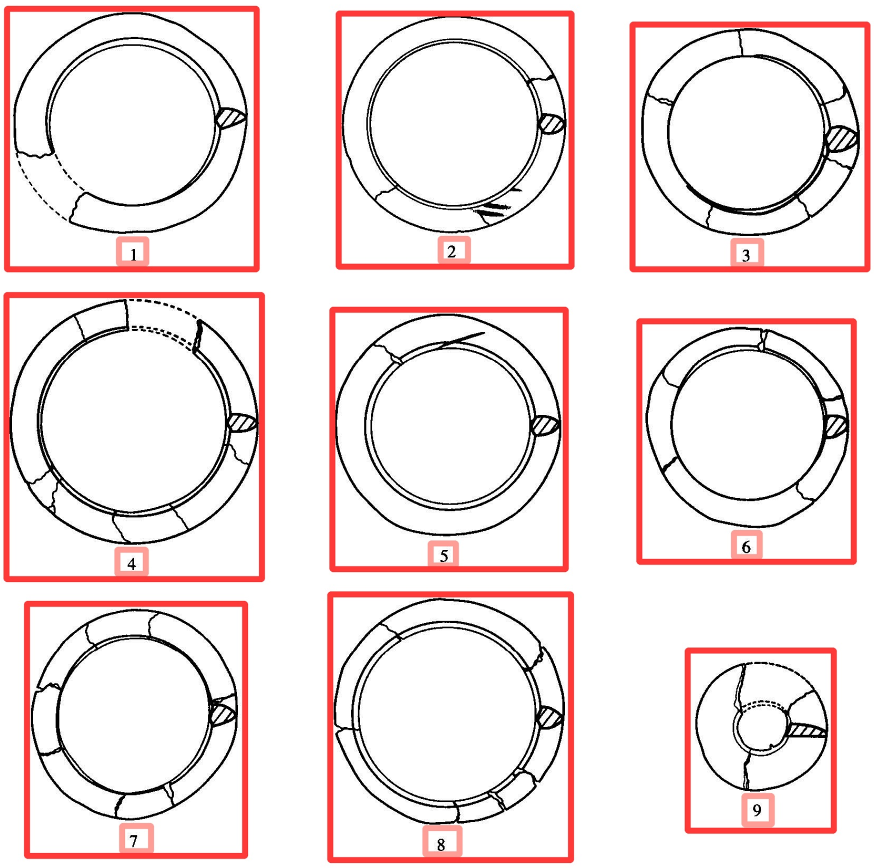
```
yolo predict model=/root/ultralytics/runs/detect/train7/weights/best.pt
source=/root/ultralytics/datasets/凌家滩 save_txt=True hide_labels=Flase
hide_conf=FLase
```

命令后添加参数save_txt=True保留位置信息，为后期裁剪做准备。hide_labels=Flase hide_conf=FLase用于隐藏标签和置信度，防止遮挡器物信息。

生成结果中会含有图片预测位置的可视化结果和txt坐标信息（每一个框的坐标数据）。

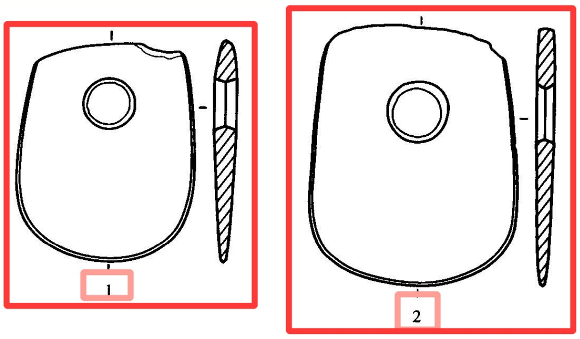
结果会生成四类框：第一类线图整体（黄色框），第二类线图（红色框），第三类序号框（粉色框），第四类图注（橙色框）。一个图注为一个线图整体，一个考古页面一般最多有两个整体。当一个页面出现多个图注时，不能简单地直接将图注作为命名，也不能计算位置距离，所以采用划分整体的版面分析方法来解决。

生成结果如图所示即可：



图一八八　98M25出土玉镯、环

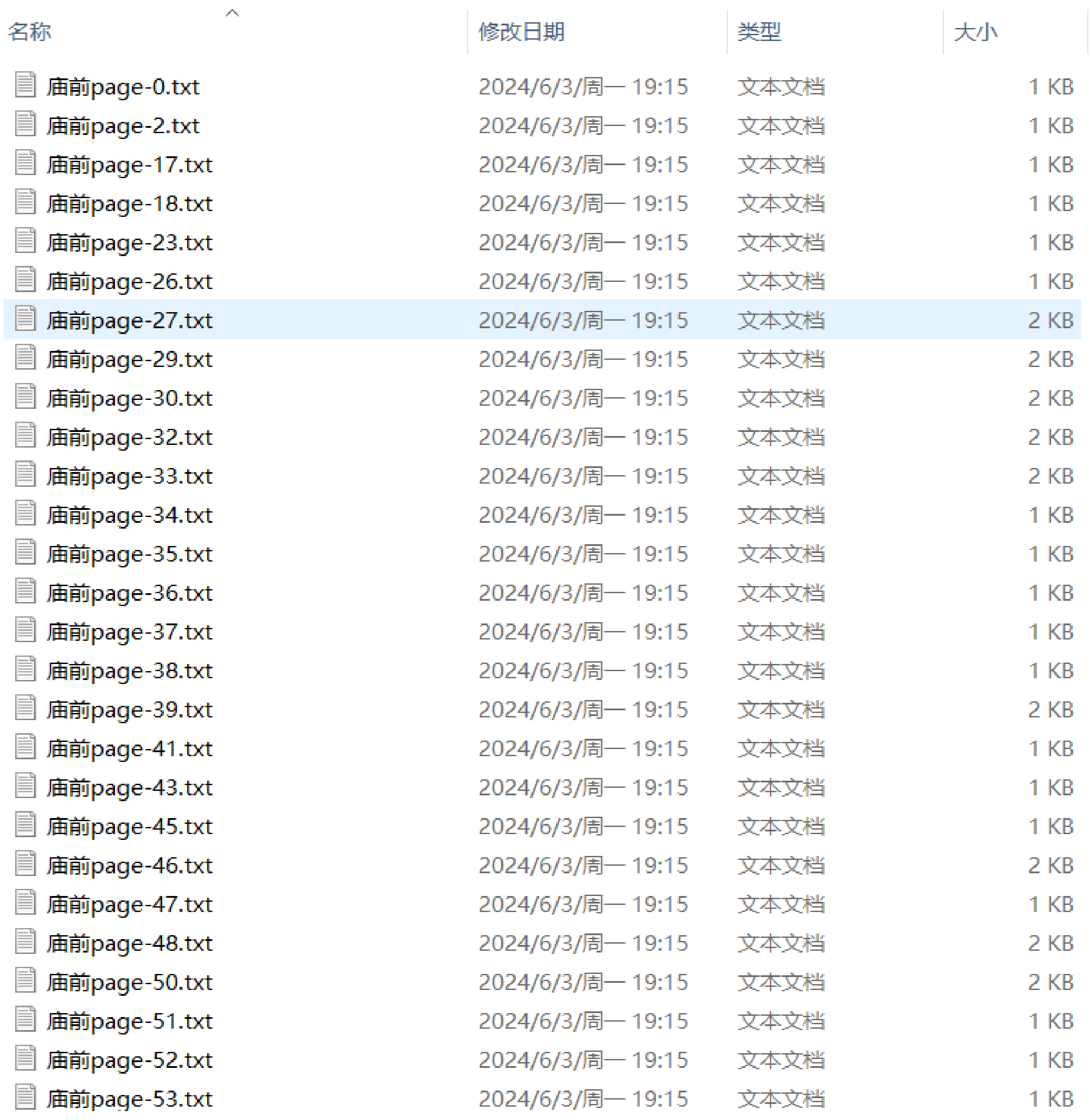
1～8. 镯（98M25：2、4、5、6、10、12、13、14）　9. 环（98M25：8）（均为1/2）



图一八九　98M25出土石钺

1～3. 98M25：16、20、21（均为1/4）

还需要找到生成结果中的坐标txt文档：



| 名称 | 修改日期 | 类型 | 大小 |
|---|---|---|---|
| 庙前page-0.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-2.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-17.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-18.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-23.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-26.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-27.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-29.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-30.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-32.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-33.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-34.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-35.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-36.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-37.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-38.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-39.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-41.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-43.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-45.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-46.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-47.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-48.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-50.txt | 2024/6/3/周一 19:15 | 文本文档 | 2 KB |
| 庙前page-51.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-52.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |
| 庙前page-53.txt | 2024/6/3/周一 19:15 | 文本文档 | 1 KB |

## 2.2 结果处理对线图裁剪命名

在该步骤中，使用上述的txt文档和原图（不要使用画框的结果图）做处理，否则会出现大量未命名的情况

代码步骤：

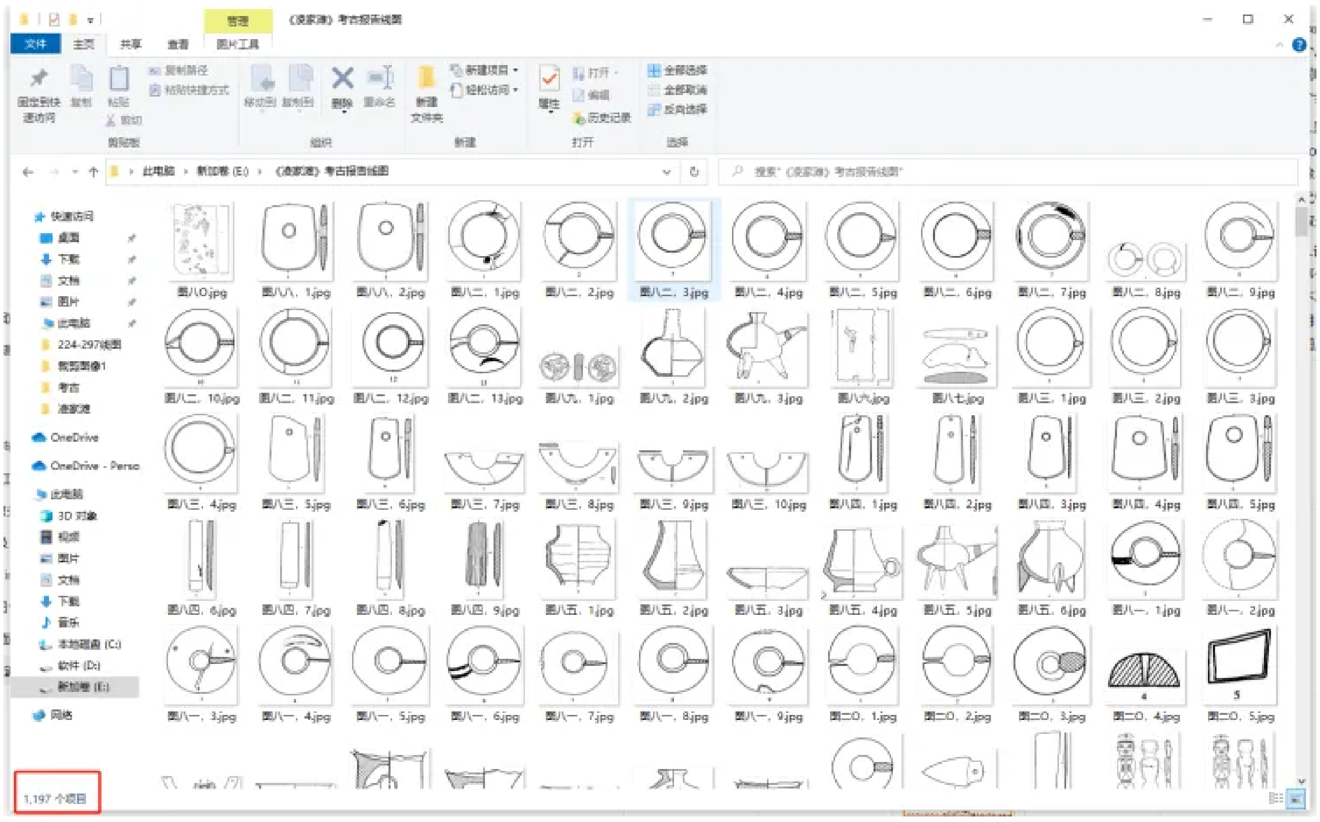
1. 更改原图及坐标文档位置。
2. 确定类别4整体框。
3. 找到该类别4整体框内的图注，序号，器物，墓葬元素。
4. 将图注与序号交给PaddleOCR识别处理。
5. 通过计算IOU，找到每一个器物对应的序号（如果有的话），并以图注，序号的方式命名。未找到或未识别到文字，使用默认命名。

```
import os
import shutil
```

```python
from PIL import Image
import numpy as np
from paddleocr import PaddleOCR
import re

# 读取坐标文档
def read_detections(txt_path):
    detections = []
    with open(txt_path, 'r') as file:
        for line in file:
            parts = line.strip().split(' ')
            detections.append([int(parts[0])] + [float(part) for part in
parts[1:]])
    return detections

# 创建保存裁剪图片的文件夹
def create_save_folder(path):
    if os.path.exists(path):
        shutil.rmtree(path)
    os.makedirs(path)

# 清理文本，确保只包含合法的文件名字符
def clean_text(text):
    # 移除非法字符
    cleaned_text = re.sub(r'[\\/:*?"<>|]', '', text)
    # 移除空白字符
    cleaned_text = cleaned_text.strip()
    # 限制文本长度
    max_length = 100  # 设定最大长度为100个字符
    if len(cleaned_text) > max_length:
        cleaned_text = cleaned_text[:max_length]
    return cleaned_text

# 全局变量初始化
last_caption = "default_caption"
# 处理图片和坐标
def process_image(txt_folder, img_folder, save_folder):
    global last_caption
    ocr = PaddleOCR(use_angle_cls=True, lang="ch")
    create_save_folder(save_folder)

    for txt_file in os.listdir(txt_folder):
        if txt_file.endswith('.txt'):
            # 为每张图片重新初始化已使用的索引集合
            used_indices = set()
            txt_path = os.path.join(txt_folder, txt_file)
            img_name = os.path.splitext(txt_file)[0] + '.jpg'
            img_path = os.path.join(img_folder, img_name)

            if os.path.exists(img_path):
                detections = read_detections(txt_path)
                img = Image.open(img_path)
                width, height = img.size

                overall_found = False

                for det in detections:
                    if det[0] == 4:
                        overall_found = True
                        overall_bounds = [det[1:5], width, height]
                        elements_within_frame = filter_elements(detections,
overall_bounds, width, height)

                        caption_text = ""
                        found_caption = False

                        for element in elements_within_frame:
                            if element[0] == 2:
                                caption_text = extract_caption_text(element,
img, ocr, width, height)
                                if caption_text:
                                    last_caption = caption_text  # 更新最后一个
成功的图注

                                    found_caption = True
                                    break

                        if not found_caption and last_caption:
                            # 如果没有找到新的图注，并且有先前的图注，生成新的图注
                            caption_text =
increment_chinese_number(last_caption)

                        for element in elements_within_frame:
                            if element[0] != 2:  # 非图注的元素处理
                                process_element(element, img, ocr,
save_folder, width, height, caption_text, elements_within_frame,used_indices)

                if not overall_found:  # 没有找到整体框，处理器物和图注
                    process_items_without_overall_box(detections, img, ocr,
save_folder, width, height)


# 处理没有整体框的情况
def process_items_without_overall_box(detections, img, ocr, save_folder,
width, height):
    items = [det for det in detections if det[0] == 0]
    captions = [det for det in detections if det[0] == 2]
    indices = [det for det in detections if det[0] == 1]

    for item in items:
        caption_text = find_closest_caption(item, captions, width, height,
img, ocr)
        index_text, _ = find_closest_index_box(item, indices, width, height,
img, ocr)
        if caption_text and index_text:
            filename = os.path.join(save_folder, f"
{caption_text}_{index_text}.jpg")
            img_roi = crop_to_box(item, img, width, height)
            img_roi.save(filename)

# 寻找最近的图注
def find_closest_caption(item, captions, width, height, img, ocr):
    x_center, y_center = item[1], item[2]
    min_distance = float('inf')
    closest_caption = None

    for caption in captions:
        cap_x_center, cap_y_center = caption[1], caption[2]
        distance = np.sqrt((x_center - cap_x_center) ** 2 + (y_center -
cap_y_center) ** 2)
        if distance < min_distance:
            min_distance = distance
            closest_caption = caption

    if closest_caption:
        img_roi = crop_to_box(closest_caption, img, width, height)
        ocr_result = ocr.ocr(np.array(img_roi), cls=True)
        if ocr_result and ocr_result[0]:
            return clean_text(ocr_result[0][0][1][0])
    return "default"

# 中文数字映射
chinese_to_arabic = {
    'O': 0, '一': 1, '二': 2, '三': 3, '四': 4,
    '五': 5, '六': 6, '七': 7, '八': 8, '九': 9,
    '十': 10
}

def chinese_to_arabic_num(chinese_str):
    """将连续的中文数字（如“二一八”）直接转换为阿拉伯数字"""
    num_map = {'O': 0, '一': 1, '二': 2, '三': 3, '四': 4, '五': 5,
               '六': 6, '七': 7, '八': 8, '九': 9, '十': 10}
    result = 0
    for char in chinese_str:
        value = num_map.get(char)
        if value is None:
            continue  # Skip invalid characters
        result = result * 10 + value
    return result
```

```python
def arabic_to_chinese(num):
    """将阿拉伯数字转换为不带单位的中文数字，例如218转换为“二一八”"""
    num_str = str(num)
    num_map = {0: 'O', 1: '一', 2: '二', 3: '三', 4: '四', 5: '五',
               6: '六', 7: '七', 8: '八', 9: '九'}
    result = ''.join(num_map[int(digit)] for digit in num_str)
    return result


def increment_chinese_number(caption):
    """自动递增图注中的末尾连续中文数字"""
    # 匹配连续中文数字部分
    pattern = re.compile(r'[一二三四五六七八九O]+')
    match = pattern.search(caption)
    if match:
        chinese_num = match.group(0)
        arabic_num = chinese_to_arabic_num(chinese_num)
        incremented_num = arabic_num + 1
        new_chinese_num = arabic_to_chinese(incremented_num)
        return caption.replace(chinese_num, new_chinese_num)
    return caption









def intersection_over_union(det, overall_bounds, width, height):
    # 解析整体框的边界
    x_center, y_center, w, h = overall_bounds[0]
    box_x_min = (x_center - w / 2) * width
    box_y_min = (y_center - h / 2) * height
    box_x_max = (x_center + w / 2) * width
    box_y_max = (y_center + h / 2) * height

    # 解析元素的边界
    ele_x_center, ele_y_center, ele_w, ele_h = det[1:5]
    ele_x_min = (ele_x_center - ele_w / 2) * width
    ele_y_min = (ele_y_center - ele_h / 2) * height
    ele_x_max = (ele_x_center + ele_w / 2) * width
    ele_y_max = (ele_y_center + ele_h / 2) * height

    # 计算交集
    inter_x_min = max(box_x_min, ele_x_min)
    inter_y_min = max(box_y_min, ele_y_min)
    inter_x_max = min(box_x_max, ele_x_max)
    inter_y_max = min(box_y_max, ele_y_max)

    if inter_x_min < inter_x_max and inter_y_min < inter_y_max:
        # 交集区域
        inter_area = (inter_x_max - inter_x_min) * (inter_y_max - inter_y_min)
        # 元素的区域
        ele_area = (ele_x_max - ele_x_min) * (ele_y_max - ele_y_min)
        # 计算交集与元素面积的比例
        iou = inter_area / ele_area
        return iou >= 0.5
    return False

def filter_elements(detections, overall_bounds, width, height):
    filtered_elements = []
    for det in detections:
        if intersection_over_union(det, overall_bounds, width, height):
            filtered_elements.append(det)
    return filtered_elements



def extract_caption_text(det, img, ocr, width, height):
    img_roi = crop_to_box(det, img, width, height)
    ocr_result = ocr.ocr(np.array(img_roi), cls=True)
    if ocr_result:
        full_text = ocr_result[0][0][1][0]
        match = re.match(r"^[^\d]*", full_text)
        if match:
            return clean_text(match.group())
    return ""

# Global counter for default indexing
default_index_counter = 0
def increment_default_index():
    global default_index_counter
    default_index_counter += 1
    return f"default_{default_index_counter}"

def process_element(det, img, ocr, save_folder, width, height, caption_text,
all_detections, used_indices):
    img_roi = crop_to_box(det, img, width, height)
    filename = None


    if det[0] == 3:
        filename = os.path.join(save_folder, f"{caption_text}.jpg")
    elif det[0] == 0:
        idx_text, closest_index_box = find_closest_index_box(det,
all_detections, width, height, img, ocr)
        if closest_index_box and idx_text not in used_indices:
            used_indices.add(idx_text)  # 标记此索引为已使用
            filename = os.path.join(save_folder, f"{caption_text},
{idx_text}.jpg")

            # 裁剪并保存序号框
            index_box_img_roi = crop_to_box(closest_index_box, img, width,
height, enlarge=True)
            index_box_img_roi = enlarge_image(index_box_img_roi, 5)  # 放大序号
框

        else:
            idx_text = increment_default_index()  # 使用自动递增的默认索引
            filename = os.path.join(save_folder, f"{caption_text},
{idx_text}.jpg")

    if filename:
        img_roi.save(filename)


def find_closest_index_box(det, all_detections, width, height, img, ocr):
    x_center, y_center, w, h = det[1], det[2], det[3], det[4]
    best_coverage = 0
    closest_index_box = None
    idx_text = increment_default_index()  # Start with a unique default index
each time

    for idx_box in all_detections:
        if idx_box[0] == 1:
            idx_x_center, idx_y_center, idx_w, idx_h = idx_box[1], idx_box[2],
idx_box[3], idx_box[4]
            inter_left = max(x_center - w / 2, idx_x_center - idx_w / 2)
            inter_top = max(y_center - h / 2, idx_y_center - idx_h / 2)
            inter_right = min(x_center + w / 2, idx_x_center + idx_w / 2)
            inter_bottom = min(y_center + h / 2, idx_y_center + idx_h / 2)
            if inter_right > inter_left and inter_bottom > inter_top:
                inter_area = (inter_right - inter_left) * (inter_bottom -
inter_top)
                idx_area = idx_w * idx_h
                coverage = inter_area / idx_area
                if coverage > best_coverage and coverage > 0.5:  # Check
against a threshold
                    best_coverage = coverage
                    closest_index_box = idx_box

    if closest_index_box:
        idx_x_center, idx_y_center, idx_w, idx_h = closest_index_box[1],
closest_index_box[2], closest_index_box[3], closest_index_box[4]
        idx_img_roi = crop_to_box(closest_index_box, img, width, height,
enlarge=True)
        idx_img_roi = enlarge_image(idx_img_roi, 5)  # 放大序号框
        ocr_result = ocr.ocr(np.array(idx_img_roi), cls=True)
        if ocr_result and ocr_result[0]:
            idx_text = clean_text(ocr_result[0][0][1][0])

    return idx_text, closest_index_box if closest_index_box else False


# 放大图片
def enlarge_image(image, scale_factor):
    # 获取原图尺寸
    original_size = image.size
    # 计算放大后的尺寸
    new_size = (int(original_size[0] * scale_factor), int(original_size[1] *
scale_factor))
    # 放大图片
    enlarged_image = image.resize(new_size, Image.LANCZOS)  # 使用 LANCZOS 替代
ANTIALIAS
    return enlarged_image

def crop_to_box(box, img, width, height, enlarge=False):
    x_center, y_center, w, h = box[1:5]
    x_min = max(0, int((x_center - w / 2) * width))
    y_min = max(0, int((y_center - h / 2) * height))
    x_max = min(width, int((x_center + w / 2) * width))
    y_max = min(height, int((y_center + h / 2) * height))
    img_crop = img.crop((x_min, y_min, x_max, y_max))

    if enlarge:
        img_crop = img_crop.resize((img_crop.width * 5, img_crop.height * 5),
Image.LANCZOS)

    return img_crop
# 示例调用
txt_folder = r'E:\庙前\labels'
img_folder = r'E:\庙前\imgs'
save_folder = r'E:\裁剪图像文件夹-庙前1'

process_image(txt_folder, img_folder, save_folder)
```

以生成一个命名完成的线图文件夹：



最终对线图进行核对，至此线图提取完毕。