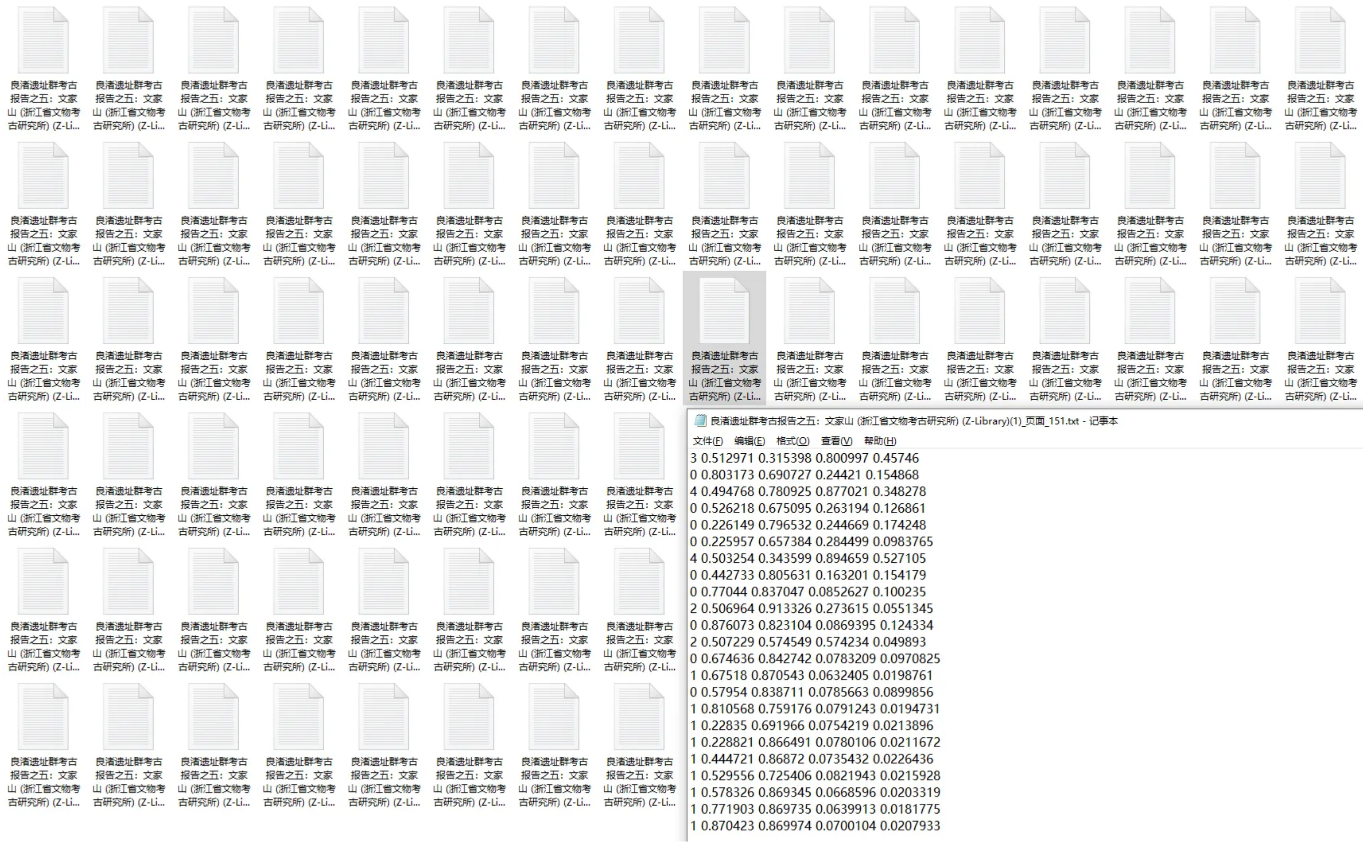
线图提取优化方案

线图提取主要分为两步：

- 线图标注：使用训练的YOLO模型对线图进行标注，识别线图中的器物，图注，序号等，以及坐标信息。结果如下图：



良渚遗址群考古报告之五：文家山（浙江省文物考古研究所）(Z-Library)(1)_页面_151.txt - 记事本

```
3 0.512971 0.315398 0.800997 0.45746
0 0.803173 0.690727 0.24421 0.154868
4 0.494768 0.780925 0.877021 0.348278
0 0.526218 0.675095 0.263194 0.126861
0 0.226149 0.796532 0.244669 0.174248
0 0.225957 0.657384 0.284499 0.0983765
4 0.503254 0.343599 0.894659 0.527105
0 0.442733 0.805631 0.163201 0.154179
0 0.77044 0.837047 0.0852627 0.100235
2 0.506964 0.913326 0.273615 0.0551345
0 0.876073 0.823104 0.0869395 0.124334
2 0.507229 0.574549 0.574234 0.049893
0 0.674636 0.842742 0.0783209 0.0970825
1 0.67518 0.870543 0.0632405 0.0198761
0 0.57954 0.838711 0.0785663 0.0899856
1 0.810568 0.759176 0.0791243 0.0194731
1 0.22835 0.691966 0.0754219 0.0213896
1 0.228821 0.866491 0.0780106 0.0211672
1 0.444721 0.86872 0.0735432 0.0226436
1 0.529556 0.725406 0.0821943 0.0215928
1 0.578326 0.869345 0.0668596 0.0203319
1 0.771903 0.869735 0.0639913 0.0181775
1 0.870423 0.869974 0.0700104 0.0207933
```

良渚遗址群考古报告之五：文家山（浙江省文物考古研究所）(Z-Li...
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

- 结果处理：使用脚本对线图坐标信息进行精确处理，对检测到的线图与相关文字信息进行匹配和裁剪。目前使用的代码如下，以生成一个命名完成的线图文件夹。

YOLO后处理 Python

```python
import os
import shutil
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 = {
    '0': 0, '一': 1, '二': 2, '三': 3, '四': 4,
    '五': 5, '六': 6, '七': 7, '八': 8, '九': 9,
```

```python
        '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



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
```

```python
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

```

```python
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'
```

目前出现了裁剪命名的线图文件夹命名率低的问题，主要原因可能有三个：

- yolo：yolo对新的考古报告线图没有很好的进行识别
- 匹配：代码逻辑没有正确的将线图和图注做匹配
- ocr：paddleocr对文字没有很好的识别

以《文家山》为例，观察其中img以解决yolo识别问题，观察裁剪文件夹以解决命名问题。

📎 文家山labels+imgs.rar

📎 裁剪图像文件夹-文家山.rar

1.YOLO

观察《文家山》发现，线图都被很好识别到，但是出现了对大量玉管命名失败的现象：

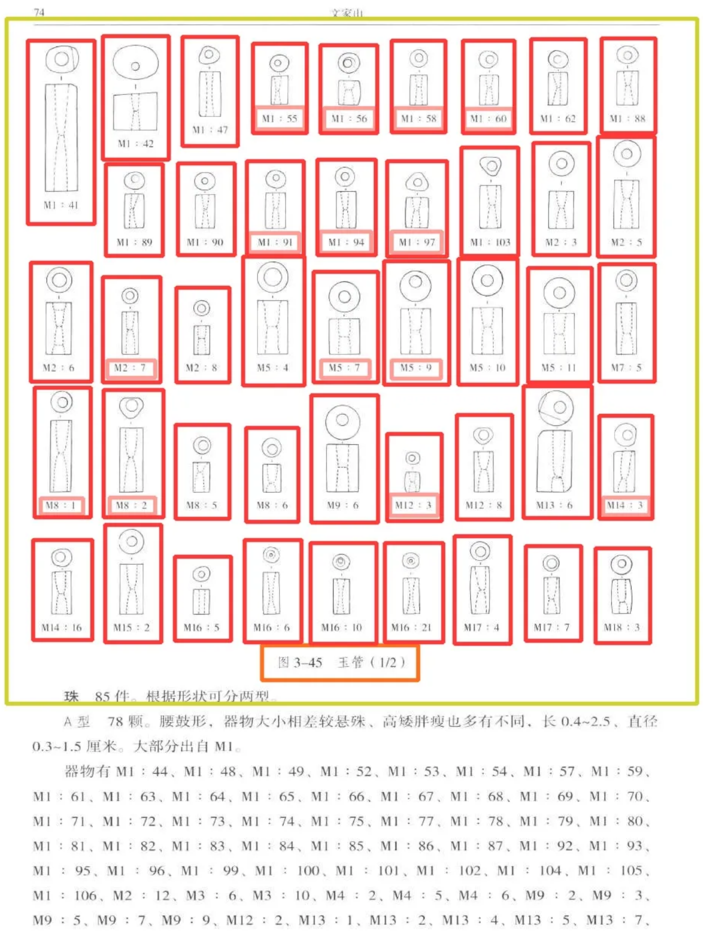


图 3–45　玉管（1/2）

珠　85 件。根据形状可分两型。

A 型　78 颗。腰鼓形，器物大小相差较悬殊、高矮胖瘦也多有不同，长 0.4~2.5、直径 0.3~1.5 厘米。大部分出自 M1。

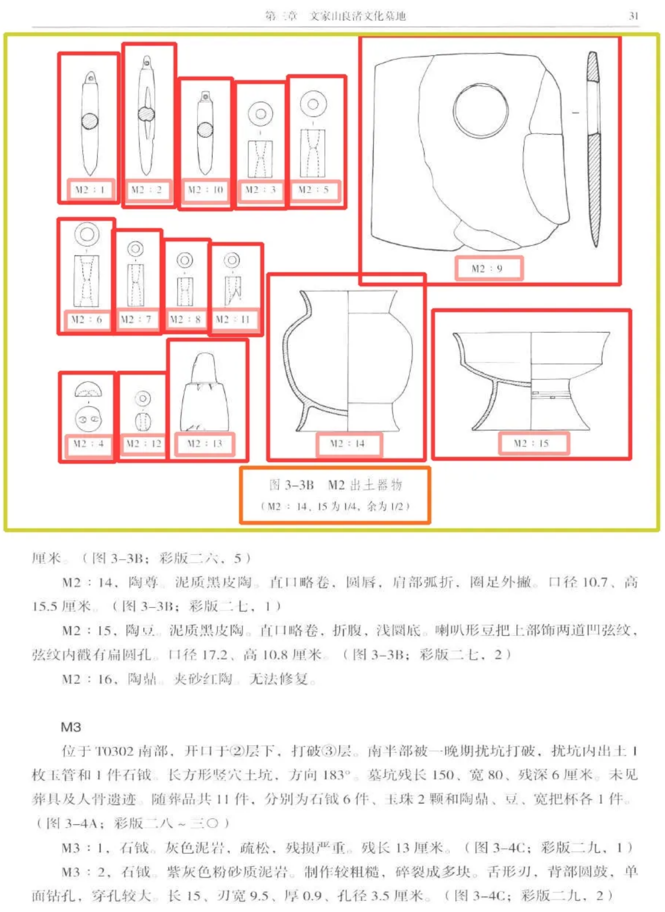
器物有 M1：44、M1：48、M1：49、M1：52、M1：53、M1：54、M1：57、M1：59、M1：61、M1：63、M1：64、M1：65、M1：66、M1：67、M1：68、M1：69、M1：70、M1：71、M1：72、M1：73、M1：74、M1：75、M1：77、M1：78、M1：79、M1：80、M1：81、M1：82、M1：83、M1：84、M1：85、M1：86、M1：87、M1：92、M1：93、M1：95、M1：96、M1：99、M1：100、M1：101、M1：102、M1：104、M1：105、M1：106、M2：12、M3：6、M3：10、M4：2、M4：5、M4：6、M9：2、M9：3、M9：5、M9：7、M9：9、M12：2、M13：1、M13：2、M13：4、M13：5、M13：7、

建议采取的办法是：

- 优化数据集：检查是否有漏标的现象，补充一些如玉管等常见器物线图的训练，测试效果
- 换用更高版本的模型：v10

2.匹配

将《文家山》考古报告其中标注正确，序号识别不好的一页143（如下图）：


M2：1
M2：2
M2：10
M2：3
M2：5
M2：9
M2：6
M2：7
M2：8
M2：11
M2：4
M2：12
M2：13
M2：14
M2：15

第三章　文家山良渚文化墓地　31

图 3-3B　M2 出土器物
（M2：14、15 为 1/4，余为 1/2）

厘米。（图 3-3B；彩版二六，5）

M2：14，陶尊。泥质黑皮陶。直口略卷，圆唇，肩部弧折，圈足外撇。口径 10.7、高 15.5 厘米。（图 3-3B；彩版二七，1）

M2：15，陶豆。泥质黑皮陶。直口略卷，折腹，浅圜底。喇叭形豆把上部饰两道凹弦纹，弦纹内戳有扁圆孔。口径 17.2、高 10.8 厘米。（图 3-3B；彩版二七，2）

M2：16，陶鼎。夹砂红陶。无法修复。

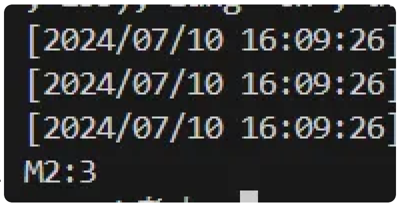
M3

位于 T0302 南部，开口于②层下，打破③层。南半部被一晚期扰坑打破，扰坑内出土 1 枚玉管和 1 件石钺。长方形竖穴土坑，方向 183°。墓坑残长 150、宽 80、残深 6 厘米。未见葬具及人骨遗迹。随葬品共 11 件，分别为石钺 6 件、玉珠 2 颗和陶鼎、豆、宽把杯各 1 件。（图 3-4A；彩版二八～三〇）

M3：1，石钺。灰色泥岩，疏松，残损严重。残长 13 厘米。（图 3-4C；彩版二九，1）

M3：2，石钺。紫灰色粉砂质泥岩。制作较粗糙，碎裂成多块。舌形刃，背部圆鼓，单面钻孔，穿孔较大。长 15、刃宽 9.5、厚 0.9、孔径 3.5 厘米。（图 3-4C；彩版二九，2）

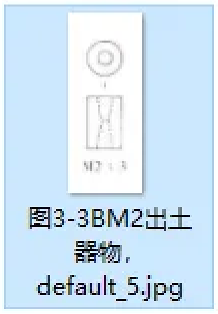
按yolo所给的坐标进行了裁剪，得到序号图片



```
[2024/07/10 16:09:26]
[2024/07/10 16:09:26]
[2024/07/10 16:09:26]
M2:3
```

写了一段使用paddleocr识别图片的代码，该图片可以识别出序号

但是在匹配中却出现了默认命名的情况



查看其识别错误的原因，均是由ocr未识别到序号导致的：

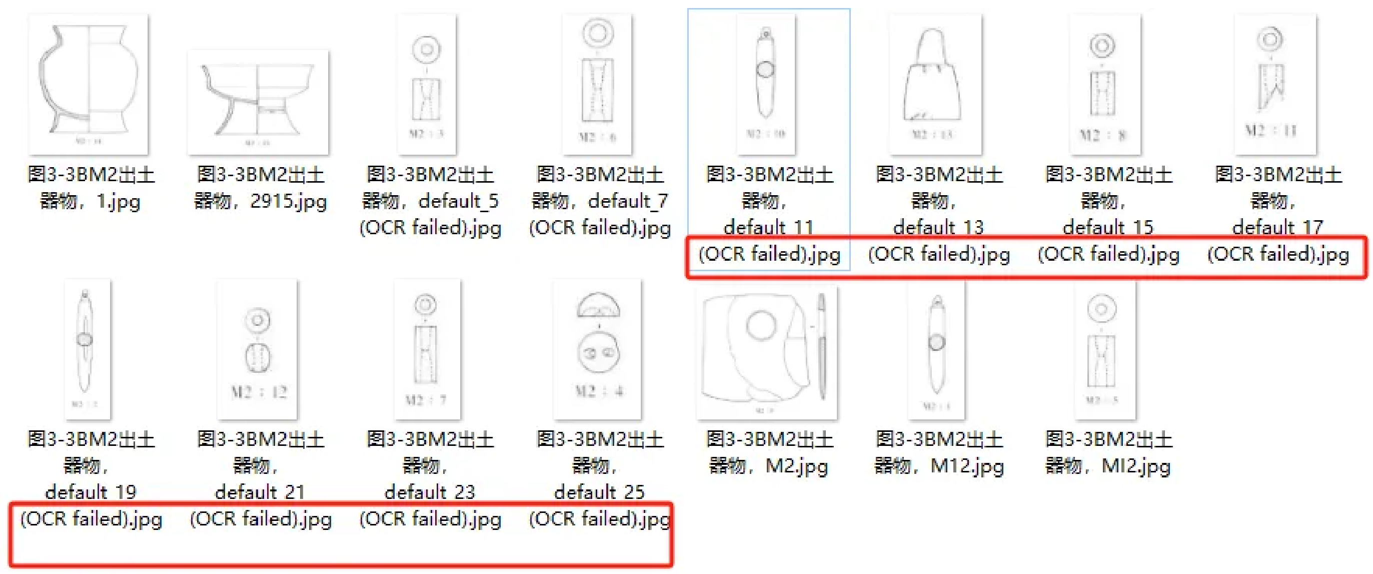


图3-3BM2出土器物，1.jpg
图3-3BM2出土器物，2915.jpg
图3-3BM2出土器物，default_5 (OCR failed).jpg
图3-3BM2出土器物，default_7 (OCR failed).jpg
图3-3BM2出土器物，default 11 (OCR failed).jpg
图3-3BM2出土器物，default 13 (OCR failed).jpg
图3-3BM2出土器物，default 15 (OCR failed).jpg
图3-3BM2出土器物，default 17 (OCR failed).jpg
图3-3BM2出土器物，default 19 (OCR failed).jpg
图3-3BM2出土器物，default 21 (OCR failed).jpg
图3-3BM2出土器物，default 23 (OCR failed).jpg
图3-3BM2出土器物，default 25 (OCR failed).jpg
图3-3BM2出土器物，M2.jpg
图3-3BM2出土器物，M12.jpg
图3-3BM2出土器物，MI2.jpg

建议采取的办法是：

- 可能是在代码的逻辑上出现了错误

3.OCR

在识别图注序号时，发现了图注基本都能被正确识别命名，但是在序号上会发现很多错误，首先解决序号识别错误的问题

建议采取的办法是：

- 可能是图像质量影响到了识别，在ocr代码之前加入更多图像预处理的操作：图像增强，二值化等
- 换用其他ocr模型