队伍编号	MC2410837
题号	B

Oracle OCR：基于 YOLOv8-MobileNet 的甲骨文文字分割与识别模型

摘要

甲骨文作为中国历史上最早的成熟文字，对于研究中国古代文明具有不可估量的价值，随着技术进步，甲骨文的数字化和自动化处理已经成为学术研究的新前沿。我们研究旨在开发一个综合性的甲骨文图像处理与识别模型，通过自动化技术精确分割和识别甲骨文中的文字。

针对问题一，我们对甲骨文原始拓片图像进行了一系列的预处理操作，包括**灰度化、去噪声、对比度增强、二值化处理及边缘增强**，并且利用**Hu矩、灰度共生矩阵和对比度**等方法进行**特征提取**，有效地捕捉了图像的关键视觉信息，以区分文字和背景，并识别不同的字符异体。

针对问题二的甲骨文图像的自动单字分割问题，我们基于**YOLOv8**构建了一个图像分割模型，利用问题二所给的已标注的数据集，将数据集划分为训练集和验证集，并对标记格式进行转换，并且在问题一的基础上对图像进行预处理，输入到预训练的YOLOv8模型中进行训练，最后根据模型训练和验证**损失、精确率、召回率、F1分数以及混淆矩阵**等多个维度对模型表现进行评估。

针对问题三，我们进一步将问题二得到的模型应用于测试集进行单字分割，并且将分割结果按规定格斯保存到**Test results.xlsx**文件中。

针对问题四，我们选用**MobileNet**这种轻量级卷积神经网络模型作为文字识别模型，在训练集上建立文字索引字典，然后将数据集划分为训练集和验证集，对MobileNet进行训练。对于测试集的图片，我们使用YOLOv8模型对图像进行检测分割，再对分割结果使用MobileNet进行单字识别，并将识别结果直接在图片上进行呈现。

关键词：YOLOv8，MobileNet，甲骨文单字分割，文字识别，OpenCV

目录

一、问题重述

1.1 问题背景

甲骨文是我国已知最早成熟的文字系统，具有极其重要的历史和学术价值。这种古老文字的研究不仅对理解中国文明的起源具有重要意义，也对全球文明研究产生了深远的影响。在政府的推动下，甲骨文研究已经进入了一个新的发展阶段，人工智能和大数据技术的应用，使得甲骨文的全息研究和数字化处理成为热点。

然而，甲骨文拓片的数字化处理面临诸多挑战。甲骨文图像分割是其中的基础问题，需要在复杂背景中准确地分割出独立的文字区域。现有的图像处理技术在处理甲骨文拓片时，常受到点状噪声、人工纹理和固有纹理的干扰，导致高误分割率。此外，甲骨文的同一个字存在多种异体字形，也增加了识别的难度。

基于以上要求，我们希望能够建立模型对甲骨文图像进行处理和分割，从而对甲骨文图像中的文字进行自动识别，从而有效地解决甲骨字识别难点，推动甲骨文研究进展，助力解决甲骨文考释分析难题。

1.2 解决目标

问题 1：图像预处理与特征提取

为附件 1 中的三张甲骨文原始拓片图片进行图像预处理，提取关键图像特征，并建立一个甲骨文图像预处理模型。该模型旨在识别和处理图像中的干扰元素，为后续的分割和识别任务提供清晰的图像基础。

问题 2：甲骨文图像分割模型开发

我们希望开发一个快速且准确的甲骨文图像分割模型，该模型能够自动从复杂背景中分割出单个甲骨文字。该模型将使用附件 2 的已标注分割数据集进行训练和评估，以确保其准确性和效率。

问题 3：模型应用与评估

利用开发的甲骨文图像分割模型，对附件 3 中的 200 张甲骨文原始拓片图像进行自动单字分割，并将分割结果整理到“Test_results.xlsx”中，并且对模型进行评估。

问题 4：甲骨文字识别

基于前三个问题的研究与模型开发，采用合适的方法对甲骨文原始拓片图像进行文字识别，使用附件 4 的已标注数据进行训练，以实现对测试集中 50 张图像的自动文字识别和结果呈现。

二、问题分析

2. 1 问题一分析

2.1.1 图像预处理

首先我们要对图像进行预处理，主要分为以下几个步骤进行处理：

步骤一：灰度化处理

灰度化处理将彩色图像转换为灰度图像，简化信息处理的复杂性，因其只涉及亮度信息，适合用于纹理和形状分析，我们通过计算彩色图像中红、绿、蓝三色通道的加权平均值来实现灰度化处理。

步骤二：去除噪声

在对图像进行预处理时，我们需要消除图像捕捉过程中引入的噪声，包括甲骨文图像中由于保存环境等因素产生的点状噪声和擦划痕迹，我们使用双边滤波和中值滤波进行噪声去除。

双边滤波是一种非线性的滤波方法，可以在保持边缘清晰的同时，有效去除图像的噪声。双边滤波对于甲骨文图像中常见的细微裂纹和纹理保持较好，有助于后续的特征提取。

中值滤波适用于去除椒盐噪声、脉冲噪声，可以保持图像的边缘特性，不会使图像产生显著的模糊，用于保持甲骨文纹理细节。

步骤三：增强对比度

通过提高图像的对比度，可以使得甲骨文的细节更为突出，便于识别和分析，我们主要运用直方图均衡化和自适应直方图均衡化，提高图像的对比度，使得甲骨文的细节更为突出，便于识别和分析。

步骤四：二值化处理

我们将灰度图像转换为黑白图像，在此步骤中我们使用自适应阈值化，在图像对比

度不均匀或背景颜色不一致的情况下更好地处理局部区域的特征。

步骤五：边缘增强

最后我们通过增强图像中的边缘信息，更清晰地分离文字与背景，使用 Canny 边缘检测识别图像中文字的轮廓。

2.1.2 特征提取

图像的 Hu 矩、灰度共生矩阵（GLCM）和对比度是图像分析中常用的特征，我们通过这些指标进行特征提取能够有效分离文字和背景。

Hu 矩：在甲骨文识别中，Hu 矩可以提供圆形度、矩形度、紧凑度等信息，从而有效区分不同字符。

灰度共生矩阵（GLCM）： GLCM 能够通过计算像素点之间的灰度空间关系，提取出图像的对比度、均匀性等纹理特征，在甲骨文图像中的能够有效区分字和背景。

对比度：在纹理分析中，对比度可以反映纹理的清晰度和复杂性。

通过进行特征提取，我们可以捕捉到图像的关键视觉信息，帮助算法理解图像内容。在甲骨文识别中可以帮助算法区分甲骨文字符和背景、识别不同字符之间的差异以及理解字符的空间排列和结构。通过结合这些特征，我们可以构建一个更强大的图像识别模型，提高对甲骨文的识别和分析能力。

2. 2 问题二分析

在问题二中，我们需要开发一个快速而准确的甲骨文图像分割模型，将单字甲骨文图像自动从复杂背景中分割出来。为了实现这一目标，我们将考虑使用 YOLOv8 算法来进行模型建立。

YOLO(You Only Look Once)是一种 one-stage 目标检测算法，即仅需要“看”一次就可以识别出图片中物体的类别和边界框。YOLOv8 模型可用于对象检测、图像分类和实例分割任务，具有灵活性与可扩展性，其核心优势在于其高效的实时检测能力，以及对小目标和复杂背景下的高精度识别[1]。

我们利用问题二所给的已标注的数据集分割，将数据集划分为训练集和验证集，并对标记格式进行转换，并且在问题一的基础上对图像进行预处理，进而输入到预训练的 YOLOv8 模型中进行迁移学习，最后根据模型训练和验证损失、精确率、召回率、F1 分数以及混淆矩阵等多个维度对模型表现进行评估。

2.3 问题三分析

问题三需要利用问题二建立的甲骨文图像分割模型对附件 3 中的 200 张甲骨文图像进行自动单字分割。我们考虑利用问题二中训练好的性能优异的 YOLOv8 图像检测模型对 200 张图像逐个进行分割，并将分割结果图片和标签文件保存。然后对标签内容进行格式转换，最后输出到 Test results.xlsx 文件中。

2.4 问题四分析

问题四在前三问甲骨文单字分割任务的基础上，需要我们对分割出的甲骨文文字进行文字识别。因此我们考虑先对甲骨文图像进行单字分割，然后对分割出来的单字图像，建立新的文字识别模型进行识别。

我们选用 MobileNet 这种轻量级卷积神经网络模型作为文字识别模型，在训练集上建立文字索引字典，然后将数据集划分为训练集和验证集，对 MobileNet 进行训练。对于测试集的图片，我们先使用 YOLOv8 模型对图像进行检测分割，再对分割结果使用 MobileNet 进行单字识别，并将识别结果直接在原始图片上进行呈现。

三、模型假设

为了便于模型的建立与求解，现作如下假设：

1、假设甲骨文原始拓片图像中的干扰元素可以有效去除，从而提高图像单字分割和文字识别的准确性。

2、假设所提供的甲骨文图像分割标注数据集可靠性高，没有出现大量的标注数据错误和漏标情况。

3、假设文字识别训练集的图片与甲骨文文字能正确对应。

四、符号说明

符号	说明	单位
TP	预测为正例而且实际上也是正例的数量	/
FP	预测为正例然而实际上却是负例的数量	/

FN	预测为负例然而实际上却是正例的数量	/
TN	预测为负例而且实际上也是负例的数量	/
Y	图片灰度	/
R/G/B	颜色图片中红色/绿色/蓝色通道强度	%
Accuracy	模型准确率	%
Precision	模型精确率	%
Recall	模型召回率	%
F1	F1 分数	/
$Loss_{bbox}$	边界框损失	/
$Loss_{cls}$	分类损失	/
α	学习率	%
β	衰减率	%

五、模型建立与求解

5.1 问题一：图像预处理和特征提取

5.1.1 图像预处理

图像预处理的目的是优化图片质量，为特征提取和进一步的图像分析提供更好的基础，我们主要分为以下几步进行图像预处理：

1. 灰度化

首先我们将彩色图像转换为灰度图像，灰度化处理能够有效处理的颜色信息，从而减少数据量，简化了图像处理的复杂性，并且专注于分析图像的结构和纹理信息而非颜色信息。灰度图像处理速度更快，数据量更小，便于后续的图像处理操作。

我们将彩色图像的每一个像素点的红、绿、蓝三个颜色通道按照一定的比例加权平均，转换成单个灰度值，使用公式

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{1}$$

来计算每个像素的灰度值。

2. 去噪声

数字图像在捕获、传输、处理过程中可能会引入各种噪声。去噪声可以提高图像质量，有助于更清楚地识别图像中的文字，能够对甲骨文图像中的细微裂纹和纹理细节处理更进一步。

在此过程中，我们使用双边滤波和中值滤波进行噪声去除：

双边滤波同时利用空间近邻度和像素值相似度进行加权平均，有效去除噪点而保持边缘，适用于图像中的文字细节区域。

中值滤波通过将每个像素点的值替换为邻域内的中位数实现噪声去除，能够有效去除随机出现的椒盐噪声，同时保持图像边界的清晰度。

3. 增强对比度

我们在这一步增加图像的动态范围，使甲骨文图像中的文字与背景之间的可视对比度增强，能够更有助于后续的文字分割和识别。

在此过程中，我们使用直方图均衡化与 CLAHE 技术来增强对比度。

对于直方图均衡化，我们需要对图像的直方图进行调整，使其分布更均匀，从而增强整个图像的对比度。

CLAHE（Contrast Limited Adaptive Histogram Equalization）是一种改进的直方图均衡化技术，能够将图像分割成小块，在每块中独立地应用直方图均衡化，能够有效避免传统直方图均衡化中的过度放大噪声，并且能够在不同区域按需增强对比度。

4. 二值化处理

二值化是将图像从灰度转换为黑白图像的过程，能够简化图像内容，使得非文字区域与文字区域有着明显的对比，便于进一步的图像分析处理。

我们使用 OTSU 算法自动选择一个阈值，这个阈值可以最大化类间方差（即前景与背景的差异），从而有效分离文字与背景。

5. 边缘增强

边缘增强是为了明确图像中文字与背景之间的界限，为文字的准确分割提供辅助。

我们使用 Canny 边缘检测，先使用高斯滤波器平滑图像以去除噪声，然后计算图像中每个像素点的梯度强度和方向，通过非极大值抑制和双阈值检测完成强边缘的连接和弱边缘的去除。

5. 1. 2 特征提取

在进行了图像预处理后，我们对关键特征进行提取，为机器学习分类和识别模型提供数据，在此过程中，我们使用以下几种方法进行特征提取：

1.Hu 矩

Hu 矩是一组基于图像矩的不变量，对于图像的平移、缩放、旋转和镜像具有不变性。所以在甲骨文图像处理以及处理文字的异体形式中，使用 Hu 矩能够更精确地对甲骨文字形进行识别和分类。

我们首先计算图像的原始矩和中心矩。原始矩反映了图像的几何特性，中心矩则提供了图像相对于其质心的空间分布。在计算之后我们对中心矩进行归一化处理，使得中心矩不受图像大小的影响。接着从归一化的中心矩中导出七个 Hu 不变矩，这些矩能够捕捉图像的基本形状信息，并且能够保持对图像变换的不变性。通过比较不同图像的 Hu 矩值，可以在数据库中进行模式匹配分类，从而识别甲骨文中的特定字符。

2.灰度共生矩阵（GLCM）

GLCM 是一种用于量化图像纹理信息的统计方法，它能够通过分析特定距离和角度上像素对的灰度关系，提取出图像纹理的对比度、均匀性、同质性和熵等属性。
我们首先创建 GLCM，选择多个方向（0°、45°、90°、135°）和距离，对每一对距离和方向计算共生矩阵。接着从每个 GLCM 中能量（或二阶矩）、对比度、相关性和同质性等特征，来描述图像纹理的粗糙度和复杂性。这些纹理特征能够区分甲骨文中具有不同纹理特性的字符区域，能提升文字识别的准确性和效率。

3.对比度特征

对比度特征可以帮助区分图像中的前景和背景，通常来说，在甲骨文图像中，对比度高的区域对应着清晰的文字符号。

我们通过分析图像的局部区域对比度来计算对比度，计算局部像素与其周围像素的亮度差来评估对比度。高对比度区域可能指示文字的存在，而低对比度区域可能是背景或受损区域。通过对比度的区分，我们可以优化分割算法，确保只有文字区域进行进一步分析和处理。

5. 1. 3 结果

三张图像预处理结果如下图所示：

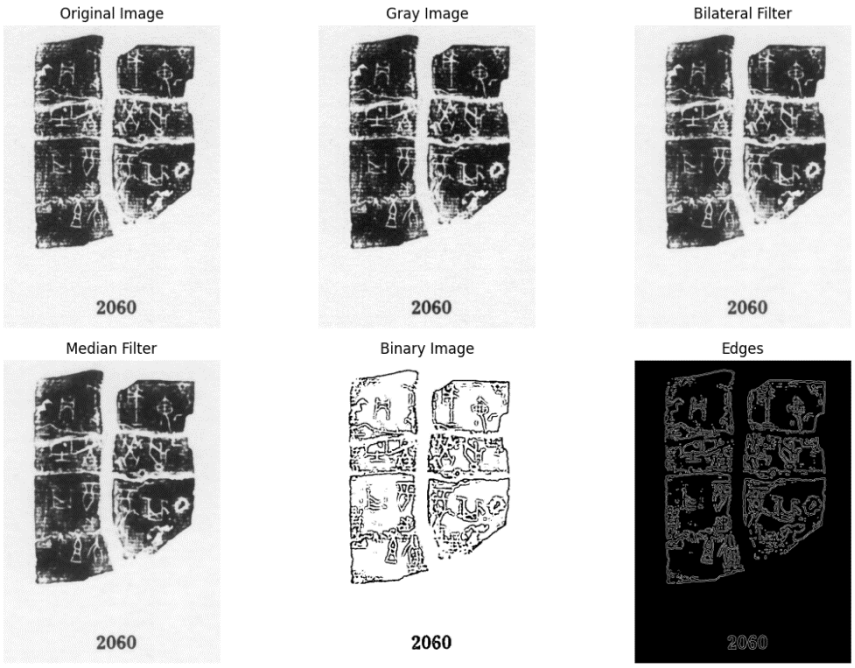


图 1 第一张图预处理

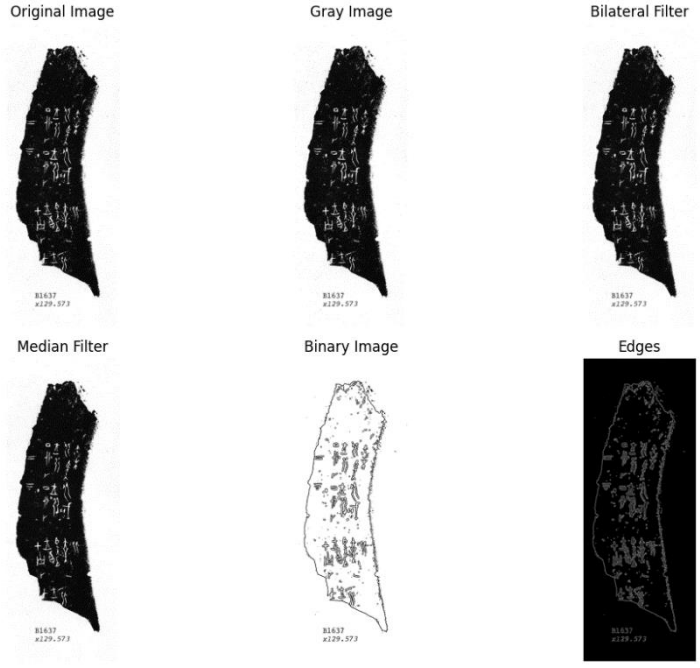


图 2 第二张图预处理

图 3 第三张图预处理

特征提取如下表所示：

表 1 三张图的特征提取

		第一张图	第二张图	第三张图
Hu 矩	H[0]	7.95162039e-04	1.08108087e-03	8.32204210e-04
	H[1]	6.51218784e-08	6.66230356e-07	1.03804527e-07
	H[2]	3.66142566e-13	2.34896731e-14	8.50421805e-13
	H[3]	7.96012311e-13	7.03248088e-15	2.09544224e-12
	H[4]	4.27342386e-25	3.15838139e-29	2.68537320e-24
	H[5]	2.01529691e-16	-7.30531546e-19	6.56574887e-16
	H[6]	4.53222812e-26	8.46882381e-29	7.83172146e-25
对比度		123.12623637971936	237.24778324468676	480.38417261080946

5. 2 问题二：图像分割模型建立

5. 2. 1 YOLOv8 模型简介

Ultralytics YOLOv8 是一款最先进的（SOTA，state of the art）模型，它在以往 YOLO 版本成功的基础上引入了新的功能和改进，进一步提高了性能和灵活性。

YOLOv8 采用了随机缩放、旋转、平移、对比度增强等更丰富多样的数据增强技术以及更稳定和学习率调度、权重初始化等更高效的训练策略。从而可以增加训练数据的多样性，提高模型的泛化能力，并且能够加速模型的收敛和提高目标检测的性能。

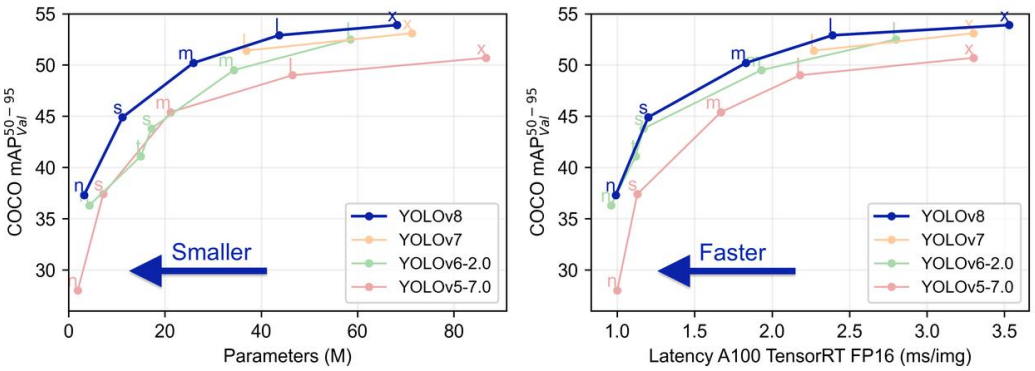


图 4 YOLO 模型发展

YOLOv8 网络结构以及模型特征如下：

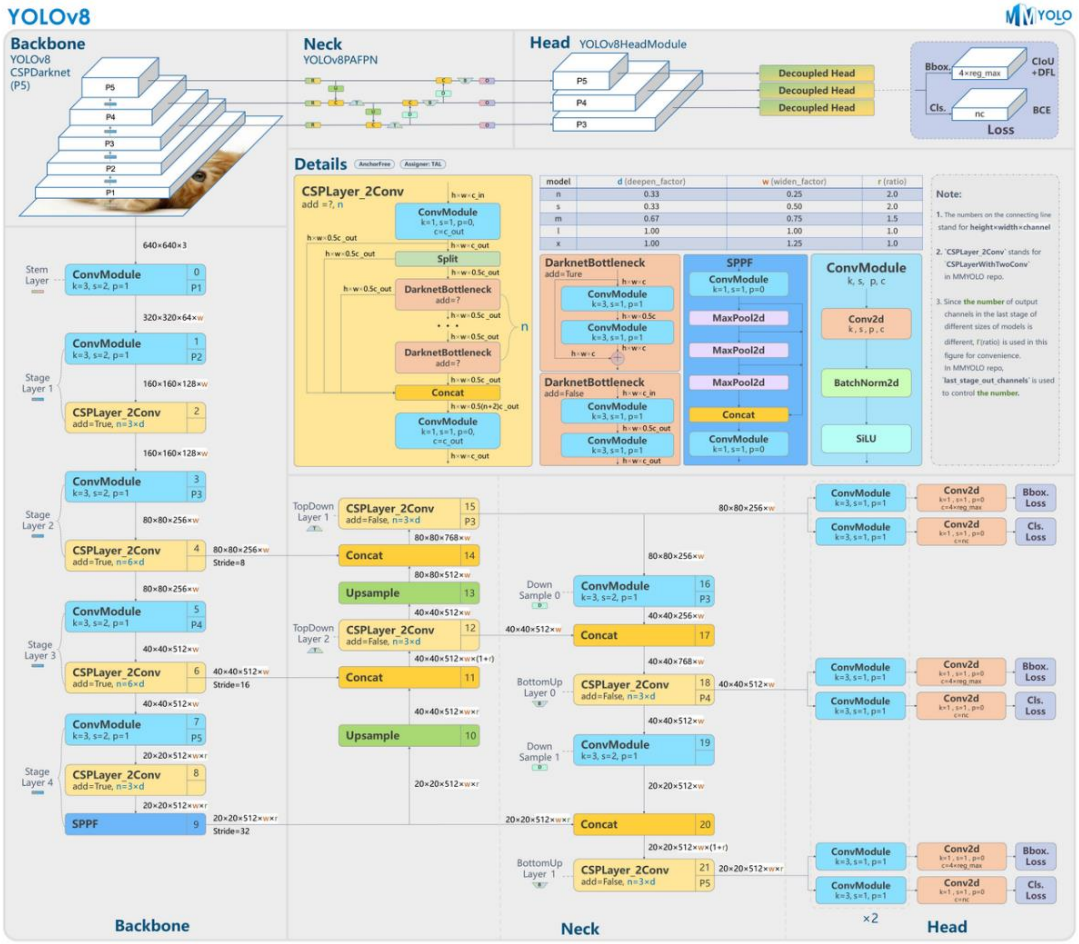


图 5 YOLOv8 网络结构

单次前向传播：YOLOv8 继续采用单次前向传播的方式进行目标检测，整个检测过程只需要一次网络前向传播就可以完成目标的定位和分类，从而实现了极高的检测速度[2]。

优化的损失函数和数据增强：为了进一步提高检测精度，YOLOv8 可以采用改进的损失函数和数据增强技术，我们使用 SoftNMS 算法替换传统的 NMS 算法能够更好地处理重叠的目标框，以及设计形状增强的数据增强方法来提高模型对伪装对象的形状感知能力[3]。

动态特征定位与并行回归：YOLOv8 通过引入动态特征定位和并行回归机制，能够提高模型在生物医学图像分析等特定领域中的检测精度。这种方法通过自适应头模块实现特征的动态定位和并行回归，有效提升了检测性能[4]。

5. 2. 2 数据预处理

对于附件 2 中提供的训练数据集，我们需要进行数据集的预处理，使得处理后的数据集格式符合我们所设计的图像分割模型的输入格式要求。具体做法如下：

1）图像标注分割标签格式转换

在提供的原始数据集中，每一张图片都有一个与其对应的 json 格式的标签文件，包含了每个图像对应的甲骨文文字目标检测标签信息。

例如，某图片的文件名为 b02519Z，该图像上已经标注好了所有甲骨文文字目标检测框的参数信息，每个检测框由五个参数进行描述，分别是$[< x_min >, < y_min >, < x_max >, < y_max >, < object_class >]$，来代表检测框的左上角点在图像中的坐标，右下角在图像中的坐标以及目标检测类别。其中的三个子数组表示了三个目标，每个目标由一个含有五个元素的数组表示，分别是目标边界框的左上角和右下角的坐标，以及目标的类别标签。于是我们可以将该图像的参数信息写为：

$$\begin{bmatrix} 64.0 & 194.0 & 134.0 & 277.0 & 1.0 \\ 150.0 & 168.0 & 204.0 & 344.0 & 1.0 \\ 206.0 & 170.0 & 259.0 & 349.0 & 1.0 \end{bmatrix}$$

具体来说，

①第一个数组$[64.0, 194.0, 134.0, 277.0, 1.0]$表示一个目标，其左上角点的坐标为$(64.0, 194.0)$，右下角点的坐标为$(134.0, 277.0)$，类别标签为$1.0$。

②第二个数组[150.0, 168.0, 204.0, 344.0, 1.0]也表示一个目标，其左上角点的坐标为(150.0, 168.0)，右下角点的坐标为(204.0, 344.0)，类别标签同样为1.0。

③第三个数组[206.0, 170.0, 259.0, 349.0, 1.0]仍然表示一个目标，其左上角点的坐标为(206.0, 170.0)，右下角点的坐标为(259.0, 349.0)，类别标签为1.0。

这种格式简单直观，易于理解和解析，但相比于其他格式，其缺点是没有提供目标边界框的归一化信息，因此在处理不同尺寸的图像时可能需要进行额外的处理。于是接下来我们需要将其转换为 yolov8 格式的标注文件：

在 YOLO 格式中，每个图像的标签信息通常以以下格式存储：

< object_class > < x_center > < y_center > < width > < height >

< object_class >：检测到的目标类别，通常使用类别的编号来表示，从 0 开始。

< x_center >：目标中心点相对于图像宽度的归一化值。

< y_center >：目标中心点相对于图像高度的归一化值。

< width >：目标宽度相对于图像宽度的归一化值。

< height >：目标高度相对于图像高度的归一化值。

下图展示了甲骨文数据集标签文件的内容统计：柱状图显示了不同类别的实例分布数量（非甲骨文和甲骨文）。散点图则展示了目标检测任务中边界框的空间分布情况，反映了常见的尺寸和长宽比。

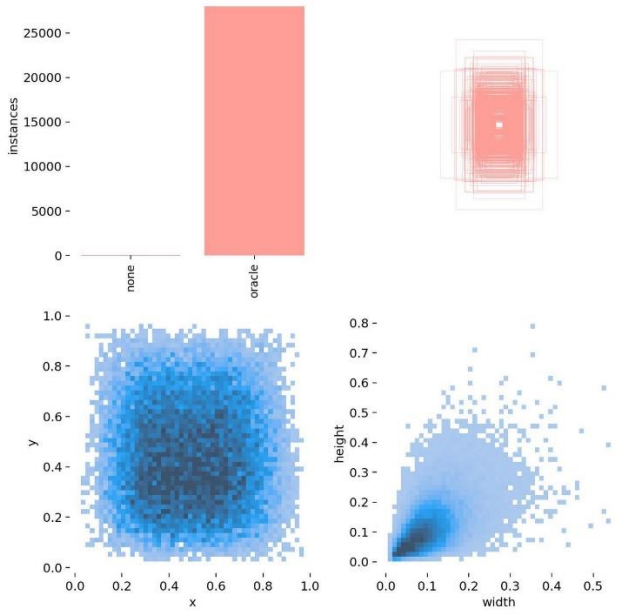


图 6 标签文件统计图

2）划分训练集和验证集

由于数据集规模有限且需要评估模型的泛化性能，我们将数据集划分为训练集和验证集。

数据集首先被随机打乱，然后按照 80:20 的比例划分为训练集和验证集。其中，前 80%的样本被分配到训练集，而后 20%的样本则用于验证。

为了确保训练集和验证集在类别分布上保持平衡，我们在随机划分过程中采用了分层抽样的方法，以防止两个数据集中类别分布不均匀导致的偏差。

验证集能够确保模型在未见过的数据上具有较好的泛化能力，防止模型过拟合，并且有助于验证模型的性能和稳定性。

5. 2. 3 YOLOv8 模型建立与训练

由于题目所给的甲骨文数据集较小，训练集标注数据比较稀缺，而迁移学习通过利用已有的知识来帮助学习新的任务，从而减少了对大量标注数据的需求。所以我们通过预训练模型在大规模数据集上进行预训练，然后在特定任务上进行微调，可以显著提高模型的性能。因此，我们采用 YOLOv8 预训练的目标检测任务模型 YOLOv8n 进行迁移学习，在其基础上采用我们所构建好的甲骨文数据集进行训练，实现对不同的甲骨文原始拓片图像进行自动单字分割。

我们选择 Ultralytics 预训练的 YOLOv8n 目标检测模型，模型训练轮次设置为 30，训练过程中每个批次的图像采样数量为 64，并且使用 Adam 优化器进行优化。

我们在模型训练过程中采用以下优化方法：

1）Warmup 学习率预热策略：

在模型刚开始训练时,模型的权重是随机初始化的，此时若选择一个较大的学习率,可能带来模型的不稳定(振荡)，选择 Warmup 预热学习率的方式，可以使得模型开始训练的几个轮次或者一些步骤内学习率较小,在预热的小学习率下，模型可以慢慢趋于稳定,等模型相对稳定后再选择预先设置的学习率进行训练,使得模型收敛速度变得更快，模型效果更佳。

下图是我们的模型在训练过程中使用 warmup 策略对学习率超参数进行动态调整后的学习率变化曲线。

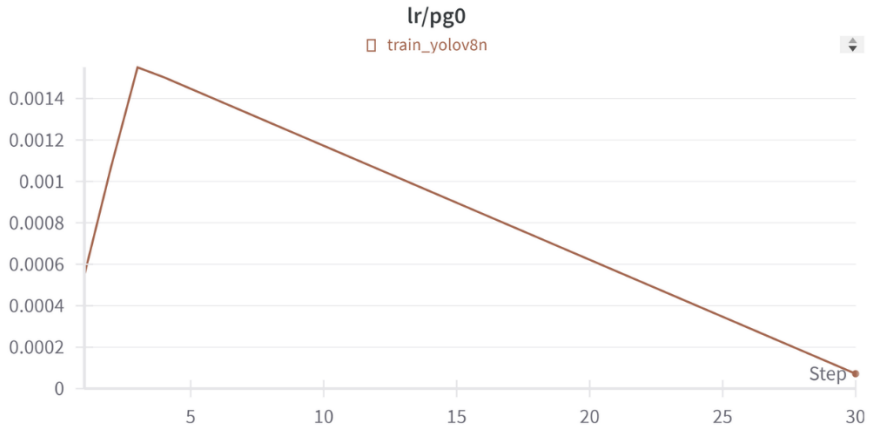


图 4 学习率曲线

2）训练过程与验证过程穿插进行

我们在模型训练过程中，采用了使模型在训练集和验证集上交叉进行训练验证的方式，让模型每次在训练集上训练一个轮次之后，就在验证集上进行一次验证，使用验证集对训练的模型进行评估，以了解模型在未见过的数据上的性能表现。这种做法可以给模型训练带来如下的好处：

①提高模型的泛化能力：通过在训练过程中不断地在验证集上评估模型的性能，可以帮助识别和解决过拟合问题和欠拟合问题，从而提高模型对未见过数据的泛化能力。验证集包含了与训练集不同的数据分布，能够有效地模拟实际应用中的情况。

②优化模型参数：在每次训练后立即进行验证，可以及时发现模型参数设置是否合理，是否需要调整学习率、正则化系数等参数，以达到更好的性能。这种即时反馈机制有助于更有效地优化模型，避免了长时间训练后才发现参数设置不当的情况，以节省我们训练模型的时间，提高训练效率。

③减少计算资源的浪费：通过在训练过程中定期进行验证，可以在早期阶段就停止无效或低效的训练过程，节省大量的计算资源。

④模型选择：交叉训练和验证可以帮助选择最佳的模型版本。通过比较不同训练轮次后模型在验证集上的性能，可以选择性能最好的模型进行最终的测试或部署。同时也能一定程度上避免模型的过拟合问题。

3）数据增强

在神经网络模型训练过程中，数据增强是一种重要的技术，主要用于通过人工增加训练数据的数量和多样性来提高模型的泛化能力和性能。这种方法可以有效地解决

因数据稀缺或标注成本高昂而导致的训练难题。

在小规模数据集上，数据增强被证明能够有效提升 AI 物体识别场景应用的效果。通过使用不同的增强程度的数据训练模型，可以在达到相同精确率的情况下，显著减少所需原始图像的数量。这种方法不仅提高了模型的泛化能力，还降低了对大量标注数据的依赖，这对于那些难以获取大规模高质量标注数据集的应用场景尤为重要。

由于题目所给的训练集图片数量有限，只有 6000 多张图片，对于训练 yolov8 这样的大型图像检测模型数据量相对较少，所以我们在训练过程中引入数据增强技术，通过定义一系列的图像增强操作，包括色彩变换，仿射变换（裁剪、缩放、镜像翻转），亮度变化，添加高斯噪声、高斯模糊等方式，保证对于同一张原始图像，增强操作前和操作后的标签文件内容不变的同时实现数据量的指数型增长，从而丰富数据库种类的多样性，提高实际检测的鲁棒性，从而有效地提升图像检测精度。

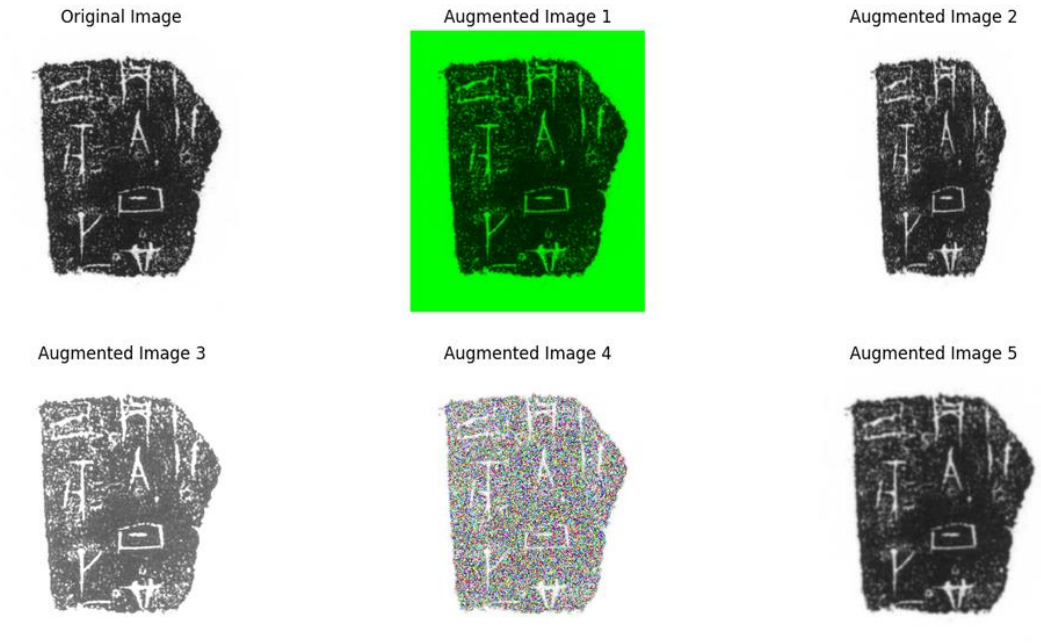


图 5 数据增强示例

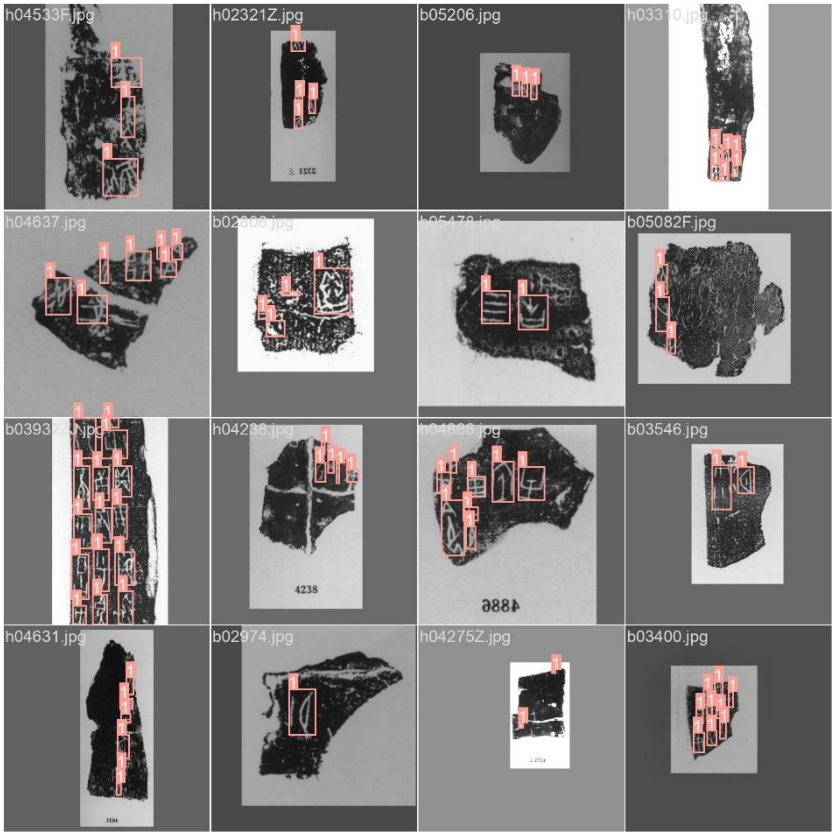


图 9 训练过程中的数据增强

我们在验证集中，只进行了一项转换操作，即将图像转换为张量格式。在验证和测试过程中，我们的目的是评估模型在未知的数据上的性能表现，因此不需要像训练集一样进行图像增强。

4）Momentum 动量优化算法

如果我们使用传统的梯度下降法，训练轨迹会呈现锯齿状，此时会大大延长训练时间。同时由于存在摆动现象，学习率只能设置到较小的数据，才不会因为步伐太大而偏离最小值。于是我们运用 Momentum 动量优化算法进行测算，能够有效减少以上存在的问题发生概率。

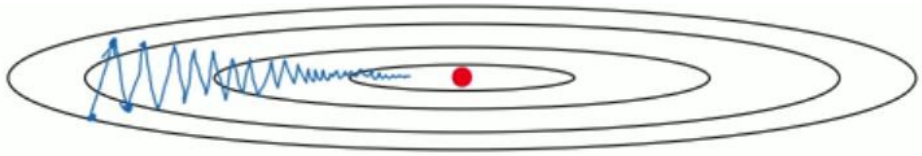


图 10 传统梯度轨迹

我们设第 t 轮迭代的梯度向量为$w^t = [w_1^t \quad w_2^t \quad \cdots \quad w_n^t]^T$，对权重梯度的每个分量的历史取值进行指数移动加权平均，，加入衰减率 $\beta \in (0,1)$，梯度向量的更新公式

如下：

$$\begin{bmatrix} \beta^{t-1}w_1^1 + \beta^{t-2}w_1^2 + \cdots + \beta^0 w_1^t \\ \beta^{t-1}w_2^1 + \beta^{t-2}w_2^2 + \cdots + \beta^0 w_2^t \\ \cdots \\ \beta^{t-1}w_n^1 + \beta^{t-2}w_n^2 + \cdots + \beta^0 w_n^t \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{t-1} \beta^i w_1^i \\ \sum_{i=0}^{t-1} \beta^i w_2^i \\ \cdots \\ \sum_{i=0}^{t-1} \beta^i w_n^i \end{bmatrix} \rightarrow v'_w \tag{2}$$

引入指数加权平均后，v_w的更新公式变为：

$$v_w \rightarrow v'_w = \beta v_w + \frac{\partial Loss}{\partial w} \tag{3}$$

$$v_b \rightarrow v'_b = \beta v_b + \frac{\partial Loss}{\partial b} \tag{4}$$

设 α 表示学习率，则新的梯度更新规则为：

$$\mathrm{w} \rightarrow w' = w - \alpha v_w \tag{5}$$

$$\mathrm{b} \rightarrow b' = b - \alpha v_b \tag{6}$$

通过指数加权平均，纵向的分量基本可以抵消，原因是锯齿状存在一上一下的配对向量，方向基本反向。因为从长期的一段时间来看，梯度优化的大方向始终指向最小值，因此，横向的更新方向基本稳定。新的梯度更新的轨迹如下图所示：

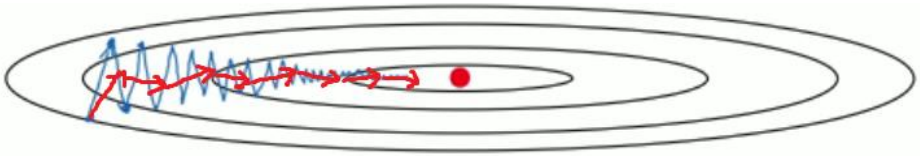


图 6 优化后的梯度轨迹

通过 momentum 算法，我们在训练模型的过程中能够克服小的局部最优解，加速学习过程，从而更快的提升训练速度和模型性能。

5）权重衰减

权重衰减（Weight Decay, WD）是一种在深度学习中广泛使用的正则化技术，旨在通过惩罚大权重参数来防止过拟合[5]。

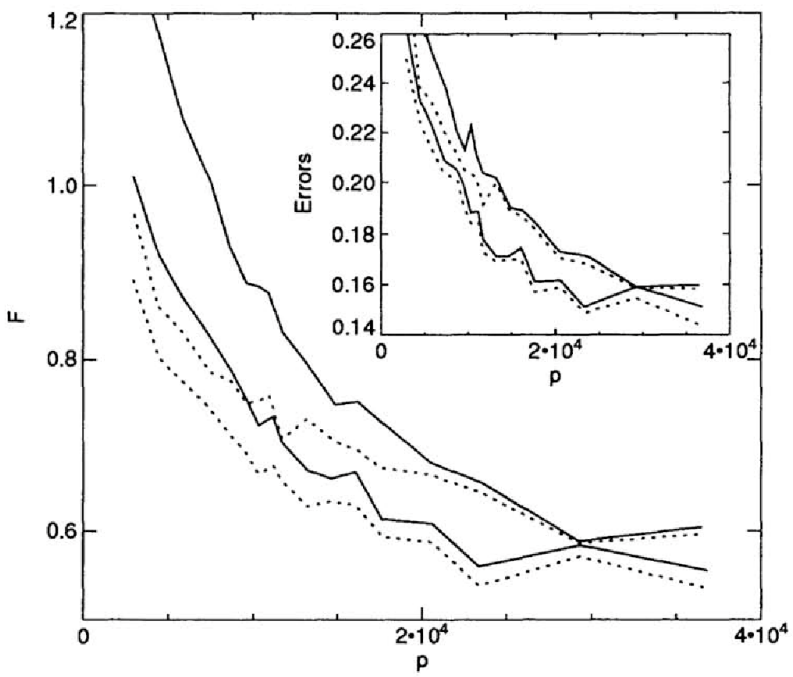


图 7 权重衰减效果对比

我们在损失函数中添加一个与权重平方成正比的项来惩罚数值较大的权重参数来完成权重衰减，一定程度上提高了模型的泛化能力。

5. 2. 4 模型评估

对于模型的训练效果我们采用多个维度的评价指标进行评估。在这里我们选择模型损失值，准确率，召回率，F1 分数等指标对训练后的模型进行评价。

1）模型损失值

在进行预测时，预测值（估计值）与实际值（预期值、参考值、Ground Truth）之间会存在差异，“损失”意味着模型因未能产生预期结果而承受的代价。

损失函数将通过比较模型的预测输出和预期输出来确定模型的性能，进而寻找优化方向。如果二者之间的偏差非常大，则损失值会很大；如果偏差很小或值几乎相同，损失值会非常低。因此，需要使用合适的损失函数，当模型在数据集上进行训练时，该函数可以适当地惩罚模型。我们选用三种损失函数对模型进行评估：

① Bbox Loss（边界框损失）：用于度量目标检测模型中预测的边界框与真实边界框之间的差异的损失函数。Bbox Loss 计算预测与实际坐标之间差异的均方误差(MSE)，计算公式如下：

$$Loss_{bbox} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{7}$$

其中x_i表示真实边界框的坐标，$\hat{x}_i$表示预测边界框的坐标。该损失函数作为优化目标，引导模型在训练过程中减少预测框和真实框之间的差距。

② Cls Loss（分类损失）：用于度量目标检测模型中预测的类别标签与真实类别标签之间的差异的损失函数。交叉熵损失函数是分类任务中常用的一种损失函数，在预测的概率和实际标签相差极大时对于错误预测给出了很大的惩罚。因此，Cls Loss 帮助模型在分类问题中优化其预测，使预测概率分布尽可能接近真实的标签分布，其计算公式为：

$$Loss_{cls} = -\sum_{c=0}^{M} y_{o,c}\log(p_o, c) \tag{8}$$

如果样本 o 属于类别 c，则$y_{o,c} = 1$，反之为 0。p_o是模型预测样本 o 属于类别 c 的概率。

③ DFL Loss（分布聚焦损失）：DFL 以交叉熵的形式，去优化与标签 y 最接近的一左一右 2 个位置的概率，从而让网络更快的聚焦到目标位置及邻近区域的分布。

$$\mathrm{DFL}(S_i, S_{i+1}) = -\big((y_{i+1} - \mathrm{y})\log(S_i) + (\mathrm{y} - y_i)\log(S_{i+1})\big) \tag{9}$$

如下图所示，表行 1 为训练集相关性能指标，表行 2 为验证集相关性能指标。

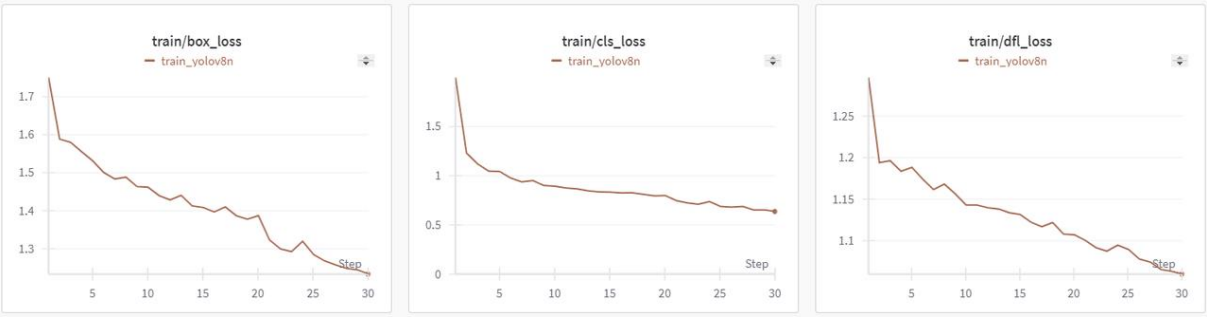


图 8 训练集损失函数曲线

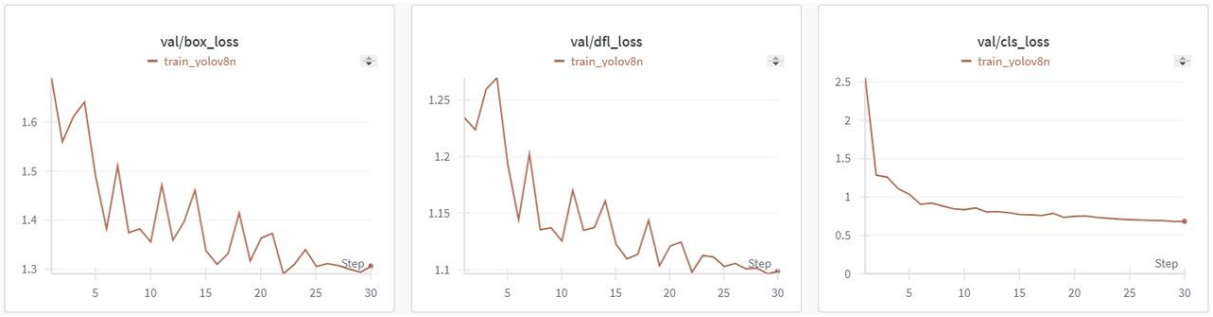


图 9 验证集损失函数曲线

2）准确率（Accuracy）

准确率是分类正确的样本数与总样本数的比例。它是最直观的性能评估指标，计算公式如下：

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{10}$$

3）召回率（Recall）

召回率是分类正确的正例样本数与所有实际正例样本数的比例。它衡量的是模型捕捉正例的能力。召回率的计算公式如下：

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

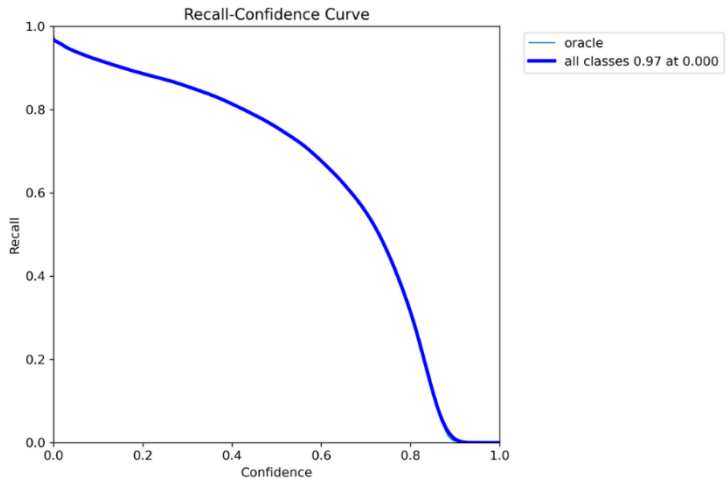


图 10 模型召回率曲线

4）精确率（Precision）

精确率是模型判定为正类别的样本中真正属于正类别的样本数占所有判定为正类别的样本数的比例，即正确预测为正例的样本数除以所有预测为正例的样本数。其计

算公式如下：

$$\text{Precision} = \frac{TP}{TP + FP} \tag{12}$$

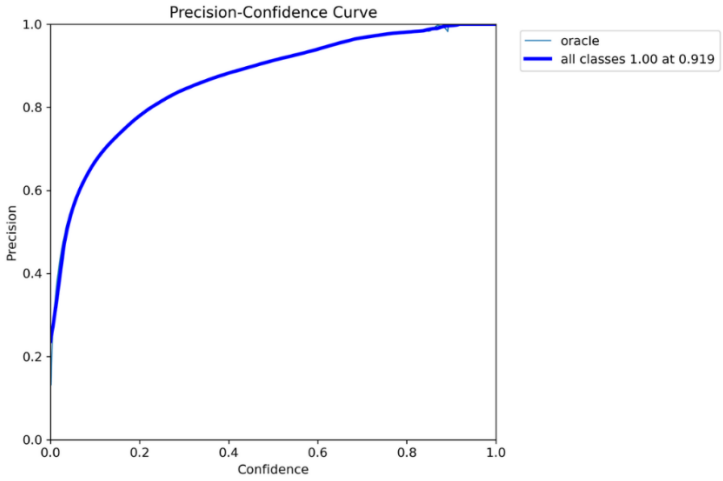


图 11 模型精确率曲线

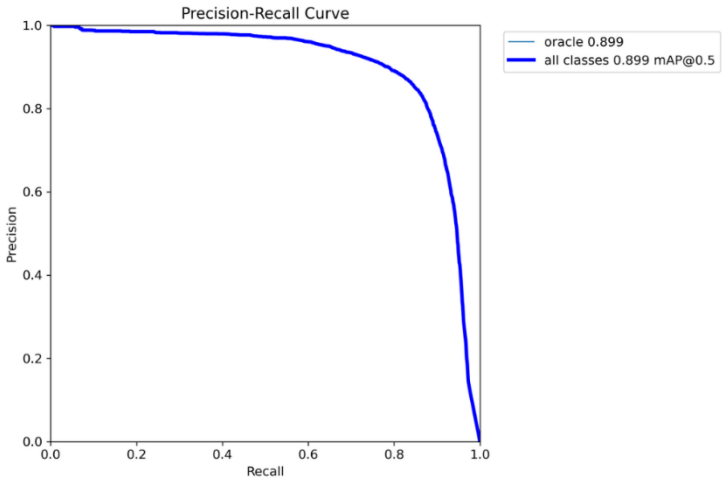


图 12 精确率召回率关系曲线

精确度-召回率曲线将分类器在不同阈值下的精确度和召回率绘制成曲线。曲线下面积（Area Under the Curve，AUC）越大，表示分类器的性能越好。

5）F1 分数（F1 Score）

F1 分数是精确率和召回率的调和平均值，其计算公式如下：

$$\text{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{13}$$

其中，精确率是分类正确的正例样本数与所有被分类为正例的样本数的比例，计算公式如下：

$$\text{Precision} = \frac{TP}{TP + FP} \tag{14}$$

F1 分数的值范围从 0 到 1，其中 1 表示模型的分类性能完美，0 表示模型的分类性能很差。F1 分数适用于需要精确率和召回率都很高的场景，比如需要同时关注正类和负类的检测性能的二分类问题。

在类别不平衡的数据集中，仅使用准确率可能会产生误导，即使模型只是简单地将所有样本分类为数量较多的类别，也可能获得较高的准确率。在这种情况下，召回率和 F1 分数提供了更全面的性能评估。

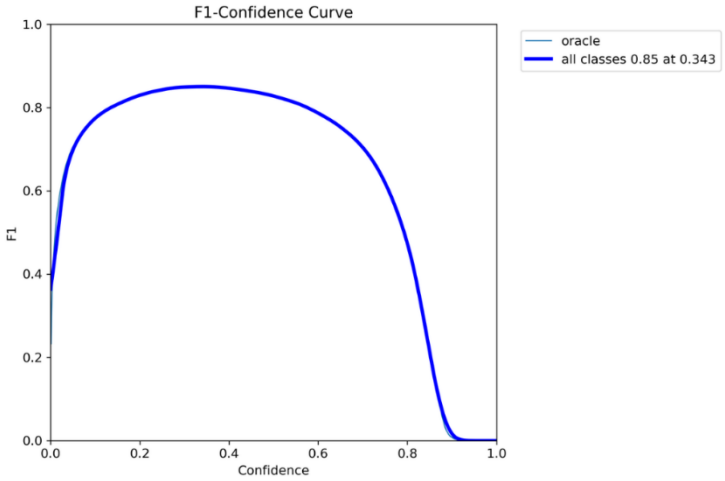


图 13 F1 分数曲线

6）混淆矩阵

混淆矩阵是针对机器学习分类问题的性能度量，其中输出可以是两个或更多类。该表包含 4 种不同的预测值和实际值组合。在图像精度评价中，混淆矩阵主要用于比较分类结果和实际测得值，可以把分类结果的精度显示在一个混淆矩阵里面。

混淆矩阵的每一列代表了预测类别，每一列的总数表示预测为该类别的数据的数目；每一行代表了数据的真实归属类别，每一行的数据总数表示该类别的数据实例的数目，每一列中的数值表示真实数据被预测为该类的数目。

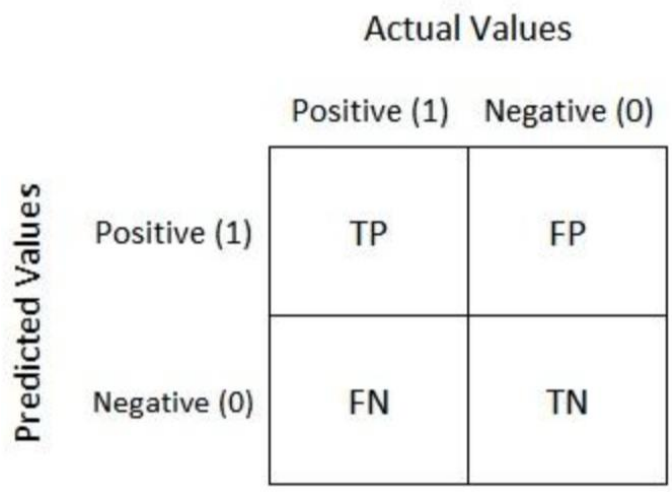


图 14 混淆矩阵

我们得到的混淆矩阵图如下图所示：

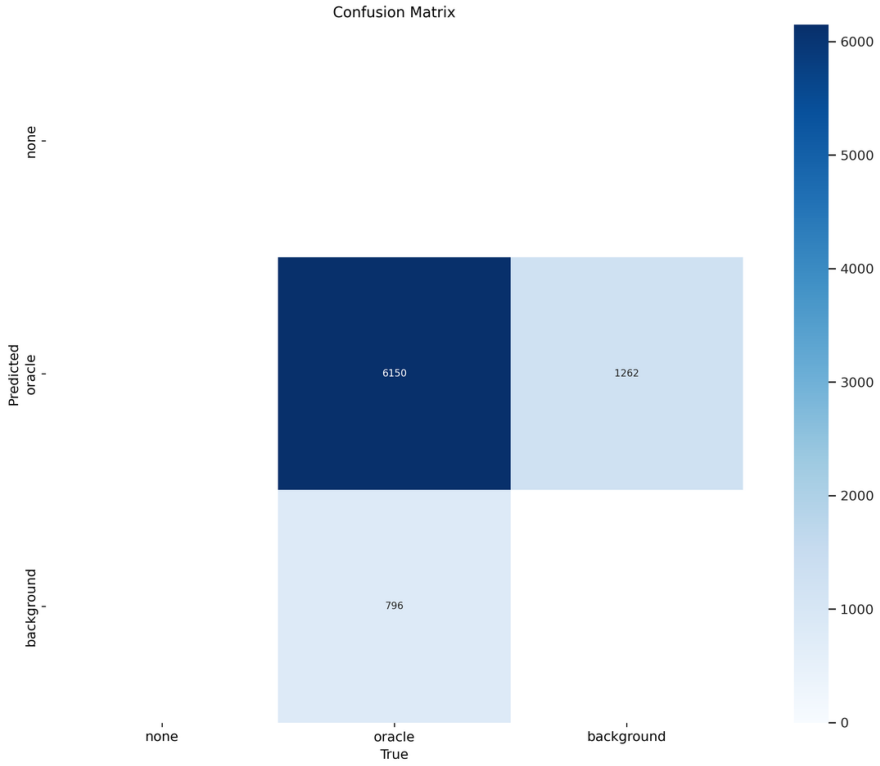


图 20 模型混淆矩阵

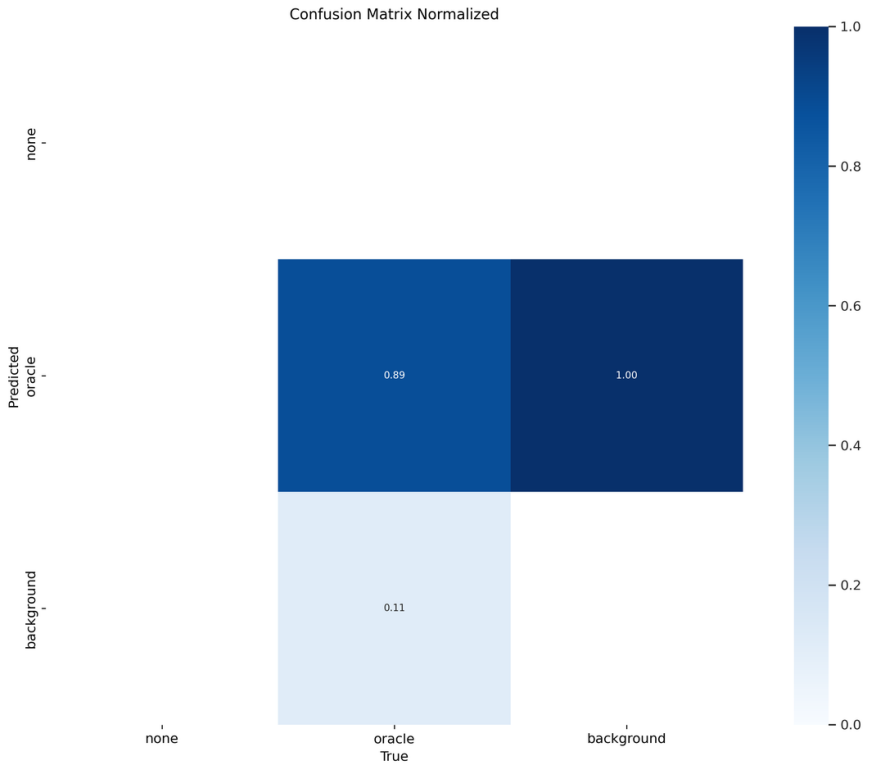


图 15 模型标准化混淆矩阵

根据上述评价指标以及对应结果对模型进行性能评估，我们所构建的模型在训练集和验证集上都表现优秀，能够精确对原始拓片图像进行甲骨文单字分割，并且具备强泛化性能和高鲁棒性。

5.3 问题三：单字分割

5.3.1 模型预测

在问题二的基础上，我们得到了基于甲骨文数据集训练好的模型，在问题三中我们需要利用问题二中建立的甲骨文图像分割模型对附件 3 的甲骨文原始拓片图像进行自动单字分割。因此我们将已经训练好的模型加载遍历附件 3，对其中的图片进行逐个检测，将图片分割结果和标签文件存储下来。

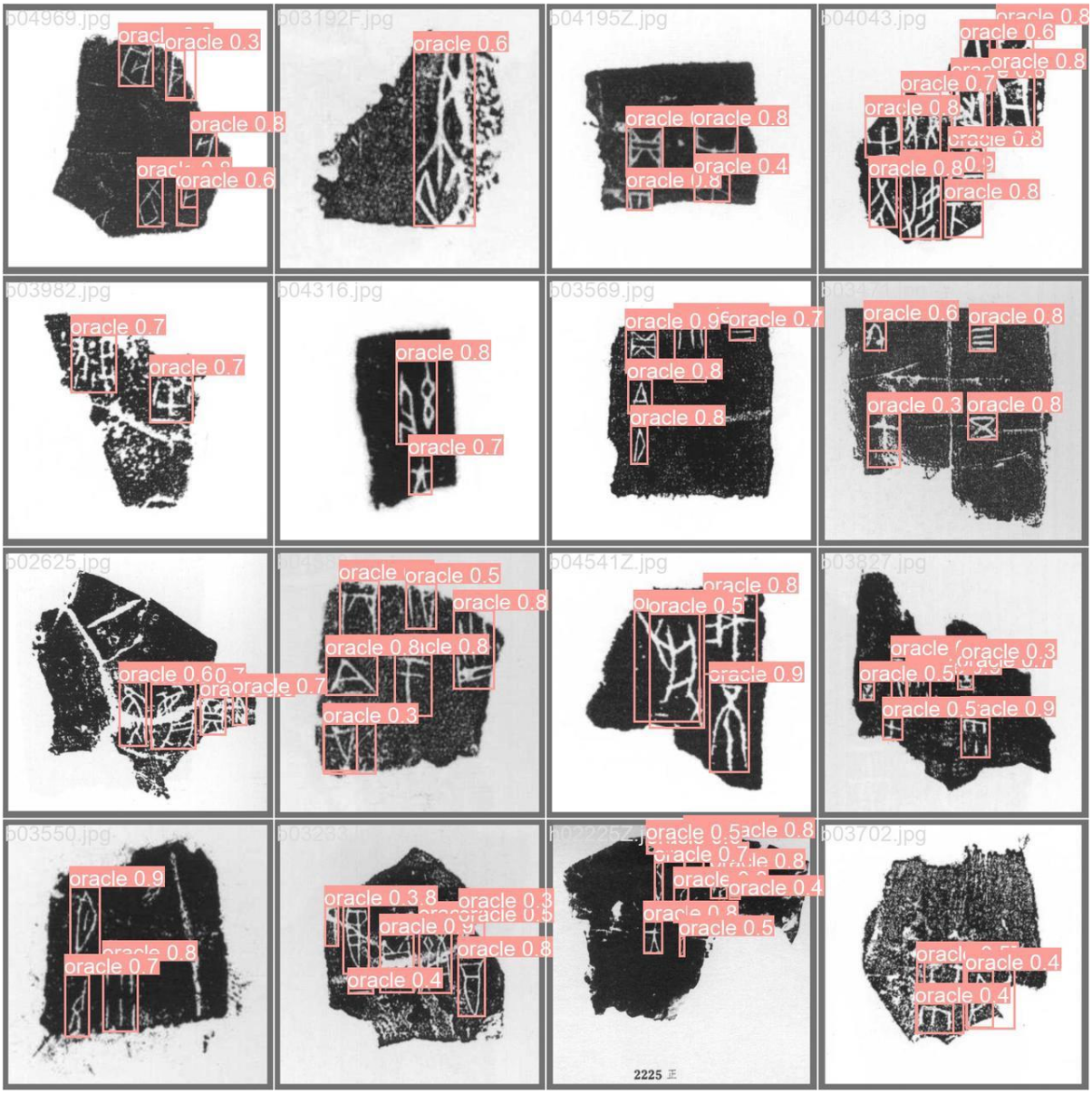


图 16 图片分割结果和标签样例

5.3.2 分割结果格式转换

由于我们使用模型预测分割出来的标签矩形框表示格式是 yolov8 格式，而 Test_results.xlsx 文件中的表示格式是原先的矩形的左上顶点和右下顶点坐标，因此我们需要将标注方式重新转换为原先的格式，并且将同一张图片上的所有矩形框标注合并，按照要求的格式将一张图片对应的标注内容有序写入 Test_results.xlsx 文件中，确保图像名称和标注一一对应。经过格式处理后，最后得到的结果部分展示如下：

表 2 分割结果示例

图像名称	标记
w01906. jpg	［111. 0，197. 0，152. 0，257. 0，1. 0］，［108. 0，283. 0，157. 0，335. 0，1. 0］，［108. 0，356. 0，144. 0，416. 0，1. 0］，［69. 0，192. 0，102. 0，301. 0，1. 0］，［106. 0，27. 0，139. 0，79. 0，1. 0］，［103. 0，77. 0，147. 0，163. 0，1. 0］
020000. jpg	［230. 0，357. 0，258. 0，391. 0，1. 0］，［313. 0，127. 0，339. 0，170. 0，1. 0］，［317. 0，208. 0，340. 0，252. 0，1. 0］，［311. 0，126. 0，339. 0，185. 0，1. 0］，［234. 0，260. 0，260. 0，316. 0，1. 0］，［269. 0，265. 0，310. 0，324. 0，1. 0］
020001. jpg	［225. 0，138. 0，273. 0，257. 0，1. 0］，［223. 0，285. 0，264. 0，332. 0，1. 0］，［275. 0，271. 0，323. 0，385. 0，1. 0］，［139. 0，121. 0，207. 0，240. 0，1. 0］，［228. 0，353. 0，256. 0，416. 0，1. 0］，［369. 0，302. 0，406. 0，395. 0，1. 0］，［322. 0，280. 0，358. 0，381. 0，1. 0］，［150. 0，267. 0，191. 0，329. 0，1. 0］，［150. 0，264. 0，211. 0，334. 0，1. 0］，［281. 0，163. 0，312. 0，226. 0，1. 0］
020002. jpg	［254. 0，188. 0，290. 0，213. 0，1. 0］，［290. 0，179. 0，305. 0，208. 0，1. 0］，［253. 0，142. 0，292. 0，174. 0，1. 0］，［302. 0，103. 0，329. 0，160. 0，1. 0］，［229. 0，52. 0，271. 0，111. 0，1. 0］，［230. 0，48. 0，271. 0，95. 0，1. 0］
020003. jpg	［242. 0，359. 0，270. 0，435. 0，1. 0］，［171. 0，348. 0，231. 0，442. 0，1. 0］，［282. 0，380. 0，316. 0，444. 0，1. 0］，［191. 0，388. 0，230. 0，443. 0，1. 0］，［172. 0，347. 0，218. 0，414. 0，1. 0］

5. 4 问题四：文字识别

5. 4. 1 处理方案

通过分析问题四，我们不难看出，问题四既包含了文字检测任务，又包含了文字识别任务。即我们需要先对目标图片集完成甲骨文文字的检测，然后对具体的甲骨文

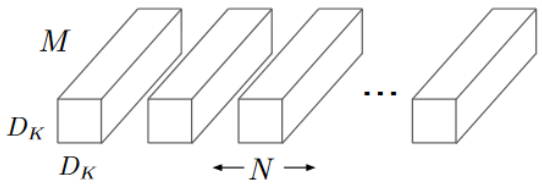
文字进行分类。

为了解决这个问题，我们可以沿用问题二、三训练好的 YOLOv8 模型。在经过了问题二数据集训练的基础上，使用 YOLOv8 模型对问题四目标数据集进行目标检测任务，确定问题四图片中甲骨文存在的区块。

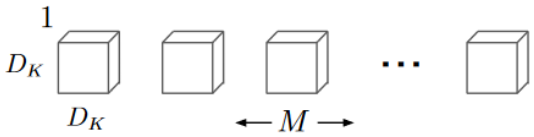
接下来，对于甲骨文存在的每个区块的具体图像，我们通过训练一个 MobileNet 模型，对图像中的文字完成 76 类甲骨文文字的多分类任务。

5. 4. 2 MobileNet 概述

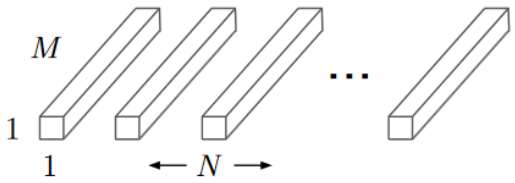
MobileNet 是一种专为移动和嵌入式设备设计的轻量级卷积神经网络（CNN）架构。MobileNet 采用了线性瓶颈和逆转残差模块来进一步优化模型结构，利用自动化机器学习（AutoML）技术和 NetAdapt 算法来优化性能，以适应不同的应用场景。



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

图 173 MobileNet 深度可分离卷积示意图

如上图所示，MobileNet 引入了深度可分离卷积的概念，将标准卷积分解为深度卷积和点卷积两部分，有效减少了模型的复杂度和计算资源的需求。

MobileNet 的骨干网络结构如下所示[6]：

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

图 18 MobileNet 骨干网络结构

5.4.3 具体处理方案

经过讨论，我们提出以下猜想，认为甲骨文具有如此特殊性：

甲骨文中有一部分属于象形文字，自然界中的事物与甲骨文的含义或许存在一定的联系。因此我们载入完成 imagenet 预训练后的 MobileNet 模型来完成问题。

为了完成问题四，我们简单编写了几个自动化工具，完成了如下子任务：

1）测试集数据的预处理

我们先通过问题二训练出的模型，将测试集数据中可能存在甲骨文的地方识别出来，并且通过图像切割将各处子图保存到文件夹中，待分类模型完成分类。

2）生成字典

我们编写了一个字典工具，将训练集中的所有文件夹的中文名称编号，将中文单字与对应的编号存到字典中对应起来。同时将文件夹的名字由中文替换为中文的字典序号，方便使用 OpenCV 进行处理。

3）划分训练集和验证集

为了避免过拟合，同时方便评估模型性能，我们按照 4:1 的数量划分方法将原有

训练集随机划分为新的训练集和验证集。

4）分类模型训练、保存和使用

我们使用深度学习 Keras 框架封装的的 MobileNet 预训练模型进行进一步训练，并且通过训练-验证交叉的方式来观察模型，方便调优。最后使用训练完成的分类模型。

训练和验证模型时的准确率折线图及损失函数折线图如下：

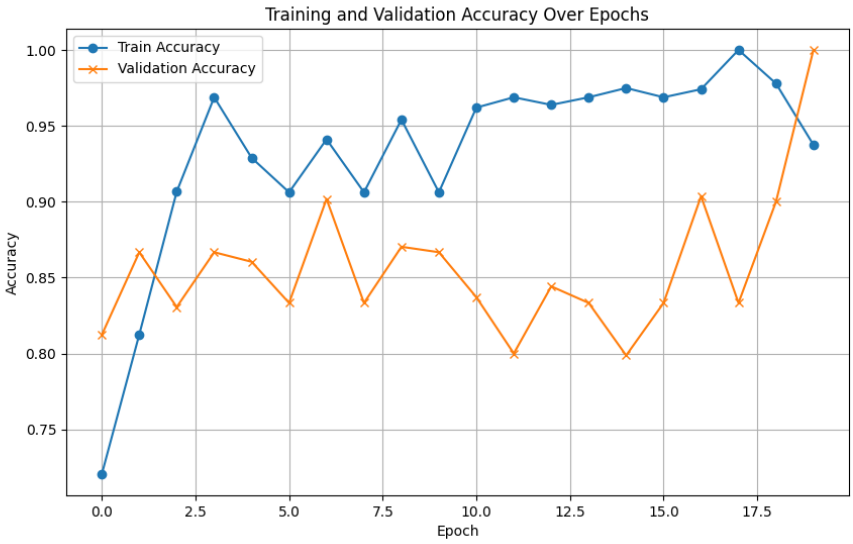


图 19 模型训练和验证的准确率折线图

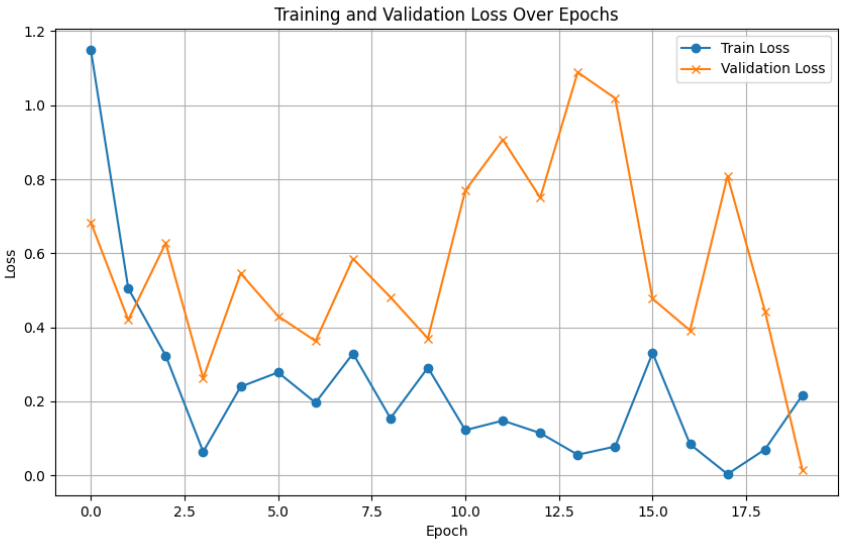


图 20 模型训练和验证的损失值折线图

从上图可以看出，20 个轮次的训练结果可以基本上将分类模型在验证集上的准确率提高到 92%-100%之间。

5.4.4 效果呈现

简单编写一个框体绘制自动化脚本后，我们将问题四的目标数据识别结果标注并且绘制到图片上，以下为两张样图，其余全部识别结果放在支撑材料中：

图 21 文字识别效果呈现图

六、模型评价和推广

6.1 模型的优点

1、 轻量化设计：MobileNet 通过使用深度可分离卷积（Depthwise Separable Convolution）来替代传统的卷积操作，显著减少了模型的参数量和计算量。

2、 可扩展性强，易上手。MobileNet 模型和 YOLOv8 模型都提供了不同大小的模型版本，以满足不同的性能和资源限制需求。卷积核大小等超参数也可以通过非常方便的配置文件进行修改和设置。

3、 高鲁棒性和准确性：YOLOv8 模型引入多尺度特征融合机制，将不同尺度的特征图进行融合，实现了对小目标和远距离目标的检测能力。MobileNet 通过引入残差结构和优化网络结构，通过增加网络层数，能有效提升整体网络的准确率。

4、 推理速度快：YOLOv8 模型采用单阶段检测方法，更加简单、高效，减少了复杂的流程和计算量，提高了检测速度和实时性。MobileNet 模型通过深度可分离卷积的方法和较少的参数量，使得模型在资源受限的环境下能够高效地部署和执行。

6.2 模型的缺点

1、 检测器模型和分类器模型之间缺少数据域适应迁移，没有将训练集的数据在分类器上做域适应处理，两个模型的配合有待改善。

2、 分类器训练集数据较少，可以通过查找数据库进一步扩充或者使用 GAN（对抗生成网络）增强数据。

6.3 模型的推广与改进

1、 得益于 YOLOv8 模块化程度高、使用简单、效果好的特点，模型可以适用于各类目标检测任务。在需要快速识别得到结果的任务上，该模型会有比较好的表现。

2、 MobileNet 适用于硬件环境比较差，计算资源有限的各种场景。除了图像识别和图像分类，在音频处理、情感分析、文本分类上都有广泛的应用空间。

七、参考文献

[1] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, ″You Only Look Once: Unified, Real-Time Object Detection,″ 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[2] Hussain, M. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines* **2023**, *11*, 677.

[3] Liyao Lu, Improved YOLOv8 Detection Algorithm in Security Inspection Image. *arXiv:2308.06452* [cs.CV]

[4] Shun Liu, Jianan Zhang, Ruocheng Song, Teik Toe Teoh, ADA-YOLO: Dynamic Fusion of YOLOv8 and Adaptive Heads for Precise Image Detection and Diagnosis, *arXiv:2312.10099* [cs.CV]

[5] Krogh, Anders and John A. Hertz. "A Simple Weight Decay Can Improve Generalization." *Neural Information Processing Systems* (1991).

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, *arXiv:1704.04861* [cs.CV]

[7] Liu Fang, Li Huabiao, Ma Jin, Yan Sheng, Jin Peiran. Automatic Detection and Recognition of Oracle Rubbings Based on Mask R-CNN. Data Analysis and Knowledge Discovery, 2021, 5(12): 88-97.

八、附录

支撑文件列表
问题 1： data_preprocess.py Figure_1.png Figure_2.png Figure_3.png 第 1 张图的 Hu 矩和对比度.txt 第 2 张图的 Hu 矩和对比度.txt 第 3 张图的 Hu 矩和对比度.txt 问题 2： best.pt　#训练的 YOLOv8 甲骨文图像分割模型 data_augment.py　#数据增强代码 train.ipynb　#模型训练代码 问题 3： best.pt　#训练的 YOLOv8 甲骨文图像分割模型 convert_to_single_line.py output_to_excel.py predict.py Test_results.xlsx　#问题 3 结果 YOLO_TO_JSON.py 问题 4： bmp2jpg.py change_fodername.py

class2chinese.py

class_dict.json　#文字索引字典

divide_class.py

draw_blocks.py

draw_capa.py

jpg2bmp.py

mobileNet_test.py

mobileNet_train.py

photo_cut.py

task4_merge.py

文字识别结果.zip

附录一：问题求解代码

问题一：预处理代码（data_preprocess.py）

```
import cv2
from skimage.feature import graycomatrix, graycoprops
import matplotlib.pyplot as plt

# 1. 读取原始图像
image = cv2.imread('image2.jpg')

# 2. 灰度化
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

image_bil = cv2.bilateralFilter(gray_image, 9, 25, 10, cv2.BORDER_DEFAULT)  # 双边滤波
image_median = cv2.medianBlur(image_bil, 3)  # 中值滤波

binary = cv2.adaptiveThreshold(  # 自适应阈值二值化 adaptive_threshold
    image_median,
    255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
    cv2.THRESH_BINARY, 11, 10)

# 边缘检测
```

```
image_canny = cv2.Canny(binary, 100, 400, 5)


# 特征提取
# 1. 形状特征 - 计算图像的 Hu 矩
moments = cv2.moments(binary)
hu_moments = cv2.HuMoments(moments)

# 2. 纹理特征 - 灰度共生矩阵和对比度
glcm = graycomatrix(image_median, distances=[1], angles=[0], symmetric=True, normed=True)
contrast = graycoprops(glcm, 'contrast')[0, 0]


# 打印提取的特征
print(f'Hu Invariants:\n {hu_moments}')
print(f'Contrast: {contrast}')


# 可视化结果
# cv2.imshow("Original Image", image)
# cv2.imshow("Gray Image", gray_image)
# cv2.imshow("image_bil", image_bil)
# cv2.imshow("image_median", image_median)
# cv2.imshow("binary", binary)
# cv2.imshow("Edges", image_canny)

# 创建一个 2x3 的子图布局
plt.figure(figsize=(12, 8))

# 显示原始图像
plt.subplot(2, 3, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')

# 显示灰度图像
plt.subplot(2, 3, 2)
plt.imshow(gray_image, cmap='gray')
plt.title("Gray Image")
plt.axis('off')
```

```
62.
63. # 显示双边滤波后的图像
64. plt.subplot(2, 3, 3)
65. plt.imshow(cv2.cvtColor(image_bil, cv2.COLOR_BGR2RGB))
66. plt.title("Bilateral Filter")
67. plt.axis('off')
68.
69. # 显示中值滤波后的图像
70. plt.subplot(2, 3, 4)
71. plt.imshow(cv2.cvtColor(image_median, cv2.COLOR_BGR2RGB))
72. plt.title("Median Filter")
73. plt.axis('off')
74.
75. # 显示二值化后的图像
76. plt.subplot(2, 3, 5)
77. plt.imshow(binary, cmap='gray')
78. plt.title("Binary Image")
79. plt.axis('off')
80.
81. # 显示边缘检测结果
82. plt.subplot(2, 3, 6)
83. plt.imshow(image_canny, cmap='gray')
84. plt.title("Edges")
85. plt.axis('off')
86.
87. # 显示子图
88. plt.tight_layout()
89. plt.show()
90. cv2.waitKey(0)
91. cv2.destroyAllWindows()
```

问题二：数据增强代码（data_augment.py）

```
1.  import cv2
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.
5.  # 加载原始图像
6.  image_path = r'./b03507.jpg'
7.  image = cv2.imread(image_path)
8.
9.  # 创建一个空列表来存储增强后的图像
```

```
augmented_images = []

# 色彩变换
augmented_image = cv2.cvtColor(image, cv2.COLOR_BGR2HLS)
augmented_images.append(augmented_image)

# 缩放
scaled_image = cv2.resize(image,None, fx=0.5, fy=0.7, interpolation=cv2.INTER_LINEAR)
augmented_images.append(scaled_image)

# 亮度变化
brightened_image = cv2.convertScaleAbs(image, alpha=1.5, beta=50)
augmented_images.append(brightened_image)

# 添加高斯噪声
noise = np.random.normal(0, 20, image.shape).astype('uint8')
noisy_image = cv2.add(image, noise)
augmented_images.append(noisy_image)

# 高斯模糊
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
augmented_images.append(blurred_image)

# 显示原始图像和增强后的图像
plt.figure(figsize=(15, 10))

# 显示原始图像
plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')

# 显示数据增强后的图像
for i, img in enumerate(augmented_images, start=2):
    plt.subplot(2, 3, i)
    plt.title(f'Augmented Image {i-1}')
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')

plt.show()
```

问题二：模型训练关键代码（train.ipynb）

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')
# model.train()

# model = YOLO('yolov8.yaml')

model.train(data='/kaggle/working/dataset/data.yaml', epochs=30, batch=64)

model.val()
```

问题三：模型预测代码（predict.py）

```
# from ultralytics import YOLO

# model =YOLO("best.pt",task="detect")


# result=model(source=r'D:\PycharmProjects\2024mathorcup\B 题\dataset\test\images\020027.jpg',save=True,save_txt=True)
# # 最后的标签填到 result 表中是浮点数，但是要四舍五入掉小数，因为 label 标签是图像上方框左上角和右下角的坐标，是整数

import os
from ultralytics import YOLO
from os import listdir
from os.path import isfile, join
from PIL import Image

# 文件夹路径
folder_path = r"D:\PycharmProjects\2024mathorcup\B 题\dataset\test\images"

# 获取文件夹中所有图片文件名
image_files = [f for f in listdir(folder_path) if isfile(join(folder_path, f)) and f.lower().endswith(('.jpg'))]

# 加载模型
model = YOLO(r"D:\PycharmProjects\2024mathorcup\B 题 \ 问 题 2&3\best.pt", task="detect")

# # 创建保存结果的文件夹
# output_folder = "output_results"
```

```
# if not os.path.exists(output_folder):
#     os.makedirs(output_folder)

# 遍历图片并进行检测
image_files = [join(folder_path, image_file) for image_file in image_files]

results = model(image_files,save=True,save_txt=True)

for r in results:
    print(r.boxes)
```

问题三：标签文件格式转换代码 1（YOLO_TO_JSON.py）

```
import os
import json
from PIL import Image
import shutil

def get_image_size(image_path):
    with Image.open(image_path) as img:
        width, height = img.size
    return width, height

def convert_yolo_to_absolute(yolo_labels_file, image_width, image_height):
    with open(yolo_labels_file, 'r') as f:
        lines = f.readlines()

    converted_labels = []
    for line in lines:
        class_id, x_center, y_center, bbox_width, bbox_height = map(float, line.st
rip().split())

        # # 将相对坐标转换为绝对坐标
        # x_min = max(0, (x_center - bbox_width / 2) * image_width)
        # y_min = max(0, (y_center - bbox_height / 2) * image_height)
        # x_max = min(image_width, (x_center + bbox_width / 2) * image_width)
        # y_max = min(image_height, (y_center + bbox_height / 2) * image_height)

        # 将相对坐标转换为绝对坐标，并四舍五入保留为浮点数但小数部分为 0
        x_min = round(max(0, (x_center - bbox_width / 2) * image_width), 0)
        y_min = round(max(0, (y_center - bbox_height / 2) * image_height), 0)
```

```
        x_max = round(min(image_width, (x_center + bbox_width / 2) * image_width), 0)
        y_max = round(min(image_height, (y_center + bbox_height / 2) * image_height), 0)


        converted_labels.append((x_min, y_min, x_max, y_max, class_id))

    return converted_labels

def process_folder(input_folder, output_folder, images_folder):
    # 确保输出文件夹存在，如果不存在则创建
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # 获取输入文件夹中所有的标签文件
    label_files = [f for f in os.listdir(input_folder) if f.endswith('.txt')]

    for label_file in label_files:
        # 获取当前标签文件的路径
        label_file_path = os.path.join(input_folder, label_file)

        # 从标签文件名中提取对应的图像文件名（假设标签文件名和图像文件名一一对应）
        image_file_name = os.path.splitext(label_file)[0] + '.jpg'

        # 根据图像文件名获取图像的宽度和高度
        image_file_path = os.path.join(images_folder, image_file_name)
        image_width, image_height = get_image_size(image_file_path)

        # 转换标签文件
        converted_labels = convert_yolo_to_absolute(label_file_path, image_width, image_height)

        # 写入转换后的标签到输出文件夹中
        output_file_path = os.path.join(output_folder, label_file)
        with open(output_file_path, 'w') as f:
            for label in converted_labels:
                f.write(' '.join(map(str, label)) + '\n')

# 示例用法
input_folder = r'D:\PycharmProjects\2024mathorcup\B题\runs\detect\predict\labels'  # 输入文件夹路径，包含所有的标签文件
```

```
output_folder = r'D:\PycharmProjects\2024mathorcup\B题\runs\detect\predict\jsons_result'  # 输出文件夹路径，用于存放转换后的标签文件
images_folder = r'D:\PycharmProjects\2024mathorcup\B题\runs\detect\predict\images'  # 图片文件夹路径，包含所有的图像文件

process_folder(input_folder, output_folder, images_folder)
```

问题三：标签文件格式转换代码 2（convert_to_single_line.py）

```
import os

def convert_labels(input_folder, output_folder):
    # 确保输出文件夹存在，如果不存在则创建
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # 遍历输入文件夹中的所有文件
    for filename in os.listdir(input_folder):
        if filename.endswith(".txt"):  # 只处理 txt 文件
            input_filepath = os.path.join(input_folder, filename)
            output_filepath = os.path.join(output_folder, filename)

            # 打开输入文件并读取内容
            with open(input_filepath, 'r') as f:
                lines = f.readlines()

            # 转换每一行的内容
            converted_lines = []
            for line in lines:
                # 移除换行符并将数字字符串转换为浮点数
                numbers = [float(num) for num in line.strip().split()]
                # 将列表转换为字符串形式
                converted_line = "[" + ", ".join(str(num) for num in numbers) + "]"
                converted_lines.append(converted_line)

            # 将转换后的内容写入输出文件
            with open(output_filepath, 'w') as f:
                f.write(", ".join(converted_lines))

            print(f"Converted {filename}")
```

```
# 输入文件夹路径和输出文件夹路径
input_folder = r"D:\PycharmProjects\2024mathorcup\B题\runs\detect\predict\jsons_result"
output_folder = r"D:\PycharmProjects\2024mathorcup\B题\runs\detect\predict\output"

# 调用函数进行批量处理
convert_labels(input_folder, output_folder)
```

问题三：结果输出代码（output_to_excel.py）

```
import os
from openpyxl import Workbook, load_workbook

# 指定文件夹路径和 Excel 文件路径,要填绝对路径，否则 excel 写不进去
folder_path = r'D:\PycharmProjects\2024mathorcup\B题\runs\detect\predict\output'
excel_path = r'D:\PycharmProjects\2024mathorcup\B 题\问题 2&3\Test_results.xlsx'

# 加载现有的 Excel 文件
wb = load_workbook(excel_path)
ws = wb.active

# 指定开始写入的行数
start_row = 3

# 遍历文件夹中的所有 txt 文件
for file_name in os.listdir(folder_path):
    if file_name.endswith('.txt'):
        file_path = os.path.join(folder_path, file_name)
        with open(file_path, 'r', encoding='utf-8') as file:
            # 读取文件内容的第一行
            first_line = file.readline().strip()
            # # 将文件名和第一行内容写入 Excel 的指定单元格位置
            # ws.cell(row=start_row, column=2, value=file_name)
            ws.cell(row=start_row, column=2, value=first_line)
            # 移动到下一行
            start_row += 1

# 保存 Excel 文件
wb.save(excel_path)
```

问题四：分类训练与预测代码（task4_merge.py）

```python
import numpy as np
import pandas as pd
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV3Small
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import img_to_array, load_img

train_data_dir = r'C:\Users\linyiwu\Desktop\Mathorcup2024\trainset'
test_data_dir = r'C:\Users\linyiwu\Desktop\Mathorcup2024\cut_photos_bmp'
output_csv = 'predictions.csv'
input_shape = (224, 224)
num_classes = 76  # 76 个类别
batch_size = 16

# 定义 ImageDataGenerator
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2  # 设置验证集比例为 20%
)

# 从目录中加载数据并划分训练集和验证集
train_generator = datagen.flow_from_directory(
    r'C:\Users\linyiwu\Desktop\Mathorcup2024\trainset',  # 替换为你的数据集目录
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'  # 指定加载训练集数据
)

validation_generator = datagen.flow_from_directory(
    r'C:\Users\linyiwu\Desktop\Mathorcup2024\trainset',  # 替换为你的数据集目录
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'  # 指定加载验证集数据
)

test_datagen = ImageDataGenerator(rescale=1. / 255)

```

```
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=input_shape,
    batch_size=batch_size,
    class_mode=None,
    shuffle=False
)


base_model = MobileNetV3Small(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_generator, epochs=10, validation_data=validation_generator)

# test_preds = model.predict(test_generator, verbose=1)
# predicted_class_indices = np.argmax(test_preds, axis=1)

# labels = (train_generator.class_indices)
# labels = dict((v, k) for k, v in labels.items())
# predictions = [labels[k] for k in predicted_class_indices]

# filenames = test_generator.filenames
# results = pd.DataFrame({"Filename": filenames, "Predictions": predictions})
# results.to_csv(output_csv, index=False)

# 加载并预测单个文件夹中的图片
images_folder = os.listdir(test_data_dir)
predictions_list = []
for img_name in images_folder:
```

```
    img_path = os.path.join(test_data_dir, img_name)
    img = load_img(img_path, target_size=input_shape)
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # 归一化
    pred = model.predict(img_array)
    predicted_class = np.argmax(pred, axis=1)[0]
    predictions_list.append((img_name, predicted_class))

# 保存预测结果到 CSV 文件
results = pd.DataFrame(predictions_list, columns=["Filename", "Predicted_Class_Index"])
results.to_csv(output_csv, index=False)
```

问题四：裁剪图片代码（photo_cut.py）

```
import os
import cv2

def crop_images(image_folder, annotation_folder, destination_folder):
    # 获取图片文件夹中的文件列表
    image_files = os.listdir(image_folder)

    # 遍历图片文件夹中的每个文件
    for image_file in image_files:
        image_name, _ = os.path.splitext(image_file)
        # 构建对应的验证框文件路径
        annotation_file_path = os.path.join(annotation_folder, image_name + '.txt')

        # 读取对应的图片
        image_path = os.path.join(image_folder, image_file)
        if not os.path.isfile(image_path):
            continue
        image = cv2.imread(image_path)

        # 如果对应的验证框文件不存在，则跳过
        if not os.path.isfile(annotation_file_path):
            continue

        # 读取验证框文件中的信息
        with open(annotation_file_path, 'r') as f:
```

```
            lines = f.readlines()

        i = 0
        # 遍历每行信息
        for line in lines:
            # 解析边界框信息
            class_id, x_center, y_center, width, height = map(float, line.strip().split())

            # 计算边界框在图片中的坐标
            image_height, image_width, _ = image.shape
            x1 = int((x_center - width / 2) * image_width)
            y1 = int((y_center - height / 2) * image_height)
            x2 = int((x_center + width / 2) * image_width)
            y2 = int((y_center + height / 2) * image_height)

            # 切割图片
            cropped_image = image[y1:y2, x1:x2]

            # 保存切割后的图片到目标文件夹
            destination_path = os.path.join(destination_folder, f"{image_name}_{int(i)}.jpg")
            cv2.imwrite(destination_path, cropped_image)
            i += 1

if __name__ == "__main__":
    image_folder = r"C:\Users\linyiwu\Desktop\Mathorcup2024\testset"  # 图片文件夹路径
    annotation_folder = r"C:\Users\linyiwu\Desktop\Mathorcup2024\labels"  # 验证框文件夹路径
    destination_folder = r"C:\Users\linyiwu\Desktop\Mathorcup2024\cut_photos"  # 目标文件夹路径

    crop_images(image_folder, annotation_folder, destination_folder)
```

问题四：划分数据集代码（divide_class.py）

```
import os
from PIL import Image
import random
import json

```

```
# 存储类别名称和对应的分类编号
class_dict = {}

# 遍历文件夹
base_path = "甲骨文智能识别中原始拓片单字自动分割与识别研究/4_Recognize/训练集"  # 修改为你的数据集文件夹路径
class_id = 1  # 从 1 开始编号类别
for folder_name in os.listdir(base_path):
    folder_path = os.path.join(base_path, folder_name)
    if os.path.isdir(folder_path):
        class_dict[folder_name] = class_id
        class_id += 1

# 打印类别名称和对应的分类编号
for class_name, class_id in class_dict.items():
    print(f"Class Name: {class_name}, Class ID: {class_id}")

# 这里可以将 class_dict 保存到文件中，以便后续使用
output_file = "甲骨文智能识别中原始拓片单字自动分割与识别研究/task4/class_dict.json"  # 修改为你要保存的文件路径
with open(output_file, "w") as f:
    json.dump(class_dict, f)

print(f"Class dictionary saved to {output_file}")

# 定义划分比例
train_ratio = 0.8  # 80% 的数据作为训练集
test_ratio = 1.0 - train_ratio  # 剩余的数据作为测试集

# 遍历文件夹
base_path = "甲骨文智能识别中原始拓片单字自动分割与识别研究/4_Recognize/训练集"  # 修改为你的数据集文件夹路径
train_output_path = "甲骨文智能识别中原始拓片单字自动分割与识别研究/task4/train"  # 修改为你的训练集输出文件夹路径
test_output_path = "甲骨文智能识别中原始拓片单字自动分割与识别研究/task4/test"  # 修改为你的测试集输出文件夹路径

# 创建输出文件夹
os.makedirs(train_output_path, exist_ok=True)
os.makedirs(test_output_path, exist_ok=True)

# 存储图像数据和对应的分类编号
```

```
data = []

for folder_name in os.listdir(base_path):
    folder_path = os.path.join(base_path, folder_name)
    if os.path.isdir(folder_path):
        class_name = folder_name
        class_id = class_dict.get(class_name, -1)  # 获取类别编号，如果不存在则为-1
        if class_id != -1:
            images = []
            for filename in os.listdir(folder_path):
                image_path = os.path.join(folder_path, filename)
                if filename.endswith(".jpg") or filename.endswith(".png") or filename.endswith(".bmp"):  # 根据你的图片格式修改
                    try:
                        image = Image.open(image_path)
                        images.append((image, class_id, filename))
                    except Exception as e:
                        print(f"Error processing image {image_path}: {e}")
            # 打乱顺序
            random.shuffle(images)
            # 划分数据
            train_size = int(len(images) * train_ratio)
            train_data = images[:train_size]
            test_data = images[train_size:]
            # 将数据添加到总体数据集中
            data.extend(train_data)
            data.extend(test_data)
            # 将数据保存到对应的文件夹中
            for img, img_class, img_filename in train_data:
                img_name = f"{class_id}_{img_filename}"
                img_path = os.path.join(train_output_path, img_name)
                img.save(img_path)
            for img, img_class, img_filename in test_data:
                img_name = f"{class_id}_{img_filename}"
                img_path = os.path.join(test_output_path, img_name)
                img.save(img_path)

print("数据集划分完成并保存到对应文件夹中")
```

问题四：绘制文字识别框代码（draw_blocks.py）

```
import os
```

```
import cv2
import pandas as pd
from PIL import ImageFont,ImageDraw,Image
import numpy as np

def read_csv(csv_file):
    # 读取 CSV 文件，返回文件名、行号和类别名的对应关系
    df = pd.read_csv(csv_file)
    return df.values.tolist()

def read_txt(txt_file, row_number):
    # 从 TXT 文件中获取指定行号的验证框信息
    with open(txt_file, 'r') as f:
        lines = f.readlines()
        line = lines[int(row_number)]  # 行号从 0 开始
        # 解析 YOLO 格式的验证框信息（假设格式为：class_name x_center y_center width height）
        class_name, x_center, y_center, width, height = line.strip().split()
        return class_name, float(x_center), float(y_center), float(width), float(height)

def draw_bbox(image, class_name, x_center, y_center, width_chuan, height_chuan, output_folder, filename):
    # Convert normalized coordinates to pixel coordinates
    height, width, _ = image.shape
    x_center_pixel = int(x_center * width)
    y_center_pixel = int(y_center * height)
    width_pixel = int(width * width_chuan)
    height_pixel = int(height * height_chuan)

    # Calculate bounding box coordinates
    x_min = x_center_pixel - (width_pixel // 2)
    y_min = y_center_pixel - (height_pixel // 2)
    x_max = x_center_pixel + (width_pixel // 2)
    y_max = y_center_pixel + (height_pixel // 2)

    # Draw bounding box
    cv2.rectangle(image, (x_min, y_min), (x_max, y_max), (0, 0, 255), 2)

    # Draw class name using PIL
    font_path = "simsun.ttc"
    font_size = 30
```

```
    font_color = (255, 0, 0)
    font = ImageFont.truetype(font_path, font_size)
    img_pil = Image.fromarray(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    draw = ImageDraw.Draw(img_pil)
    draw.text((x_min, y_min - font_size), class_name, font=font, fill=font_color)


    # Convert PIL image back to OpenCV format and save
    image_with_text = cv2.cvtColor(np.array(img_pil), cv2.COLOR_RGB2BGR)
    cv2.imwrite(output_folder + '/' + filename + '.jpg', image_with_text)

def main():
    src_folder = r'C:\Users\linyiwu\Desktop\Mathorcup2024\src_data\labels'  # 存放txt文件的文件夹
    photo_folder = r'C:\Users\linyiwu\Desktop\Mathorcup2024\src_data\testset'  # 存放图片的文件夹
    csv_file = 'predictions.csv'  # CSV文件名
    output_foder = r'C:\Users\linyiwu\Desktop\Mathorcup2024\src_data\testset'  # 输出图片的文件夹
    data = read_csv(csv_file)

    for filename, class_number, class_name in data:
        filename, row_number_and_ext = filename.split('_')
        row_number = row_number_and_ext.split('.')[0]
        txt_file = os.path.join(src_folder, filename + '.txt')
        photo_file = os.path.join(photo_folder, filename + '.jpg')
        # print(f"Processing {filename}")
        # print(f"Row_number: {row_number}")
        if os.path.exists(txt_file) and os.path.exists(photo_file):
            image = cv2.imread(photo_file)
            _, x_center, y_center, width, height = read_txt(txt_file, row_number)

            # print(f"x_center: {x_center}, y_center: {y_center}, width: {width}, height: {height}")
            draw_bbox(image, class_name, x_center, y_center, width, height, output_foder, filename)
            # cv2.imshow(image)

            # print(f"Processed {filename}")

if __name__ == "__main__":
```

```
    main()
```

问题四：绘制准确率和损失折线图代码（draw_capa.py）

```
import pandas as pd
import matplotlib.pyplot as plt

# 从文本文件读取数据
data = pd.read_csv(r'task4 性能参数.txt', delimiter='\s+')

# 绘制训练准确率和验证准确率的折线图
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Train_Accuracy'], label='Train Accuracy', marker='o')
plt.plot(data.index, data['Validation_Accuracy'], label='Validation Accuracy', marker='x')
plt.title('Training and Validation Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# 绘制训练损失和验证损失的折线图
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Train_Loss'], label='Train Loss', marker='o')
plt.plot(data.index, data['Validation_Loss'], label='Validation Loss', marker='x')

plt.title('Training and Validation Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```