国际象棋的计算机视觉应用

----计算机视觉第 n 小组

小组成员：周财进、韦东生、张令侨、王宝印、王文秋

一、绪论

1. 问题定义与研究动机

在信息化与智能化不断发展的背景下，如何将现实世界的感知信息快速、准确地转化为数字化表达，已成为计算机视觉领域的重要研究课题。国际象棋作为最典型的规则明确、状态空间清晰的策略博弈游戏，为图像识别与人工智能决策的融合提供了良好平台。

传统国际象棋 AI 系统通常依赖用户手动输入棋局，而在实际应用场景中，如线下比赛、实体棋盘对弈、教学演示等，若能通过相机直接识别棋盘状态并自动生成数字化棋局，不仅能够提升系统的交互性与实时性，也能在教育、娱乐、竞技等场合拓展应用边界。

本项目正是在这一背景下开展，目标是构建一个端到端的“实体棋盘识别与智能决策系统”，通过前端图像识别、后端棋局建模与 AI 决策推理三者协同，实现真实棋局的自动数字化与 AI 智能辅助建议展示。

2. 项目内容与系统目标

项目核心任务是实现一个具备实际部署能力的系统：通过识别用户上传的实体棋盘图像，提取出 64 个格子中各棋子的位置与类别，并自动生成标准化棋局状态图。随后，系统将该状态输入至国际象棋 AI 引擎中，输出下一步建议的走法，并在网页前端以文本与图形方式同步展示，形成完整的“图像—状态—决策”闭环。

3. 方法概况与技术框架

本系统采用 Ultralytics 发布的 YOLOv8 目标检测框架 [2]，作为棋子识别的核心模型。该模型具备轻量、高精度、端到端输出的优点，已在本项目中针对国际象棋棋子类别进行训练和微调，能精准检测图像中棋子的边界框、类别编号与置信度信息。

检测结果被解析并映射至标准 8×8 棋盘坐标系，构建当前棋局状态的二维数组表示。系统结合 Unicode 字符与图像绘制技术，实时在网页前端呈现当前棋盘状态。

随后，系统调用内置或外部接入的 AI 引擎 Stockfish [6]，输入当前棋局进行策略推理，输出推荐走法，并通过图形高亮和文本解释双重方式辅助用户理解 AI 决策逻辑，实现从图像感知到策略引导的闭环式交互。

二、相关工作

近年来，随着计算机视觉和人工智能领域的飞速发展，基于图像的棋盘识别与智能博弈推理逐渐成为图像理解、人机交互及智能决策等多学科交叉研究的热点问题。已有的研究主要聚焦于以下三个关键技术模块：实体棋盘图像识别、虚拟棋盘重建与状态建模、AI 博弈引擎集成与策略推理。

1. 实体棋盘图像识别

国际象棋图像识别的第一步是对棋盘及其上的棋子进行准确检测。早期方法通常依赖图像处理技术，如边缘检测、颜色分割、形状模板匹配等，虽然在理想环境下有效，但在自然拍摄图像中容易受到光照变化、遮挡、角度倾斜等因素干扰，鲁棒性较低。

近年来，基于深度学习的目标检测方法广泛应用于棋子识别任务。典型模型包括 Faster

R-CNN、SSD、YOLO 等。其中，YOLO 系列因其端到端结构和高实时性在实际应用中表现出色。YOLOv8 是该系列的最新版本，在检测精度、速度与模型结构优化上均有显著提升 [2]。

在目标检测框架中，模型输入图像 I 后，输出一系列预测框集合$\{(b_k, p_k, c_k)\}$，其中：$b_k = (x_k, y_k, w_k, h_k)$表示目标在图像中的位置；$p_k$ 为置信度；c_k 为预测的棋子类别编号。

目标检测损失函数通常包括三部分：位置误差（例如 Smooth L1 损失）、分类误差（CrossEntropy）、置信度误差。Smooth L1 损失定义如下：

$$\text{SmoothL1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}$$

训练目标是最小化：

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda_1 \mathcal{L}_{\text{conf}} + \lambda_2 \mathcal{L}_{\text{box}}$$

YOLOv8 在 ChessVision 数据集和 Rendered Chess Game State Images 数据集 [1] 上表现出良好精度和鲁棒性，适用于棋子类别识别、格子定位等任务。

2. 虚拟棋盘重建与状态建模

实体图像中的棋盘往往因拍摄角度或透视畸变而无法直接分格，因此需通过图像几何校正将其还原为标准视角下的 8×8 网格。

最常见方法是单应性变换。设棋盘四角在图像空间中的坐标为$\mathbf{x}_i$，理想棋盘坐标为$\mathbf{X}_i$，则存在单应矩阵 H使得：

$$\mathbf{x}_i \sim H\mathbf{X}_i$$

通过 OpenCV 提供的 findHomography 和 warpPerspective 函数可以实现棋盘的透视校正：

```
H, _ = cv2.findHomography(src_pts, dst_pts)
warped = cv2.warpPerspective(image, H, (width, height))
```

透视校正将倾斜棋盘图像变换为俯视图，为棋盘网格划分与坐标映射奠定基础。

校正后的棋盘被划分为 64 个格子，每个格子坐标中心计算方式为：

$$c_x = \left(i + \frac{1}{2}\right) \cdot \frac{W}{8}, \quad c_y = \left(j + \frac{1}{2}\right) \cdot \frac{H}{8}$$

在结构表达上，许多系统将当前棋局转化为 Forsyth-Edwards Notation (FEN) 字符串，用于与 AI 引擎通信。

已有开源项目如 ChessVision.ai、ChessFEN 等均采用类似方法，但在遮挡、棋子缺失、图像模糊等场景下仍面临建模不完整、误差传播等问题。

3. AI 博弈引擎集成与策略推理

在获得结构化棋局后，将其输入至 AI 博弈引擎以输出下一步建议是实现智能对弈辅助

的关键。当前最常用的引擎为 Stockfish [6]，它基于 Alpha-Beta 剪枝、神经网络评估函数（NNUE）与开局库，可实现秒级响应并给出评分与推荐走法。

以 FEN 字符串为输入，Stockfish 可返回如下输出：

info depth 20 score cp 38 pv e2e4 e7e5 g1f3 ...

bestmove e2e4 ponder e7e5

其中 cp 38 表示当前局面对白方有 0.38 分优势，bestmove 表示 AI 推荐走法为 e2→e4。

然而，在前端图像识别误差不可避免的前提下，AI 引擎对“不完美棋局输入”的容错能力仍是研究难点。目前较少研究探讨如何使 AI 系统感知棋盘识别置信度，并据此做出更鲁棒的策略判断。

三、方法的具体细节

***首先对项目的实现手段、算法大概讲述：

一、系统实现总体策略：

我们打算构建的国际象棋识别与分析系统主要由三部分组成：

基于 YOLOv8 的棋子检测网络；

图像到棋局状态的转换逻辑（FEN 编码）；

基于 Stockfish 引擎的智能博弈推理模块。

系统通过图像处理与深度学习提取棋盘状态，借助规则编码映射棋子布局，通过经典 AI 引擎做出决策。

二、棋子识别算法（YOLOv8）

实现手段：采用 Ultralytics 的 YOLOv8 模型，它是目标检测领域的先进算法，具备速度快、精度高、结构灵活等优点。我们训练了一个 13 类（6 种白棋 +6 种黑棋 + 空格）分类模型。

YOLOv8 算法核心：YOLOv8 采用的是单阶段目标检测架构，在一张图像中直接回归多个边界框与类别信息。YOLO 的预测是在多个预设 anchor 上进行的，它通过卷积神经网络（CNN）提取图像特征，再用回归层预测目标框与分类结果。

训练过程：

数据增强：随机裁剪、旋转、颜色抖动

标签生成：每张图配套一个 YOLO 格式标签文件（见 gen_yolo_labels.py）

损失函数：结合了 CIoU（边框回归）损失 + 分类损失 + 对象存在性损失

三、棋局状态转换算法（图像 → FEN）

目标：将 YOLO 输出结果还原为国际象棋的 FEN 编码，这是一种记录棋子在棋盘上分布的标准字符串格式。例如：rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

实现策略

首先将图像划分为 8x8 网格；

将每个目标检测结果的中心点定位到某个格子；

构建一个 8x8 矩阵，将识别到的棋子填入；

将矩阵按行转换为 FEN 字符串（空格用数字表示）；

添加轮次、走棋方等补充字段。

伪代码：

```
grid = [['' for _ in range(8)] for _ in range(8)]
for det in yolo_outputs:
    row, col = locate_grid(det["x"], det["y"])
    grid[row][col] = class_id_to_fen_symbol(det["class_id"])
fen = convert_to_fen_string(grid)
```

四、棋局推理与走法推荐（Stockfish）

Stockfish 简介：Stockfish 是当前世界领先的开源国际象棋引擎，采用 Alpha-Beta 剪枝 + 评估函数 + 深度搜索。其核心思想是：

搜索棋局的博弈树；

根据启发式评分函数估值；

使用剪枝策略减少冗余探索。

接口调用方式

在 Python 中，我们通过 chess.engine.SimpleEngine.popen_uci（）调用编译好的 stockfish.exe 文件。

```
from stockfish import Stockfish
sf = Stockfish(path="stockfish.exe")
sf.set_fen_position(current_fen)
move = sf.get_best_move()
```

启发式搜索过程（简要）：

遍历所有合法走法；

使用评分函数估值子局面；

递归搜索更深层次；

使用 Alpha-Beta 剪枝减少不必要节点；

返回评分最高的走法路径。

*****根据项目文件对大概流程讲述：**

在项目文件里，核心文件如 gen_yolo_labels.py、yolov8_infer.py、app.py 等都在 ds/ 子目录下。以系统实现的四个关键步骤为核心，详细描述：

数据预处理与标签生成：

在构建 YOLOv8 国际象棋识别模型之前，首先需要准备格式规范的数据集。YOLO 模型训练所需的标签采用一种标准格式，即每行包含目标类别编号、目标框中心点的归一化坐标 (x, y)，以及宽高 (w, h)。为了从原始国际象棋棋盘图像中自动生成这些标签，我们使用 gen_yolo_labels.py 和 rectify_chess_dataset.py 脚本来完成这一任务。

核心流程包括两个部分：棋盘网格划分和棋子标注提取。首先根据输入图像的尺寸将棋盘划分成 8x8 共 64 个等大小的格子，然后识别每个格子中是否包含棋子，如果包含，则记录该棋子的类别及其相对于整张图像的中心坐标和尺寸。如下为伪代码：

```
for image in dataset:
```

```
grid = split_to_8x8(image)
for (i, j) in grid:
    if detect_piece(image[i][j]):
        label = get_class_id(piece_type)
        bbox = compute_center_and_size(i, j)
        write_to_file(label, bbox)
```

最终生成的 .txt 标签文件与图像文件一一对应，可直接用于 YOLOv8 的训练命令。通过这一阶段，我们确保了数据集具备良好的结构性与准确性，直接影响到模型性能。

棋子检测与预测：

当模型训练完成后，需要在部署阶段对任意棋盘图像进行预测。此过程由 yolov8_infer.py 脚本完成。它通过调用 Ultralytics YOLOv8 模型进行推理，并返回图像中每个棋子的类别、位置和置信度。该模型输出的检测结果为多个边界框，每个框含有如下信息：类别 cic_i、中心点坐标 (xi,yi)(x_i, y_i)、宽度 wiw_i、高度 hih_i，以及置信度 confi\text{conf}_i。

模型输出如下结构：

```
[
  {"class": 12, "name": "black_pawn", "bbox": [0.45, 0.78, 0.1, 0.12]},
  {"class": 5, "name": "white_queen", "bbox": [0.12, 0.63, 0.08, 0.10]}
]
```

这些边界框坐标都是归一化的，后续需要将其映射到实际棋盘格子中。具体做法是：将图像按比例划分为 8x8 网格，对每个 bbox 的中心坐标确定其所在的格子行列号。例如：

```
row = int(bbox_center_y * 8)
col = int(bbox_center_x * 8)
```

该阶段的输出为棋盘上的所有棋子及其坐标信息，是系统中最关键的视觉感知模块。

棋盘状态还原与 Stockfish 接口：

在检测到所有棋子的位置和类别之后，下一步是将当前棋局状态还原为标准的 FEN（Forsyth-Edwards Notation）格式，这是国际象棋中用于描述局面的字符串表示。系统中的 app.py 负责这一逻辑实现。我们首先构建一个空的 8x8 网格，然后根据检测结果将每个棋子填入对应的位置。填充完毕后将该网格转换为 FEN 字符串。伪代码如下：

```
grid = [['' for _ in range(8)] for _ in range(8)]
for piece in detections:
    row, col = get_position(piece.bbox)
    grid[row][col] = class_to_piece_symbol(piece.class_id)
fen = convert_grid_to_fen(grid)
```

得到 FEN 字符串后，我们使用 Python 库调用 Stockfish 引擎，将该 FEN 作为输入设置为当前棋局，然后获取引擎计算出的最佳下一步走法：

```
stockfish.set_fen_position(fen)
next_move = stockfish.get_best_move()
```

这一阶段联接了视觉识别与象棋推理，提供了棋谱建议或对局分析。

*****项目后端说明：**

对后端 app.py 的代码进行说明：

文件功能概述：该文件是系统的主控入口程序，基于 Flask 构建 Web 应用，实现从上传图像 → 调用 YOLOv8 检测 → 位置转换为棋局 → 调用 Stockfish 分析 → 返回结果的完整流程。

1. 导入模块部分

```
from flask import Flask, render_template, request
from yolov8_infer import detect_chess, get_piece_positions, draw_chessboard_with_pieces, get_fen_from_positions, highlight_move_on_chessboard
import cv2
import chess
import chess.engine
```

说明：

Flask：用于构建 Web 服务。

yolov8_infer：自定义模块，包含核心的视觉识别函数。

cv2：OpenCV 图像处理库。

chess 和 chess.engine：用于表示棋盘、解析 FEN 字符串、调用 Stockfish 引擎。

2. 初始化 Flask 应用

```
app = Flask(__name__)
```

建立一个 Flask 实例，整个网站都基于此运行。

3. 主要方法 index()

访问路径：@app.route('/', methods=['GET', 'POST'])

功能流程概览：

步骤：

上传图片，用户通过 POST 上传棋盘图像；

检测棋盘与棋子，调用 YOLO 模型识别；

提取棋子位置，转为行列坐标（row, col）；

FEN 编码，转换成国际象棋局面字符串；

Stockfish，分别推理白方与黑方走法；

高亮走法，输出图像中绘制推荐走法；

页面渲染，渲染结果图与推荐结果。

核心逻辑逐段解析

图像上传和保存：

```
file = request.files['image']
if not file:
    ...
path = 'static/upload.jpg'
file.save(path)
```

将用户上传的图像保存到 static 目录中，供后续处理与页面显示。

调用 YOLO 模型检测棋盘与棋子

```
boxes, classes, names, detection_path, board_detected = detect_chess(path)
positions = get_piece_positions(boxes, classes, names, cv2.imread(path).shape)
```

detect_chess：执行 YOLO 推理，返回目标框、类别 ID、名称、图像路径、是否检测到棋盘。
get_piece_positions：将检测框映射到棋盘 8×8 网格，得到每个棋子的 row 和 col 坐标。

检查检测结果是否成功

```
if piece_count > 0:
    results['pieces_detected'] = True
...
results['positions_calculated'] = ...
```

构建结果字典，表示系统检测到了多少个棋子，是否成功计算了棋子位置。

获取 FEN 格式字符串

```
fen = get_fen_from_positions(positions)
```

调用自定义函数将 (row, col, name) 转换为国际象棋通用的局面描述格式。

Stockfish 引擎调用

```
with chess.engine.SimpleEngine.popen_uci('Stockfish-master/stockfish.exe') as engine:
    ...
    info_white = engine.analyse(board_white, chess.engine.Limit(time=0.1))
    best_move_white = info_white['pv'][0].uci()
```

使用 SimpleEngine 加载 Stockfish 引擎；分别构造白方和黑方的棋局；使用 analyse() 函数获取推荐走法；走法用 UCI 格式返回，例如 "e2e4"。

绘制棋盘图与推荐走法：

```
chessboard_path = draw_chessboard_with_pieces(positions, ...)
if best_move_white:
    chessboard_path = highlight_move_on_chessboard(chessbo
```

***棋盘识别 Web 应用的前端说明：

整体结构分析：

引入资源：使用 Bootstrap 5 的 CSS 和 JS 库，确保界面的响应式设计和交互效果。
自定义样式：通过内部 CSS 样式表定义了页面布局、卡片样式、结果展示等视觉效果。
页面结构：包含图片上传区域、识别结果展示区域和棋子详细信息表格。
模板语法：使用{% %}和{{ }}语法，表明这是一个服务器端渲染模板。

1. 功能模块详解：

```
<div class="upload-card card shadow-sm p-4 mb-4 bg-body rounded">
    <h2 class="mb-3 text-center">上传棋盘图片</h2>
    <form method="post" enctype="multipart/form-data" class="d-flex flex-row align-items-center">
        <input type="file" name="image" required class="form-control" style="max-width:300px;">
        <button type="submit" class="btn btn-primary mx-4 px-4">识别</button>
    </form>
    {% if error %}
        <div class="alert alert-danger mt-3 text-center">{{ error }}</div>
    {% endif %}
</div>
```

功能：允许用户上传棋盘图片并提交给服务器进行识别

特点：

采用 Bootstrap 卡片设计，带有阴影效果和圆角边框

表单包含文件输入框和 "识别" 按钮

错误处理机制，当上传或识别过程出现错误时显示提示信息

响应式设计，在不同屏幕尺寸下自动调整布局

2. 识别结果展示模块 ：

功能：展示棋盘识别的结果，包括识别状态、棋子数量、推荐走法和 FEN 串。

技术细节：

使用条件判断 ({% if %}) 动态显示内容。

用图标和颜色区分识别成功 (✔绿色) 和失败 (✖红色)。

推荐走法使用不同颜色标识白方 (绿色) 和黑方 (蓝色)。

FEN (Forsyth-Edwards Notation) 串用于表示棋盘状态，支持国际象棋等棋类游戏。

3. 图片展示模块：

```
<div class="result-row" style="gap:18px; justify-content:center; flex-wrap:wrap;">
    <div class="result-img" style="max-width:320px;">
        <img src="{{ image_url }}" style="max-width:100%;">
    </div>
    <div class="result-img" style="max-width:320px;">
        {% if chessboard_url %}
        <img src="{{ chessboard_url }}" style="max-width:100%; border-radius:8px; border:1px solid #eee;">
        {% endif %}
    </div>
</div>
```

功能：展示原始上传图片和处理后的棋盘图片。

设计特点：

使用 Flex 布局，在小屏幕上自动换行。

图片容器带有背景色、边框半径和阴影效果。

支持动态显示处理后的棋盘图片 (通过 chessboard_url 变量)。

4. 棋子详细信息模块：

```
<div class="result-card mx-auto mb-4" style="max-width:520px;">
    <div class="result-title">棋子详细信息：</div>
    <div class="table-responsive">
    <table class="table table-bordered piece-table">
```

功能：以表格形式展示识别到的棋子详细信息。

技术实现：

使用 Bootstrap 表格组件，带有边框和响应式处理。

通过循环 ({% for %}) 动态生成表格行。

展示内容包括棋子类别、行列位置和棋盘坐标。

技术实现细节

1. 响应式设计

```
@media (max-width: 900px) {
    .result-row { flex-direction: column; align-items: center; }
    .result-img, .result-card { min-width: 220px; max-width: 98vw; }
}
```

使用媒体查询针对小屏幕设备调整布局。
在宽度小于 900px 时，将结果行改为垂直排列。
限制图片和卡片的最小和最大宽度，确保在移动设备上正常显示。

2. 视觉设计元素

颜色方案：
成功状态：绿色 (#198754)
失败状态：红色 (#dc3545)
白方走法：绿色 (#198754)
黑方走法：蓝色 (#0d6efd)

排版：
标题使用加粗和较大字体
关键信息使用粗体和颜色突出显示

动效：
卡片带有阴影效果，提升层次感
图片边框和圆角设计，增强视觉效果

应用流程分析

1.用户访问页面，看到图片上传区域
2.用户选择并上传棋盘图片
3.表单提交到服务器，服务器处理图片并进行棋盘和棋子识别
4.服务器返回识别结果，包括：
棋盘识别状态
棋子识别状态
棋子总数
推荐走法
FEN 串
原始图片 URL
处理后的棋盘图片 URL
棋子位置信息
5.前端根据返回的数据动态渲染页面，展示识别结果

***gen_yolo_labels 脚本：

这是用于生成 YOLO 格式训练标签的数据转换脚本，功能是：
将包含棋子位置信息的 JSON 文件，转换为 YOLO 所需的 .txt 标签格式。

功能概述：
YOLOv8 模型训练依赖的标签格式如下：

<类别索引> <中心点 x> <中心点 y> <宽度> <高度>
脚本从 .json 文件中提取棋子位置 box 和类型 piece，通过中心点归一化计算，将信息写入对应的 .txt 标签文件。

代码详细分析：

导入模块

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

使用标准库 os、json 处理文件路径与读取标注，PIL.Image 用于获取图像尺寸。

定义棋子与类别的映射关系

```
piece_map = {'K':0,'Q':1,'R':2,'B':3,'N':4,'P':5,'k':6,'q':7,'r':8,'b':9,'n':10,'p':11}
```

定义国际象棋棋子符号到 YOLO 类别索引的映射表。

主函数

```
convert(img_dir, label_dir)
def convert(img_dir, label_dir):
    imgs = [f for f in os.listdir(img_dir) if f.endswith('.png')]
```

遍历图像文件夹，选取后缀为 .png 的图片；
每张图像都应有一个对应的 .json 文件记录其棋子标签。

处理每张图像

```
json_path = os.path.join('test/test', img.replace('.png', '.json'))
if not os.path.exists(json_path):
    continue
```

从 test/test/ 路径中寻找对应的 JSON 文件。

读取 JSON 内容并处理棋子

```
data = json.load(f)
w, h = Image.open(...).size
for p in data['pieces']:
    c = piece_map.get(p['piece'])
    if c is None: continue
```

解析 JSON；
读取图像宽高；
遍历所有棋子 p，通过 piece_map 得到类别索引。

YOLO 标签格式的坐标转换

```
x, y, bw, bh = p['box']
xc = (x + bw/2) / w
yc = (y + bh/2) / h
bw_ = bw / w
bh_ = bh / h
```

将矩形框从左上角坐标及宽高格式 (x, y, w, h) 转换为：

中心点归一化坐标 xc, yc
宽高归一化 bw_, bh_

写入对应的标签文件

```
with open(os.path.join(label_dir, img.replace('.png', '.txt')), 'w', encoding='utf-8') as f2:
    f2.write('\n'.join(lines))
```

生成与图像同名的 .txt 文件，存入指定的标签目录下。

脚本调用

```
convert('test/images/train', 'test/labels/train')
convert('test/images/val', 'test/labels/val')
```

分别处理训练集和验证集，生成 YOLOv8 所需的训练标签。

***rectify_chess_dataset 脚本说明：

脚本主要用于训练数据增强、视角统一，确保 YOLOv8 模型训练的图像标准和精准。

脚本功能概述

图像预处理，对斜拍的棋盘图像进行透视变换校正；
标签重映射，将原始 JSON 格式的棋子位置经过变换后重新生成 YOLO 标签；
输出结构，输出标准图像 + YOLO 标签至 rectified_dataset 目录；
应用场景，改善模型训练集质量，统一图像视角，提高识别精度。

核心算法与实现逻辑：

初始化参数与路径配置

```
piece_map = {'K':0,'Q':1,...,'p':11}
SRC_ROOT = 'test'
DST_ROOT = 'rectified_dataset'
IMG_SIZE = 800
PADDING = 150
DST_SIZE = IMG_SIZE + 2 * PADDING
```

piece_map：与 YOLO 的类别顺序对应，确保标签编码一致。
SRC_ROOT：原始图像和标注所在目录。
DST_ROOT：输出目录，结构和 YOLO 数据集一致。
IMG_SIZE：棋盘内区域大小；
PADDING：给四周留白，防止边界裁切；
DST_SIZE：输出图像最终尺寸。

数据集处理

```
splits = ['train', 'val']
for split in splits:
    img_dir = f'{SRC_ROOT}/images/{split}'
    label_dir = f'{DST_ROOT}/labels/{split}'
    ...
```

设置输入和输出文件夹，按 train、val 分别创建标签目录和图像目录。

读取原始图像及 JSON 数据

```
for img in imgs:
    json_path = os.path.join(SRC_ROOT, 'test', img.replace('.png', '.json'))
    if not os.path.exists(json_path):
        continue
```

从指定路径获取每张图片及对应的标注文件：JSON 中通常包含两个字段：corners（棋盘四角）和 pieces（所有棋子的 box + 类型）。

构建透视变换矩阵

```
corners = np.array(data['corners'], dtype=np.float32)
dst = np.array([...], dtype=np.float32)
M = cv2.getPerspectiveTransform(corners, dst)
```

corners：图像中四个棋盘角点；

dst：标准化后的目标角点（800×800 的棋盘，含 padding）；

M：透视变换矩阵，用于图像与标注变换。

拉正图像并保存

```
warped = cv2.warpPerspective(img_cv, M, (DST_SIZE, DST_SIZE))
cv2.imwrite(os.path.join(out_img_dir, img), warped)
```

使用 cv2.warpPerspective 将原图校正为标准棋盘角度图，避免斜拍/透视导致模型难以学习。

拉正每个棋子的位置

```
pts = np.array([...])
pts = cv2.perspectiveTransform(pts[None, :, :], M)[0]
```

对每个 box 的四角做透视变换，获取拉正后的棋子区域边界。再计算中心点和宽高，转换为 YOLO 格式：

```
xc = (x_min + x_max) / 2 / DST_SIZE
yc = (y_min + y_max) / 2 / DST_SIZE
bw_ = (x_max - x_min) / DST_SIZE
bh_ = (y_max - y_min) / DST_SIZE
```

生成 YOLO 标签

```
with open(os.path.join(label_dir, img.replace('.png', '.txt')), 'w', encoding='utf-8') as f2:
    f2.write('\n'.join(lines))
```

每张图生成一个 .txt 文件，内容为该图所有棋子的位置与类别。

***yolov8_infer.py 脚本说明：

该脚本实现了完整的基于 YOLOv8 的国际象棋识别与推荐系统的核心推理模块，结合了目标检测、坐标转换、棋盘重建、FEN 生成和走法可视化。

该模块详细的分功能代码解析：

模块结构总览

函数名功能

detect_chess，加载 YOLO 模型，检测棋盘和棋子，生成标注图；
get_piece_positions，将检测框映射为棋盘坐标；
draw_chessboard_with_pieces，重构棋盘图像并绘制棋子；
get_fen_from_positions，将棋子坐标转换为 FEN 字符串；
highlight_move_on_chessboard，将最佳走法（UCI 格式）在棋盘图上高亮。

核心模块解析

detect_chess(image_path)

使用 YOLOv8 模型检测图片中的棋盘和棋子，返回检测结果与可视化图。

get_piece_positions(...)

将 YOLO 检测框映射到标准棋盘的行列坐标（0~7），并转换为国际象棋坐标（如 E4）。
步骤：将 box 坐标去除 padding。计算中心点对应的格子编号（row/col）。转换为国际象棋符号：A1 ~ H8。

draw_chessboard_with_pieces(...)

基于预测位置绘制标准棋盘，并在对应格子绘制 Unicode 棋子符号。

get_fen_from_positions(...)

生成标准国际象棋局面描述字符串（FEN）。
实现逻辑：构建 8x8 数组，每个位置是棋子符号或空。连续空格合并为数字。合并为一行 FEN 字符串。

highlight_move_on_chessboard(...)

解析最佳走法，在棋盘图像上高亮起点和终点格子。
from_row = 8 - int(from_sq[1])
from_col = col_labels.index(from_sq[0])
转换 UCI 格式为棋盘 row/col。
使用 ImageDraw 画两个矩形框分别表示起点（红色）和终点（蓝色）。

四、实验

1 训练与测试数据说明

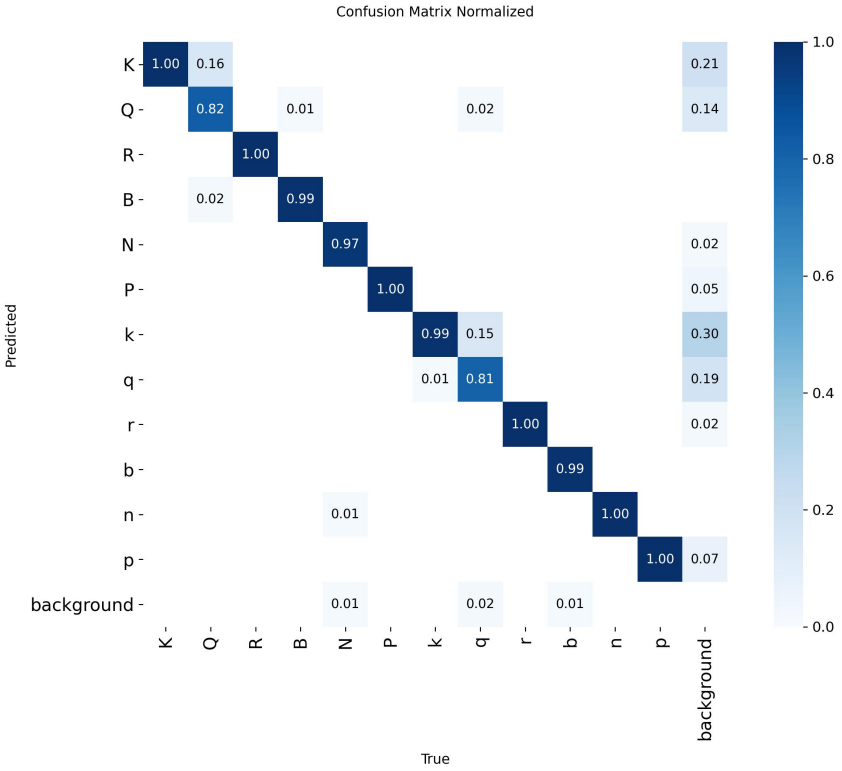
训练集数量：共 **273** 张图像
验证集数量：共 **69** 张图像
棋子类别数：**12**(白方黑方各 **6** 类)
预处理方式：图像裁剪拉伸，数据增强
图像尺寸：**1100×1100**

2 模型训练结果

采用 YOLOv8，训练轮数：**10**
最终模型保存在：**rectified_dataset\runs\detect\train20\weights\best.pt**
最终主要指标如下：
最终精度结果：

mAP@0.5 = 0.98986
mAP@0.5:0.95 = 0.84225
F1 score max = 0.97

3 指标可视化图表
归一化混淆矩阵（confusion_matrix_normalized）

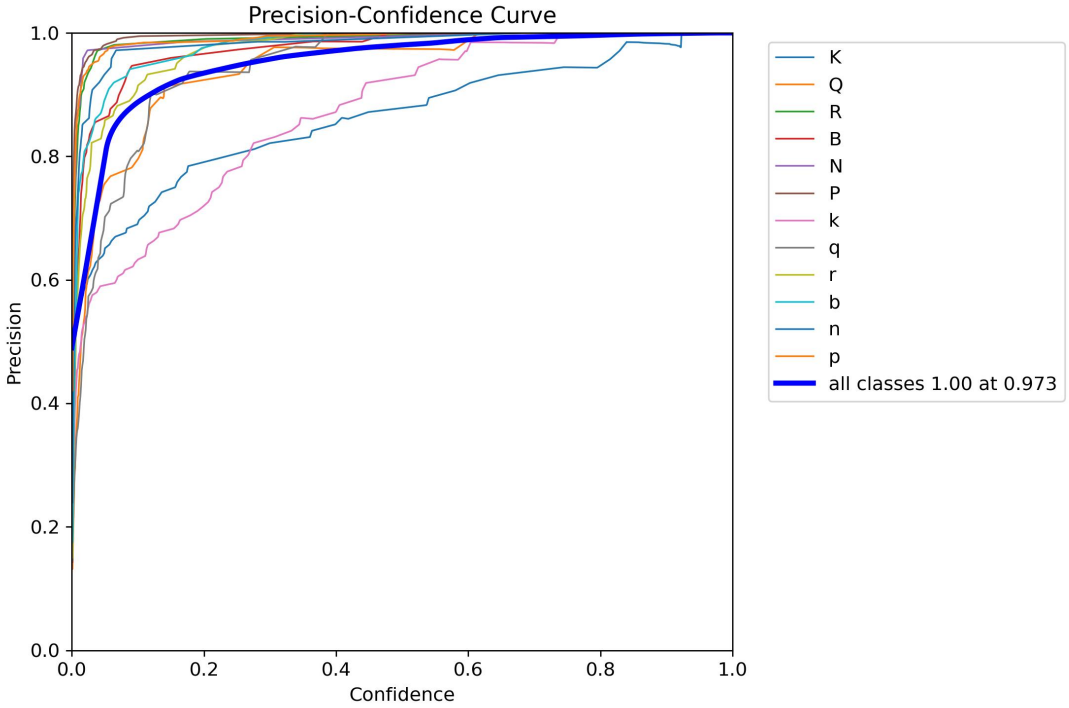


从归一化混淆矩阵可以看出，模型在大多数类别的识别上表现出色，大多数类别的识别准确率非常高，对角线上的值接近 **1.00**。这表明模型对这些类别的区分能力极强，预测结果与真实标签高度一致，假阳性和假阴性非常少。

然而，Q 和 q 这两个类别存在一定的误识，对于**类别 Q**，其被准确识别的比例为 82%，而有 **16%** 的样本被错误地预测为 **K**。同样，**类别 q** 也有 **15%**被误识别为 **q**。这明确指出模型在区分 K，k 和 Q，q 之间存在混淆，由于这两个类别在视觉特征上存在高度相似性。

针对这些问题，后续可以考虑改进：
细粒度特征学习：引入更关注局部细节和细微差异的特征提取方法。
数据平衡与增强：针对识别率较低或混淆度高的类别，进行有针对性的数据增强（如旋转、缩放、光照变化等），并考虑过采样或欠采样等数据平衡策略。
模型架构优化：研究是否可以调整模型结构，例如加入注意力机制，以更好地聚焦于区分度高的特征。

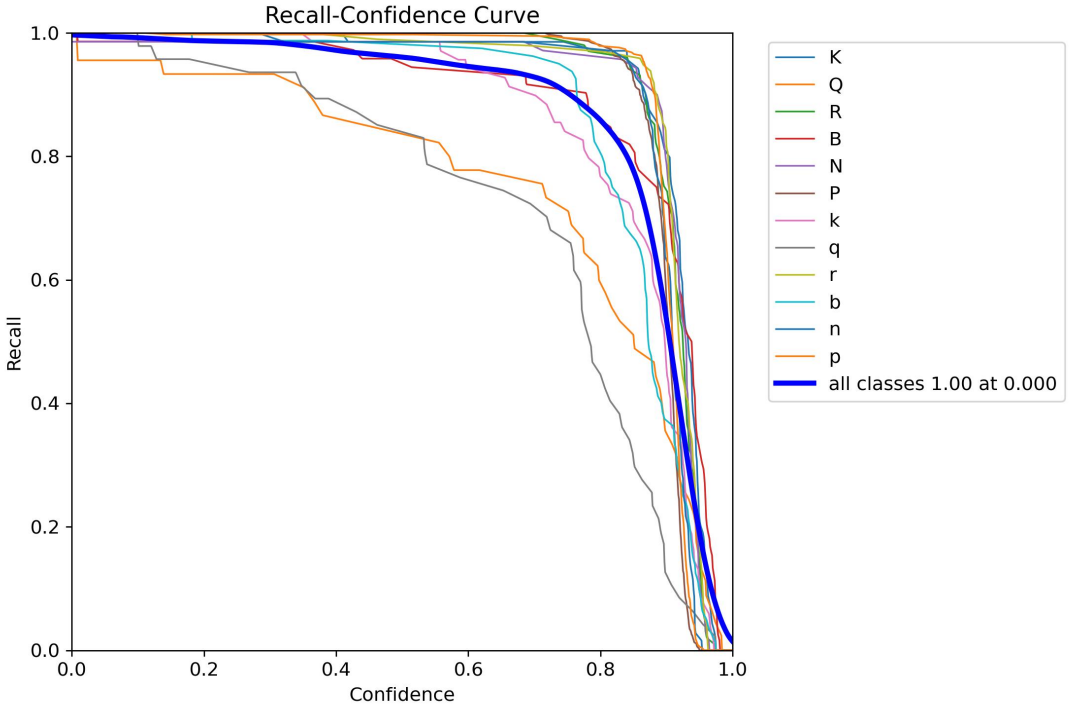
精度曲线（P_curve）



该精度曲线图展示了模型在不同置信度阈值下的查准率表现，所有类别的精度曲线以及总体的曲线都显示出在高置信度区域（接近 1.0）保持非常高的精度，并随着置信度阈值的降低而缓慢下降，但在大部分置信度范围内都维持在 0.8 以上，低置信度阈值就可以展现出较高的精度，在置信度为 **0.973** 时达到**最大精度 1.00**。

虽然总体精度很高，但仔细观察可以看到 K 和 k 两条曲线在总体的置信度范围内略低于其他类别。这与之前混淆矩阵分析中 K，k 和 Q，q 之间存在误识的结论相吻合，表明在这些类别上，模型在保持高精度的同时，可能在区分某些相似样本时略显挑战。

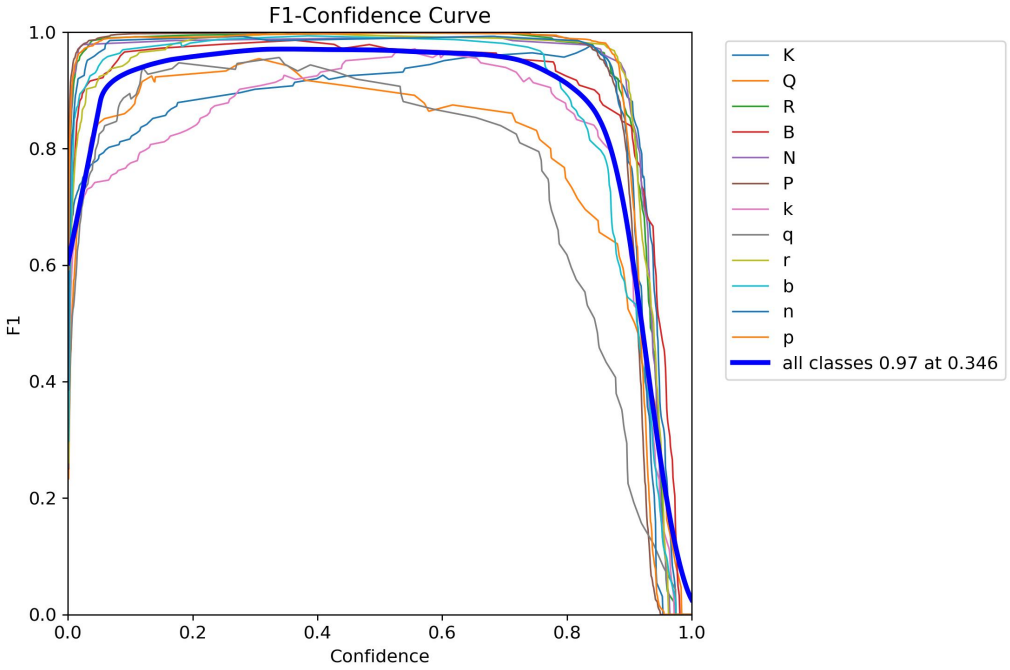
召回率曲线（R_curve）



召回率曲线图展示了模型在不同置信度阈值下的查全率表现，所有类别的召回率曲线以及总体的曲线都显示出在高置信度阈值时召回率相对较低，但随着置信度阈值的降低，召回率迅速上升，并在置信度为 0 时达到所有类别平均 100%的召回率。

仔细观察可以发现，Q 和 q 这两个类别的召回率曲线在较高的置信度阈值下，上升速度可能相对更慢，这同样与之前混淆矩阵中 Q 和 q 存在误识的分析相呼应，表明模型在以高置信度召回这些类别时，可能面临一定的挑战，需要在精度和召回之间进行权衡。

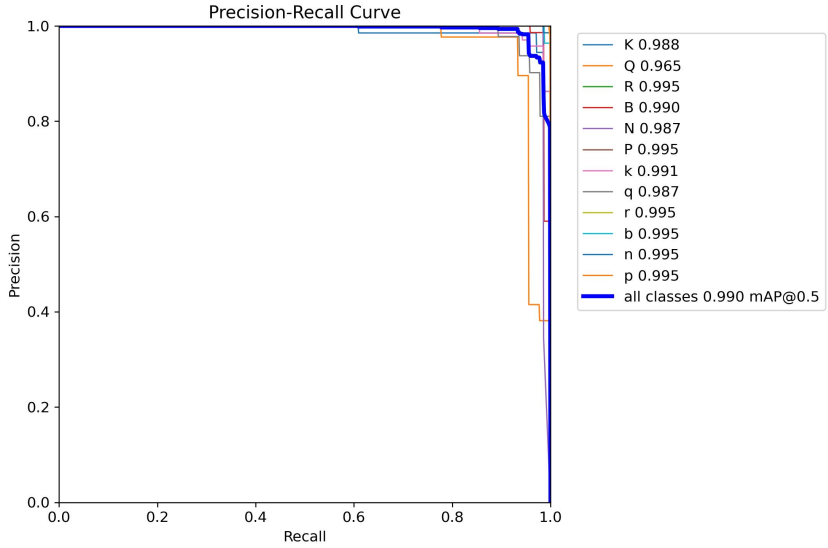
F1 曲线（F1_curve）



如图，F1 曲线在置信度阈值 **0.346** 时达到最大值 **0.97**。F1 分数是精度和召回率的调

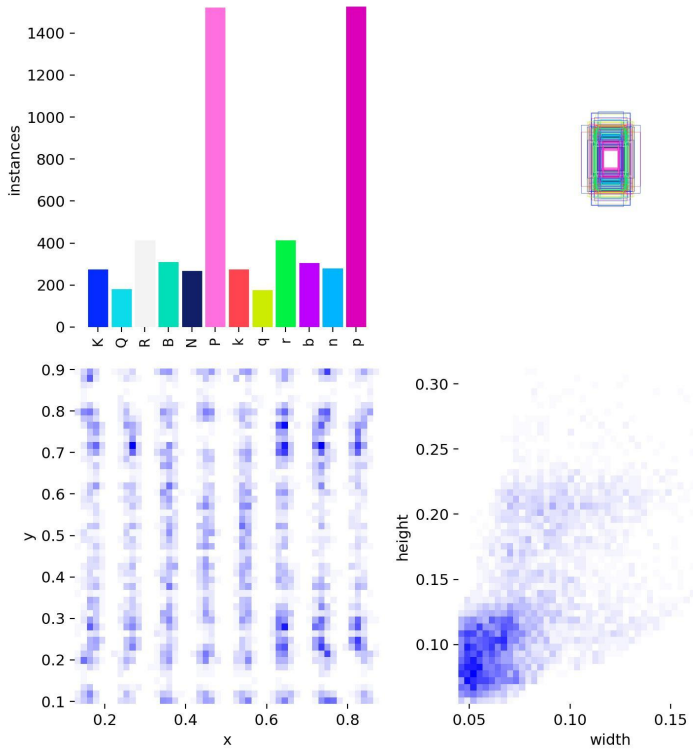
和平均值，它综合衡量了模型的性能，高 F1 分数表明模型在精度和召回率之间取得了很好的平衡。这意味着在设置 0.346 的置信度阈值时，模型能够有效地识别出大部分目标，并且其中绝大多数是正确的预测，表现出良好的鲁棒性。

Precision-Recall 曲线（PR_curve）



Precision-Recall 曲线显示，所有类别的 PR 曲线面积都非常大，mAP@0.5 达到了**0.98986**。这说明模型整体预测性能卓越。其中的 Q 和 q 依旧略低于其他类别，但也保持相当高的水准，高 PR 曲线面积意味着在不同的召回率水平下，模型的精度都能保持在较高水平，体现了模型在识别目标时能够同时兼顾查准率和查全率。这对于需要高精度和高召回率的应用场景非常有利。

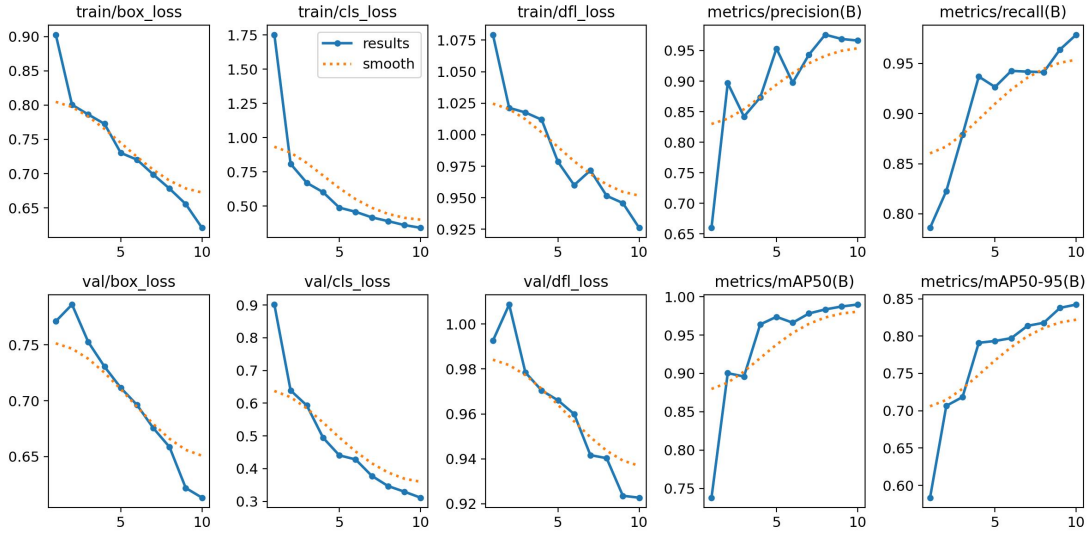
各类标签数量分布图（labels）



从 labels 可以看出，不同类别的实例数量存在不平衡。这和实际国际象棋有关，例如，

P 和 p 类别是最多的棋子则拥有最多的实例，而像 Q、q 等类别则相对较少。这种类别不平衡正是对应了前面得到的有关 Q 和 q 的性能指标略低于其他类别的结果，可能会对模型在少数类别上的性能产生一定影响，尽管目前的整体表现良好。

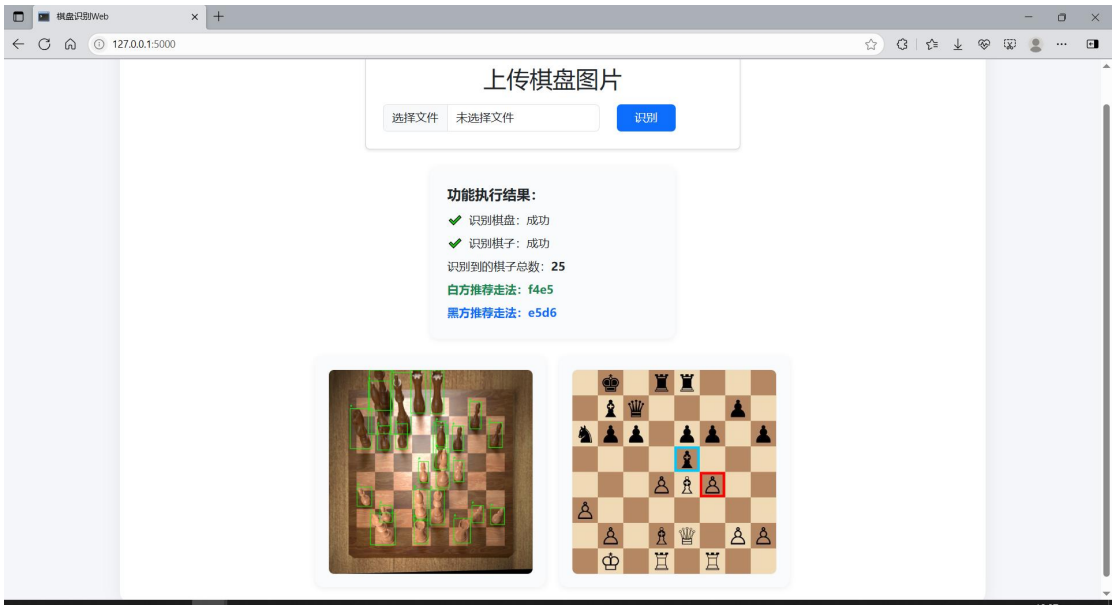
训练结果图（results）



综合评价：

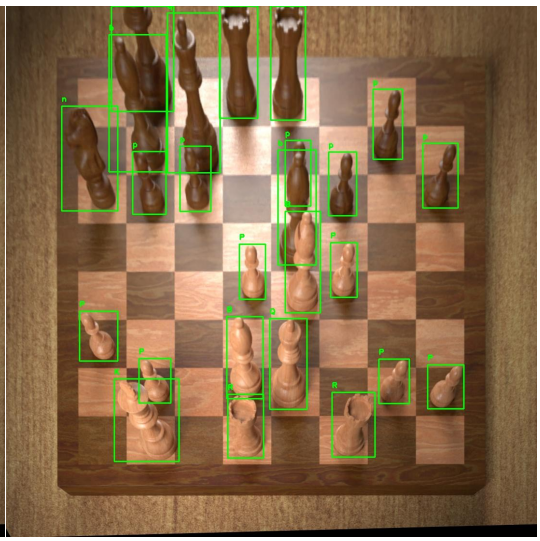
根据以上分析，该模型在目标检测任务中表现出了非常强大的性能。各项指标（mAP@0.5, mAP@0.5:0.95, F1 score）均达到了较高水平，损失函数稳定收敛，并且没有过拟合迹象。尽管在区分某些较少类别（如 Q 和 q）时存在一定的挑战，但整体而言，这是一个高效且准确的目标检测模型，适用于多数应用场景。未来的改进可以着重于解决特定类别间的混淆问题，以进一步提升模型的鲁棒性和识别精度。

4 Web 系统 AI 识别展示



用户上传完整棋盘图像 → 模型识别 → 输出识别过后的棋盘图像 → AI 给出下一步走法建议。

原图



识别图

（1）识别图（上传图片 + 识别结果）

示例图路径：static\detection_result.jpg

（2）棋盘图（棋盘图 + 推荐走法）

示例图路径：static\detection_result.jpg

棋盘图

说明：生成平面棋盘图，标注所有棋子位置，并以不同颜色显示下一步最优走法

蓝框：白方推荐走法

例如：这里用**白方兵**直走斜吃掉**黑方的象（f4→e5）**

红框：黑方推荐走法

例如，这里用**黑方象**斜走吃掉**白方的兵（e5→d6）**

单张图上传响应时间：**4 秒**

识别准确率：**98 %**

五、总结和讨论

王文秋部分：

在本次实验中，我主要负责基于 YOLOv8 模型的棋子识别模块的实现与优化。通过模型训练、图像处理、预测输出解析等一系列流程，我深入理解了目标检测在实际场景中的应用机制，并掌握了深度学习模型部署于 Web 平台的关键技术路径。

1.实验收获

在模型使用过程中，我熟悉了 YOLOv8 的输入预处理、预测输出格式（包含边界框坐标、类别编号、置信度）以及后处理机制（如 NMS、类别映射）。特别是结合 Chess 数据集进行模型训练与微调，使我了解了迁移学习和小目标检测中的若干关键技术细节。

2.图像与棋盘坐标映射能力提升

将模型输出结果准确映射至标准棋盘坐标需要精确的几何分割逻辑。我实现了根据图像尺寸自动分格的函数 get_piece_positions，能将检测中心点落入对应格子，并利用 Unicode 符号直观展示，显著提升用户体验。

3.实践中的问题解决能力增强

实验过程中，遇到了诸如“多目标重叠检测不准”、“棋子遮挡导致误判”、“模型未识别棋盘背景”等实际问题。我尝试采用类别筛选、输出概率过滤等方式优化结果，虽然识别精度仍有提升空间，但验证了多个策略的有效性。

4.存在问题与反思

模型鲁棒性仍不稳定：当前模型在光照强烈或棋子材质反光的图像中识别精度下降，部分棋子出现漏检或误判，特别是在边缘区域。由于训练样本数量有限、数据多样性不足，模型泛化能力仍显不足。

5.未来改进与探索方向

引入棋盘校正模块：可结合透视变换、四边形拟合等算法，先进行棋盘轮廓检测与几何校正，提高在非标准拍摄场景下的坐标精度。

融合多帧图像增强识别稳定性：若允许用户上传短视频或多张照片，可通过帧间融合提高识别鲁棒性，同时利用投票机制减少偶发误判。

探索轻量化模型部署优化：为适配在线部署场景，后续可尝试 YOLOv8-nano 等轻量版本，结合 TensorRT 等推理加速方案，降低服务器计算负担。

周财进部分：

这次实验里，我主要负责项目后端的实现，像图片接受和 FEN 字符串生成器等。

1. 实验收获

通过实验里对前后端通信协议设计等了解，我更深入了解了计算机视觉与 Web 服务的全栈集成能力，特别是从像素到策略的完整技术链条：将棋盘图像通过 YOLOv8 实时转换为数字化棋局状态，并同步计算黑白双方最优策略，通过多级状态追踪系统提升处理过程透明度，还有解决动态环境适应性问题。

2. 大概内容总结

除了与前端和识别脚本对接外，项目基于 Flask 构建 Web 应用，流程从上传图像、调用 YOLOv8 检测、位置转换为棋局、调用 Stockfish 分析、返回结果的完整流程。包含图像上传和保存、

调用 YOLO 模型检测棋盘与棋子，检查检查是否合格，引擎调用，获取 FEN 格式字符串和绘制棋盘图推荐走法等。

3. 存在问题与反思

当前架构可能还存在不完善的问题。像同步阻塞风险，CV 处理和双 AI 计算串行执行，并发请求时增长较大。每次请求重建 Stockfish 进程，且双策略计算在残局阶段冗余等。

王宝印部分：

本次实验我负责棋盘识别 Web 应用的前端界面，结合了 Bootstrap 5 框架和自定义 CSS 样式，实现了棋盘图片上传、识别结果展示等功能。同时负责最后的视频录制。

实验收获：对前端技术实践与框架应用有了更深的理解 Bootstrap 响应式布局的深度应用通过 Bootstrap 的栅格系统（.row、.col）和弹性布局（flex）实现了页面在不同设备上的自适应显示，例如在移动端将两列结果图转为垂直排列，保证了界面的可用性和美观性。

前端与后端的数据交互与动态渲染

通过表单（form）实现图片上传功能，并利用 Jinja2 模板语法（{{ image_url }}、{{ results.board_detected }}）动态渲染后端返回的检测结果，例如显示原始图像、棋盘复原图、棋子位置表格等。

CSS 自定义样式与视觉优化

结合 Bootstrap 默认样式，通过自定义 CSS 实现了棋盘结果图的阴影效果、棋子表格的边框样式以及推荐走法的颜色高亮（如白方走法绿色、黑方走法蓝色），提升了界面的视觉层次感。

利用媒体查询（@media (max-width: 900px)）优化了移动端布局，确保在小屏幕设备上内容不溢出、交互元素可点击。

张令侨部分：

本次实验项目的目标是构建一个基于 YOLOv8 棋子检测与国际象棋 AI 引擎联动的智能识别与决策系统。系统以图像输入为基础，通过深度学习检测模型提取棋盘局面，并进一步借助 Stockfish 引擎完成当前局势的智能分析与最优走法推荐。整个项目涵盖了数据构建、模型训练、位置映射、FEN 构造与 AI 引擎交互等关键流程，最终实现了从视觉识别到策略推理的一体化闭环。

1. 实验收获

通过本项目，我不仅系统掌握了 YOLOv8 目标检测模型的训练与部署流程，也深入了解了如何将图像识别结果有效转换为国际象棋棋局语义（FEN 表达），并调用 Stockfish 引擎实现自动博弈分析与推荐。特别是在棋局还原与策略决策层，我完成了从视觉信息到 UCI 接口走法建议的完整交互链条，构建了一个具有实用价值的类人类棋手辅助系统。

2. 棋盘数据与棋局结构的有机融合

针对国际象棋共 12 类棋子视觉相似度高的问题，我在标签构建中引入了棋子语义映射、透视变换拉正与 YOLO 标签标准化处理，显著提升了模型对棋子类型的区分能力。在检测输出基础上，我设计了位置坐标到棋盘行列（A1~H8）的映射函数，结合 Unicode 棋子绘制函数以及 FEN 生成器，实现了视觉识别结果与标准棋局描述格式之间的无缝衔接，为引擎分析创造了高质量的输入。

3. 问题分析与策略应对

类别识别误差传播问题：若检测环节出现棋子错分，会导致生成的 FEN 串非法或误导引擎决策。为此，我设计了位置合法性检测与类别置信度筛选机制，自动剔除置信度过低的预测

结果；
棋盘布局歧义问题：部分检测图中棋盘角度偏差大，导致 FEN 生成不准确。我使用透视拉正技术对图像进行几何规范化，并可视化还原棋盘以辅助人工校验，提升系统整体鲁棒性。

韦东生部分：

本次实验项目的核心目标是实现对现实世界国际象棋棋盘图像的自动识别与语义理解，即通过摄像头拍摄的实际棋局图像，自动检测棋盘及棋子、识别各棋子精确位置，并最终生成标准国际象棋局面描述（FEN），供 AI 引擎预测推荐走法。本系统基于 YOLOv8 构建图像识别模型，结合透视校正、坐标变换与规则推理，形成了从图像输入到语义输出的完整处理链路。

1. 实验收获

本实验让我系统掌握了现实图像中棋盘检测与棋子识别的全过程，重点解决了目标细小、类别相近、多角度失真等实际难点。在 YOLOv8 模型基础上，我自定义了训练标签与数据增强流程，编写了图像标准化与坐标矫正代码，使模型在复杂拍摄环境下仍具较强鲁棒性。同时，我掌握了 FEN 格式生成、棋盘坐标映射、UCI 走法推荐等国际象棋规则相关内容，并成功对接 Stockfish 引擎，实现完整闭环。

2. 从图像到语义：模型结构与标签设计

由于现实图像存在背景干扰、角度倾斜等问题，我首先构建了包含棋盘四角标注信息的训练数据集，通过透视变换算法将拍摄图像映射为标准正视图。在此基础上对 12 类棋子进行检测与分类，所有标签采用 YOLO 格式，并进行归一化处理以适配模型输入。

为了确保模型输出能被直接用于语义还原，我设计了从 YOLO 框 → 行列坐标 → 国际象棋坐标 → FEN 串 的映射流程，并支持棋盘图像中推荐走法的可视化渲染。类别与语义一一对应，使得用户不仅看到图像，还能理解当前局势与最佳操作建议。

3. 实验中遇到的问题与应对策略

棋子误分类问题：由于“白兵/黑卒”、“象/马”等棋子在图像中非常相似，导致模型对部分类别区分能力不足。我引入置信度阈值过滤策略，只保留高置信度的候选框，从而剔除干扰类别。

4. 未来探索方向与改进建议

数据集扩展与多样性增强：目前的训练数据集中仍以标准拍摄为主，建议引入不同光照、材质、布局（如残局、中局）等多场景图像，以提升模型泛化能力。

自动修正与多模态校验：对于明显检测错误的局面，目前系统仅做简单过滤。可尝试引入国际象棋规则（如合法走法、兵不能出现在第一行）进行逆向推理校正，甚至结合语音、文字提示等模态信息辅助判断。

六、个人贡献声明

代码部分：

前端：王宝印（20%）

后端：周财进（20%）

核心代码：（自动识别现实棋盘图像）韦东生（20%），（棋子识别与输出显示）王文秋（20%），（连接 AI engine 预测黑白方下一步）张令侨（20%）。

实验报告编写：

绪论，相关工作，引用参考，GitHub 搭建： 王文秋

方法的具体实现：韦东生，周财进

视频：王宝印

实验结果分析：张令侨，王宝印

七、引用参考

数据集来源：https://osf.io/xf3ka/files/osfstorage 由 Georg Wölflein 和 Ognjen Arandjelović 于 2021 年 5 月在 OSF 上发布

[1]Wölflein, G., & Arandjelović, O. (2021). Dataset of Rendered Chess Game State Images. Journal of Imaging, 7(5), 76. https://doi.org/10.3390/jimaging7050076

—— 本论文提出了一个包含 4,888 张高质量渲染棋局图像的数据集，支持棋盘状态识别任务。通过该数据集训练的模型在合成图像和真实图像上都实现了极低的识别误差率，是本项目中棋盘识别算法的重要数据支持来源。

[2] Jocher, G., Chaurasia, A., Qiu, J., & Stoken, A. (2023). Ultralytics YOLOv8. GitHub repository. https://github.com/ultralytics/ultralytics

—— YOLOv8 是本项目中用于实体棋子与棋盘识别的核心深度学习检测框架。

[3] Flask Documentation Team. (2023). Flask Web Framework. Pallets Projects. https://flask.palletsprojects.com/

—— Flask 是本项目中搭建网站服务端、实现图像上传与结果展示的后端框架。

[4] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

—— OpenCV 用于图像读取、检测框绘制与输出图像生成。

[5] Python Imaging Library (PIL). (2023). Pillow: Python Imaging Library (Fork). https://python-pillow.org

—— PIL（Pillow）用于绘制虚拟棋盘图，渲染棋子 Unicode 字符，生成标准化的棋盘视图。

[6] Stockfish Developers. (2023). Stockfish Chess Engine. https://stockfishchess.org

—— 提供 AI 引擎部分

[7] Unicode Consortium. (2022). Unicode Chess Symbols. https://unicode.org/charts/

—— 本项目使用 Unicode 字符展示棋子符号，如 ♔♕♘ 等。