# 基于图像识别的国际象棋局面识别与走法推荐系统

郭馨匀，万展廷，夏冰，何梓硕，李浩弘

2025 年 6 月

## 1 绪论

随着人工智能和计算机视觉技术的快速发展，传统棋类游戏也迎来了智能化变革。国际象棋作为一项规则复杂、策略性强的博弈游戏，一直以来都是人工智能研究的重要方向。本项目旨在构建一个基于图像识别的国际象棋局面识别与走法推荐系统，通过对真实棋盘图像的分析，实现棋子识别、棋盘定位和最优走法推荐。

虽然已有如 ChessVision.ai、商汤元萝卜等产品实现了棋子识别或对弈支持，但大多依赖数字棋盘或商业硬件，难以推广到普通用户的现实场景中。因此，我们希望探索如何通过纯视觉方式，仅依赖普通摄像头，对现实国际象棋棋局进行识别，并结合开源棋力引擎进行走法推荐，从而降低技术门槛，提升系统的普适性。

整体系统由图像处理、棋子识别、棋局还原与最优走法推荐四个核心模块组成。我们采用端到端的分布式开发策略，使用 Python 与 OpenCV 进行图像预处理，借助 YOLOv8 深度学习模型完成棋子检测，并通过 FEN 字符串生成与 Stockfish 引擎接口实现走法推理。前端则以 React 框架为基础，整合视觉识别结果与交互展示功能。

具体流程如下：

- **图像预处理与棋盘提取**：通过 OpenCV 实现图像灰度化、滤波与边缘检测，定位并透视变换棋盘区域。
- **棋子识别模型训练、检测及棋盘数据结构生成**：训练棋子识别模型，使用训练好的模型，对每个格子中的棋子进行识别，提取类型与阵营信息，生成 8×8 的一维棋盘局面数组。
- **FEN 字符串生成**：将棋盘局面数组转换为标准化的棋局描述格式，作为后续分析的输入。
- **最优走法推荐**：调用 Stockfish 引擎，分析局势，输出最优策略；
- **前端交互展示**：提供图像上传、识别结果可视化、推荐走法显示等功能，提升用户体验。

## 2 相关工作

目前围绕国际象棋视觉识别与智能辅助决策的研究已有一定的积累，主要有以下方向：

- **棋盘与棋子识别**：例如 ChessVision.ai 提供了一套基于摄像头的图像识别系统，能够自动识别棋盘与棋子布局，实现走法标注和保存。该系统在实际运行中效果良好，但其核心技术为闭源，难以深入分析其实现原理。
- **数字-物理转换系统**：已有如《Convert a Physical Chessboard into a Digital One》使用摄像头将物理棋盘转换为数字布局，并可用于远程对弈或复盘。这类系统多依赖边缘检测与特征点匹配来实现棋盘定位，但在复杂背景下鲁棒性有待提升。
- **项目论文示例**：如 “CVChess: Computer Vision Chess Analytics” 项目利用 TensorFlow 和 OpenCV 实现了端到端的棋盘识别，但该项目依赖特定灯光和背景条件，通用性仍存在问题。

- **学生项目报告**：部分高校课程中也出现了类似项目，如“Chess-CV”与“Chess-ID”等，尝试将识别结果转化为 FEN 格式并结合 Stockfish 实现走法推荐，对我们具有参考价值。

  尽管上述系统已初步验证了计算机视觉在国际象棋识别中的可行性，但在真实棋盘识别的稳定性、通用数据集支持不足、对弈引擎整合不便等方面仍存在一定问题。

# 3 方法的具体细节

## 3.1 图像预处理及棋盘区域提取

这部分采用 OpenCV 来实现对输入棋盘图像的预处理与棋盘区域提取。具体步骤如下：

- **图像读取与缩放**：首先读取输入图像，并统一缩放宽度至 800 像素，以保证处理效率与视觉一致性。


- **灰度化与二值化**：将缩放后的图像转为灰度图，并应用高斯模糊与自适应阈值法（Adaptive Thresholding）进行二值化处理，以增强棋盘边缘的可识别性。
- **形态学运算**：对二值图进行闭运算与开运算，进一步清除噪声并连通棋盘格子边缘。最后进行一次膨胀操作，使轮廓更加连续。
- **轮廓提取与筛选**：使用 `cv2.findContours` 提取所有外轮廓，根据轮廓面积筛选出可能为棋盘的候选区域，选取面积最大的四边形进行后续处理。
- **四边形拟合与透视变换**：对候选轮廓应用多次 `cv2.approxPolyDP` 逼近多边形，筛选出恰好四个角点的轮廓；随后进行顶点排序，并使用透视变换（Perspective Transform）将棋盘校正为标准正方形形态。
- **棋盘网格划分**：对校正后的图像进行 8×8 网格切分，每个格子作为后续棋子识别的基本单元，同时输出每个格子的像素坐标区域。

**Algorithm 1** 图像预处理与棋盘提取算法

**Require:** 原始图像 $I$

**Ensure:** 标准化棋盘图像 $B$

1: 将 $I$ 缩放至固定宽度（如 800 像素），得到 $I_{resized}$

2: 灰度化 $I_{resized}$，得到 $I_{gray}$

3: 对 $I_{gray}$ 应用高斯模糊 + 自适应阈值，获得二值图 $I_{bin}$：

$$T(x, y) = \mu(x, y) - C$$

4: 进行形态学操作（闭运算 + 开运算 + 膨胀），增强连通性

5: 使用 `findContours` 提取轮廓，筛选面积较大的外接四边形

6: 对符合条件的轮廓进行多边形逼近，获取四个顶点

7: 使用透视变换将四边形映射为正方形图像 $B$，其中变换矩阵 $\mathbf{H}$ 满足：

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

8: 将标准棋盘图 $B$ 划分为 $8 \times 8$ 网格，保存每格位置

## 3.2 棋子识别模型训练、检测及棋盘数据结构生成

**国际象棋棋子模型推理测试脚本**：

1. 基本用法：python test_inference.py –model [模型路径] –image [图片路径]
2. 使用默认模型测试：python test_inference.py –image test_images/chess.jpg
3. 带参数调整：python test_inference.py –model best.pt –image test.jpg –conf 0.5 –iou 0.45

- **模型加载**：检查模型文件是否存在，加载 YOLO 模型。验证模型类别数是否为 13（对应 13 种国际象棋棋子）；
- **图片加载**：检查图片文件是否存在，读取图片并获取其尺寸;
- **推理过程**：调用 YOLO 模型的 predict 方法进行推理，设置置信度阈值和 IOU 阈值。遍历检测结果，获取每个检测框的类别、置信度和坐标。将检测框的中心坐标映射到棋盘坐标，并更新棋盘状态。添加颜色校验逻辑，通过 HSV 色彩空间检测棋子颜色，避免颜色反色问题;
- **结果展示**：显示检测结果图像。打印棋盘状态，包括棋盘数组和棋盘布局。将棋盘数据保存到文件。

## 3.3 FEN 格式生成

输入为上一步得到的 8×8 棋盘的一维数组。
先将数组中的棋子标签映射为 FEN 规定的单字符表示：
接着进行 FEN 字符串组装，格式为：

$$\text{FEN} = \underbrace{R_1/R_2/\cdots/R_8}_{\text{棋盘布局}} \quad \underbrace{a}_{\text{活跃方}} \quad \underbrace{b}_{\text{王车易位权利}} \quad \underbrace{c}_{\text{吃过路兵目标格}} \quad \underbrace{d}_{\text{半回合计数}} \quad \underbrace{e}_{\text{全回合计数}}$$

表 1: 棋子标签与 FEN 字符的映射关系

| 棋子标签 | 描述 | FEN 字符 |
|---|---|---|
| wP | 白兵 (White Pawn) | P |
| wR | 白车 (White Rook) | R |
| wN | 白马 (White Knight) | N |
| wB | 白象 (White Bishop) | B |
| wQ | 白后 (White Queen) | Q |
| wK | 白王 (White King) | K |
| bP | 黑兵 (Black Pawn) | p |
| bR | 黑车 (Black Rook) | r |
| bN | 黑马 (Black Knight) | n |
| bB | 黑象 (Black Bishop) | b |
| bQ | 黑后 (Black Queen) | q |
| bK | 黑王 (Black King) | k |
| ’ ’ | 空格符 | . |

$$\text{其中，} \quad \begin{cases} R_i: & \text{第} i \text{ 行的棋盘表示，} i = 1, \ldots, 8 \\ a \in \{w, b\}: & \text{当前执子方（白或黑）} \\ b \in \{\mathrm{K}, \mathrm{Q}, \mathrm{k}, \mathrm{q}, -\}: & \text{王车易位权利} \\ c \in \{-, a1, b3, \ldots\}: & \text{吃过路兵目标格} \\ d \in \mathbb{Z}_{\geq 0}: & \text{半回合计数} \\ e \in \mathbb{Z}_{\geq 1}: & \text{全回合计数} \end{cases}$$

最后输出 FEN 字符串即可。

### 3.4 Stockfish 接入及最优走法获取

Stockfish 是一个免费、开源、强大的 UCI 国际象棋引擎，可从 stockfishchess.org 或 github 上免费获取。可以通过 UCI 选项调整引擎配置，例如：

- **思考时间/深度/节点数**：控制引擎计算量。
- **线程数**：利用多核 CPU 并行计算。
- **哈希表大小**：存储搜索过的局面信息，提高效率。

经过配置后，可以调用我们的 FEN 字符串来获取下一步的最佳走法。

### 3.5 后端代码整合

项目后端采用 FastAPI 框架搭建，定义/api/recognize 为后端接口，用以接收用户上传的图片。接着进行图像预处理、棋子识别、生成 FEN 字符串、获取最佳走法，最后将 FEN 字符串及最优走法返回给前端。

### 3.6 Web 前端页面开发及整体功能整合

前端主要功能模块包括：

- **图片上传**：ImageUpload 组件允许用户上传棋盘图片，App.jsx 负责将图片通过 FormData 发送到后端 http://localhost:8000/api/recognize 接口。

- **后端数据处理**:App.jsx 接收后端返回的包含 FEN 字符串和 best_move 的 JSON 数据，并管理应用状态。
- **棋盘可视化**:ChessboardDisplay 组件使用 react-chessboard 库，根据后端返回的 FEN 字符串动态渲染棋盘局面。
- **结果显示**:FenDisplay 组件显示 FEN 字符串并提供复制功能；MoveRecommendation 组件显示后端计算的最佳走法。
- **错误与加载管理**: 前端实现了加载状态提示，并能有效处理后端识别失败（如 best_move 为 null）的情况，向用户显示“无法计算最佳走法”等友好提示。

# 4 实验

## 4.1 数据集准备

本项目采用公开数据集 **Chess Pieces Dataset v2.4**（YOLOv8 标注格式），该数据集包含 416×416 分辨率的棋盘图像以及对应的棋子标注信息。数据集中共包含多个不同视角、光照条件下的真实棋盘图像，涵盖国象中的全部 12 类棋子（黑白双方的 King、Queen、Rook、Bishop、Knight、Pawn）。

数据集结构如下所示：

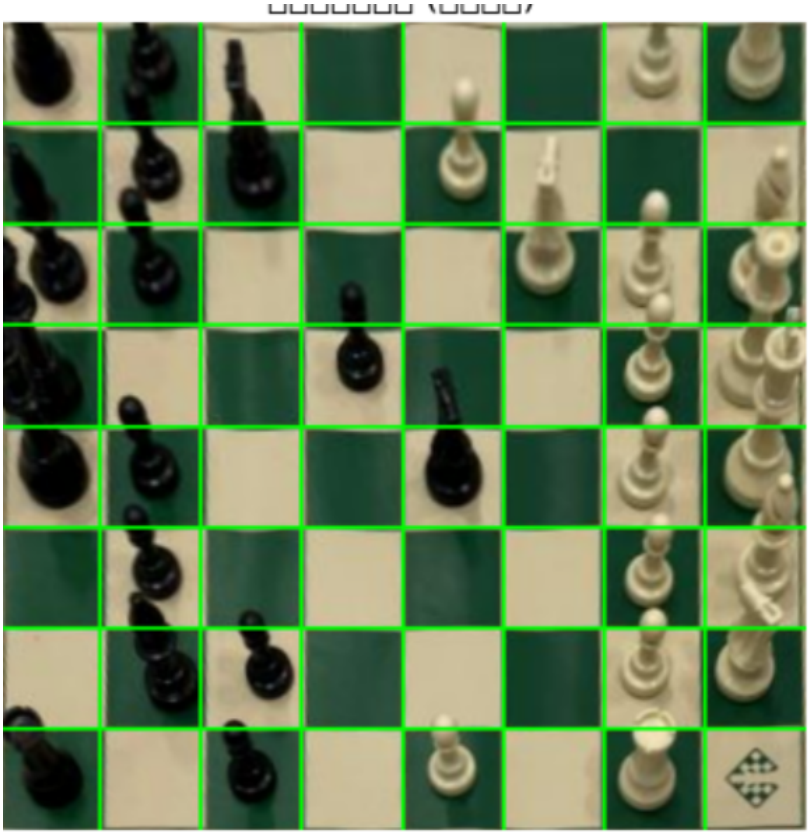
- `images/train`：训练图像，共计约 600 张；
- `images/test`：验证图像，约 30 张；
- `labels/train`：对应的 YOLOv8 格式标注文件；
- `data.yaml`：包含类名定义与路径配置。

我们使用该数据集训练 YOLOv8 棋子识别模型，并在验证集上评估模型性能。图像均为棋盘俯视角，增强了模型的泛化能力和鲁棒性。

## 4.2 图像预处理及棋盘区域提取

本项目采用 OpenCV 实现对输入棋盘图像的预处理与棋盘区域提取。处理流程如下：

- **图像读取与缩放**：统一缩放图像宽度为 800 像素，保证处理效率与后续操作一致性；
- **灰度化与二值化**：使用高斯模糊与自适应阈值算法（Adaptive Thresholding）进行图像增强与边缘提取；
- **形态学运算**：闭运算与开运算相结合，增强轮廓连通性并抑制噪声；
- **轮廓提取与四边形拟合**：提取大轮廓并进行多边形逼近，获取棋盘四个顶点；
- **透视变换与网格划分**：对棋盘区域执行透视变换，使其变换为标准正方形，随后进行 $8 \times 8$ 网格切分。



### 4.3 棋子识别模型训练、检测及棋盘数据结构生成

#### 4.3.1 棋子识别模型的训练

- **数据集准备**：def prepare_dataset(data_path)；
- **训练 YOLOv8 模型**：def train_model(data_yaml, pretrained='yolov8n.pt', epochs=100, imgsz=640):

  data_yaml：数据配置文件的路径，包含数据集路径、类别数和类别名称。

  pretrained：预训练模型的路径，默认为 yolov8n.pt。

  epochs：训练轮数，默认为 100, 后改为 200。

  imgsz：输入图像的尺寸，默认为 640。
- **评估模型**：def evaluate_model(model_path, data_yaml):

  model_path：训练好的模型文件路径。

  data_yaml：数据配置文件的路径。

  加载模型，并使用验证集评估模型性能。

  打印 mAP@0.5 和 mAP@0.5:0.95 等评估指标。

#### 4.3.2 测试脚本 test_inference.py

通过命令行参数指定模型路径、图片路径、置信度阈值和 IOU 阈值，运行模型推理并输出结果。使用 YOLO 模型对国际象棋棋子进行检测，并将检测结果映射到棋盘上，最终生成棋盘状态。

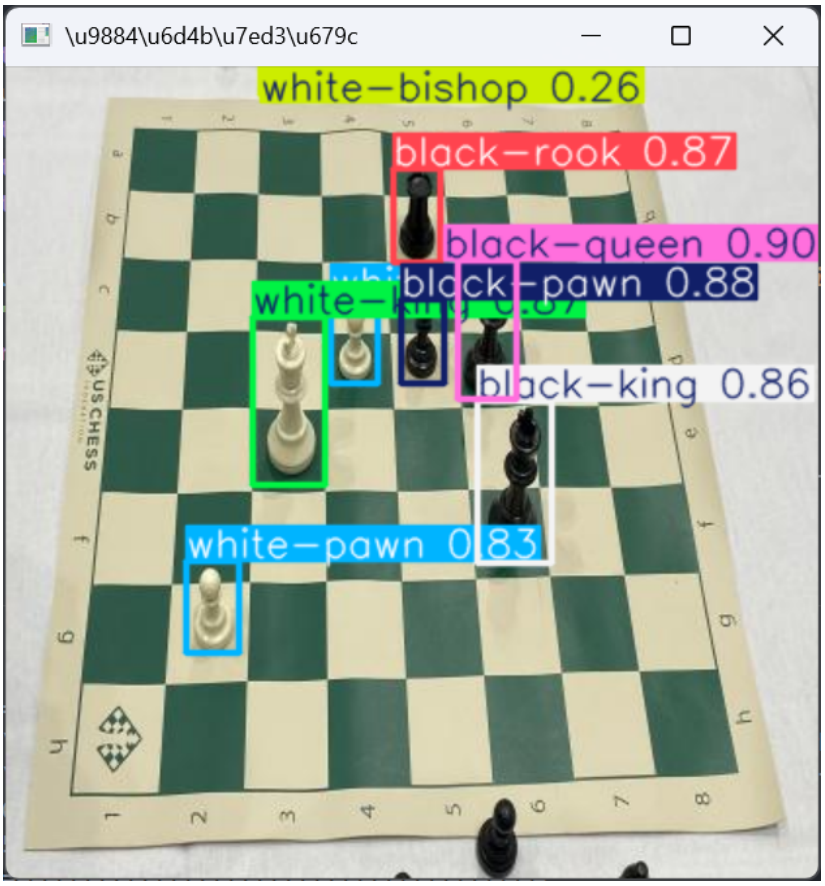
- **参数解析**:import argparse: 使用 argparse 模块解析命令行参数，包括模型路径、图片路径、置信度阈值和 IOU 阈值。提供默认值，方便用户直接运行脚本测试默认图片。
- **辅助函数**:get_chess_notation(pred_name)：将模型预测的类别名称（如 black_pawn）转换为标准棋子表示（如 bP）。使用字典 piece_map 实现映射。

create_empty_board()：创建一个 8x8 的空棋盘，用空字符串表示每个位置。

map_to_board_coords(x, y, img_width, img_height)：将图像坐标（x, y）映射到棋盘坐标（0-7, 0-7）。根据图像宽度和高度计算每个棋格的大小，然后确定棋子所在的棋格。

save_board_to_file(board, output_dir="output")：将棋盘数据保存到文件中。创建输出目录（如果不存在），生成带时间戳的文件名，并将棋盘状态写入文件。

- **模型推理函数**：def run_inference(model_path, image_path, conf_thres=0.25, iou_thres=0.45)




### 4.3.3 棋盘数据结构生成

- **def create_empty_board()**: return [[" for _ in range(8)] for _ in range(8)]//创建一个 8×8 的空棋盘，每个位置用空字符串' '表示。

  生成的棋盘一维数组如下：

$$\begin{bmatrix} ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \end{bmatrix}$$

- **def map_to_board_coords(x, y, img_width, img_height)**:

  square_width = img_width / 8

  square_height = img_height / 8

board_x = int(x / square_width)

board_y = int(y / square_height)

return board_x, board_y

计算每个棋格的宽度 square_width 和高度 square_height。使用整除运算将图像坐标转换为棋盘坐标。将图像坐标（x, y）映射到棋盘坐标（0-7, 0-7）。

### 4.4 FEN 格式生成

生成 FEN 格式大致可分为三步：

- **棋子标签映射到 FEN 规定的单字符 (piece_labels_to_board 函数)**: 检测输入的 8×8 一维数组中的棋子标签，映射为 FEN 规定的单字符，检测结果为一维列表，长度为 64，按照行优先顺序排列。这个函数的存在主要是因为对接不清晰，导致上一步生成的棋盘数组中使用的棋子标签与 FEN 规定的单字符不一致。如果对接时能确认清楚，即可省去这步。
- **存放 FEN 规定单字符的一维列表生成 FEN 串 (board_to_fen 函数)**: 将一维列表构造成 8×8 棋盘矩阵，然后逐行生成 FEN 串，其中连续空位用数字代替。
- **FEN 字符串构建的主函数 (generate_fen 函数)**: 调用 piece_labels_to_board 函数和 board_to_fen 函数，最后返回 FEN 字符串。

测试使用的一维数组如下：

$$\begin{bmatrix} ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & bR & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & wP & bP & bQ & ,, & ,, \\ ,, & ,, & wK & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & bK & ,, & ,, \\ ,, & wP & ,, & ,, & ,, & ,, & ,, & ,, \\ ,, & ,, & ,, & ,, & ,, & ,, & ,, & ,, \end{bmatrix}$$

测试输出的 FEN 字符串：

$$8/1P6/5k2/2K5/3Ppq2/8/4r3/8\ w\ -\ 0\ 1$$

### 4.5 Stockfish 接入及最优走法获取

接入 stockfish 引擎，对 stockfish 进行配置。depth=30 为思考深度，thinktime=5 为引擎计算时间，timeout=30.0 为超时提醒。将测试输出的 FEN 格式进行输入获取最佳走法：”c5d5”。

### 4.6 后端代码整合

项目后端采用 FastAPI 框架搭建, 文件结构如下：

```
project/
├─ app.py                    # 后端主干部分，定义接口
├─ logic/
│  ├─ fen_generator.py       # 将检测结果转为FEN字符串
│  ├─ image_preprocess.py    # 棋盘检测及透视变换
│  ├─ move_recommend.py      # 调用Stockfish引擎获取最优走法
│  └─ piece_detection.py     # 棋子检测及坐标映射
├─ stockfish/
```

```
│  └─ stockfish-windows-x86-64-avx2.exe
├─ best.pt                    # YOLOv8训练好的棋子识别模型
├─ requirements.txt           # 项目依赖库
```

app.py 是后端的主干部分，允许所有来源跨域请求，方便前端测试与多设备访问，使用 async def recognize 支持异步文件读取。主要负责接收用户上传的国际象棋局面照片，调用后端图像预处理、棋子识别、FEN 字符串生成和最优走法计算模块，最终返回 FEN 字符串和最优走法。

image_preprocess.py 根据图像预处理及棋盘区域提取部分的原代码进行修改，去除了非必要部分。原代码给透视变换后的图片添加了方便观察的 8×8 网格，此项便于开发时进行测试，但实际不需要此功能，故删去。原代码还对棋盘进行了网格划分，但由于对接不清晰，棋子识别部分也进行了棋盘网格划分。考虑到如若将网格划分交给这一步，调用函数时需要多一个返回值，且棋子识别函数也需多一个输入值，同时棋子识别部分的棋盘网格划分部分再更改比较麻烦，所以删去了该步骤代码的这一部分。

piece_detection.py 根据棋子识别模型检测及棋盘数据结果生成部分的原代码进行修改。原先的代码是独立存在，需要在代码中更改图片的本地文件路径来进行测试，有许多不必要的输出，同时还会保存检测结果图片和一维棋盘数组在本地。所以更改成了输入直接接收上一步棋盘区域提取得到的图片，去除不必要输出，只输出一个一维棋盘数组。同时，原先的棋子映射到棋盘中，使用的是检测框的中点，经过测试发现由于某些棋子（如王、后）较高，检测框的中点映射到棋盘中容易出错，映射到实际行数的上一行，所以改成使用检测框的下四分之一点进行映射。

fen_generator.py 与后端代码整合是同一人完成，故代码没有需要更改的部分。

move_ecommend.py 是根据 Stockfish 接入及最优走法获取部分进行更改。原先使用的 chess.engine.SimpleEngine.pop 在测试中报错，无法正常运行，于是改用了更原始的 subprocess.Popen，更改后可正常运行。原代码相对独立，没有对接清晰，没有输入 FEN 字符串的部分，且输入也不是 FEN 字符串，而是棋子标签是 FEN 标准单字符的二维数组。整合到后端后更改了这一部分代码，也精简了部分代码。

后端测试使用的图片：

后端服务启动后，浏览器访问 http://localhost:8000/docs 进行测试。

测试结果为:

```
{
  "fen": "8/1P6/5k2/2K5/3Ppq2/8/4r3/8 w - - 0 1",
  "best_move": "c5d5"
}
```


与本地测试结果一致，对接前端后，后端代码整合部分完成。

### 4.7 Web 前端页面开发及整体功能整合

前端界面基于 React 框架和 Vite 构建工具进行开发，采用模块化的组件设计，以确保代码的高效性、可维护性和可扩展性。图片上传模块由 ImageUpload 组件负责。它提供了用户友好的文件选择界面，并在用户选择图片后，将图片数据封装为 FormData 对象，通过异步的 fetch API，发送至后端 http://localhost:8000/api/recognize 接口进行处理。

应用主控与状态管理方面，App.jsx 作为前端应用的中央控制器，负责协调各个组件间的交互。它接收并解析后端返回的 JSON 响应（包含 FEN 字符串和 best_move），并据此更新应用的全局状态。同时，App.jsx 精确地管理着加载状态（isLoading）和错误信息（error）的显示，确保用户能够实时了解系统运行状况。

棋盘可视化功能由 ChessboardDisplay 组件承载，该组件集成了 react-chessboard 库。它能够根据后端提供的 FEN 字符串，实时且精确地在界面上绘制出当前的棋盘局面。为聚焦展示，棋盘被设定为不可拖动，并在未识别出有效 FEN 或等待图片上传时，显示相应的提示信息。

识别结果展示模块包含两个关键组件：FenDisplay 组件专注于展示后端识别出的 FEN 字符串，并提供了一键复制功能，极大地方便了用户的操作；MoveRecommendation 组件则负责呈现后端计算出的最佳走法。值得注意的是，该组件内置了鲁棒性处理：当后端未能返回有效的推荐走法（如 best_move 为 null 或空）时，它将明确显示“无法计算最佳走法”，从而确保用户始终获得清晰的反馈，避免信息缺失。

通过上述组件的精心设计与协同工作，前端界面实现了与后端 API 的无缝集成，共同完成了国际象棋局面识别与走法推荐的核心功能。

## 5 总结和讨论

我们小组通过合作完成了国际象棋局势的识别与分析系统，获得了以下成果：

- **棋盘识别**：通过 OpenCV 的灰度化、自适应阈值和形态学运算等操作，系统能够稳定提取棋盘区域，解决了复杂背景下的棋盘定位问题。
- **棋子检测**：基于 YOLOv8 训练的棋子识别模型在测试集上达到了一定的效果，能够区分 12 类棋子并映射到棋盘格位。
- **当前局势分析**：通过与 Stockfish 引擎链接对当前棋盘形势进行了下一步分析。
- **端到端流程整合**：通过后端与前端的协同，实现了用户与系统交互的操作。

通过整合 OpenCV、YOLOv8 和 Stockfish 等技术，体验了计算机视觉算法的协同开发流程，实现了从图像输入到最优走法推荐的完整流程。

实验有收获也有不足。我们组的棋子识别模型取得了一定效果，但由于此次模型训练的数据集较小，棋子识别效果并不是太理想。

通过查阅相关资料，我们总结了一些提升棋子识别准确率的方法：

- **数据集方面**：

  增加数据集多样性和规模：使用更多样化、规模更大的数据集进行训练，涵盖不同的棋盘类型、棋子样式、光照条件、拍摄角度等，有助于模型学习到更广泛的特征，提高泛化能力。例如，新引入的 ChessReD 数据集包含 10800 张真实世界图像，从不同角度、不同光照条件和不同相机规格拍摄，覆盖了各种棋子配置。

数据增强：对训练数据进行数据增强操作，如旋转、缩放、裁剪、颜色变换、添加噪声等，可以增加模型对不同情况的适应性，减少过拟合。

- **模型架构方面**：

  端到端模型：采用端到端的深度学习模型，直接从整个图像预测棋盘配置，避免了传统方法中因分步处理导致的误差累积。例如，使用 ResNeXt 模型进行棋盘识别，在 ChessReD 数据集上取得了较好的效果。

  改进目标检测模型：尝试使用改进的目标检测模型，如对 DETR 进行修改使其预测相对目标坐标，虽然在棋盘识别任务中 DETR 变体未能成功收敛，但使用变换器进行端到端相对目标检测是一个有前景的领域。

  多模型融合：结合多个不同架构的模型，通过集成学习的方式提高识别准确率。

- **训练策略方面**：

  优化训练参数：调整学习率、优化器、训练周期等参数，找到更适合模型训练的超参数组合。例如，在训练 ResNeXt 模型时，使用 AdamV 优化器，为骨干网络和编码器 - 解码器架构设置不同的学习率，并使用调度器每 300 个周期将两者都减少 10 倍。

  迁移学习：在预训练模型的基础上进行迁移学习，利用预训练模型在大规模数据集上学习到的通用特征，再在特定的棋盘识别数据集上进行微调，可以提高模型的性能。

展望未来，该系统在棋子识别精度提升、多引擎分析对比、以及 AR 走法可视化等方面仍有改进空间，同时还可以拓展到中国象棋、日本将棋等不同棋盘识别上。

# 6 个人贡献声明

**郭馨匀**：提供项目创意及大致思路、编写项目提案、FEN 格式生成部分代码编写、后端代码整合、编写技术报告、项目描述视频录制制作与上传、上传项目到 GitHub 仓库、编写项目简介 README.MD
**万展廷**：提供项目创意及部分思路、图像预处理及棋盘区域提取部分代码编写、编写技术报告
**夏冰**：Web 前端页面开发及整体功能整合、编写技术报告、项目描述视频录制
**李浩弘**：棋子识别模型训练、棋子识别模型检测及棋盘数据结构生成部分代码编写、编写技术报告、项目描述视频录制
**何梓硕**：Stockfish 接入、最优走法获取部分代码编写、编写技术报告

# 7 引用参考

数据集来源：https://public.roboflow.com/object-detection/chess-full
YOLOv8:https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt
stockfish 引擎：https://stockfishchess.org/