课程名称：计算机视觉

2025 年春季学期

报告题目： 国际象棋视觉分析系统的研究与实现

小组成员： 李佳倚、王天煜、姜骜、方智、庞宇浩

撰写日期： 2025 年 6 月 22 日

任课教师： 亓琳

学生所在学院： 信息科学与工程学部

目录

国际象棋视觉分析系统的研究与实现

李佳倚、王天煜、姜骜、方智、庞宇浩

（中国海洋大学信息科学与工程学部）

摘要

国际象棋作为策略性棋类游戏的典范，其数字化分析需求日益增长。传统棋局记录依赖人工输入，效率低下且易出错。随着计算机视觉和深度学习技术的发展，通过图像自动识别棋盘状态成为可能。本项目构建一套端到端的国际象棋视觉分析系统，采用多模块协同架构，从图像采集到 AI 决策的完整技术，为棋手提供辅助分析工具。

1 绪论

国际象棋作为策略性棋类游戏的典范，其数字化分析需求日益增长。传统棋局记录依赖人工输入，效率低下且易出错。随着计算机视觉和深度学习技术的发展，通过图像自动识别棋盘状态成为可能。本项目构建一套端到端的国际象棋视觉分析系统，采用多模块协同架构，从图像采集到 AI 决策的完整技术，为棋手提供辅助分析工具。

1.1 棋盘检测

通过 OpenCV 实现自适应预处理流程，采用改进的霍夫变换检测棋盘边界，结合几何约束筛选有效轮廓，最后通过透视变换校正为 8×8 标准视图，一定程度上解决倾斜拍摄导致的形变问题。

1.2 高精度棋子识别

基于 YOLOv8 框架训练专用棋子检测模型。设计了双阶段坐标映射机制：先获取棋子在图像中的边界框坐标，再通过中心点判别法与棋盘网格进行匹配，配合容错边界处理，显著降低边缘棋子误判率。

1.3 动态棋局状态重构

构建 FEN 生成引擎，将视觉识别结果转化为标准 FEN 字符串。针对静态图像的历史信息缺失问题，引入用户交互机制，系统默认保留王车易位权利，确保棋局合规性。

1.4 智能决策与交互

集成 Stockfish 引擎，通过 safe_play 函数实现实时走法推荐。开发 Flask Web 应用，支持图像上传 → 状态分析 →AI 决策的可视化流程，为棋手提供辅助分析工具。当棋手上传的棋盘图片合规可行时，系统会进行下一步的提示，给棋手一个较优方案。当棋盘不合规时，系统会提示该棋盘不合规。

2 前沿进展与尚未解决的问题

国际象棋识别是一个典型的目标检测问题，目标检测的输入图像往往不止包含一个目标，同时不仅需要知道输入图像中包含了哪些种类的物体，而且需要对它们进行准确的定位。如今研究者们开发了许多基于特征或基于神经网络的目标检测算法。

2.1 相关工作与前沿进展

在深度学习广泛应用于人工智能之前，研究者们开发了许多目标检测算法。尽管这些早期算法性能不及现代技术，它们的设计理念对现代目标检测算法产生了重要影响。随着深度学习在计算机视觉中的成功应用，诞生了众多基于神经网络的检测方法。

2.1.1 VJ（Viola-Jones）检测器

Viola 和 Jones [1, 2] 为人脸检测开发了 VJ 检测器，这是首个实现实时性能和有效检测率的目标检测框架。

2.1.2 方向梯度直方图（histogram of oriented gradients, HOG）

2005 年，Dalal [3] 等人引入了 HOG 特征描述符用于行人检测。不同于 VJ 检测器，HOG 采用固定窗口大小，并通过图像金字塔对图像进行缩放，以处理不同尺寸目标。

2.1.3 可变形部件模型（deformable part-based model, DPM）

DPM 是一种基于部件（part）的检测模型，在 2008 年作为 HOG 特征的扩展由 Felzenszwalb 等人提出 [4]。相较于 HOG 检测，DPM 为了进一步提高精度，在提取 HOG 特征的时候选择无符号梯度和有符号梯度 2 种特征共同构建特征向量。DPM 仍然沿用部分 HOG 检测的步骤，但是为了提高对姿态变化的鲁棒性，DPM 模型提出了对目标的部件进行检测。

2.1.4 CNN 模型

1989 年，LeCun 等人 [5] 首次用反向传播算法训练卷积网络，成功实现手写数字识别。此后，2015 年提出的 VGG 结构 [6] 进一步加深了 CNN 模型，但是人们逐渐发现，单纯加深网络不仅不会获得更优秀的效果，还会造成梯度消失或梯度爆炸等问题，使得网络难以训练。He 等人 [7] 在 2016 年提出了 ResNet，通过引入捷径连接（shortcut connection）缓解了深度神经网络训练中的梯度消失问题，使用恒等映射（identity mapping）避免了模型退化。ResNet 的成功应用在图像分类、目标检测、语义分割等任务上，成为深度学习领域的重要里程碑之一，其引入的残差学习思想为设计更深层次的网络提供了有效的方法，自此绝大部分深度神经网络都会采取残差结构，这一思想且对之后许多深度神经网络的设计产生了深远的影响。

2.1.5 Fast R-CNN

Girshick [8] 于 2015 年提出了 Fast R-CNN。该模型首先使用深度卷积网络提取图像特征图，然后通过选择性搜索确定感兴趣区域（region of interest，RoI）。每个 RoI 被重新映射到特征图的对应位置，并通过感兴趣区域池化（region of interest pooling，RoI Pooling）处理，使不同大小区域转换为统一尺寸特征。Fast R-CNN 设计了多任务损失函数，结合 Softmax 分类和边界框回归进行联合训练，同时采用非极大值抑制（non-maximum suppression，NMS）[9] 获得最终检测结果。该网络通过共享特征图加速处理，并将分类和定位任务整合到一个框架中，实现一次性模型更新，简化训练和减少开销。Fast R-CNN 训练速度比 R-CNN 快 9.5 倍，测试速度提升 213 倍，精度也有显著提升。

2.1.6 YOLO（you only look once）

2015 年，Redmon 等人 [10] 提出首个不基于区域的目标检测算法 YOLO 模型。YOLO 不预生成感兴趣区域，而是一次性直接在整张输入图像上进行目标检测。它将图像划分为网格，每个网格预测多个检测框和类别概率，利用置信度和 IOU 进行非极大值抑制以去除冗余检测框，大大提高了检测速度。YOLO 作为一种新颖范式，开辟了目标检测研究的新方向。后续众多算法基于 YOLO 设计思想提出。YOLO 的后续版本包括 YOLO9000 [11]。使用更有效的 DarkNet-19 骨干网络，引入批归一化 [12] 加速网络收敛，使用统计生成的先验锚框优化查询能力，预测边界框偏移量而非坐标。YOLOv3 [13] 使用更大的 DarkNet-53 骨干网络，引入多尺度预测。YOLOv7 由 Wang 等人 [14] 提出，旨在进一步提升 YOLO 系列的性能，同时保持实时目标检测的特性，特别适合于需要快速检测的应用如多目标跟踪、自动驾驶等。YOLOv7 着重于优化梯度传播路径和动态标签分配策略，以提高深层网络的学习效率和收敛速度，同时减少参数和计算量。

YOLOv7 在不同层引入模型重参数化技术 [15]，控制网络中的最短和最长梯度路径，使得更深的网络也能有效学习和收敛。

2.2 待解决的问题

20 世纪 90 年代和 21 世纪初期，基于手工特征和传统机器学习的目标检测算法到如今各种基于深度学习方法的算法，目标检测领域经历了从传统方法到深度学习的革命性转变。到如今，最新的深度学习模型架构和为不同领域的检测特殊设计的针对性方法已经能够在一定程度上满足人们对目标检测算法性能的需求。但是现有的目标检测算法仍不能良好的解决所有检测问题，仍有很多待解决的空间。

2.2.1 多样性

多样性挑战是目标检测领域中的一个重要难题，尤其随着数据集的不断扩展和应用场景的多样化。早期传统基于机器学习的目标检测任务由于计算资源和数据限制，主要集中在人脸检测和行人检测等单一场景。随着技术发展，数据集的规模和类别迅速增加，因此普通的目标检测效果可能会很差。对目标检测中的多样性挑战，未来的发展趋势将集中在模型的适应和对不同数据模式的适应性上。

2.2.2 增量学习

也称为连续学习或终生学习，指的是模型在学习新任务或新数据时，能够保留以前学到的知识，并且利用过去的知识来帮助学习新任务的能力。增量学习目标检测 [16] 是指在已有的目标检测模型基础上，不断添加新类别的能力，同时保持对已学习类别的识别能力。在实际应用中，经常会出现新的目标类别，增量学习使模型能够适应这种变化。增量学习不需要每次都重新训练整个模型，节省了大量的时间和计算资源，并且增量学习赋予了模型持续学习和适应新任务的能力，如何克服灾难性遗忘，实现真正的智能，需要对增量学习进行探索，符合人工智能的长期目标。

3 实验方法

本实验旨在通过图像处理和机器学习技术，从包含棋盘的图像中提取棋盘信息，并转换为国际象棋的 FEN 表示，同时利用 Stockfish 引擎获取 AI 走法。主要涉及图像处理、目标检测和国际象棋规则处理等方面。

3.1 图像处理

使用 OpenCV 库对输入的棋盘图像进行预处理，包括灰度转换、二值化、边缘检测、形态学操作等，以提取棋盘的轮廓和方格信息。通过霍夫变换检测直线，找到棋盘的边界，并进行透视变换，将棋盘图像转换为标准的 8x8 方格布局。

- **图像预处理**：读取图像，将其转换为灰度图与 RGB 图，采用 Otsu 二值化和 Canny 边缘检测，再进行膨胀操作。
- **棋盘检测**：通过霍夫直线变换检测直线，找出可能的棋盘轮廓，筛选出符合条件的正方形轮廓，确定棋盘的四个顶点。
- **透视变换**：依据四个顶点进行透视变换，把棋盘校正为正视图。
- **方格划分**：将校正后的棋盘划分为 8x8 的方格，记录每个方格的坐标。
- **棋子检测**：使用 YOLO 模型检测图片中的棋子，根据方格坐标确定每个棋子所在的方格。
- **FEN 生成**：依据棋子的位置和颜色，生成国际象棋的 FEN（Forsyth - Edwards Notation）表示。
- **SVG 与 PNG 生成**：根据 FEN 生成 SVG 格式的棋盘图像，再将其转换为 PNG 格式。

3.2 目标检测

使用 YOLO（You Only Look Once）模型对棋盘上的棋子进行检测和分类。模型会识别出每个棋子的位置和类型（如白棋、黑棋的不同棋子类型），并将检测结果映射到棋盘的方格上。

3.3 国际象棋规则处理

将检测到的棋子信息转换为国际象棋的 FEN 表示，使用 Python 的 chess 库创建棋盘对象，并根据 FEN 表示初始化棋盘状态。借助 Stockfish 引擎计算 AI 的最佳走法。当用户选择颜色方后，依据当前棋盘的 FEN 表示创建 `chess.Board` 对象，调用 `safe_play` 函数让引擎在 0.1 秒的时间限制内计算最佳走法。

3.4 YOLO 棋子识别模型训练

我们的数据集是在 roboflow 平台上拉取的棋子标注数据集，通过平台的添加噪声、图像裁剪等图像预处理方法，完成一个适合 YOLO 模型进行训练的棋子数据集，提高模型的最终泛化性能。

4 实验过程

4.1 环境准备

确保系统中安装了必要的库和依赖，如 OpenCV、NumPy、Matplotlib、Pandas、Ultralytics、cairosvg、chess 等。同时，下载并配置好 YOLO 模型和 Stockfish 引擎。

4.2 数据准备

收集包含国际象棋棋盘的图像数据，将其放置在指定的 images 目录下。确保图像的质量和清晰度，以便于后续的处理和分析。

4.3 运行测试脚本

使用 `test_script.py` 文件对单张图像进行处理和测试，该脚本会调用 `processer` 脚本，对指定的图像进行处理，并输出处理结果（包括图像路径、FEN 表示和提取的棋盘图像路径）。

4.4 运行 Web 应用

使用 `app.py` 文件启动一个 Flask Web 应用，用户可以通过浏览器访问应用，上传图像、处理图像并获取 AI 走法。具体步骤如下：

1. 启动 Flask 应用。
2. 打开浏览器，访问应用的 URL（通常为 `http://127.0.0.1:5000/`）。
3. 在页面上选择上传图像，点击上传按钮将图像保存到 images 目录。
4. 选择要处理的图像，点击处理按钮，应用会调用 `processer` 脚本对图像进行处理，生成 FEN 表示和提取的棋盘图像，并将处理结果显示在页面上。
5. 选择自己的颜色（白棋或黑棋），点击获取 AI 走法按钮，应用会调用 Stockfish 引擎计算并显示 AI 的最佳走法。

5 数据描述

5.1 输入数据

- **图像数据**：包含国际象棋棋盘的图像文件，格式为 png、jpg、jpeg 或 gif。图像应清晰显示棋盘和棋子，以便于后续的处理和分析。
- **YOLO 模型**：用于棋子检测和分类的预训练 YOLO 模型文件（如 `chess.pt`），该模型可以识别出棋盘上的不同棋子类型。
- **Stockfish 引擎**：用于计算 AI 走法的国际象棋引擎文件（如 `stockfish_10_x64.exe`），该引擎可以根据当前棋盘状态和用户选择的颜色，计算并输出 AI 的最佳走法。

5.2 输出数据

- **FEN 表示**：国际象棋的 FEN 表示，用于描述当前棋盘的状态。FEN 表示包含了棋子的位置、回合信息等，是国际象棋中常用的一种表示方法。
- **提取的棋盘图像**：经过处理后提取出的标准 8x8 方格布局的棋盘图像，格式为 jpeg（如 `ExtractedBoard.jpeg`），该图像可以直观地显示棋盘的状态。
- **AI 走法**：根据当前棋盘状态和用户选择的颜色，Stockfish 引擎计算出的 AI 最佳走法，以国际象棋的代数记谱法表示。

6 关键代码

以下代码片段展示了 Flask 后端应用处理用户请求的核心逻辑。

```
# 处理图片
elif 'process_image' in request.form:
    selected_file = request.form.get('selected_file')
    session['selected_file'] = selected_file

    if selected_file:
        image_path = os.path.join(IMAGE_DIR, selected_file)
        results = process_chessboard_image(
            image_paths=image_path,
            yolo_model_path=YOLO_MODEL_PATH,
            output_dir=OUTPUT_DIR,
            save_debug=False
        )
```

```
        if results:
            res = results[0]
            extracted_image_path = os.path.join(OUTPUT_DIR, 'Extracted-Board.jpeg')
            fen = res['fen']
            session['fen'] = fen
            session['extracted_image_path'] = extracted_image_path
            print(f"FEN updated: {fen}")
        else:
            error_message = "图片处理失败。"
    else:
        error_message = "请选择图片。"
# 获取 AI 走法
elif 'get_ai_move' in request.form:
    selected_file = request.form.get('selected_file')
    session['selected_file'] = selected_file
    our_color_str = request.form.get('our_color')

    if our_color_str:
        our_color = chess.WHITE if our_color_str == 'white' else chess.BLACK
        session['our_color'] = our_color
    else:
        error_message = "请选择颜色方。"

    if fen and our_color is not None:
        try:
            board = chess.Board(fen)
            result = safe_play(board, chess.engine.Limit(time=0.1))
            best_move = result.move
            if best_move is None:
                error_message = "当前局面没有合法走法，可能已结束。"
            else:
                ai_move = board.san(best_move)
                print(f"AI move: {ai_move}")
        except Exception as e:
            error_message = f"获取 AI 走法时出错: {e}"
    else:
        error_message = "请先处理图片并选择颜色方。"
```

Listing 1: 处理图片与获取 AI 走法的 Flask 后端逻辑

7 实验结果

本项目的实验旨在展示我们所构建的“国际象棋视觉分析系统”的各项功能实现情况与实际运行效果。实验内容严格依照项目的设计方案进行，结果主要通过对系统各模块功能的描述以及 Web 应用最终呈现的效果来展示。

7.1 系统功能实现与成果展示

我们系统的整体架构如**图 1**所示，它清晰地描绘了从图像输入到 AI 决策输出的完整流程。

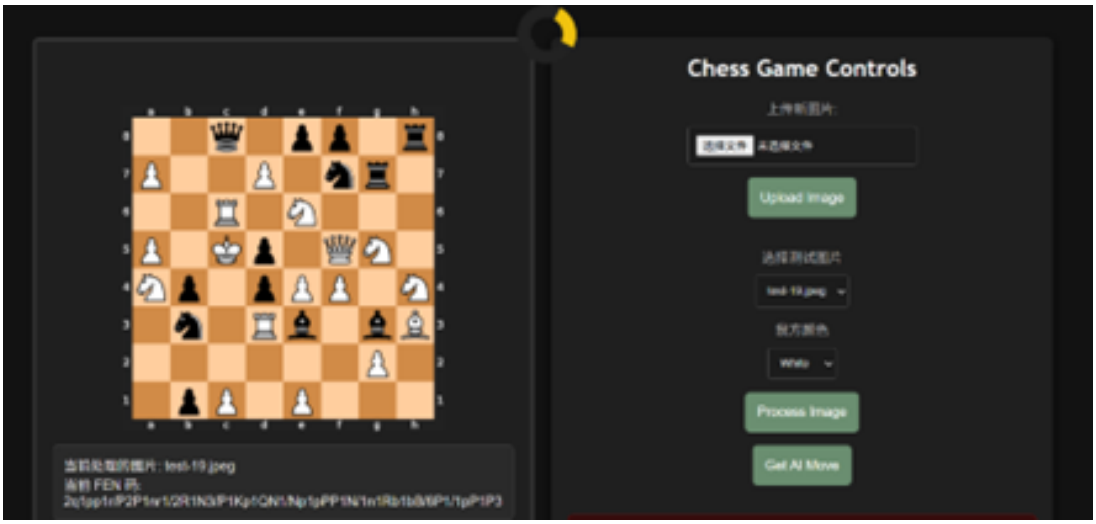

图 1: 系统整体架构图

7.1.1 图像处理与棋盘识别模块

我们利用 OpenCV 库编写了 `chess_processer` 脚本作为核心处理单元。该函数接收一张用户上传的图像，通过灰度转换、Otsu 二值化、Canny 边缘检测及形态学膨胀等一系列操作，对图像进行预处理以凸显棋盘特征。

7.1.2 棋子检测与定位模块

在棋盘被校正并划分为 64 个方格后，我们基于 YOLOv8 框架训练了一个棋子检测模型（`chess.pt`）对棋盘图像进行对象检测。该模型能够识别出图像中存在的国际象棋棋子，并返回每个棋子的类别（如王、后、兵等）、颜色（黑/白）以及其在图像中的边界框（Bounding Box）坐标。我们的代码进一步将这些像素坐标与之前划分好的 64 个方格进行匹配，从而确定了每一个棋子在棋盘上的具体位置（如 e4, d8 等）。

7.1.3 棋局状态转换与 AI 引擎集成模块

我们编写了逻辑代码，将棋子在棋盘上的二维位置信息，转换为国际象棋通用的FEN(Forsyth-Edwards Notation)字符串表示。该 FEN 字符串随后被传递给 `python-chess` 库，用于创建一个合法的棋盘对象。最后，系统通过进程调用，将此棋盘状态交给 Stockfish 引擎（`stockfish_10_x64.exe`）进行分析。系统能够生成描述当前棋局的 FEN 字符串，并在前端页面上显示。当用户选择自己执棋的颜色并发出请求后，系统能够调用 `safe_play` 函数，在 0.1 秒的时间限制内从 Stockfish 引擎获取最佳走法，并以代数记谱法（如“Qxd6”）的形式清晰地展示给用户，如**图 2**所示。

图 2: AI 走法建议展示（示例）

7.1.4 Web 应用交互界面与流程

为了提供用户友好的交互平台，我们基于 Flask 框架开发了一个 Web 应用。整个应用的交互流程和功能界面如下。

- **布局与风格：**我们设计了一个简洁的单页应用界面。为了优化视觉效果和可用性，我们对前端页面的颜色方案进行了重新渲染，增加了加载动画（loading spinner），并使用边框将不同的功能区域（文件上传区、图像显示区、结果展示区）清晰地划分开来。
- **交互元素：**界面上包含“选择文件”、“上传图片”、“处理图片”和“获取 AI 走法”等核心功能按钮。我们为这些按钮增加了细微的动态效果以增强交互感。此外，我们还为按钮增加了动态效果，并设计了错误信息闪烁提示。
- **请求处理：**`app.py` 作为后端的核心，负责处理前端发来的 HTTP 请求。例如，当收到`'process_image'` 请求时，后端会调用 `chess_processer` 脚本。当收到`'get_ai_move'` 请求时，则会调用 Stockfish 引擎。
- **状态保持：**我们利用 Flask 的 session 机制来保持用户的会话状态。当一张图片处理成功后，其生成的 FEN 码和处理后的图像路径会被存储在 session 中。这样，当用户后续请求 AI 走法时，后端可以直接从 session 中获取当前棋局的信息，避免了数据丢失或混淆。
- **错误反馈：**我们为系统设计了清晰的错误提示机制。如果用户在未上传图片时点击“处理”，或在未选择颜色时请求 AI 走法，前端页面会显示相应的错误提示信

息（如“请选择图片”、“请选择颜色方”），引导用户进行正确的操作。例如，当系统检测到不合规的棋盘局面时，会给出明确提示（见**图 3**）。

图 3: 不合规棋局的错误提示

7.2 系统运行的实际效果与观察

在对系统进行实际操作和测试时，我们观察到了以下一些具体现象，这些现象真实反映了当前系统的能力和局限。

7.2.1 理想情况下的运行

对于从正上方拍摄、光照良好、棋子摆放清晰的图像，系统在理想条件下运行稳定，结果准确无误。

7.2.2 非理想情况下的表现

- **棋盘识别的脆弱性：**当测试图像的拍摄角度较为倾斜，或棋盘材质存在反光时，霍夫变换经常无法准确定位棋盘边缘，导致整个识别流程失败。
- **棋子识别的错误：**在某些光照和角度下，YOLO 模型会出现误判。我们记录到的一个典型案例是，系统在一张图片中错误地将一个“后”识别为了“王”，从而在棋盘上识别出了两个“黑王”。这个不符合规则的识别结果，在后续我们增加的棋局合规性校验函数中被成功拦截，系统提示错误，避免了向 AI 引擎传递一个无效的棋局。
- **AI 走法的合理性：**在大多数情况下，Stockfish 引擎给出的走法建议是合理且有效的。但我们注意到，在一些极端的、不平衡的局面（有时是由于棋子识别错误导致的），AI 给出的推荐有时会偏离预期，这反映了“垃圾输入，垃圾输出”（Garbage In, Garbage Out）的原则——AI 的决策质量完全依赖于前端视觉识别的准确性。

8 总结与讨论

本项目是将计算机视觉理论应用于复杂实际问题的一次综合性实践。我们成功搭建了一个系统，但更重要的是在开发过程中遇到的挑战，以及我们对这些挑战的分析与思考。这部分内容是对我们工作量、遇到的困难以及从中获得的经验教训的真实记录。

8.1 遇到的主要挑战及解决方案

项目推进过程并非一帆风顺，我们遇到的技术和工程难题涵盖了从环境配置到算法集成的方方面面。

- **霍夫变换的参数调优困境：**棋盘识别采用的霍夫变换虽然经典，但其参数（如 minLineLength, maxLineGap）极为敏感。我们发现，针对某一张图片调优好的参数，换一张光照或角度稍有不同的图片就可能完全失效。我们尝试了动态调参策略，即根据图像的尺寸和灰度直方图自适应调整参数，但效果提升有限。这让我们认识到，依赖固定参数的传统 CV 算法在泛化能力上存在瓶颈，难以适应真实世界的多样性。未来需要采取其他办法。
- **YOLO 坐标到棋盘坐标的精确映射：** YOLO 模型返回的是棋子在整张图片中的边界框（Bounding Box）坐标 (x, y, w, h)。要确定棋子属于 8x8 网格中的哪一个，需要将这个像素坐标精确地映射到逻辑坐标（如 e4）。这个过程需要基于透视变换后的棋盘角点进行精确的几何计算。初期，由于计算误差或边界处理不当，经常出现棋子被错误地归入相邻格子的问题，特别是在格子边缘的棋子。我们通过引入中心点判别法，并为每个格子设置容错边界，才基本解决了这一问题。
- **FEN 生成逻辑的完整性缺失：**我们最初实现的 FEN 生成器只能描述棋子的位置。但在一次测试中，AI 给出了一个“王车易位”的走法，而根据图像，王和车都已经移动过，失去了易位资格。这让我们意识到，一个完整的 FEN 字符串除了棋子布局，还包含行棋方、王车易位权利、吃过路兵目标格等关键信息。从单张静态图片中推断出这些历史对局信息是不可能的。这让我们对项目的能力边界有了清醒的认识：我们的系统是一个“棋局快照分析器”，而非“对局过程跟踪器”。我们最终决定，在 Web 界面上让用户手动指定行棋方，并默认双方均保留易位权利，作为一种功能上的妥协。
- **数据集偏差导致的模型泛化问题：**在我们训练 YOLO 模型时，虽然模型表现出较好的能力，但我们推断其训练集（或我们可获取的数据集）主要是标准的斯汤顿（Staunton）棋具。当我们用一些设计感较强、形态较为艺术化的棋子进行测试时，我们训练的模型的识别率显著下降。例如，一种极简风格的“象”常常被误识别为“兵”。这暴露了深度学习模型对训练数据分布的强依赖性，即存在数据集偏差（Dataset Bias）问题。

- **后端逻辑与前端状态管理的耦合：**在 Flask 应用的开发初期，我们的后端逻辑与前端状态管理混乱。例如，当用户处理完一张图片后，如果直接上传第二张图片并点击“获取 AI 走法”，系统可能会用第二张图片的棋局和第一张图片用户选择的颜色去请求 AI。我们通过在后端引入更严格的会话（Session）管理机制，确保每次处理请求都会生成一个唯一的任务 ID，并将 FEN、校正图像路径、用户颜色等状态绑定在该 ID 下，才解决了这种状态不一致的问题。
- **训练模型需要 GPU：**为应对此问题，我们购置了并行云 GPU 服务器，解决训练量较大问题。

8.2 思考与未来展望

基于本次项目的实践和遇到的瓶颈，我们对该课题的未来发展方向进行了思考，并提出了以下改进路径。

- **引入混合视觉模型提升棋盘检测鲁棒性：**针对传统霍夫变换的脆弱性，未来的关键改进是采用两阶段的混合视觉模型。第一阶段，训练一个轻量级的 CNN 或 YOLO 模型，其唯一任务是高精度地检测棋盘的四个角点。这个模型可以针对各种透视、光照和遮挡进行专门优化。第二阶段，在通过角点完成精确的透视变换后，再调用我们现有的、更大型的 YOLO 模型进行棋子识别。这种“分而治之”的策略能有效提升系统在复杂场景下的适应性。此外，探索端到端的空间变换网络（Spatial Transformer Networks, STN）也可能提供一个更优的解决方案。
- **开发基于视频流的棋局状态跟踪系统：**从静态图像分析升级到动态视频流分析是一个重要的进步。这不仅仅是简单地对每一帧进行处理，而是需要一套高效的状态跟踪和差异检测算法。具体可以分解为：运动检测即利用背景减除或帧间差分算法，高效地判断场景中是否发生了变化（即有手进入或棋子移动），避免在静止画面上空耗计算资源。状态确认，当检测到运动结束后，对前后两个稳定状态的棋盘进行识别，通过对比 FEN 差异来确定具体的走法（例如，e2 的兵消失，e4 出现一个兵，即为 e2e4）。性能优化，为实现准实时处理，必须对模型进行优化，例如使用 TensorRT 进行模型量化和加速，或在不显著影响精度的情况下对模型进行剪枝，以适应普通计算设备。
- **利用合成数据生成技术突破数据瓶颈：**针对我们当前训练模型所面临的泛化能力不足问题，单纯依靠搜集和标注真实照片成本高昂且效率低下。一个先进的解决方案是利用 3D 渲染引擎（如 Blender、Unity）生成合成数据。我们可以编写脚本，通过输入任意 FEN 字符串，自动在虚拟 3D 场景中摆放棋子，并随机化棋具模型、材质、纹理、光照方向与强度、摄像机角度等参数，从而生成大规模、标注准确且多样性丰富的用于模型训练的数据集。这将有效缓解我们训练的模型对特定棋具风格的过拟合问题。

- **增强系统的可解释性与人机交互：**当前的系统仅直接给出最佳走法。未来的系统可以向**可解释性 AI（XAI）**方向发展。例如，在返回 AI 建议的同时，可以在棋盘上用热力图或彩色箭头，可视化 Stockfish 引擎对关键棋格的控制力评估、不同候选走法的优劣评分，甚至展示出 AI 预测的未来几步棋的变化。这不仅能提升用户体验，更能将系统从一个“对弈工具”转变为一个“教学和复盘助手”。

9 个人贡献声明

李佳倚： 主导系统模块化设计，负责项目的整体架构，如代码设计、棋盘识别和棋子识别、图像预处理和数据预分析、代码仓库的构建，参与项目总结与分析的撰写等。

王天煜： 管理负责项目报告绪论，相关研究，部分前端页面设计和后端算法的逻辑实现，构建了多视角棋局图像的测试集，YOLO 模型棋子识别能力优化，最终整合提交等。

姜骜： 负责撰写项目实验过程，数据描述。以及设计 YOLO 模型的训练方法，构建了多视角棋局图像的测试集，协调各模块接口定义，项目代码的整体优化等。

方智： 负责对项目进行总结讨论，设计 AI 引擎接口供前端进行调用，优化前端页面，优化图像处理脚本，确保图像处理系统与 AI 决策模块良好对接，编写报告等。

庞宇浩： 负责录制项目视频，测评与展示，撰写实验结果，优化棋盘识别和图像检测功能，前端 API 接口的定义，棋盘合法性判断的逻辑与代码编写等。

10 参考文献

参考文献

[1] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features[C]//Proc of the IEEE Computer Society Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2001: 511-518.

[2] Viola P, Jones M J. Robust real-time object detection[J]. International Journal of Computer Vision, 2001, 57(2): 137-154.

[3] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//Proc of the IEEE Computer Society Conf on Computer Vision and Pattern Recognition. Piscataway, NJ:IEEE, 2005: 886-893.

[4] Felzenszwalb P, McAllester D, Ramanan D. A discriminatively trained, multiscale, deformable part model[C]//Proc of the IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ:IEEE, 2008: 1-8.

[5] Lecun Y, Boser B, Denker J S, et al. Backpropagation applied to handwritten zip code recognition[J]. Neural Computation, 1989, 1(4): 541-551.

[6] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint, arXiv:1409.1556, 2014.

[7] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition[C]//Proc of the IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2016: 770-778.

[8] Girshick, R. Fast R-CNN[C]//Proc of the IEEE Int Conf on Computer Vision. Piscataway, NJ: IEEE, 2015: 1440-1448.

[9] Neubeck, A., & Van Gool, L. Efficient non-maximum suppression[C]//Proc of the 18th Int Conf on Pattern Recognition. Piscataway, NJ: IEEE, 2006, 1: 850-855.

[10] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, realtime object detection[C]//Proc of the IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ:IEEE, 2016: 779-788.

[11] Redmon J, Farhadi A. YOLO9000: Better, faster, stronger[C]//Proc of the IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ:IEEE, 2017: 7263-7271.

[12] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//Proc of the 32nd Int Conf on Machine Learning. New York:ACM, 2015: 448-456.

[13] Redmon, J., & Farhadi, A. YOLOv3: An incremental improvement[J]. arXiv preprint, arXiv:1804.02767, 2018.

[14] Wang C Y, Bochkovskiy A, Liao H Y M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors[C]//Proc of the IEEE/CVF Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2023: 7464-7475.

[15] Ding Xiaohan, Zhang Xiangyu, Ma Ningning, et al. RepVGG: Making vggstyle convnets great again[C]//Proc of the IEEE/CVF Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2021: 13733-13742.

[16] Shmelkov K, Schmid C, Alahari K. Incremental learning of object detectors without catastrophic forgetting[C]//Proc of the 16th IEEE Int Conf on Computer Vision. Piscataway, NJ:IEEE, 2017: 3400-3409.