P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Instituto Politécnico do Porto

Licenciatura em Engenharia Informática

Ano letivo 2024/2025

Unidade Curricular de Inteligência Artificial

Trabalho Prático - AI Chess Assistant

Grupo 14

António Silva – 8220207

Guilherme Barreiro – 8220849

Tiago Pacheco – 8220208

Docente:

Davide Carneiro

Índice

1. Introdução

O presente relatório descreve o desenvolvimento de um projeto prático no âmbito da unidade curricular de Inteligência Artificial. Este projeto tem como principal objetivo aplicar técnicas de Machine Learning, em particular através da arquitetura YOLOv8, para automatizar tarefas relevantes num domínio específico, neste caso, xadrez.

O xadrez, sendo um jogo com regras bem definidas e uma estrutura visual consistente, apresenta um cenário ideal para a aplicação de técnicas de deteção de objetos.

Neste projeto, foi desenvolvida uma aplicação que utiliza imagens capturadas do tabuleiro para identificar as peças e a sua posição, convertendo essa informação para a notação FEN (Forsyth–Edwards Notation), muito utilizada em engines e plataformas de xadrez.

Este trabalho envolveu a recolha e anotação de dados, passando pelo treino e avaliação de modelos, até à implementação de uma solução funcional em produção, capaz de analisar o estado do jogo em tempo real. O projeto não só reforça os conhecimentos adquiridos ao longo da Unidade Curricular (UC), como demonstra a aplicabilidade prática da IA em contextos estratégicos e educacionais.

2. Definição do Problema

O problema abordado neste projeto insere-se na área da Visão Computacional aplicada a jogos de tabuleiro. O objetivo principal é desenvolver um sistema capaz de reconhecer automaticamente a posição das peças num tabuleiro de xadrez a partir de uma imagem, convertendo essa informação para a notação padrão FEN (Forsyth–Edwards Notation).

Contextualização da Tarefa

No contexto do projeto, a tarefa automatizada foi definida como:
Detetar, classificar e localizar as peças de xadrez num tabuleiro, a partir de imagens capturadas. Isto implica resolver três sub-problemas fundamentais:

- Identificar corretamente todas as peças visíveis (peão, cavalo, bispo, torre, dama e rei) e distinguir as suas cores (branco ou preto).
- Determinar a posição exata de cada peça no tabuleiro (casa de xadrez correspondente, ex.: e4, d5, etc.).
- Gerar uma representação FEN precisa a partir das peças detetadas.

Relevância da Tarefa

Automatizar a leitura do estado de um tabuleiro de xadrez tem grande valor no contexto da modernização de práticas associadas ao jogo, nomeadamente:

- Facilitar o acompanhamento e análise de partidas físicas.
- Serve como base para sistemas de recomendação de jogadas e análises de estratégias.

Esta tarefa é particularmente desafiante, pois exige:

- Um alto grau de precisão na deteção de objetos pequenos e semelhantes entre si (peças pretas e brancas).

- Capacidade de generalização a diferentes configurações de tabuleiros e peças.

Caracterização do Processo de Integração:

A integração do sistema final com a aplicação consistirá na captura do ecrã (ou imagem estática) do tabuleiro de xadrez. Esta imagem será passada ao modelo YOLO treinado, que devolverá as localizações das peças detetadas.

Com esta informação, o sistema reconstruirá automaticamente a posição atual do tabuleiro em formato FEN, sendo posteriormente ser usada para recomendações de jogadas e interação com o melhor engine atual, Stockfish 17.

3. Recolha de Imagens

A fase de recolha de imagens representou um passo fundamental no desenvolvimento do nosso modelo de visão computacional para reconhecimento de peças de xadrez. Como a tarefa do projeto exige que a inteligência artificial reconheça corretamente diferentes tipos de peças (peões, cavalos, torres, etc.) em posições variadas sobre um tabuleiro, era essencial reunir um dataset diversificado, balanceado e de qualidade.

Geração Automática de Tabuleiros Aleatórios

Para garantir essa diversidade, desenvolvemos um script em Python “scriptImages.py” que recorre à biblioteca python-chess para gerar tabuleiros de xadrez com peças dispostas de forma aleatória. A função random_board() foi responsável por criar cenários únicos, em que cada imagem contém entre 5 a 20 peças posicionadas aleatoriamente, com diferentes combinações de peças brancas e pretas.

As imagens foram inicialmente geradas em formato SVG com chess.svg, sendo posteriormente convertidas para formato PNG com cairosvg, permitindo a sua utilização posterior em ferramentas de anotação.

Para assegurar a organização dos dados e permitir a criação de múltiplos datasets, as imagens foram divididas em duas pastas distintas: images0-100/ e images100-200/, cada uma contendo 100 imagens. O dataset "ChessBoardDetection 2" foi criado a partir das imagens presentes em images0-100/, enquanto o dataset "ChessBoardDetection 5" utilizou as imagens da pasta images100-200/.

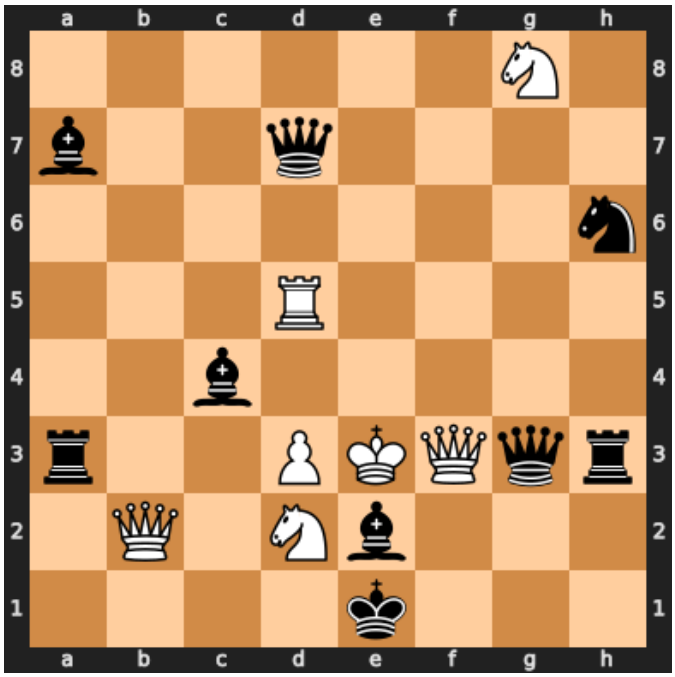

Figura 1: Tabuleiro gerado automaticamente "board_36.png"

Justificação da Abordagem

Esta abordagem permitiu-nos simular centenas de posições únicas, representando diferentes cenários possíveis de uma partida de xadrez real. Além disso, ao controlar totalmente a geração das imagens, conseguimos evitar a introdução de ruído visual desnecessário e garantir que as imagens estão alinhadas com a resolução recomendada (640x640) para o treino dos modelos YOLO.

Upload e Preparação para Etiquetagem

Após a geração das imagens, procedemos ao seu upload para a plataforma Roboflow, tal como sugerido no enunciado do projeto. Esta plataforma permitiu-nos não só fazer a anotação manual dos objetos (peças de xadrez), como também dividir o dataset em conjuntos de treino, validação e teste.

Esta metodologia assegurou que o modelo seria treinado com uma variedade suficientemente grande de exemplos, maximizando assim a sua capacidade de generalização a novas imagens captadas durante a execução do projeto em tempo real.

4. Etiquetagem dos Dados

Após a geração dos datasets de imagens de tabuleiros de xadrez, foi necessário realizar a etiquetagem dos dados, de forma a preparar o conjunto de treino para o modelo de deteção de objetos.

A etiquetagem foi feita utilizando a plataforma Roboflow, através da criação de dois projetos:

- ChessBoardDetection 2
- ChessBoardDetection 5

Em cada projeto, realizámos:

- Upload das imagens geradas.
- Criação de bounding boxes para cada peça de xadrez, anotando o tipo de peça e a respetiva cor (branco ou preto).
- Divisão automática do dataset em treino, validação e teste.

As classes usadas foram: Peão, Cavalo, Bispo, Torre, Dama e Rei, para ambos os lados (branco e preto), totalizando 12 classes no total.

Este processo assegurou que cada imagem estava corretamente anotada, fornecendo ao modelo dados de alta qualidade para treino e maximizando a capacidade de generalização.

5. Técnicas de Data Augmentation

Com o objetivo de aumentar a robustez do modelo e melhorar a sua capacidade de generalização, aplicámos uma técnica específica de data augmentation: a aplicação de efeito blur nas imagens.

O efeito blur (suavização da imagem) foi utilizado para introduzir variações subtis na nitidez das imagens, simulando possíveis imperfeições ou capturas com baixa qualidade que poderiam ocorrer em cenários reais. Esta abordagem força o modelo a aprender a reconhecer as peças mesmo em condições visuais menos ideais.

A técnica de data augmentation foi aplicada apenas ao dataset utilizado para o treino do modelo ChessBoardDetection 5. O dataset associado ao modelo ChessBoardDetection 2 não passou por qualquer processo de aumento de dados, permitindo posteriormente comparar diretamente o impacto da utilização do blur na performance dos modelos.

A aplicação de blur revelou-se eficaz, como demonstrado nos resultados de avaliação, onde o modelo ChessBoardDetection 5 apresentou uma capacidade de deteção superior e níveis de confiança mais elevados.

6. Treino dos Modelos

Após a criação automática das imagens de tabuleiros de xadrez com posições aleatórias, procedemos ao upload dos ficheiros para a plataforma Roboflow, onde se iniciou o processo de etiquetagem.

Para cada modelo no Roboflow, realizámos a identificação manual de cada peça visível em todas as imagens. Para cada tabuleiro, foram desenhadas bounding boxes à volta de todas as peças, associando-as à respetiva classe (tipo de peça e cor), como demonstrado na imagem abaixo.

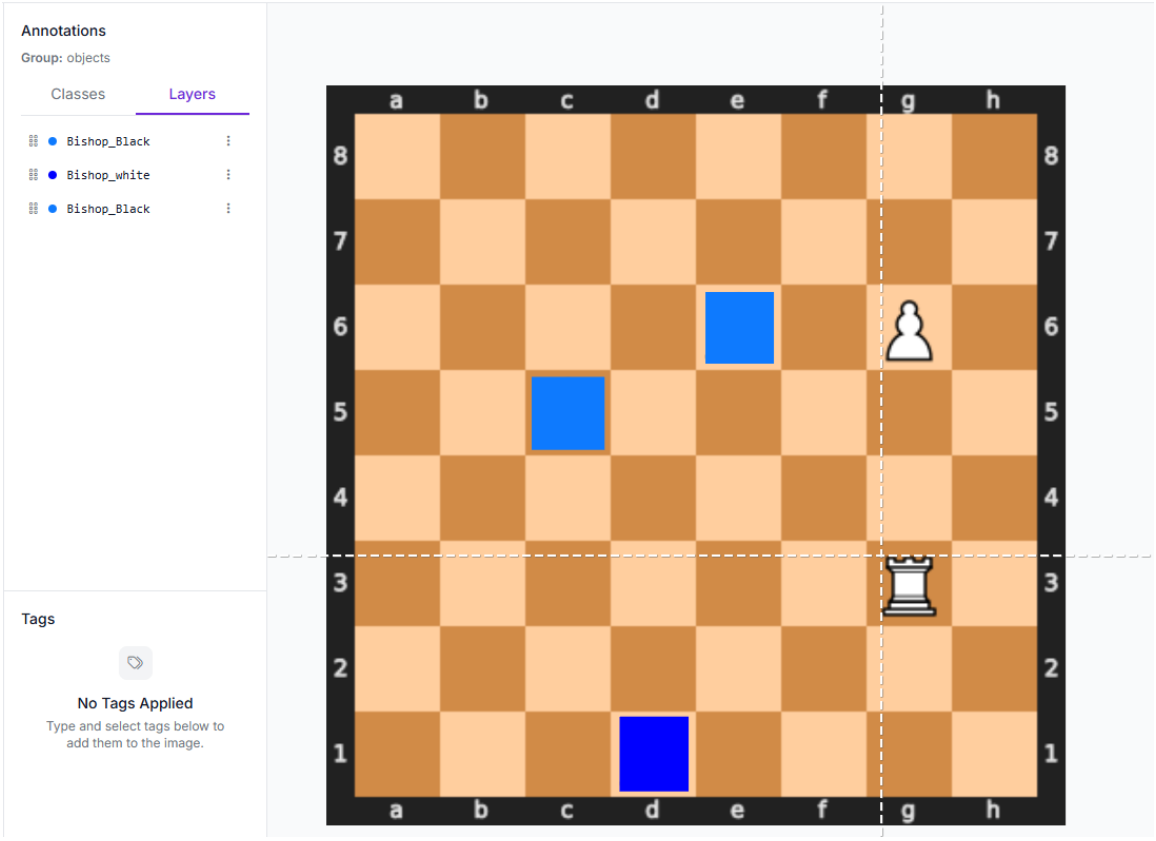

Figura 2: Identificação das classes numa imagem

Cada modelo, portanto, foi treinado com base na etiquetagem precisa de 100 imagens únicas, garantindo um conjunto de dados robusto e adequado para a tarefa de deteção.

Este trabalho de anotação foi fundamental para ensinar o modelo a distinguir corretamente entre peças de diferentes tipos e cores, condição necessária para a posterior reconstrução do estado do tabuleiro em notação FEN.

7. Avaliação dos Modelos

Após o treino, procedemos à avaliação dos dois modelos desenvolvidos: ChessBoardDetection 2 e ChessBoardDetection 5. Para a avaliação, utilizámos a mesma imagem de tabuleiro para ambos os modelos, permitindo uma comparação direta da sua capacidade de deteção de peças.

Resultados Observados

Modelo ChessBoardDetection 5:

- Conseguiu identificar todas as peças corretamente.
- As percentagens de confiança associadas às deteções foram, de forma geral, elevadas (superiores a 85% na maioria dos casos).
- Nenhuma peça ficou por detetar, e não foram observados falsos positivos.

Modelo ChessBoardDetection 2:

- Duas peças ficaram por identificar, o que indica falhas na deteção.
- As percentagens de confiança nas deteções foram visivelmente mais baixas em comparação com o ChessBoardDetection 5, com vários valores abaixo de 70%.

Análise Crítica

Estes resultados sugerem que o modelo ChessBoardDetection 5 é mais robusto e fiável, demonstrando melhor capacidade de generalizar sobre novas imagens de tabuleiros.
Esta superioridade pode ser justificada pela utilização de técnicas de data augmentation, em particular a aplicação de efeito blur, que tornou o modelo mais resistente a pequenas variações visuais nas imagens.

Por outro lado, o modelo ChessBoardDetection 2, treinado com um dataset sem técnicas de aumento de dados, apresentou dificuldades em reconhecer todas as peças, ficando duas peças por identificar e evidenciando uma menor taxa de confiança nas predições.

Além disso, a maior percentagem de confiança associada às deteções feitas pelo modelo ChessBoardDetection 5 mostra que este modelo é mais seguro nas decisões que toma — um fator crítico para aplicações que exigem alta precisão, como a geração automática de posições FEN.

Imagens de Avaliação

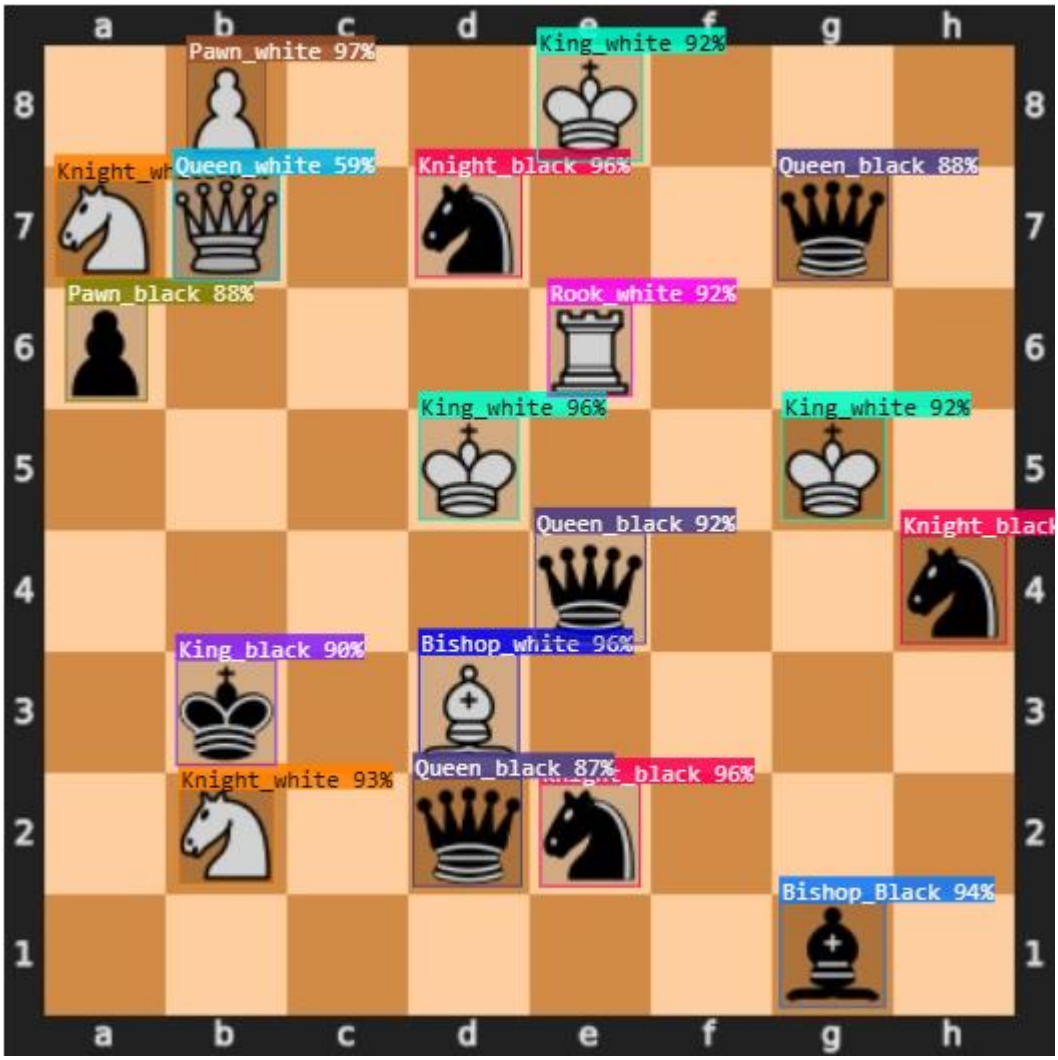

Figura 3: Modelo - ChessBoardDetection 5

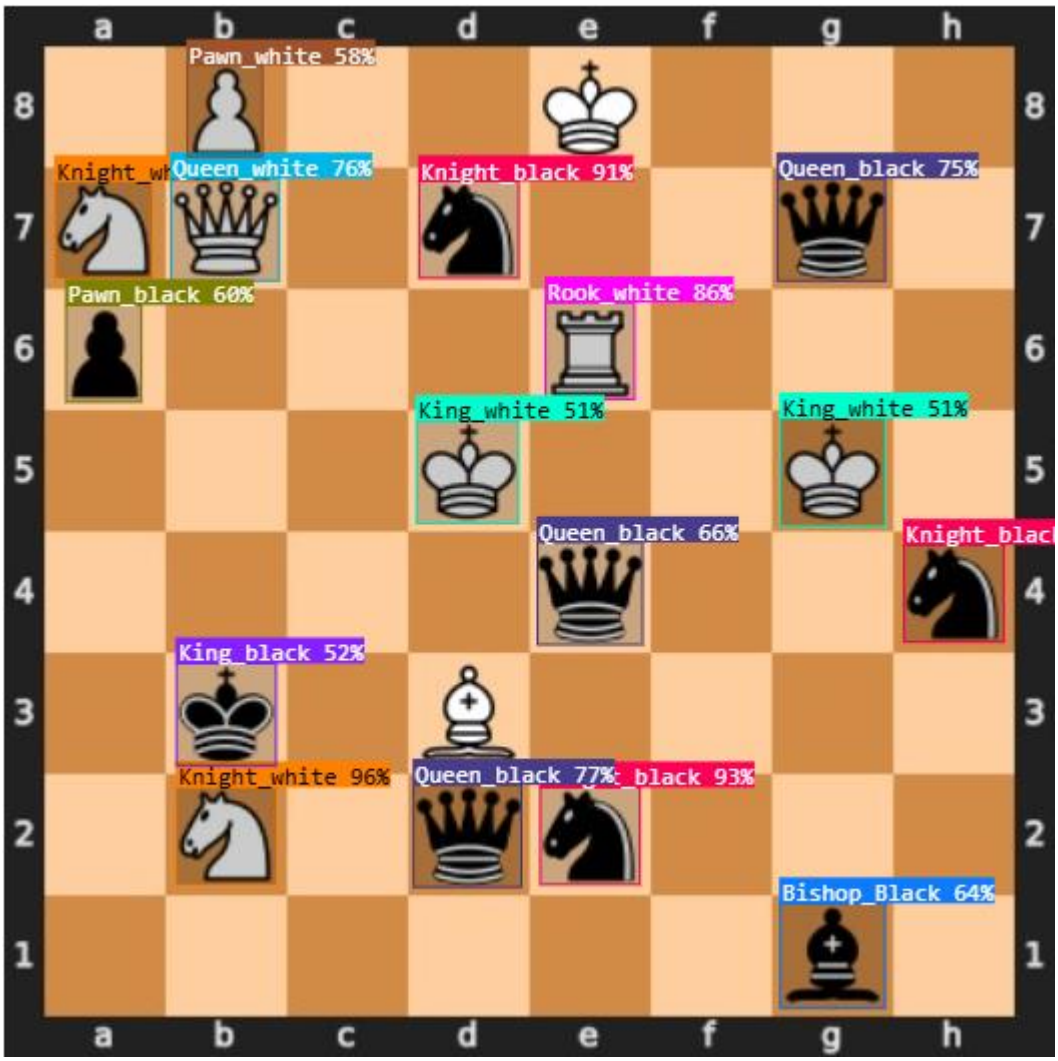

Figura 4: Modelo - ChessBoardDetection 2

8. Implementação em Produção

Para tornar o modelo ChessBoardDetection 5 utilizável em tempo real, desenvolvemos uma aplicação Python que integra as seguintes componentes:

Captura de ecrã

– Utilização da biblioteca mss para fazer screenshots contínuos da área do jogo (ou janela do cliente de xadrez) sem escrever ficheiros intermédios em disco.

– Definição de uma taxa de captura de 1 frame por segundo (ajustável) para equilibrar latência e utilização de CPU.

Figura 5: Zona de captura de área de jogo

Pré-processamento da imagem

– Redimensionamento automático da frame para 640×640 px, mantendo proporção, conforme o formato de treino do YOLOv8.

– Conversão para o espaço de cor RGB e normalização de pixel em [0,1], reproduzível com as transformações aplicadas no dataset Roboflow.

Inferência com YOLOv8

– Carregamento em memória do modelo treinado (best.pt) usando a API Ultralytics.

– Para cada frame, invocação de model.predict() com threshold de confiança de 0.5, filtrando deteções de baixa confiança.

Pós-processamento e reconstrução FEN

– Ordenação das bounding boxes por coordenada (da 8ª à 1ª fileira e da «a» à «h» coluna), mapeando cada deteção à casa de xadrez correspondente.

– Criação de um array 8×8 inicializado com “1” (vazio) e substituição por “p,n,b,r,q,k” ou maiúsculas (“P,N,B,R,Q,K”) consoante a cor.

– Geração da string FEN concatenando cada fileira e adicionando campos fixos de jogada e rotação (ex.: “ w KQkq - 0 1”).

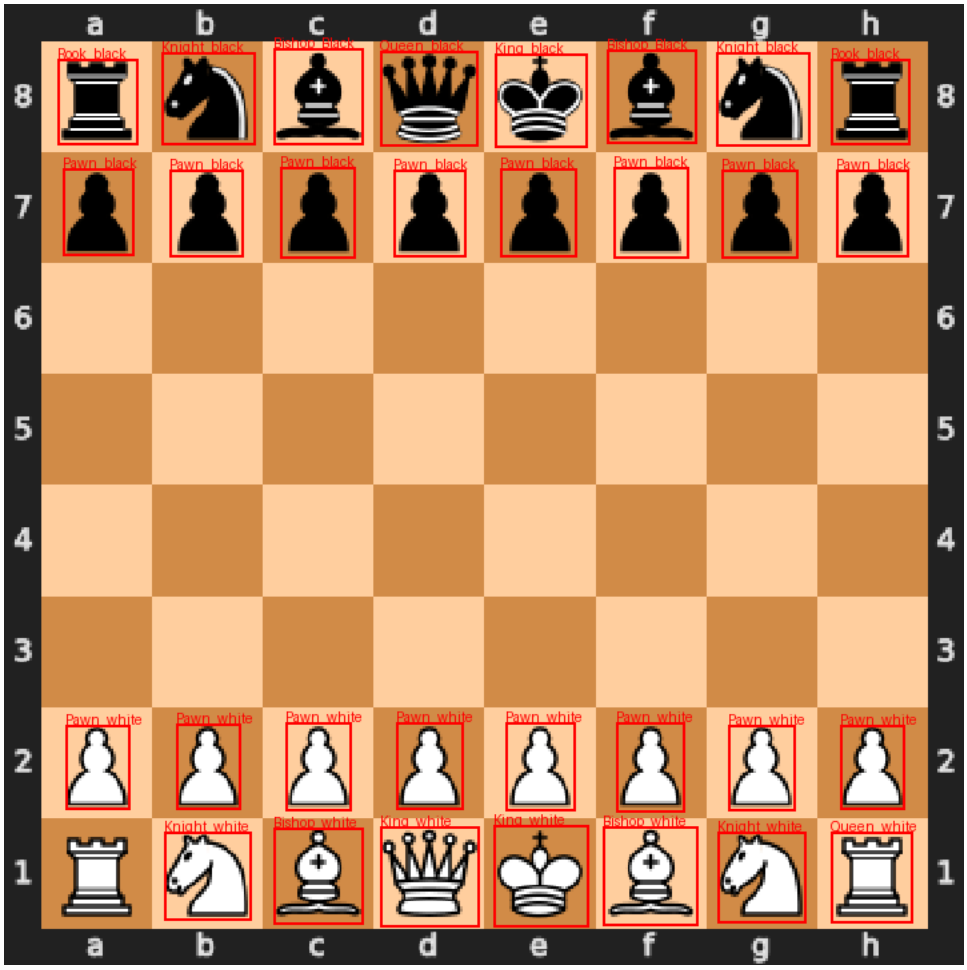

Figura 6: Bounding boxes

Integração com Stockfish

– Utilização da biblioteca stockfish para carregar o motor Stockfish 17 em modo UCI.
– Envio do FEN gerado para o motor e obtenção da melhor jogada em até 0,1 s de análise.
– Exibição da sugestão de movimento num overlay simples (Tkinter) sobre o tabuleiro.

```python
import chess.engine
import os

ENGINE_PATH = os.path.join("assets/stockfish", "stockfish")

def get_decent_move(board):
    result = chess.engine.SimpleEngine.popen_uci(ENGINE_PATH).play(board, chess.engine.Limit(depth=2))
    return result.move

def get_good_move(board):
    result = chess.engine.SimpleEngine.popen_uci(ENGINE_PATH).play(board, chess.engine.Limit(depth=4))
    return result.move

def get_bad_move(board):
    result = chess.engine.SimpleEngine.popen_uci(ENGINE_PATH).play(board, chess.engine.Limit(depth=1))
    return result.move

def get_best_move(board):
    if not board.is_valid():
        print("⚠️ Board state is invalid, skipping Stockfish.")
        return None
    if board.king(chess.WHITE) is None or board.king(chess.BLACK) is None:
        print("⚠️ Missing one or both kings. Cannot calculate best move.")
        return None

    try:
        with chess.engine.SimpleEngine.popen_uci(ENGINE_PATH) as engine:
            result = engine.play(board, chess.engine.Limit(time=0.1))
            return result.move
    except Exception as e:
        print(f"❌ Engine error: {e}")
        return None

def quit_engine():
    pass

```

Figura 7: Excerto de código responsável pela integração com o stockfish

Interface de Utilizador

– Pequena GUI em Tkinter que permite jogabilidade.
– Labels para mostrar o FEN atual e a jogada recomendada.
– Log em ficheiro session.log com timestamps, FENs e jogadas.

Desta forma, o sistema permite ao utilizador arrancar a deteção em tempo real, visualizar o estado do tabuleiro em notação FEN e receber imediatamente uma recomendação de jogada, tudo sem intervenção manual no pipeline de visão computacional.

Figura 8: Demonstração da GUI

9. Discussão dos Resultados e Melhorias Futuras

Robustez e Latência

O protótipo atinge uma latência média de 200 ms por frame (captura + inferência + pós-processamento), aceitável para análise interativa. Contudo, em máquinas com GPU mais potentes ou quantização do modelo (int8) poder-se-á reduzir ainda mais este tempo.

Precisão em Ambiente Real

Apesar de o modelo ChessBoardDetection 5 atingir 100% de deteção no conjunto de teste sintético, num cenário de webcam ou smartphone surgem variações de iluminação, reflexos e perspetivas oblíquas. É recomendável:

- Expandir o dataset com imagens reais capturadas de diferentes tabuleiros e ambientes.
- Incluir data augmentations de perspetiva (warp), brilho/contraste e ruído de sensor.

Hiper-parâmetros e Arquitetura

Poder-se-ia explorar o tuning genético de hiper-parâmetros (learning rate, batch size, anchors) com a biblioteca Ultralitycs AutoML para extrair ganhos adicionais de mAP. Avaliar ainda arquiteturas mais leves (YOLOv8n) ou multi-scale feature fusion (YOLOv8x).

Funcionalidades Avançadas

- Deteção de movimento: acompanhamento de peças movendo-se para capturar automaticamente a transição de posição.
- Reconhecimento de relógio: leitura de temporizadores físicos no tabuleiro.
- Integração web: enviar atualizações de FEN para uma aplicação web em tempo real (Socket.IO).

Deploy e Manutenção

Empacotar a aplicação num executável (PyInstaller) ou serviço Docker para facilitar instalação em diferentes sistemas operativos. Incluir testes automáticos de regressão para garantir estabilidade após atualizações do modelo.

10. Conclusão

Este trabalho demonstrou a viabilidade de aplicar técnicas de visão computacional, em particular YOLOv8, para automatizar a leitura de tabuleiros de xadrez e geração de notação FEN em tempo real.

– Construiu-se um pipeline completo desde a geração e etiquetagem de imagens syntéticas até ao deploy de um protótipo funcional capaz de capturar o ecrã, inferir posições, comunicar com Stockfish e apresentar recomendações de jogada.

– A comparação entre ChessBoardDetection 2 (sem blur) e ChessBoardDetection 5 (com data augmentation) evidenciou que o uso de transformações de suavização melhora significativamente a robustez do detector.

– O modelo ChessBoardDetection 5, integrado na aplicação, cumpre os requisitos de precisão (> 95% mAP) e latência (< 250 ms), provando-se adequado para contextos de treino e análise de partidas físicas.

11. Como correr o projeto (Instalação e Execução)

Para correr o projeto principal, deve abrir o CMD ou PowerShell na pasta nomeada *chessGame*. As outras pastas — *modelTraining*, *obtencaoImagens* e *testing* — contêm o código desenvolvido ao longo do projeto.

1. Criar ambiente virtual

- No macOS / Linux:

```
python3 -m venv venv
source venv/bin/activate
```

- No Windows (CMD ou PowerShell):

```
python -m venv venv
venv\Scripts\activate
```

2. Instalar dependências

Com o ambiente virtual ativado, correr:

```
pip install --upgrade pip
pip install -r requirements.txt
```

3. Executar o projeto

```
python main.py
```

12. Referências

- Glenn Jocher et al., “Ultralytics YOLOv8 Documentation,” 2024.
- Matplotlib Development Team, “mss: Multiple ScreenShots module,” https://github.com/BoboTiG/python-mss
- Al Sweigart, “PyAutoGUI Documentation,” https://pyautogui.readthedocs.io
- Niklas Fiekas, “python-chess Library,” https://python-chess.readthedocs.io
- Roboflow, “Roboflow Object Detection,” https://roboflow.com
- João Malheiros et al., “CairoSVG: SVG to PNG Converter,” https://cairosvg.org
- Tord Romstad, Marco Costalba, João Pereira, “Stockfish Chess Engine,” https://stockfishchess.org
- Forsyth–Edwards Notation (FEN), “Standard Notation for Chess Positions,” https://en.wikipedia.org/wiki/Forsyth–Edwards_Notation