

# Projeto Aprendizagem Profunda para Visão por Computador

Extração Automática do Estado do Tabuleiro de Damas

André Eusébio

Filipe Pereira

Vicente Chã

**Resumo**—Este projeto apresenta o desenvolvimento de um sistema para detetar, através de uma imagem real, o estado atual de um jogo de damas e transferi-lo para um ambiente virtual com as peças nas mesmas posições. O sistema utiliza o modelo de deteção de objetos *YOLO12* para identificar e classificar as peças brancas e pretas. Para garantir a precisão, a imagem original passa por um pré-processamento de correção geométrica, transformando perspectivas arbitrárias numa vista de topo.

## REPOSITÓRIO

<https://github.com/RICADINHO/TrabalhoAPVC>

## I. INTRODUÇÃO

Como projeto final da unidade curricular de "Aprendizagem Profunda para Visão por Computador" foi-nos proposto desenvolver um projeto livre no qual aplicássemos, de forma prática, algumas das técnicas aprendidas ao longo da UC. Decidimos criar um jogo de damas em que o utilizador carrega uma imagem de um tabuleiro com as respetivas peças, usada como estado inicial do jogo na aplicação.

O mapeamento entre a posição das peças na imagem e a matriz que representa o estado interno do jogo é realizado recorrendo ao detetor de objetos *YOLO*. A imagem fornecida pelo utilizador, pode corresponder a um tabuleiro capturado sob uma perspetiva não ideal (por exemplo, inclinado ou torto), no entanto, antes de se proceder ao mapeamento, a imagem é transformada para uma vista *top-down*. Este pré-processamento permite uma deteção mais precisa das peças e, consequentemente, uma correta identificação da sua posição real no tabuleiro.

## II. Fine tuning DO MODELO

Um dos objetivos deste projeto era o *fine tuning* de um modelo de visão por computador para uma determinada função

(identificar peças de damas neste caso). Para tal escolhemos o modelo *YOLO12* pela sua eficiência e facilidade em fazer *fine tune*.

### A. Conjunto de dados

Para este projeto, utilizamos um conjunto de dados especializado em peças de jogo de damas, estruturado para permitir a deteção e classificação precisa das peças em diferentes condições de iluminação e posição.

O *dataset* já vem definido em três conjuntos distintos:

- **Train Set (84%):** 1062 imagens.
- **Valid Set (5%):** 68 imagens.
- **Test Set (11%):** 135 imagens.

#### 1) Pré-processamento e Augmentação

Todas as imagens foram submetidas a um pré-processamento de *Auto-Orient* e redimensionadas para uma resolução de  $640 \times 640$  pixels através do método de *stretch*, garantindo a compatibilidade com o tamanho do *input* do *YOLO12*.

Para aumentar a robustez do modelo e evitar o *overfitting*, aplicámos técnicas de *data augmentation*, gerando 3 variações por cada exemplo de treino. As transformações incluíram:

- **Rotação:** Entre  $-14^\circ$  e  $+14^\circ$ .
- **Brilho:** Ajustes entre  $-20\%$  e  $+20\%$ .
- **Desfoque:** Até 2.7px.

### B. Treino do Modelo

O treino foi realizado utilizando o *YOLO12*, ideal para aplicações de tempo real. Configurámos o treino para 100 épocas, com um *batch size* de 16. Implementámos também um mecanismo de *early stopping* com paciência de 20 para interromper o treino caso a métrica de validação deixasse de melhorar.

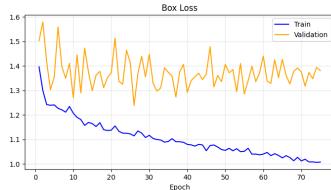


Figura 1. *Box Loss* ao longo das épocas de treino

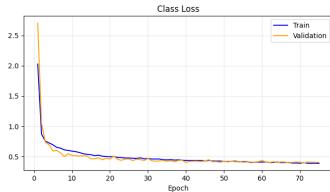


Figura 2. *Class Loss* ao longo das épocas de treino

### C. Resultados

Os resultados obtidos demonstram uma excelente capacidade de deteção das peças, conforme detalhado na Tabela II-C.

Classe	Imagens	Instâncias	Box(Precision)	Recall	mAP50	mAP50-95
all	135	1526	0.929	0.977	0.975	0.622
black	123	775	0.922	0.959	0.970	0.613
white	120	751	0.936	0.995	0.979	0.631

Tabela I  
MÉTRICAS DO *fine tune* PARA DUAS CLASSES

#### 1) Análise das Métricas

As curvas de aprendizagem revelam a convergência do modelo:

- **Box Loss (Figura 1):** Indica a precisão com que o modelo localiza as peças (*bounding boxes*).
- **Class Loss (Figura 2):** Demonstra a eficácia na distinção entre peças brancas e pretas.
- **mAP50 (Figura 3):** O modelo atingiu uma precisão média de 0.975 a 50% de *IoU*, o que é considerado excelente para esta aplicação.

#### 2) Resultados de 3 vs 2 classes

Um ponto crucial do desenvolvimento foi a decisão de filtrar as classes utilizadas. Inicialmente, o *dataset* incluía a classe



Figura 3. *mAP* ao longo das épocas de treino

BOARD. No entanto, como demonstrado na Tabela II-C2, a inclusão desta classe prejudicava a performance global.

A classe BOARD apresentou uma precisão muito baixa (0.259), provavelmente devido à grande área que ocupa na imagem e à confusão com o fundo. Ao restringir o modelo apenas às peças, observamos uma melhoria na *Precision* global (de 0.684 para 0.929) e no *mAP50* (de 0.760 para 0.975). Esta simplificação permitiu que o modelo se focasse exclusivamente nos objetos de maior interesse tático para o jogo.

Class	Tipo	Box(Precision)	Recall	mAP50	mAP50-95
Board	3 Classes	0.259	1.000	0.369	0.367
Black	2 Classes	0.922	0.959	0.970	0.613
	3 Classes	0.860	0.969	0.940	0.584
	Diff	-0.062	+0.010	-0.030	-0.029
White	2 Classes	0.936	0.995	0.979	0.631
	3 Classes	0.934	0.996	0.970	0.615
	Diff	-0.002	+0.001	-0.009	-0.016
All	2 Classes	0.929	0.977	0.975	0.622
	3 Classes	0.684	0.988	0.760	0.522
	Diff	-0.245	+0.011	-0.215	-0.100

Tabela II  
COMPARAÇÃO DA PERFORMANCE COM DIFERENTES CLASSES

### III. PROCESSAMENTO DE IMAGEM E EXTRAÇÃO DO ESTADO DO TABULEIRO

Esta secção descreve o processamento de imagem desenvolvido para extrair automaticamente o estado de um tabuleiro de damas. O processamento é composto por quatro etapas principais: deteção das extremidades do tabuleiro através da Transformada de Hough, correção de perspetiva por estimativa de uma homografia, deteção de peças com um modelo *YOLO* aplicado à imagem retificada e, por fim, construção de uma matriz  $8 \times 8$  que representa o estado do jogo.



Figura 4. Exemplos de imagens de entrada consideradas no processamento: vista superior do tabuleiro (esquerda) e vista oblíqua do tabuleiro (direita).

Ao longo deste trabalho consideram-se dois cenários distintos de aquisição de imagem: (i) uma vista aproximadamente

ortogonal do tabuleiro, com distorção de perspetiva reduzida, e (ii) uma vista capturada a partir de um ângulo oblíquo, introduzindo deformações geométricas mais significativas.

#### A. Deteção do Tabuleiro com Transformada de Hough e Correção de Perspetiva

Numa primeira fase, a imagem de entrada é convertida para tons de cinzento e submetida a um filtro bilateral, com o objetivo de reduzir o ruído preservando simultaneamente as arestas. De seguida, é aplicado um limiar adaptativo gaussiano, permitindo acomodar variações de iluminação. As arestas são então extraídas através do operador de Canny.

Sobre a imagem de arestas é aplicada a Transformada de *Hough* probabilística (*HoughLinesP*), que permite detetar segmentos de reta correspondentes às linhas do tabuleiro. Os pontos extremos desses segmentos são agregados e utilizados para calcular o fecho convexo da região detetada. Posteriormente, este fecho é aproximado a um polígono, sendo selecionados quatro vértices que representam os cantos do tabuleiro.



Figura 5. Segmentos de reta detetados pela Transformada de *Hough* probabilística, correspondentes às linhas do tabuleiro.

Os cantos detetados são ordenados de forma consistente (canto superior esquerdo, canto superior direito, canto inferior direito e canto inferior esquerdo), condição essencial para a estimação correta da homografia e consequente correção de perspetiva.

No caso de imagens adquiridas numa vista aproximadamente superior, a deteção automática dos cantos revela-se robusta, permitindo proceder diretamente à correção de perspetiva. No entanto, quando a imagem é capturada a partir de um ângulo oblíquo, a presença de distorções acentuadas e possíveis oclusões das linhas do tabuleiro pode comprometer a fiabilidade da deteção automática.

Nessas situações, é adotada uma abordagem de seleção manual dos cantos do tabuleiro, garantindo uma correspondência

correta entre os pontos da imagem original e o plano retificado, e assegurando assim uma correção de perspetiva consistente.

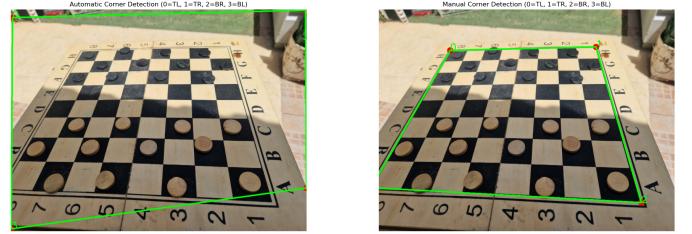


Figura 6. Estimativa dos cantos do tabuleiro numa vista oblíqua: deteção automática (esquerda) e seleção manual dos vértices (direita).

#### B. Homografia e Retificação do Tabuleiro

A homografia  $\mathbf{H}$  é estimada a partir da correspondência entre os quatro cantos do tabuleiro na imagem original e os vértices de um quadrado alinhado num plano cartesiano. Esta matriz permite mapear a perspetiva original para uma vista ortogonal do tabuleiro.

A estimação da homografia é realizada através do método *Direct Linear Transformation* (DLT), recorrendo à função `findHomography` da biblioteca OpenCV. De seguida, a transformação de perspetiva é aplicada à imagem original utilizando a função `warpPerspective`, resultando numa imagem retificada do tabuleiro.

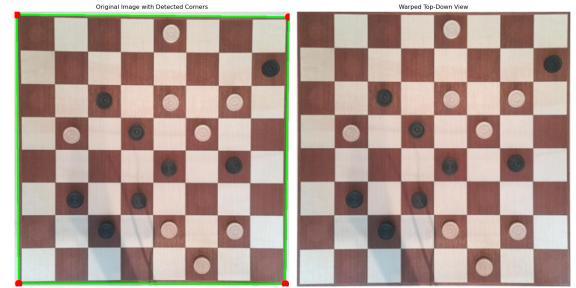


Figura 7. Exemplo de retificação do tabuleiro numa vista superior após estimação automática da homografia.

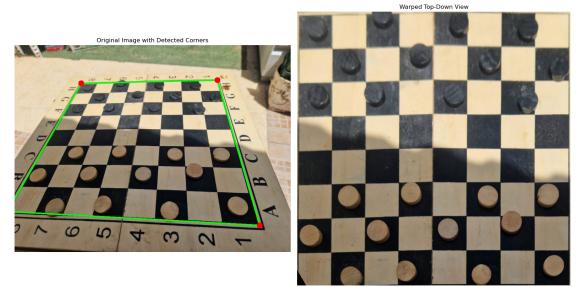


Figura 8. Exemplo de retificação do tabuleiro numa vista oblíqua após seleção manual dos cantos.

A imagem retificada apresenta dimensões proporcionais ao tamanho real do tabuleiro, sendo escalada de forma a facilitar a deteção subsequente das peças.

### C. Detecção de Peças com YOLO na Imagem Retificada

Com o tabuleiro corretamente alinhado, é aplicado um modelo *YOLO* previamente treinado para a deteção das peças de jogo. O modelo é executado exclusivamente sobre a imagem retificada, garantindo que cada peça se encontra corretamente alinhada com as células do tabuleiro.

Para cada deteção, o modelo fornece a classe da peça, a confiança associada e as coordenadas da caixa delimitadora. O centro geométrico de cada caixa é calculado e utilizado para determinar a célula do tabuleiro onde a peça se encontra.

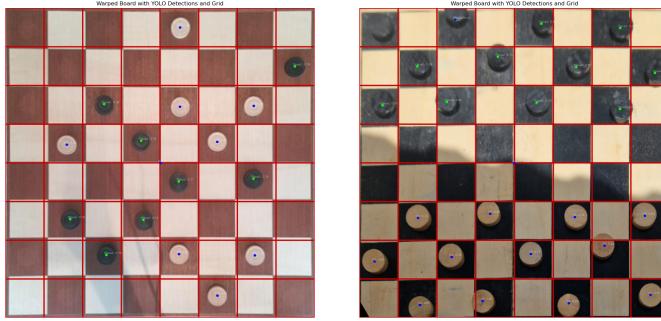


Figura 9. Resultados da deteção de peças com *YOLO* na imagem retificada: vista superior (esquerda) e vista oblíqua (direita).

Quando ocorrem múltiplas deteções na mesma célula, é selecionada apenas a deteção com maior valor de confiança, reduzindo a ocorrência de falsos positivos.

### D. Construção da Matriz $8 \times 8$ do Tabuleiro

A etapa final consiste na construção de uma matriz discreta  $8 \times 8$  que representa o estado do tabuleiro. A imagem retificada é dividida uniformemente em 64 células, correspondentes às casas do tabuleiro.

Cada deteção é associada a uma célula  $(i, j)$  com base nas coordenadas do centro da peça. A matriz resultante é definida da seguinte forma:

- 0 – célula vazia;
- 1 – peça da classe 1 (peça preta);
- 2 – peça da classe 2 (peça branca).

$$\mathbf{T} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 \end{pmatrix}$$

Esta matriz constitui uma representação simbólica e compacta do estado do jogo, sendo posteriormente utilizada para inicializar o processamento lógico do jogo de damas.

## IV. PIPELINE DO SISTEMA DE RECONSTRUÇÃO DO JOGO DE DAMAS

Esta secção descreve a pipeline do sistema desenvolvido para reconstruir automaticamente um jogo de damas a partir de uma imagem real do tabuleiro. O sistema utiliza técnicas de visão por computador, homografia projetiva e deteção de objetos, anteriormente apresentadas.

Para jogar, basta executar o ficheiro `main.py` disponível no repositório do projeto, iniciando assim o jogo.

### A. Upload da Imagem do Tabuleiro

O primeiro passo da *pipeline* consiste no carregamento de uma imagem do tabuleiro de damas pelo utilizador. Este processo é iniciado no menu principal da aplicação, onde o utilizador seleciona a imagem através do explorador de ficheiros do sistema operativo.

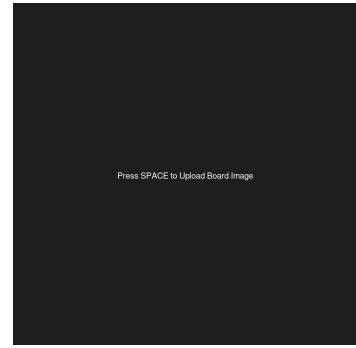


Figura 10. Ecrã inicial da aplicação onde é solicitado o *upload* da imagem do tabuleiro.

Após a seleção, a imagem é carregada na sua resolução original e armazenada para processamento posterior. Este procedimento permite utilizar imagens capturadas com diferentes dispositivos, ângulos de visão e condições de iluminação.

### B. Detecção do Tabuleiro e Cálculo da Homografia

Depois de carregada a imagem, o sistema realiza a deteção automática dos quatro cantos do tabuleiro de damas. Esta

deteção baseia-se em métodos clássicos de processamento de imagem, nomeadamente extração de arestas, deteção de segmentos de reta e cálculo do invólucro convexo, permitindo identificar a região correspondente ao tabuleiro.

Com base nos quatro cantos detetados, é calculada uma homografia que transforma a imagem original, potencialmente capturada com perspetiva inclinada, numa vista superior do tabuleiro. O resultado desta transformação é apresentado ao utilizador para validação.

Caso o utilizador verifique que os cantos detetados automaticamente não estão corretamente posicionados, o sistema permite o seu ajuste manual através da deslocação direta dos pontos. Esta abordagem combina automatização com intervenção do utilizador, aumentando a robustez do sistema.

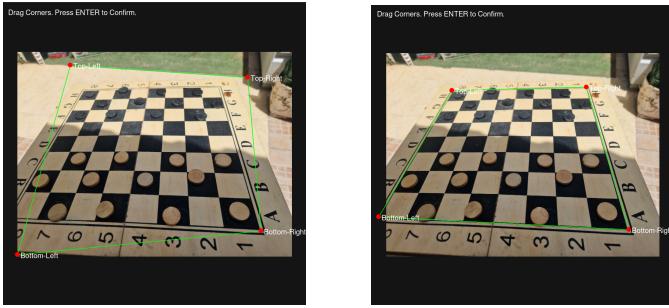


Figura 11. Deteção automática dos cantos do tabuleiro (esquerda) e ajuste manual efetuado pelo utilizador (direita).

Após a validação final, a homografia é recalculada e aplicada, resultando numa imagem do tabuleiro retificada e alinhada.

### C. Reconstrução do Estado do Jogo

Com a imagem do tabuleiro corrigida geometricamente, o sistema procede à deteção das peças de jogo utilizando um modelo *YOLO* previamente treinado para distinguir peças de diferentes cores. As posições das peças detetadas são projetadas para uma grelha regular correspondente às casas do tabuleiro.

Com base nesta informação, é construída uma matriz  $8 \times 8$  que representa o estado do jogo, onde cada posição indica a presença ou ausência de uma peça, bem como o seu tipo. Esta matriz é depois convertida para o formato interno utilizado pelo *game engine*.

Por fim, o jogo é reconstruído no ambiente gráfico, permitindo ao utilizador continuar a partida a partir do estado real capturado na imagem inicial.

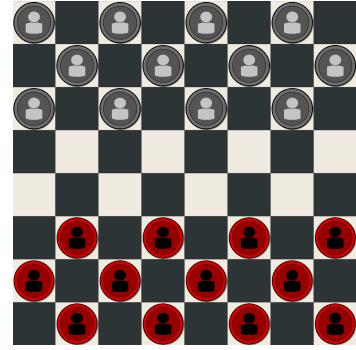


Figura 12. Estado do jogo reconstruído no ambiente gráfico com base na matriz obtida.

O *game engine* e a lógica base do jogo de damas utilizados neste trabalho foram adaptados a partir de um projeto de código aberto disponível no repositório *pythoncode-tutorials*, mais concretamente da implementação de um jogo de damas em Python com interface gráfica desenvolvida em Pygame<sup>1</sup>. Sobre esta base foram integrados os módulos de visão por computador e de reconstrução automática do estado do jogo.

## V. CONCLUSÕES

O projeto atingiu com sucesso o objetivo de converter o estado de um jogo de damas real para um ambiente virtual a partir de uma imagem. Através do *fine-tuning* do modelo *YOLOv2*, obteve-se uma precisão média (*mAP50*) de 0.975. O sistema demonstrou robustez graças à combinação de:

- Pré-processamento Geométrico: O uso de transformações de homografia permitiu converter perspetivas arbitrárias numa vista de topo ideal para a análise.
- Interatividade: A possibilidade de ajuste manual dos cantos do tabuleiro garantiu a precisão do mapeamento mesmo em condições adversas.
- Integração de Dados: A extração das posições para uma matriz  $8 \times 8$  permitiu a reconstrução fiel do estado do jogo.

Conclui-se que a *pipeline* desenvolvida é uma solução eficiente para a conexão entre ambientes reais e virtuais através da integração de visão computacional.

<sup>1</sup><https://github.com/x4nth055/pythoncode-tutorials/tree/master/gui-programming/checkers-game/>