INSTITUTO FEDERAL DE SANTA CATARINA

ANDREY ADRIANO DA ROSA

Reconhecimento por Imagem de Lances de Xadrez com Visão Computacional e Redes Neurais Convolucionais

São José - SC

fevereiro/2025

RECONHECIMENTO POR IMAGEM DE LANCES DE XADREZ COM VISÃO COMPUTACIONAL E REDES NEURAIS CONVOLUCIONAIS

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Prof. Roberto Wanderley da Nobrega, Dr.

Coorientador: Prof. Marcos Moecke, Dr.

São José - SC

fevereiro/2025

Andrey Adriano da Rosa

Reconhecimento por Imagem de Lances de Xadrez com Visão Computacional e Redes Neurais Convolucionais

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 28 de fevereiro de 2025:

gov.br
Documento assinado digitalmente
ROBERTO WANDERLEY DA NOBREGA
Data: 20/03/2025 19:40:21-0300
Verifique em https://validar.iti.gov.br

Prof. Roberto Wanderley da Nobrega, Dr.
Orientador
Instituto Federal de Santa Catarina

GOV.BR
Documento assinado digitalmente
RAMON MAYOR MARTINS
Data: 21/03/2025 10:29:29-0300
CPF: ***.866.756-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Ramon Mayor Martins, Dr.
Instituto Federal de Santa Catarina

GOV.BR
Documento assinado digitalmente
MARIO DE NORONHA NETO
Data: 24/03/2025 17:38:11-0300
CPF: ***.859.519-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Mário de Noronha Neto, Dr.
Instituto Federal de Santa Catarina

AGRADECIMENTOS

Gostaria de expressar minha gratidão a todos que fizeram parte desta jornada e contribuíram para a realização deste trabalho. Esse período foi cheio de desafios e muito transformador.

Agradeço ao meu pai, que sempre trabalhou muito para me proporcionar as melhores oportunidades e me ensinou a importância da dedicação e do esforço para alcançar meus objetivos, e à minha mãe, que sempre esteve ao meu lado, cuidou de mim, me apoiou e incentivou a seguir em frente. Eles me ensinaram os valores que me trouxeram até aqui e sem o seu apoio incondicional nada disso seria possível.

Aos meus orientadores, Prof. Roberto Wanderley da Nobrega e Prof. Marcos Moecke, por me orientarem neste tema e por todo o apoio durante o desenvolvimento deste trabalho.

Agradeço também ao Prof. Ramon Mayor Martins, que me deu sugestões que foram importantes para o projeto e que me ajudaram muito.

Aos meus amigos Lucas Coelho Raupp e Fausto Cristiano, que estiveram ao meu lado desde o início dessa caminhada. Compartilhamos muitos desafios, conquistas e aprendizados, nos apoiamos e crescemos juntos.

À minha amiga Rebeca, pelas conversas diárias ao longo do último ano e por todo o apoio e incentivo, mesmo que à distância.

A todos os amigos que fiz dentro do IFSC nesses cinco anos de graduação, e aos colegas do Inova IFSC, com quem compartilhei muitos momentos bons. Não vou citar nomes pois são muitos e não quero esquecer de mencionar ninguém.

Aos amigos que fiz trabalhando para a Intelbras. Especialmente ao Vitor, com quem aprendi muito todos os dias.

Sou grato ao Instituto Federal de Santa Catarina (IFSC), pelo ensino de qualidade e pela infraestrutura fornecida durante a graduação, e a todos os professores e colaboradores do IFSC, que contribuíram para a minha formação e crescimento pessoal e profissional.

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação de visão computacional, baseada em processamento digital de imagens e redes neurais convolucionais, para o reconhecimento de lances de xadrez a partir de uma sequência de imagens capturadas durante uma partida, lance a lance. A solução foi implementada em Python, utilizando a biblioteca OpenCV e um modelo YOLOv11 treinado na plataforma Roboflow com um conjunto de imagens de partidas de xadrez criado pelo autor. O modelo alcançou uma precisão de 97,3% na identificação das peças no tabuleiro. A aplicação é capaz de reconhecer todos os lances realizados ao longo da partida e gerar, ao final, um relatório detalhado em notação algébrica de xadrez.

Palavras-chave: Visão computacional. Xadrez. Reconhecimento por imagem. Redes neurais convolucionais. YOLO.

ABSTRACT

This work presents the development of a computer vision application based on digital image processing and convolutional neural networks for recognizing chess moves from a sequence of images captured during a match, move by move. The solution was implemented in Python, using the OpenCV library and a YOLOv11 model trained on the Roboflow platform with a dataset of chess game images created by the author. The model achieved a precision of 97.3% in identifying the pieces on the board. The application is capable of recognizing all moves made throughout the game and generating a detailed report in chess standard algebraic notation at the end.

Keywords: Computer vision. Chess. Image recognition. Convolutional neural networks. YOLO.

LISTA DE ILUSTRAÇÕES

Figura 1 – Posição inicial das peças em um jogo de xadrez 14
Figura 2 – Movimentação do bispo 16
Figura 3 – Movimentação da torre 16
Figura 4 – Movimentação da dama 17
Figura 5 – Movimentação do cavalo 17
Figura 6 – Movimentação do peão 18
Figura 7 – Movimentação do rei 18
Figura 8 – Captura *en passant* 19
Figura 9 – Roque curto para as brancas e roque longo para as pretas 20
Figura 10 – Promoção do peão 20
Figura 11 – Notação do tabuleiro 21
Figura 12 – Desambiguação com cavalo 22
Figura 13 – Desambiguação com peças na mesma coluna 22
Figura 14 – Representação de uma imagem digital bidimensional 25
Figura 15 – Representação de uma imagem digital colorida (modelo RGB) 25
Figura 16 – Exemplo de aplicação do detector de bordas de Canny 26
Figura 17 – Operações morfológicas 27
Figura 18 – Transformada de Hough 28
Figura 19 – Ruído sal e pimenta reduzido por filtro de mediana 29
Figura 20 – Exemplo de fecho convexo 29
Figura 21 – Exemplo de transformação de perspectiva 30
Figura 22 – Funcionamento do DBSCAN 31
Figura 23 – Funcionamento de um neurônio artificial 32
Figura 24 – Uma rede neural profunda 33
Figura 25 – Exemplo de convolução 2D 34
Figura 26 – Exemplo de *pooling* 34
Figura 27 – Funcionamento do YOLO 36
Figura 28 – Conjunto de imagens anotadas no Roboflow 37
Figura 29 – Detalhes do conjunto de dados 38
Figura 30 – Visualização do modelo treinado 39
Figura 31 – Configuração para captura de imagens 40
Figura 32 – Exemplo de cálculo da IoU entre duas caixas delimitadoras 41
Figura 33 – Fluxo geral de funcionamento do algoritmo. 45
Figura 34 – Sequência de técnicas usada para seleção do tabuleiro 47
Figura 35 – Resultados intermediários das técnicas para seleção do tabuleiro 48

Figura 36 – Imagem anotada na plataforma Roboflow 49
Figura 37 – Estatísticas de treinamento do modelo YOLOv11 51
Figura 38 – Sequência de técnicas usada para o mapeamento das casas do tabuleiro 52
Figura 39 – Resultados intermediários das técnicas para mapeamento do tabuleiro . 54
Figura 40 – Peças detectadas pelo modelo YOLO 55
Figura 41 – Relatório da partida gerado pelo algoritmo 57

LISTA DE ABREVIATURAS E SIGLAS

AP *Average Precision.*

API *Application Programming Interface.*

CNN *Convolutional Neural Networks.*

DBSCAN *Density-Based Spatial Clustering of Applications with Noise.*

DGT *Digital Game Technology.*

DL *Deep Learning.*

DNN *Deep Neural Networks.*

FEN *Forsyth-Edwards Notation.*

FIDE Federação Internacional de Xadrez.

IA Inteligência Artificial.

IoU *Intersection over Union.*

JSON *JavaScript Object Notation.*

MAP *mean Average Precision.*

ML *Machine Learning.*

OpenCV Open-Source Computer Vision.

SDK *Software Development Kit.*

YOLO *You Only Look Once.*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Objetivos 12
1.1.1 Objetivo geral 12
1.1.2 Objetivos específicos 13
1.2 Justificativa 13

2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 O Xadrez 14
2.1.1 Regras do Xadrez 15
2.1.2 Movimentação das peças 15
2.1.3 Lances especiais 19
2.1.4 Notação Algébrica 21
2.1.5 Notação FEN 23
2.2 Visão Computacional 23
2.2.1 Importância e Aplicações 24
2.2.2 Composição de uma Imagem Digital 24
2.2.3 Detecção de Bordas 26
2.2.4 Operações Morfológicas 26
2.2.5 Transformada de Hough 27
2.2.6 Filtro de Mediana 28
2.2.7 Fecho Convexo 29
2.2.8 Transformação de Perspectiva 29
2.2.9 DBSCAN 30
2.3 *Machine Learning* 31
2.3.1 Tipos de Aprendizado 31
2.3.2 *Deep Learning* 32
2.3.3 Redes Neurais Convolucionais 33
2.3.3.1 Convolução 33
2.3.3.2 *Pooling* 34
2.3.3.3 Camada Totalmente Conectada 35
2.3.4 You Only Look Once (YOLO) 35
2.3.5 Roboflow 36
2.3.5.1 Anotação de Imagens 37
2.3.5.2 Treinamento de Modelos 38

3 METODOLOGIA . . . 40
3.1 Conjunto de imagens . . . 40
3.1.1 Configuração utilizada . . . 40
3.1.2 Anotação de objetos . . . 41
3.2 Métricas . . . 41
3.2.1 Interseção sobre a União (*IoU*) . . . 41
3.2.2 Confiança . . . 41
3.2.3 Precisão . . . 42
3.2.4 Sensibilidade (*Recall*) . . . 42
3.2.5 Precisão Média (AP) . . . 42
3.2.6 Precisão Média Geral (*mAP*) . . . 42
3.2.7 Perda da Caixa Delimitadora (*Box Loss*) . . . 43
3.2.8 Perda de Classificação (*Classification Loss*) . . . 43
3.2.9 Perda de Distribuição Focal (*Focal Loss*) . . . 43
3.3 Ferramentas . . . 43

4 DESENVOLVIMENTO . . . 45
4.1 Fluxo de funcionamento do algoritmo . . . 45
4.2 Conjunto de Imagens . . . 46
4.2.1 Seleção do tabuleiro . . . 46
4.2.2 Anotação das imagens . . . 49
4.2.3 Distribuição de classes . . . 49
4.3 Treinamento do modelo *YOLO* . . . 50
4.4 Algoritmos . . . 52
4.4.1 Mapeamento das casas do tabuleiro . . . 52
4.4.2 Detecção das peças . . . 55
4.4.3 Determinação da posição das peças no tabuleiro . . . 55
4.4.4 Detecção de erros . . . 56
4.4.5 Identificação do lance e geração da notação algébrica . . . 56

5 CONCLUSÕES . . . 58
5.1 Trabalhos Futuros . . . 58

REFERÊNCIAS . . . 60

APÊNDICES 62

APÊNDICE A – LOGS DA APLICAÇÃO NA LINHA DE COMANDO 63

1 INTRODUÇÃO

Nos últimos anos, a Inteligência Artificial (IA) tem se evidenciado cada vez mais com o surgimento de novas tecnologias. A visão computacional, um dos ramos da IA, engloba o processamento de imagens e reconhecimento de padrões e proporcionou diversas inovações no ramo da tecnologia, como veículos autônomos, análise de imagens médicas e aplicações em segurança e monitoramento (FORBES, 2023).

O xadrez é um dos jogos de tabuleiro mais antigos da humanidade. Embora haja controvérsias quanto a sua origem, os primeiros registros do jogo que deu origem ao xadrez, chamado chaturanga, datam do século VI, na Índia. Posteriormente, o jogo se espalhou pelo mundo e sofreu alterações até chegar à Europa, onde se estabeleceram as regras modernas entre os séculos XV e XVI (MURRAY, 2015).

Atualmente, nos tempos da *world wide web*, o xadrez pode ser jogado *online*, permitindo que pessoas de todo o mundo se enfrentem em partidas de diversas modalidades, através de plataformas como a *Chess.com*[1]. Apesar disso, torneios presenciais ainda são muito comuns, tanto de grande porte como o Campeonato Mundial de Xadrez, administrado pela Federação Internacional de Xadrez (FIDE)[2], quanto torneios de pequeno porte organizados por clubes locais.

Os estudos da inteligência artificial voltados ao xadrez são quase tão antigos quanto a própria ciência da computação, iniciando-se na década de 1950, com Alan Turing e Claude Shannon, que criaram os primeiros algoritmos capazes de jogar xadrez. Em 1957, Alex Bernstein, engenheiro da IBM, criou a primeira máquina completamente automatizada capaz de jogar xadrez (HEATH; ALLUM, 1997).

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver uma aplicação capaz de fazer o reconhecimento por imagem de lances em um jogo de xadrez utilizando algoritmos de visão computacional e aprendizado de máquina e os apresentar ao usuário em notação algébrica de xadrez.

Inicialmente, os objetivos incluiam desenvolver um aplicativo para dispositivos móveis. Porém, devido à complexidade do projeto e dificuldades encontradas durante o desenvolvimento, o escopo foi reduzido para uma prova de conceito com uma sequência

[1] https://www.chess.com/
[2] https://www.fide.com/

de imagens.

1.1.2 Objetivos específicos

- Investigar e aplicar técnicas e algoritmos de visão computacional para detecção de objetos;
- Detectar isoladamente o tabuleiro de xadrez e mapear suas casas, reconhecendo individualmente cada peça presente no tabuleiro;
- Reconhecer todos os lances de um jogo de xadrez utilizando a combinação de visão computacional e aprendizado de máquina;
- Entregar ao usuário um registro dos lances da partida em notação algébrica de xadrez.

1.2 Justificativa

Existem poucas alternativas acessíveis para determinar a posição das peças em tabuleiros físicos de xadrez ou registrar partidas de forma automática. Atualmente, isso é feito manualmente ou utilizando tabuleiros eletrônicos, como o da marca *Digital Game Technology* (DGT). Este último é usado em torneios de xadrez, porém seu preço ainda é demasiadamente elevado. A marca DGT, por exemplo, possui apenas uma loja representante oficial no Brasil[3], cujo tabuleiro eletrônico mais barato custa quase R$4.000,00. Desta forma, o presente trabalho visa proporcionar uma alternativa mais acessível para o registro automático de partidas de xadrez.

[3] https://www.pontodoxadrez.com.br/

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O Xadrez

O xadrez é um jogo em que dois oponentes movem suas peças em um tabuleiro de 8 fileiras e 8 colunas, com o objetivo de pôr o rei oponente em uma posição que não exista qualquer movimento legal a ser feito, a posição de xeque-mate (FIDE, 2022).

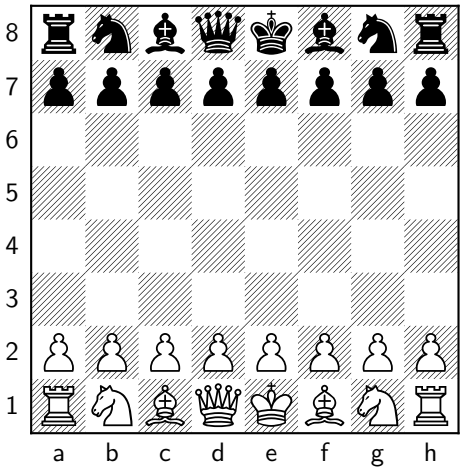
Quadro 1 – Peças do xadrez

Nome (*inglês*)	**Letra Representativa**	**Símbolo Representativo**
Rei (*King*)	K	♔ ♚
Dama (*Queen*)	Q	♕ ♛
Bispo (*Bishop*)	B	♗ ♝
Cavalo (*Knight*)	N	♘ ♞
Torre (*Rook*)	R	♖ ♜
Peão (*Pawn*)	P	♙ ♟

Fonte: Elaborada pelo autor.

Inicialmente, 32 peças são dispostas no tabuleiro, sendo 1 rei, 1 dama, 2 bispos, 2 cavalos, 2 torres e 8 peões de cada cor, cujos símbolos podem ser observados no Quadro 1. A posição inicial das peças no tabuleiro pode ser observada na Figura 1. As casas do tabuleiro são designadas por letras nas colunas e número nas fileiras, assim cada casa pode ser designada por um par letra - número.

Figura 1 – Posição inicial das peças em um jogo de xadrez



Fonte: Elaborada pelo autor.

2.1.1 Regras do Xadrez

As regras oficiais do xadrez são definidas pela FIDE e conhecidas como as Leis do Xadrez, descritas em um documento disponível publicamente no site oficial da federação (FIDE, 2022).

O jogador que controla as peças brancas sempre joga primeiro, depois é a vez daquele que controla as peças pretas. Eles seguem alternando a vez até o fim do jogo. Um jogador tem a vez apenas quando o oponente tiver finalizado seu lance. O jogador cujo rei sofrer um xeque-mate terá perdido a partida.

Se ocorrer uma situação em que nenhum dos dois jogadores possam realizar um xeque-mate, chamada de "posição morta", o jogo terá empatado. Também será considerado empate se o jogador que tiver a vez não puder realizar qualquer movimento legal e seu rei não estiver em xeque, neste caso diz-se que a partida terminou com o rei "afogado". A partida também pode terminar empatada mediante comum acordo entre os dois jogadores, contanto que ambos tenham feito ao menos o primeiro lance.

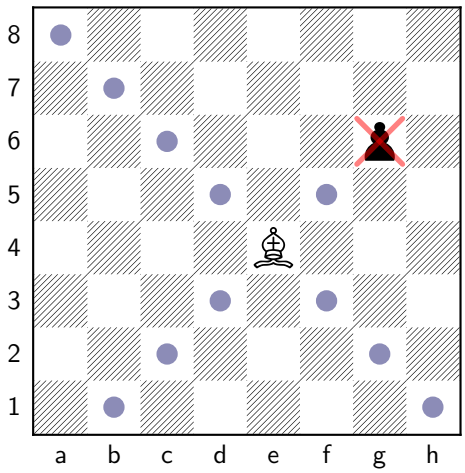
2.1.2 Movimentação das peças

Cada peça do xadrez segue diferentes regras de movimentação no tabuleiro, com suas peculiaridades em relação à direção e quantas casas podem andar. Não é permitido mover uma peça para uma casa em que há uma peça de mesma cor. Se uma peça move-se para uma casa em que há uma peça da outra cor, é realizada uma captura (FIDE, 2022).

Bispo

O bispo move-se apenas na diagonal, quantas casas forem possíveis, e captura peças oponentes seguindo a mesma regra, conforme demonstrado na Figura 2.

Figura 2 – Movimentação do bispo

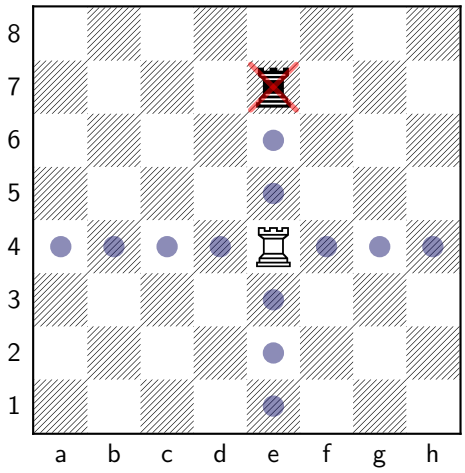


Fonte: Elaborada pelo autor.

Torre

A torre move-se e captura peças apenas nas direções vertical e horizontal, em qualquer sentido, conforme mostra a Figura 3. Se cumpridos os requisitos, a torre pode ser movimentada em um lance especial em conjunto com o rei, chamado roque.

Figura 3 – Movimentação da torre

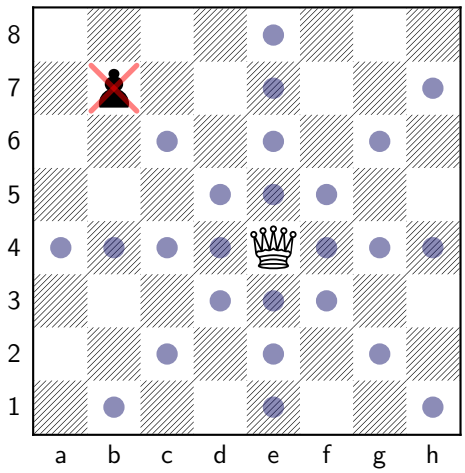


Fonte: Elaborada pelo autor.

Dama

A dama, sendo uma das peças mais poderosas do xadrez, pode ser movida e capturar peças nas direções vertical, horizontal e diagonal, em qualquer sentido, conforme Figura 4.

Figura 4 – Movimentação da dama

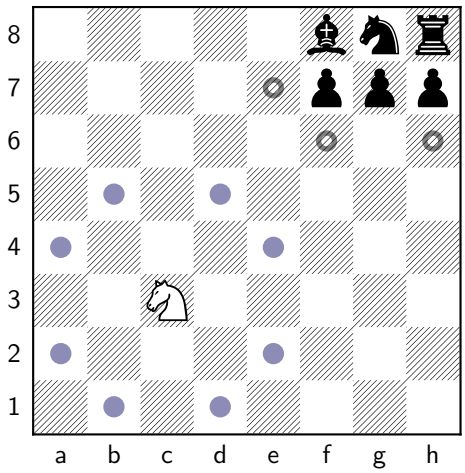


Fonte: Elaborada pelo autor.

Cavalo

O cavalo se move em um padrão de 'L', ou seja, pode mover-se duas casas em uma direção (horizontal ou vertical) e mais uma casa na direção perpendicular, conforme demonstrado na Figura 5. Além disso, ele é a única peça que pode pular sobre outras peças no tabuleiro, o que significa que ele pode se mover para sua casa de destino mesmo que existam outras peças no caminho.

Figura 5 – Movimentação do cavalo



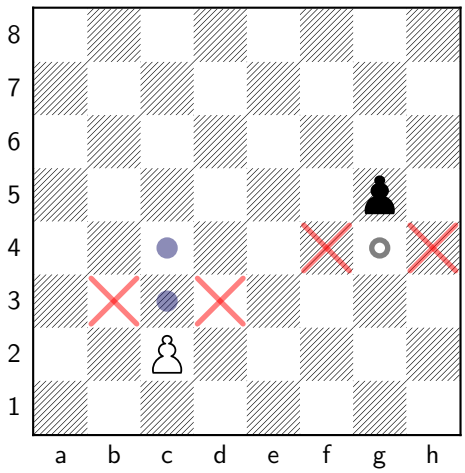
Fonte: Elaborada pelo autor.

Peão

Os peões das brancas movem-se da fileira 2 em direção a fileira 8, enquanto os peões das pretas fazem o movimento inverso, da fileira 7 em direção a fileira 1. O peão

pode andar duas casas apenas na primeira vez em que é movido. Após isso, pode andar apenas uma casa e pode realizar a captura de uma peça apenas nas casas imediatamente à sua diagonal, no sentido em que se move, conforme exemplificado na Figura 6. Sob circunstâncias específicas, pode ser promovido a outra peça ou realizar uma captura *en passant*.

Figura 6 – Movimentação do peão

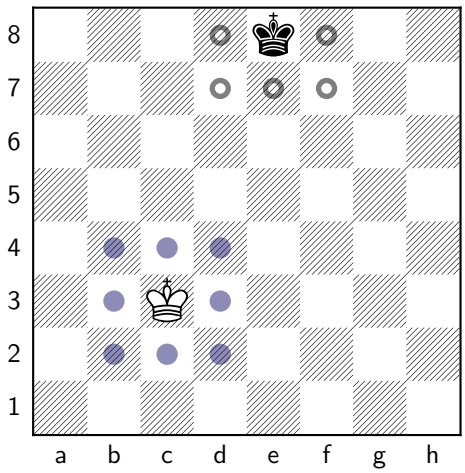


Fonte: Elaborada pelo autor.

Rei

O rei, assim como a dama, movimenta-se e captura peças oponentes em todas as direções. No entanto, ele pode andar apenas 1 casa, conforme demonstra a Figura 7, e não pode colocar-se em xeque.

Figura 7 – Movimentação do rei



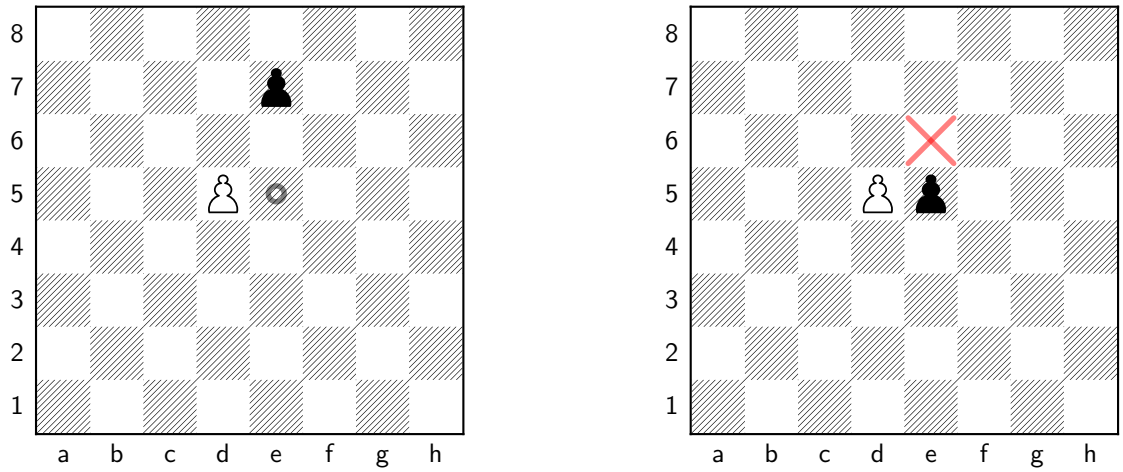
Fonte: Elaborada pelo autor.

2.1.3 Lances especiais

En passant

Um peão que estiver na mesma fileira e em uma coluna adjacente a um peão oponente que acabou de andar 2 casas, pode capturar o peão oponente como se este tivesse andado apenas 1 casa, conforme demonstrado na Figura 8. A captura *en passant* deve ser realizada imediatamente após o peão oponente mover-se duas casas.

Figura 8 – Captura *en passant*



(a) Peão anda duas casas

(b) Possibilidade de captura

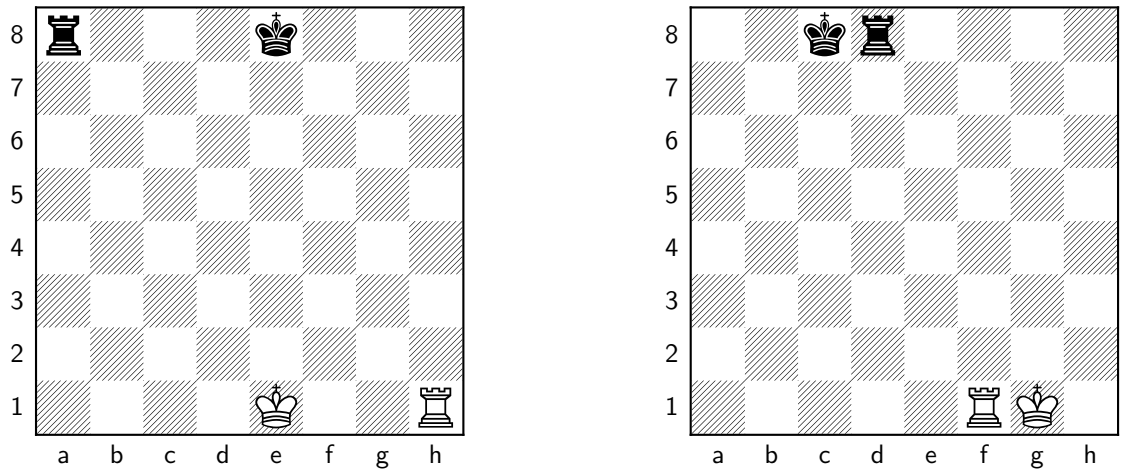
Fonte: Elaborada pelo autor.

Roque

O roque é um lance especial que move duas peças (o rei e uma torre) e conta como um único lance. Como requisitos, ambos o rei e a torre não podem ter sido movidos durante a partida e todas as casas entre as duas peças devem estar desocupadas. Além disso, o rei não pode passar por ou terminar em uma casa sob ataque.

Existem duas variações, o roque longo, realizado com a torre mais distante do rei, e o roque curto, realizado com a torre mais próxima. No roque, o rei anda duas casas em direção à torre, depois a torre move-se para a casa por qual o rei passou. A Figura 9a mostra uma situação anterior à realização do roque, sendo o roque curto para as peças brancas e o roque longo para as peças pretas, enquanto a Figura 9b mostra a posição das peças após o roque.

Figura 9 – Roque curto para as brancas e roque longo para as pretas



(a) Antes do roque

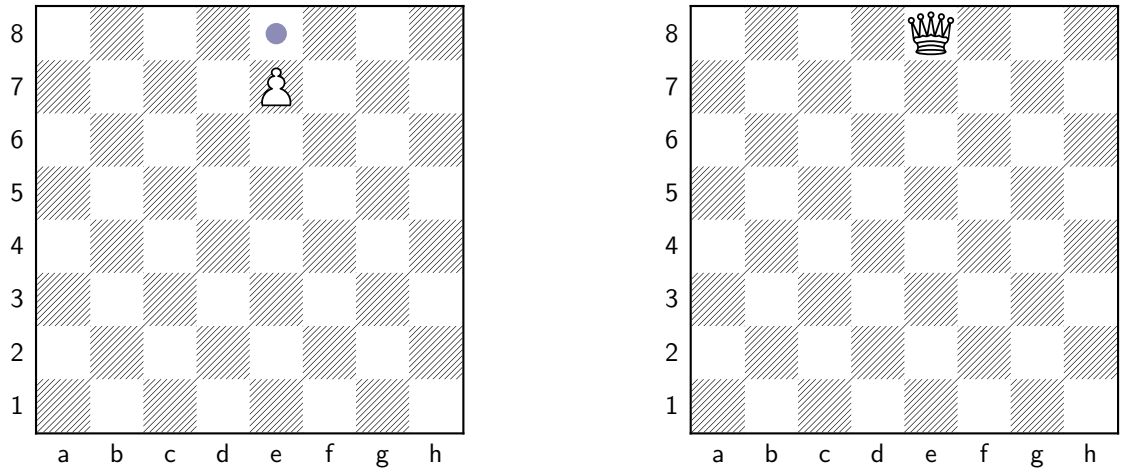
(b) Depois do roque

Fonte: Elaborada pelo autor.

Promoção do peão

Quando um peão chega à fileira mais distante de sua casa inicial (fileira 8 para as brancas e 1 para as pretas), ocorre a promoção, em que o peão deve ser substituído, como parte do movimento, por uma nova dama, torre, bispo ou cavalo de mesma cor, conforme demonstrado na Figura 10. A escolha do jogador não é restrita a peças capturadas anteriormente (FIDE, 2022).

Figura 10 – Promoção do peão



(a) Peão alcança a fileira mais distante

(b) Peão é promovido a uma Dama

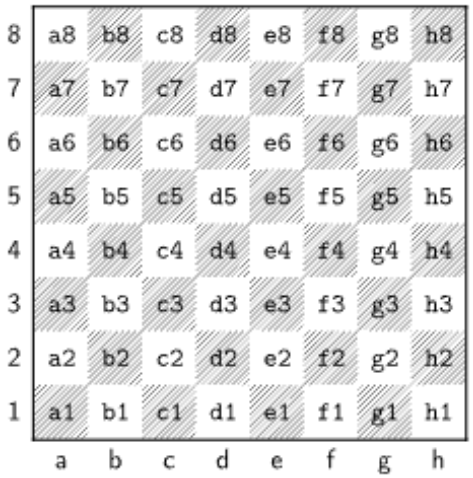
Fonte: Elaborada pelo autor.

2.1.4 Notação Algébrica

No xadrez, existem padrões a serem seguidos ao registrar jogadas. A FIDE reconhece em seus torneios e partidas apenas um sistema de notação, chamado Sistema Algébrico (FIDE, 2022).

Neste sistema, cada casa do tabuleiro é mapeada por sua coluna e sua fileira, como demonstrado na Figura 11. A casa 'a1', por exemplo, refere-se à casa na primeira fileira da coluna 'a', enquanto 'e7' é a casa na sétima fileira da coluna 'e'.

Figura 11 – Notação do tabuleiro



Fonte: (OVERLEAF, 2024)

Conforme o Quadro 1, cada peça do xadrez é representada por uma letra, sempre maiúscula, que é utilizada na notação de lances. O registro do movimento de uma peça é composto pela peça movida, seguida pela casa de destino. Para o peão, não é utilizada a letra representativa. Por exemplo, se a Dama for movida de 'd1' para 'g4', a notação será 'Qg4' (Dama se move para 'g4'), enquanto um peão movido de 'e2' para 'e4' será anotado apenas como 'e4' (Peão se move para 'e4').

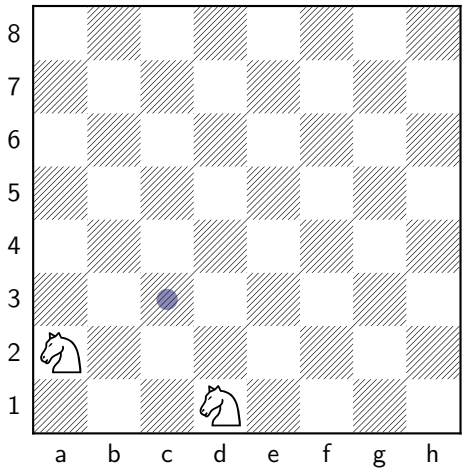
Para jogadas com captura, deve ser adicionada a letra 'x' entre a peça movida e a casa de destino. Por exemplo, se uma torre captura uma peça em 'h5', a notação será 'Rxh5'. Quando um peão faz uma captura, utiliza-se a coluna da qual o peão partiu, seguida pela casa de destino, por exemplo 'exd5' (Peão na coluna 'e' captura a peça em 'd5'). Para capturas *en passant*, utiliza-se a coluna de partida do peão, seguido pelo 'x', depois a casa para qual o peão se moveu (não a do peão capturado), seguido pela notação 'e.p', porém esta última não é obrigatória.

Em alguns casos, é possível que duas peças iguais e de mesma cor possam mover-se para a mesma casa. Nestes casos, a inicial da peça é seguida pela coluna de partida, se estiverem na mesma fileira, ou pela fileira de partida, se estiverem na mesma coluna.

Quando as duas peças estiverem em fileiras e colunas diferentes, é preferível utilizar a coluna de partida (FIDE, 2022).

Conforme exemplificado na Figura 12, quando dois cavalos podem mover-se para a casa ‘c3’, a notação ‘Nc3’ não é o suficiente para descrever a jogada, pois seria perdida a informação sobre qual cavalo moveu-se para a casa de destino. Nesta situação, deve-se indicar a coluna de partida da peça com ‘Nac3’ ou ‘Ndc3’.

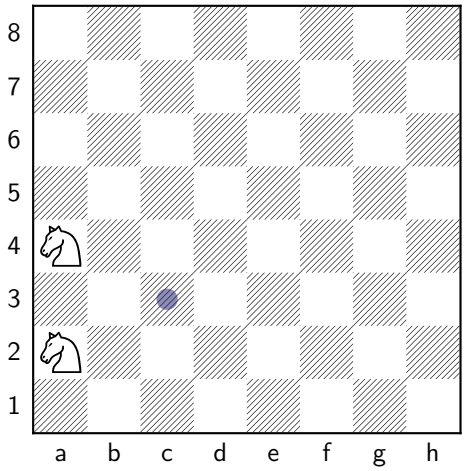
Figura 12 – Desambiguação com cavalo



Fonte: Elaborada pelo autor.

No entanto, existem casos ainda mais críticos para desambiguação, como quando dois cavalos estão na mesma coluna e podem mover-se para a mesma casa, conforme demonstrado na Figura 13. Nestes casos, deve-se utilizar a fileira de partida da peça, resultando na notação ‘N2c3’ ou ‘N4c3’.

Figura 13 – Desambiguação com peças na mesma coluna



Fonte: Elaborada pelo autor.

Quando um peão atinge a última fileira e ocorre a sua promoção, a peça escolhida é indicada após a anotação da jogada, por exemplo ‘d8Q’, quando o peão branco atinge a fileira 8 e é promovido a uma dama, ou ‘exf8N’, quando um peão branco atinge a fileira 8 capturando uma peça e é promovido a um cavalo. Além disso, o roque curto é indicado pela notação especial ‘0-0’, enquanto o roque longo é registrado como ‘0-0-0’ (FIDE, 2022).

Uma jogada que coloca o rei oponente em xeque deve ter a notação ‘+’ adicionada ao final, enquanto o xeque-mate deve ser indicado como ‘#’ ou ‘++’, embora o primeiro seja mais indicado. A oferta de um empate deve ser anotada como ‘(=)’.

Exemplo do uso da notação algébrica para registrar o xeque-mate do pastor: 1 - e4 e5 2 - Bc4 Nc6 3 - Qh5 Nf6 4 - Qxf7#.

2.1.5 Notação FEN

A notação *Forsyth-Edwards Notation* (FEN) é uma notação que descreve as posições de um jogo de xadrez. Sua principal vantagem é representar qualquer posição do tabuleiro em uma única linha de texto, facilitando a sua leitura e armazenamento. Ela também é fácil de interpretar, permitindo que jogadores possam rapidamente entender a posição de um jogo a partir de uma sequência de caracteres (CHESS.COM, 2020).

Esta notação é composta exclusivamente por letras e números e possui seis campos separados por espaços, que representam, respectivamente, a disposição das peças, o jogador a jogar, os direitos de roque, a captura *en passant*, o contador de meia-jogadas, usado para a regra dos 50 lances, e o contador de jogadas completas. Letras minúsculas representam peças pretas, enquanto letras maiúsculas representam peças brancas. As peças são representadas pelas suas iniciais em inglês, conforme o Quadro 1. Os números representam a quantidade de casas vazias e a barra (/) separa as fileiras do tabuleiro.

O tabuleiro da Figura 1, com a posição inicial das peças, por exemplo, é representado pela seguinte notação FEN:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

A letra ‘w’ (de *white*) indica que as brancas são as próximas a jogar, enquanto ‘KQkq’ indica que ambos os jogadores podem realizar tanto o roque do rei quanto o roque da dama. A captura *en passant* não é possível, indicada pelo traço (-). O contador de meia-jogadas e o contador de jogadas completas são 0 e 1, respectivamente.

2.2 Visão Computacional

A visão computacional é uma área de estudo cujo objetivo é permitir que máquinas façam decisões úteis sobre objetos físicos e cenas reais com base em imagens capturadas. Segundo Shapiro e Stockman (2001), para tomar essas decisões, é quase sempre necessário

construir uma descrição ou modelo dos objetos a partir da imagem, o que pode ser definido como a construção de descrições de cenas a partir de imagens.

Complementando essa definição, Szeliski (2022) destaca que a visão computacional envolve a recuperação da forma tridimensional e a aparência de objetos em imagens, tratando-se de um "problema inverso". Isso significa que se busca recuperar informações desconhecidas a partir de dados insuficientes, recorrendo a modelos físicos, probabilísticos ou ao aprendizado de máquina a partir de grandes conjuntos de exemplos.

Pode-se entender, então, que a visão computacional é um campo que combina técnicas matemáticas e computacionais para interpretar e entender imagens digitais, reconstruindo propriedades como forma, iluminação e distribuição de cores dos objetos presentes nessas imagens, visando replicar a capacidade humana de percepção visual.

2.2.1 Importância e Aplicações

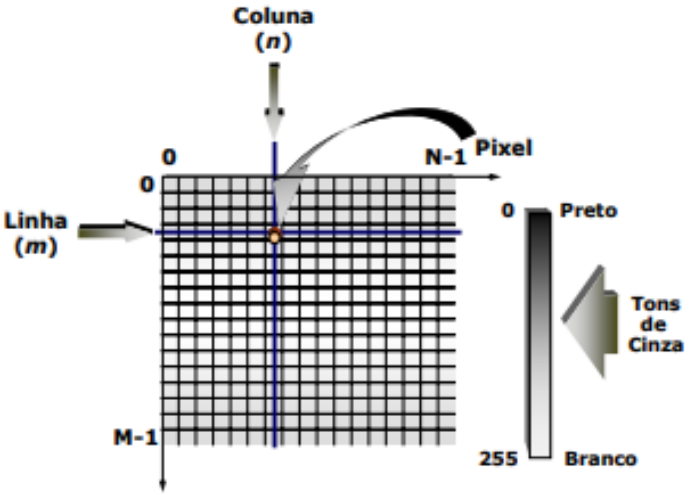
A visão computacional é essencial em várias áreas, pois permite automatizar e melhorar a análise de imagens e vídeos. Esta tecnologia é utilizada na indústria para a inspeção de máquinas, onde a verificação rápida e precisa de defeitos garante a qualidade e a segurança dos produtos. Em armazéns, a visão computacional automatiza tarefas de embalagem e transporte, aumentando a eficiência e reduzindo custos. Na área médica, é usada para a análise de imagens pré-operatórias e intra-operatórias, além de monitorar a morfologia cerebral dos pacientes ao longo do tempo (SZELISKI, 2022).

Entre as aplicações mais inovadoras da visão computacional estão os veículos autônomos, que utilizam essa tecnologia para navegar e tomar decisões em tempo real, transformando a forma de deslocamento e aumentando a segurança nas estradas. A modelagem 3D fotogramétrica permite a construção automática de modelos 3D detalhados a partir de fotografias. No entretenimento, a visão computacional é usada na captura de movimento e efeitos visuais, criando experiências cinematográficas imersivas. Além disso, a detecção facial para autenticação visual e a vigilância inteligente são exemplos de como a visão computacional está melhorando o dia a dia, trazendo avanços em segurança e conveniência (SZELISKI, 2022).

2.2.2 Composição de uma Imagem Digital

Uma imagem é representada digitalmente como uma matriz bidimensional $f(m, n)$ e possui sua origem em $(0, 0)$, no topo à esquerda. Cada elemento desta matriz é chamado *pixel* e, em uma imagem em tons de cinza, pode assumir um valor de 0 a 255, que representa a intensidade luminosa naquele ponto, conforme a Figura 14 (QUEIROZ; GOMES, 2006).

Figura 14 – Representação de uma imagem digital bidimensional

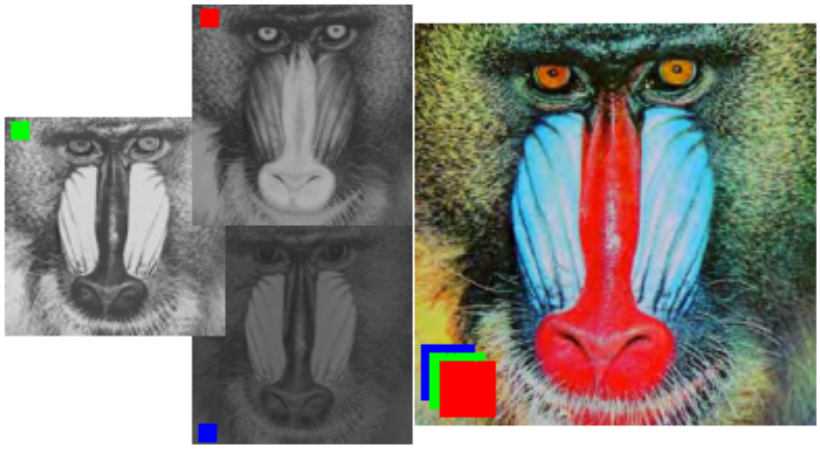


Fonte: (QUEIROZ; GOMES, 2006)

Segundo Gonzalez e Woods (2018), uma imagem digital colorida pode utilizar diferentes modelos de cor, como o RGB (vermelho, verde, azul), CMY (ciano, magenta, amarelo), CMYK (ciano, magenta, amarelo, preto) ou HSI (matiz, saturação, intensidade). Neste trabalho, será abordado apenas o modelo aditivo de cores, o RGB.

Em um sistema RGB, a imagem colorida é formada pela composição de três imagens monocromáticas em que cada *pixel* representa a intensidade de sua cor. A Figura 15 apresenta os planos monocromáticos de uma imagem e o resultado da composição dos três planos (QUEIROZ; GOMES, 2006).

Figura 15 – Representação de uma imagem digital colorida (modelo RGB)



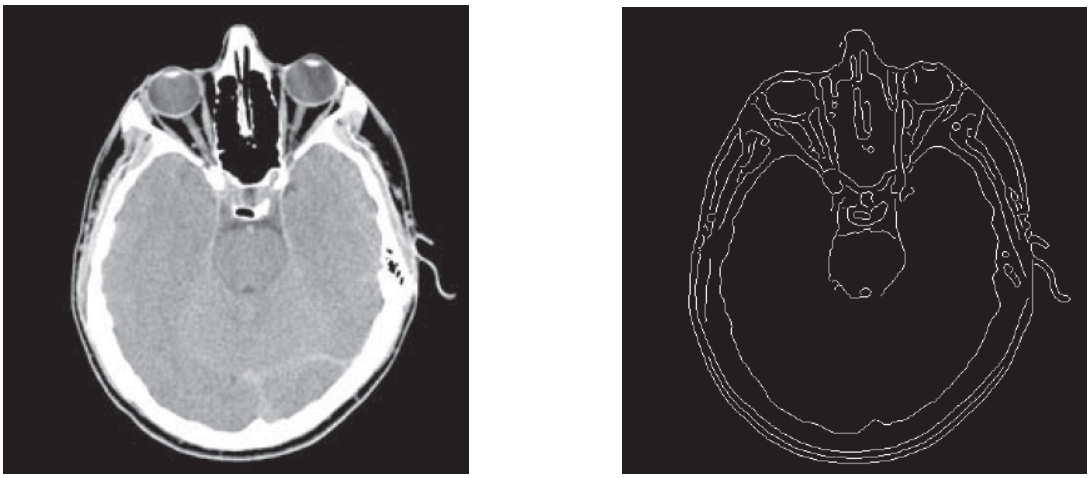
Fonte: (QUEIROZ; GOMES, 2006)

2.2.3 Detecção de Bordas

A detecção de bordas é uma técnica essencial no processamento de imagens, utilizada para identificar os contornos dos objetos presentes na imagem. Segundo Szeliski (2022), bordas são locais onde há uma mudança brusca na intensidade de cor ou luz. O método de detecção de bordas de Canny (CANNY, 1986) é um algoritmo popular devido a sua alta precisão e baixa taxa de erro, conforme observado na Figura 16. Segundo Gonzalez e Woods (2018), este detector segue uma série de etapas:

- **Redução de ruído**: Aplica um filtro Gaussiano para reduzir ruídos e suavizar a imagem.
- **Cálculo do gradiente**: Encontra a direção e magnitude do gradiente de intensidade da imagem.
- **Supressão de não-máximos**: Procura por valores máximos locais e realiza a supressão de não-máximos para remover *pixels* que não fazem parte das bordas, resultando em linhas mais finas.
- **Histerese com dois limiares**: Utiliza um limiar duplo para determinar se os *pixels* restantes são parte de uma borda.

Figura 16 – Exemplo de aplicação do detector de bordas de Canny



(a) Imagem original (b) Resultado do detector

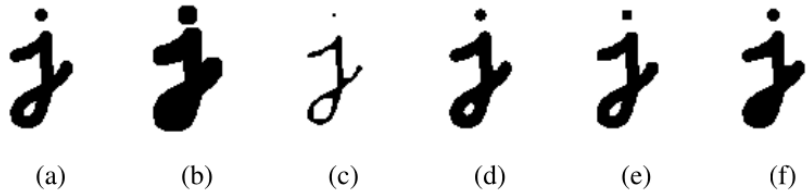
Fonte: (GONZALEZ; WOODS, 2018)

2.2.4 Operações Morfológicas

A forma mais comum de operações sobre imagens binárias são as chamadas operações morfológicas, pois elas mudam a forma dos objetos presentes na imagem, conforme visto na Figura 17. Estas operações são realizadas convolvendo a imagem binária com

um elemento estruturante e então selecionando o valor binário de saída a partir do resultado da convolução e comparando-o com um limiar. O elemento estruturante pode ter diferentes formas, de uma simples matriz 3×3 até formas mais complexas (SZELISKI, 2022). Segundo Gonzalez e Woods (2018), as operações morfológicas mais comuns são a dilatação e a erosão, que são a base para muitas outras, como a abertura e o fechamento.

Figura 17 – Operações morfológicas



Morfologia de uma imagem binária: (a) imagem original; (b) dilatação; (c) erosão; (d) maioria; (e) abertura; (f) fechamento. O elemento estruturante é uma matriz 5×5

Fonte: (SZELISKI, 2022)

A erosão é uma operação que remove *pixels* de borda de um objeto, diminuindo seu tamanho. Sua definição formal é dada pela Equação 2.1, em que A é uma imagem binária ou um conjunto de *pixels*, B é o elemento estruturante, z são posições em A na qual o elemento B é posicionado e B_z é o elemento B transladado para a posição z. Ou seja, a erosão de A por B é o conjunto de todos os *pixels* z em que o elemento B se encaixa completamente em A.

$$A \ominus B = \{z \mid B_z \subseteq A\} \tag{2.1}$$

A dilatação, por outro lado, expande um objeto, preenche buracos e conecta objetos próximos. A dilatação de A por B é dada pela Equação 2.2 e é definida como o conjunto de todos os *pixels* z em que ao menos uma parte de B e A se sobrepõem.

$$A \oplus B = \{z \mid B_z \cap A \neq \emptyset\} \tag{2.2}$$

2.2.5 Transformada de Hough

A Transformada de Hough, nomeada em homenagem ao seu criador (HOUGH, 1962), é uma técnica utilizada no processamento de imagens para detectar linhas, formas geométricas e outros elementos em uma cena. Ela é especialmente útil em situações em que os contornos de objetos estão fragmentados ou compostos por múltiplos segmentos colineares. Seu objetivo principal é agrupar esses segmentos em linhas contínuas.

Segundo Szeliski (2022), na Transformada de Hough, em sua formulação original, cada ponto de uma borda "vota" em todas as possíveis linhas que passam por ele. Para os

cálculos desta técnica, as linhas são expressas no sistema de coordenadas polar, onde uma linha é representada por sua distância r do ponto de origem e seu ângulo θ em relação ao eixo horizontal, conforme a Equação 2.3.

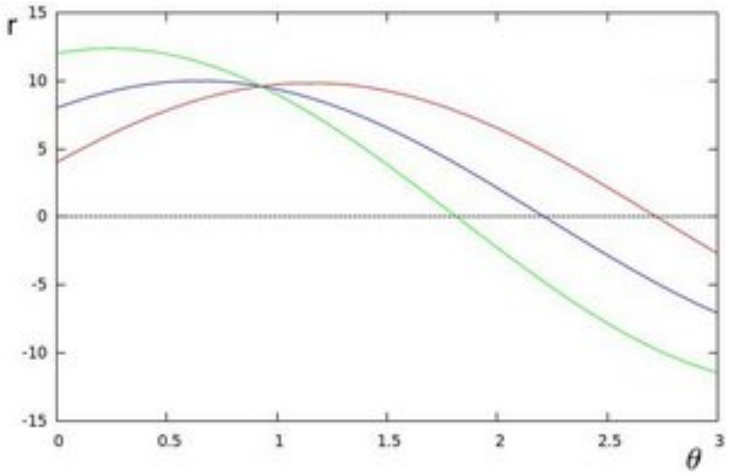
$$r = x\cos(\theta) + y\sin(\theta) \tag{2.3}$$

Para cada ponto, pode-se definir a família de retas que passa por ele como:

$$r_i = x_i\cos(\theta) + y_i\sin(\theta) \tag{2.4}$$

De modo que cada par (r_i, θ) representa uma reta que passa pelo ponto (x_i, y_i). Para cada ponto (x_i, y_i), a Equação 2.4 define uma curva senoidal no espaço (r, θ). A interseção das curvas de dois pontos distintos, como na Figura 18, indica que eles pertencem à mesma linha.

Figura 18 – Transformada de Hough



Fonte: (OPENCV, 2025)

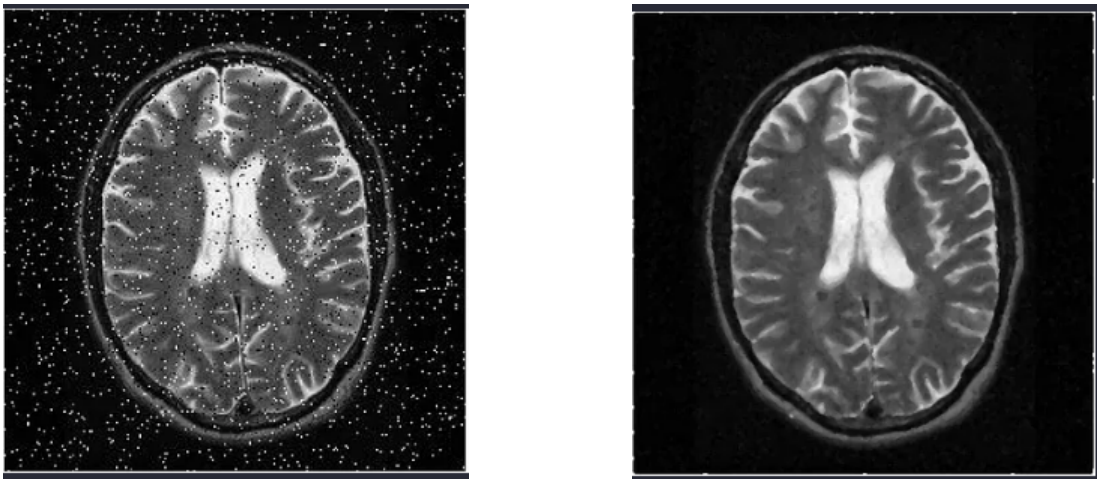
Vários parâmetros podem ser usados nesse processo de "votação", como a quantidade mínima de interseções ou o seu comprimento mínimo para ser considerado uma linha válida, e o espaçamento máximo entre dois segmentos colineares para que sejam considerados parte da mesma linha.

2.2.6 Filtro de Mediana

O filtro de mediana é um filtro não linear utilizado para suavizar imagens, remover ruídos e preservar bordas. Ele substitui o valor de um *pixel* pela mediana dos valores de intensidade na vizinhança do *pixel*. O tamanho da vizinhança é definido por uma janela de tamanho $M \times M$, onde M é um número ímpar. O filtro de mediana é eficaz na

remoção de ruídos impulsivos, como o ruído sal e pimenta, conforme visto na Figura 19 (GONZALEZ; WOODS, 2018).

Figura 19 – Ruído sal e pimenta reduzido por filtro de mediana



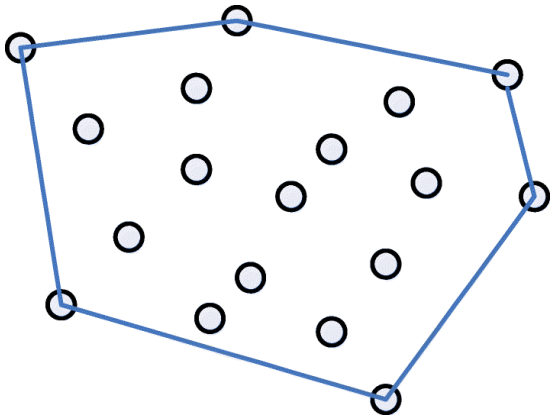
(a) Imagem original (b) Filtro de mediana

Fonte: (FLORES, 2019)

2.2.7 Fecho Convexo

O fecho convexo, ou envoltória convexa, é uma técnica utilizada para envolver um conjunto de pontos em um polígono convexo. Este polígono é formado pela menor área convexa possível que contém todos os pontos do conjunto, como mostrado na Figura 20.

Figura 20 – Exemplo de fecho convexo



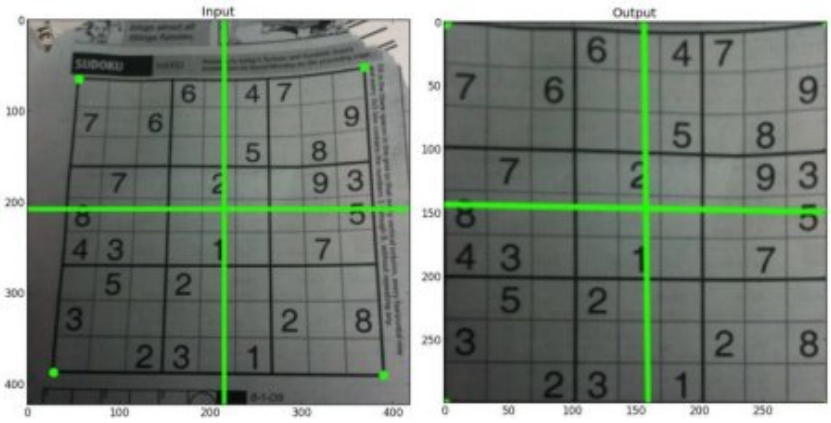
Fonte: (LIU; TANG; CHAN, 2018)

2.2.8 Transformação de Perspectiva

A transformação de perspectiva é uma técnica usada para corrigir o ângulo ou mudar o ponto de vista de uma imagem. Essa técnica é utilizada, por exemplo, em

aplicativos de escaneamento de documentos, onde as fotos podem ser tiradas em ângulos variados.

Figura 21 – Exemplo de transformação de perspectiva



Fonte: (OPENCV, 2024)

Conforme exemplificado na Figura 21, o processo envolve a definição de quatro pontos na imagem original que correspondem aos cantos da superfície ou objeto a ser ajustado. Esses pontos são então mapeados para uma nova posição em um plano frontal. A transformação calcula a relação entre esses pontos e ajusta todos os outros *pixels* da imagem de acordo. Essa operação é realizada utilizando uma matriz de transformação que aplica uma série de cálculos para reposicionar cada *pixel* da imagem.

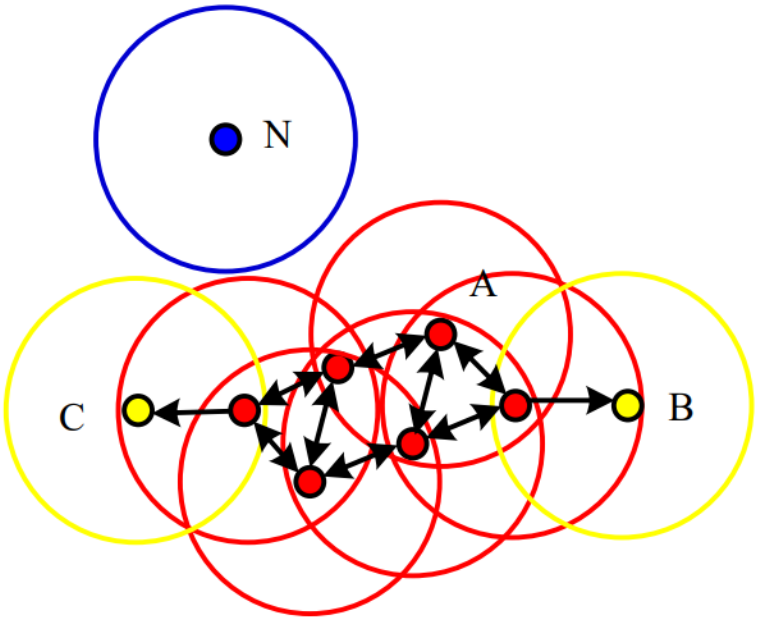
2.2.9 DBSCAN

O *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) (ESTER et al., 1996) é um algoritmo usado para agrupamento espacial de dados em *clusters*, identificando regiões de alta densidade. Ele é capaz de separar *clusters* de diferentes formas e tamanhos, além de identificar e remover ruídos. Ele utiliza dois parâmetros, ϵ (epsilon) e *MinPts* (número mínimo de pontos), para definir um *cluster*.

O algoritmo começa com um ponto aleatório e verifica se há pontos vizinhos a uma distância ϵ. Se o número de pontos for maior que *MinPts*, ele é considerado um *cluster*. Caso contrário, o ponto é marcado como ruído. Todos os pontos são verificados, encontrando outros *clusters* e eliminando pontos que não façam parte de nenhum.

A Figura 22 exemplifica o funcionamento do DBSCAN. O ponto azul é um ruído, os vermelhos são pontos centrais e os amarelos são pontos de borda. O algoritmo é capaz de identificar *clusters* de diferentes formas e tamanhos, além de remover ruídos.

Figura 22 – Funcionamento do DBSCAN



Fonte: (ZHANG; SHEN; OUYANG, 2022)

2.3 *Machine Learning*

O *Machine Learning* (ML), ou aprendizado de máquina, é uma área da inteligência artificial focada em desenvolver algoritmos capazes de aprender e tomar decisões a partir de dados. Um algoritmo de ML é capaz de melhorar seu desempenho em uma tarefa específica à medida que é exposto a mais dados relacionados a essa tarefa (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.1 Tipos de Aprendizado

Segundo Szeliski (2022), existem dois tipos principais de aprendizado de máquina, sendo eles o aprendizado supervisionado e o não supervisionado.

No aprendizado supervisionado, o algoritmo é treinado com pares de entradas e saídas corretas. O objetivo é fazer com que o modelo preveja corretamente as saídas para novas entradas. As principais tarefas são classificação (prever categorias) e regressão (prever valores numéricos).

Por outro lado, no aprendizado não supervisionado, o algoritmo recebe apenas dados de entrada, sem saídas rotuladas. O objetivo é encontrar padrões ou agrupamentos nos dados. Exemplos incluem *clustering* (agrupamento de dados similares) e análise de componentes principais (PCA) para reduzir a dimensionalidade dos dados.

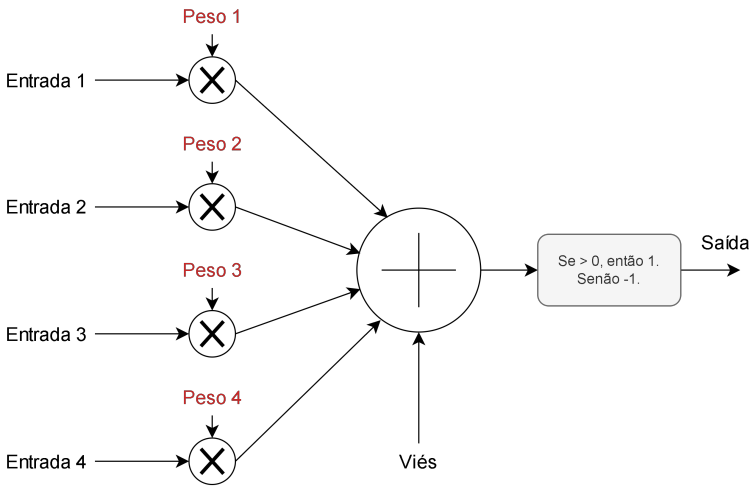
2.3.2 *Deep Learning*

O *Deep Learning* (DL) é um subcampo de *Machine Learning* que utiliza redes neurais profundas, ou *Deep Neural Networks* (DNN), para aprender a partir de dados brutos. Segundo Glassner (2021), os algoritmos de DL, dado treinamento suficiente, são capazes de descobrir automaticamente as regras de tomada de decisão, examinando os dados de entrada e extraindo padrões.

Segundo Glassner (2021), neurônios são a unidade básica de uma rede neural artificial, inspirados nos neurônios reais do sistema nervoso humano, mas de forma muito mais simplificada. Eles recebem dados de entrada, processam essas informações e produzem uma saída.

Em um neurônio moderno, cada entrada pode assumir valores numéricos de ponto flutuante. Esses valores são multiplicados por um peso, que determina a importância de cada entrada. Depois, os resultados das multiplicações são somados. Deste valor, é então subtraído o viés, um valor que não vem da saída de outro neurônio. Se o resultado for maior que zero, a saída será 1, senão será -1 (ou 1 e 0), conforme exemplo da Figura 23 (GLASSNER, 2021).

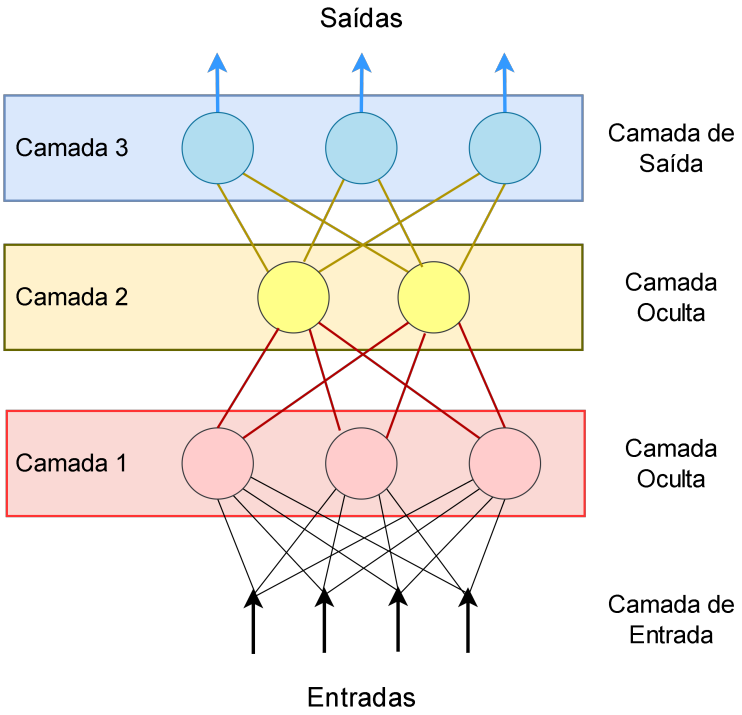
Figura 23 – Funcionamento de um neurônio artificial



Fonte: Adaptado de (GLASSNER, 2021)

Redes neurais artificiais são, como o nome diz, redes formadas pela conexão de neurônios artificiais. Quando estruturadas em camadas, cujos neurônios em uma mesma camada não se comunicam, conforme visualizado na Figura 24, obtém-se uma rede neural profunda, ou DNN. Essas redes neurais profundas são capazes de analisar dados de forma hierárquica, em que cada camada utiliza informações vindas de neurônios da camada anterior para processar maiores conjuntos de dados (GLASSNER, 2021).

Figura 24 – Uma rede neural profunda



Fonte: Adaptado de (GLASSNER, 2021)

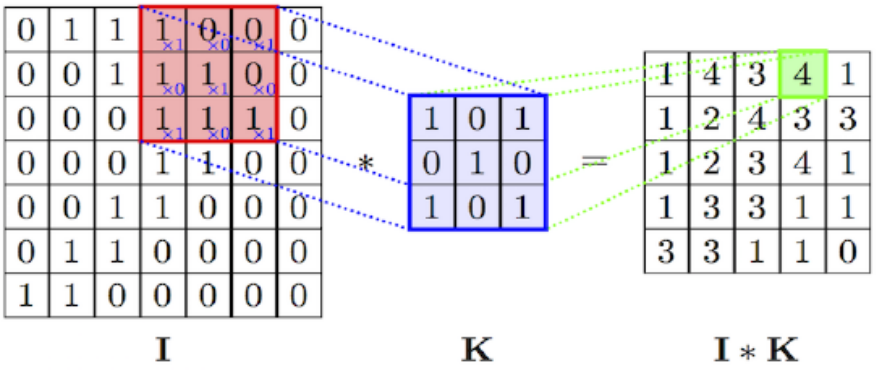
2.3.3 Redes Neurais Convolucionais

Redes neurais convolucionais, ou *Convolutional Neural Networks* (CNN), são um tipo especial de rede neural artificial que utiliza a operação matemática chamada convolução em ao menos uma de suas camadas (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.3.1 Convolução

Na visão computacional, a convolução 2D é a mais utilizada. Nessa operação, a matriz de entrada (a imagem) é sobreposta por um filtro chamado *kernel* (uma matriz menor). As duas matrizes são então multiplicadas pixel por pixel, gerando novos valores que são somados para produzir um único valor. O filtro é "deslizado" (*striding*) sobre a imagem, repetindo a operação até cobrir toda a área. Cada valor resultante dessa operação compõe uma nova matriz chamada mapa de atributos (*feature map*), conforme visualizado na Figura 25.

Figura 25 – Exemplo de convolução 2D



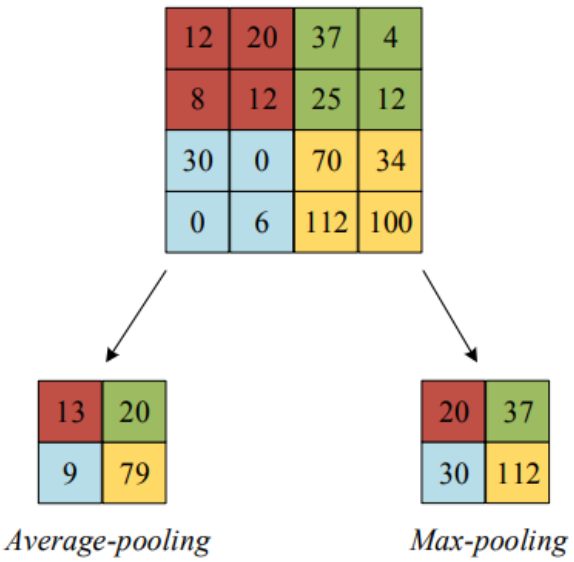
Fonte: (MOHAMED, 2017)

As camadas de convolução possuem como principal função extrair *features*, como bordas, texturas e padrões, mantendo a relação espacial entre pixels e reduzindo o tamanho da entrada de forma controlada, mantendo as informações mais relevantes.

2.3.3.2 *Pooling*

Outro conceito fundamental das CNNs é o *pooling*, uma forma não linear de subamostragem. Entre as diversas funções de *pooling*, as principais são a *max-pooling* e *average-pooling*. Nesta camada, aplica-se um outro filtro, de forma que não haja sobreposições, conforme Figura 26. Este filtro escolherá o maior valor (*max-pooling*), ou calculará a média dos valores (*average-pooling*), gerando como resultado uma matriz de menor dimensão (MOHAMED, 2017).

Figura 26 – Exemplo de *pooling*



Fonte: (MOHAMED, 2017)

A principal função dessa camada é diminuir a dimensão da entrada, “resumindo” informações das *features* extraídas nas camadas de convolução e, por consequência, reduzir a quantidade de parâmetros e custo computacional da rede neural.

As camadas de convolução e *pooling* podem ser repetidas múltiplas vezes em sequência, a fim de extrair *features* maiores da imagem de entrada, aumentando a robustez da rede.

2.3.3.3 Camada Totalmente Conectada

Após múltiplas camadas de convolução e *pooling*, os *feature maps* são então convertidos em um vetor unidimensional e passados como entrada de uma rede neural, em que cada neurônio está conectado a todos os neurônios da camada anterior. Essa conexão densa forma uma camada totalmente conectada, que permite que a rede combine as *features* extraídas para formar padrões mais complexos (MOHAMED, 2017).

2.3.4 You Only Look Once (YOLO)

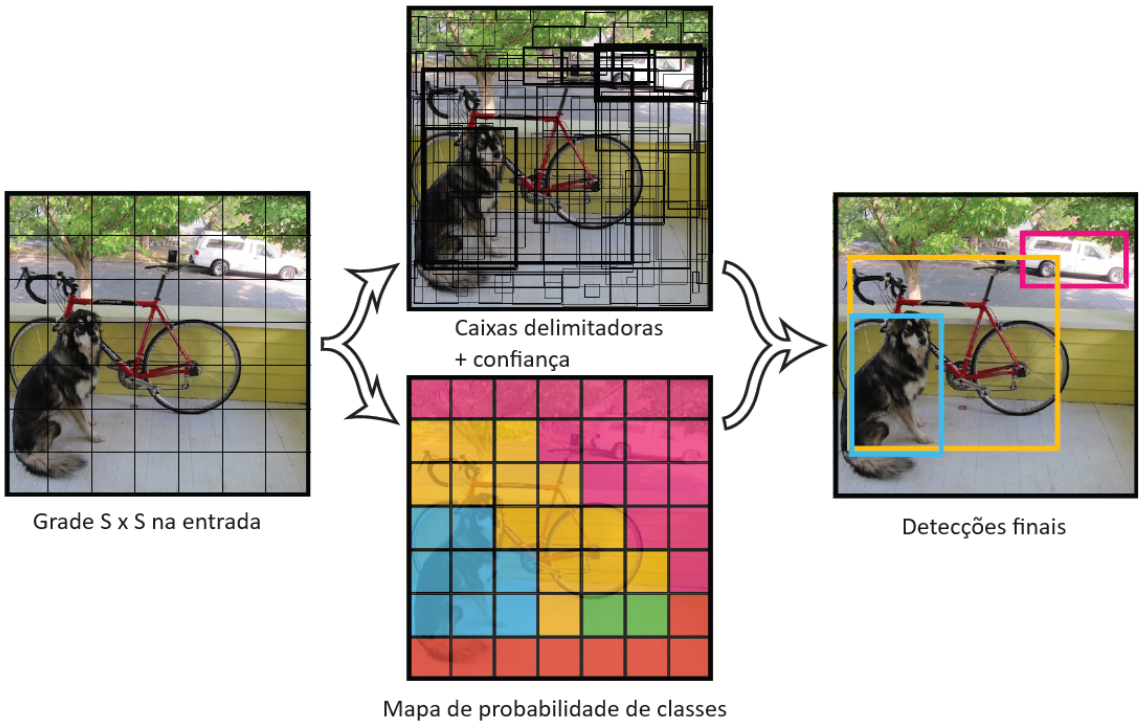
O *You Only Look Once* (YOLO) é uma técnica de estado-da-arte para detecção de objetos voltada a aplicações em tempo real. Ele trata a detecção de objetos como um problema de regressão única e aplica uma única rede neural à imagem. A rede neural divide a imagem em regiões e prevê as coordenadas das caixas delimitadoras e a probabilidade de classes em uma única avaliação para prever quais objetos estão presentes e onde estão (REDMON et al., 2016).

Conforme visualizado na Figura 27, o sistema divide a imagem de entrada em uma grade $S \times S$. Se o centro de um objeto estiver em uma célula da grade, esta célula será a responsável por detectar esse objeto. Cada célula prevê B caixas delimitadoras e gera uma pontuação de confiança, que reflete o quão confiante o modelo está de que aquela caixa contém um objeto.

Cada caixa delimitadora prevê as coordenadas do seu centro em relação aos limites da célula, sua altura e largura em relação à imagem completa e a sua confiança. Cada célula da grade prevê apenas C probabilidades de classe, independente da quantidade de caixas delimitadoras presentes.

A pontuação de confiança e a probabilidade de classe são combinadas para obter a probabilidade de cada caixa incluir um objeto de um tipo específico.

Figura 27 – Funcionamento do YOLO



Fonte: Adaptado de (REDMON et al., 2016)

O YOLO pode ser utilizado como um modelo pré-treinado ou ser treinado do zero. É possível encontrar versões mais atuais do YOLO pré-treinadas com o conjunto de dados COCO[1]. Além disso, é possível realizar um ajuste fino do modelo através do treinamento com um conjunto de imagens personalizado.

2.3.5 Roboflow

O Roboflow[2] é uma plataforma que oferece ferramentas para anotação de imagens, treinamento de modelos de aprendizado de máquina e implantação de modelos em produção. A plataforma permite que usuários treinem modelos de visão computacional sem a necessidade de conhecimento em programação.

Dentro da plataforma, os usuários podem criar projetos, que podem ser públicos ou privados, e compartilhá-los entre si. Existem diversos projetos e conjuntos de imagens públicos e gratuitos hospedados na plataforma.

Dentro de um projeto, um usuário pode carregar novas imagens, anotá-las manualmente ou com auxílio de inteligência artificial, treinar modelos e visualizar métricas de desempenho. A plataforma fornece uma *Application Programming Interface* (API) para

[1] https://cocodataset.org
[2] https://roboflow.com/

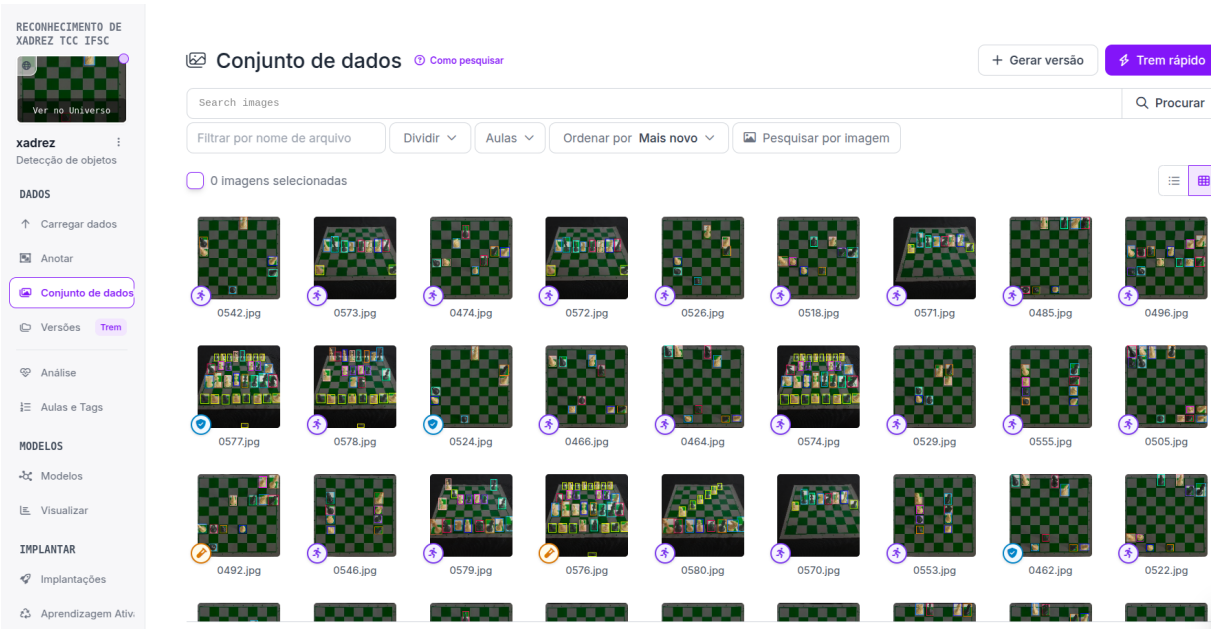
acesso aos modelos dos usuários, permitindo a integração com outras aplicações, e um ambiente de inferência para testar os modelos treinados. O Roboflow também disponibiliza *Software Development Kit* (SDK) gratuitos em diversas linguagens de programação para facilitar a integração com a sua API.

É possível testar modelos diretamente na plataforma, sem a necessidade de criar um *software* específico para isso. A plataforma oferece uma interface gráfica para realizar inferências em imagens, vídeos e *streams* de câmeras, além de permitir a exportação de modelos.

2.3.5.1 Anotação de Imagens

É possível anotar imagens e gerenciar o versionamento de seus conjuntos de imagens. A plataforma permite fazer a separação entre conjuntos de treinamento, teste e validação automaticamente. Além disso, é possível importar conjuntos de imagens de outras plataformas de armazenamento em nuvem ou exportar os conjuntos anotados para uso em outras aplicações. A Figura 28 apresenta a página de visualização de imagens anotadas em um projeto.

Figura 28 – Conjunto de imagens anotadas no Roboflow

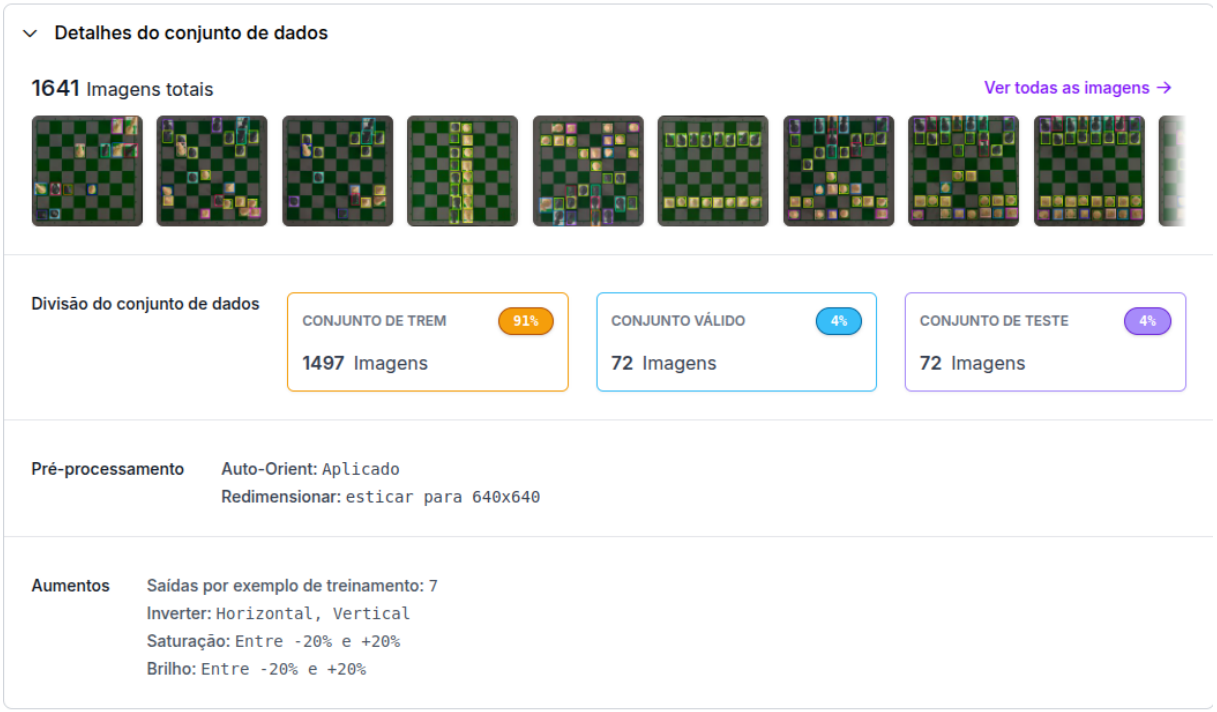


Fonte: Captura de tela da plataforma Roboflow

É possível aumentar o tamanho de um conjunto de dados em dezenas de vezes automaticamente aplicando técnicas para criar variações de uma mesma imagem, como rotação, espelhamento, variação de brilho, contraste, saturação e outras características da imagem. Pode-se também utilizar um modelo já treinado para auxiliar na anotação automática de imagens. A Figura 29 apresenta a página de detalhes de um conjunto

de dados, mostrando a divisão entre os subconjuntos e as técnicas de aumento de dados aplicadas.

Figura 29 – Detalhes do conjunto de dados



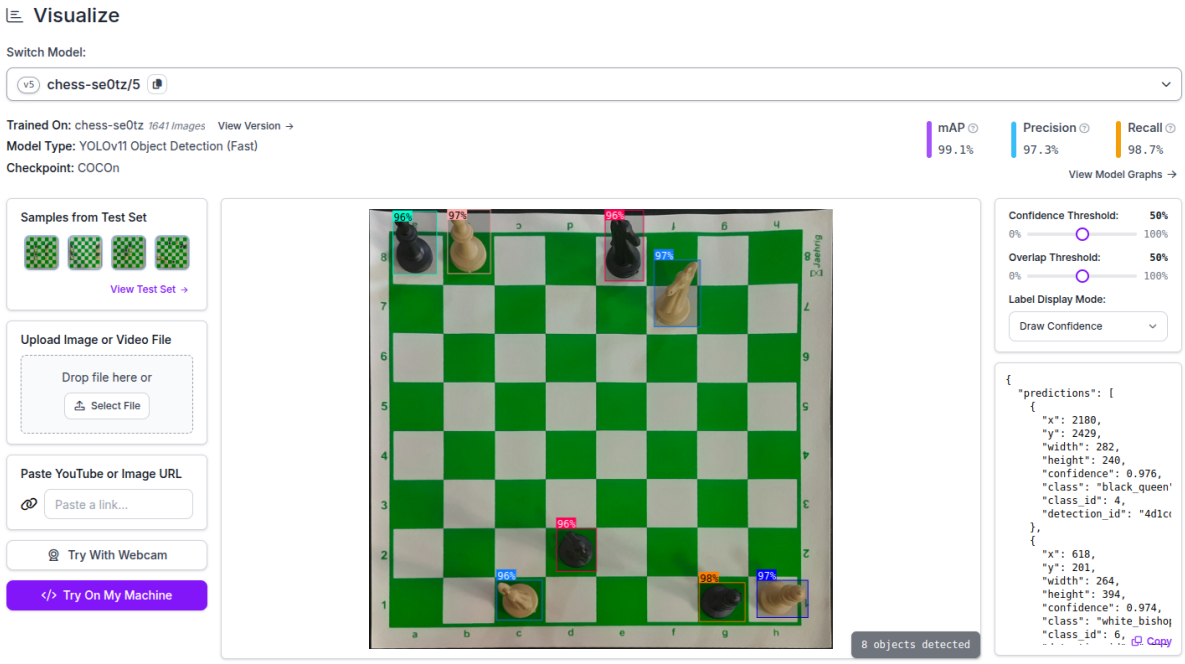
Fonte: Captura de tela da plataforma Roboflow

2.3.5.2 Treinamento de Modelos

O Roboflow permite o treinamento do modelo em nuvem de forma gratuita, com a possibilidade de escolher entre os modelos Roboflow 3.0 e YOLOv11, podendo partir de um modelo pré-treinado a partir do conjunto COCO, treinar do zero ou continuar a partir dos parâmetros de um modelo treinado anteriormente.

A Figura 30 apresenta a página de visualização de um modelo treinado, que permite testá-lo com o conjunto de imagens usado no treinamento, com novas imagens e vídeos que podem ser carregados diretamente na plataforma, com vídeos do YouTube ou com a imagem da câmera do dispositivo.

Figura 30 – Visualização do modelo treinado



Fonte: Captura de tela da plataforma Roboflow

O Código 2.1 mostra um exemplo de como um objeto detectado é representado na saída gerada pelo modelo YOLOv11 em formato *JavaScript Object Notation* (JSON), que contém informações sobre as caixas delimitadoras encontradas, suas coordenadas, a classe prevista e a probabilidade associada a essa classe. As coordenadas 'x' e 'y' indicam a posição do centro da caixa delimitadora, enquanto 'width' e 'height' indicam a largura e altura da caixa, respectivamente. Este formato é o mesmo recebido pela API de inferência do Roboflow.

Código 2.1 – Representação de um objeto detectado pelo YOLOv11 em formato JSON

```
{
    "predictions": [
      {
        "x": 2180,
        "y": 2429,
        "width": 282,
        "height": 240,
        "confidence": 0.976,
        "class": "black_queen",
        "class_id": 4,
        "detection_id": "4d1cd1fd-58e1-4ccb-9e3b-7c2dcafaa810"
      }
    ]
  }
```

3 METODOLOGIA

Neste capítulo é apresentada a metodologia empregada no desenvolvimento do trabalho, detalhando as etapas do processo e as ferramentas utilizadas.

3.1 Conjunto de imagens

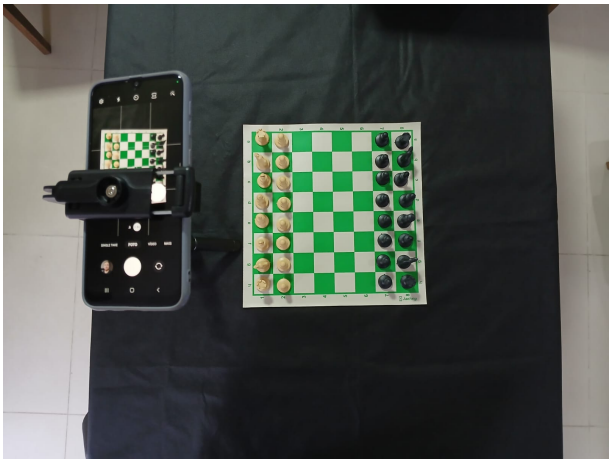
A criação e preparação de um conjunto de imagens é uma etapa fundamental no desenvolvimento de modelos de visão computacional voltados para a detecção e classificação de objetos. Para este estudo, foi elaborado um conjunto de imagens personalizado, destinado à detecção de peças de xadrez em um tabuleiro.

3.1.1 Configuração utilizada

A configuração do experimento foi baseada em imagens capturadas com um *smartphone* Samsung Galaxy M31, equipado com uma câmera de 16 MP, em resolução de 4624 x 3468 pixels na proporção 4:3. O tabuleiro utilizado possui dimensões de 34 cm x 34 cm, com cada casa medindo 4 cm de lado. Optou-se por um tabuleiro com casas verdes e brancas para evitar possíveis problemas de contraste entre peças pretas e casas pretas.

Com o auxílio de um tripé, foram ajustadas a altura da câmera, entre 40 e 60 centímetros, e variaram-se o ângulo e o *zoom*, buscando posicionar o tabuleiro de xadrez como o elemento principal da cena. Em parte das imagens, foi utilizado um tecido TNT preto sob o tabuleiro para minimizar reflexos de luz, que poderiam comprometer o processamento das imagens. A Figura 31 ilustra a configuração descrita.

Figura 31 – Configuração para captura de imagens



Fonte: Elaborada pelo autor.

3.1.2 Anotação de objetos

As imagens foram processadas com uma transformação de perspectiva, delimitando elas e ajustando o seu ângulo para conter apenas o tabuleiro. Posteriormente, utilizou-se a plataforma Roboflow para realizar a anotação do conjunto de imagens. Essa ferramenta permite gerar um modelo de visão computacional com base no conjunto já anotado, facilitando a anotação assistida por IA e acelerando o restante do processo.

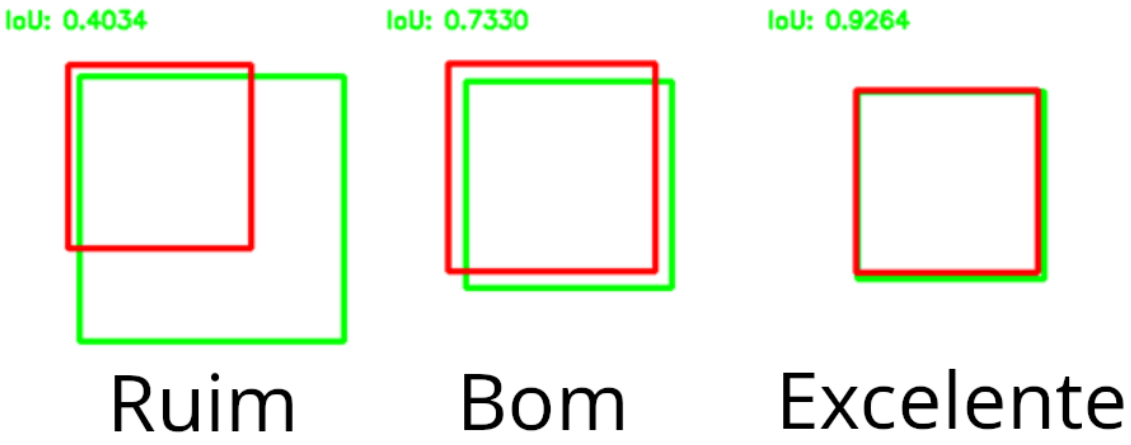
3.2 Métricas

A avaliação de modelos de detecção de objetos é uma tarefa que envolve diversas métricas, cada uma com um propósito específico. As principais métricas utilizadas são apresentadas a seguir.

3.2.1 Interseção sobre a União (*IoU*)

A *Intersection over Union* (IoU) quantifica a sobreposição entre duas caixas delimitadoras, sendo calculada pela razão entre a área da interseção e a área da união entre a caixa real e a prevista pelo modelo, assumindo um valor entre 0 e 1. É uma métrica comumente utilizada para avaliar a precisão de modelos de detecção de objetos. Um exemplo pode ser visto na Figura 32.

Figura 32 – Exemplo de cálculo da IoU entre duas caixas delimitadoras



Adaptada de Rosebrock (2016)

3.2.2 Confiança

A pontuação de confiança é uma medida de quão confiante o modelo está de que uma caixa delimitadora contém um objeto e que sua previsão da caixa delimitadora é

precisa. Seu cálculo é feito através da multiplicação da probabilidade de que a caixa contenha um objeto pela IoU entre a caixa prevista e a caixa real, usada como referência no treinamento. Ela assume um valor entre 0 e 1, sendo 1 a pontuação máxima.

3.2.3 Precisão

Mede a porcentagem de objetos detectados corretamente. É calculado através da soma de objetos detectados corretamente dividido pelo total de objetos detectados. É calculado pela razão entre verdadeiros positivos e a soma de verdadeiros positivos e falsos positivos.

3.2.4 Sensibilidade (*Recall*)

Mede a capacidade de identificar todas as instâncias de objetos nas imagens. É calculado através da soma de objetos detectados corretamente dividido pelo total de objetos presentes. É obtido pela razão entre verdadeiros positivos e a soma de verdadeiros positivos e falsos negativos.

3.2.5 Precisão Média (AP)

A curva precisão-recall relaciona o valor da precisão com o valor da sensibilidade para diferentes valores de limiar de confiança (ANWAR, 2022). A *Average Precision* (AP) é calculada como a área sob a curva precisão-recall, sendo uma métrica que varia de 0 a 1, onde 1 representa a melhor precisão possível. Com esta métrica, é possível avaliar o desempenho do modelo independentemente do limiar de confiança escolhido. A Equação 3.1 mostra a fórmula para o cálculo da AP, onde $p(r)$ é a precisão em um determinado valor de sensibilidade r.

$$AP = \int_{r=0}^{1} p(r)\, dr \tag{3.1}$$

3.2.6 Precisão Média Geral (*mAP*)

A *mean Average Precision* (mAP), ou precisão média geral, quantifica o desempenho de um modelo de detecção de objetos. Ela é calculada como a média aritmética das AP de cada classe de objeto, como visto na Equação 3.2, onde k é o número de classes.

$$mAP = \frac{1}{k} \sum_{i}^{k} AP_i \tag{3.2}$$

Ela pode ainda ser calculada para diferentes valores de IoU, como mAP50, que considera um IoU de 0,5, ou o mAP50-95, que é mais rigorosa e dá uma visão mais abrangente do desempenho do modelo, considerando valores de IoU entre 0,5 e 0,95.

3.2.7 Perda da Caixa Delimitadora (*Box Loss*)

Mede a diferença entre a caixa delimitadora prevista e a caixa real do objeto na imagem, garantindo que o modelo aprenda a localizar os objetos corretamente. Essa perda pode ser baseada em métricas como IoU, que penalizam previsões mal posicionadas e incentivam caixas mais precisas (BRIENZA et al., 2023).

3.2.8 Perda de Classificação (*Classification Loss*)

É uma métrica que indica quão bem um modelo é capaz de classificar corretamente os objetos detectados (BRIENZA et al., 2023). No modelo YOLO, essa perda é calculada utilizando a função de entropia cruzada, que penaliza previsões incorretas de classes de objetos.

3.2.9 Perda de Distribuição Focal (*Focal Loss*)

É uma função de perda que quantifica a qualidade da detecção de objetos, representando quão bem o modelo detecta a presença de um objeto em uma dada área de interesse na imagem (BRIENZA et al., 2023). Esta métrica é utilizada no treinamento para tornar o modelo mais confiável e preciso, garantindo que ele aprenda a lidar com a distribuição de classes desbalanceada e detectar objetos mais difíceis de serem identificados (TORRES, 2024) (KIM, 2024).

3.3 Ferramentas

Para o desenvolvimento do projeto, foi utilizado um computador pessoal modelo Lenovo Ideapad 3 15ALC6, com um processador Ryzen 5 5500U e 8 GB de memória RAM. O sistema operacional utilizado foi o Ubuntu 24.04 LTS.

A preparação do conjunto de dados e o treinamento do modelo foi realizado exclusivamente na plataforma Roboflow, escolhida por sua interface fácil de usar e pela disponibilidade gratuita de recursos, como a anotação assistida por IA.

O desenvolvimento dos algoritmos do projeto foi conduzido utilizando principalmente a linguagem de programação Python. Diversas bibliotecas foram empregadas para atender às diferentes necessidades do trabalho, destacando-se: Open-Source Computer Vision (OpenCV) versão 4.10.0.84, para o processamento de imagens; *Roboflow Python*

Package[1] versão 1.1.50, usada para acessar o modelo YOLO através da API de sua plataforma; e *python-chess*[2] versão 1.11.1, utilizada para validação de jogadas e implementação de funcionalidades específicas relacionadas às regras do xadrez.

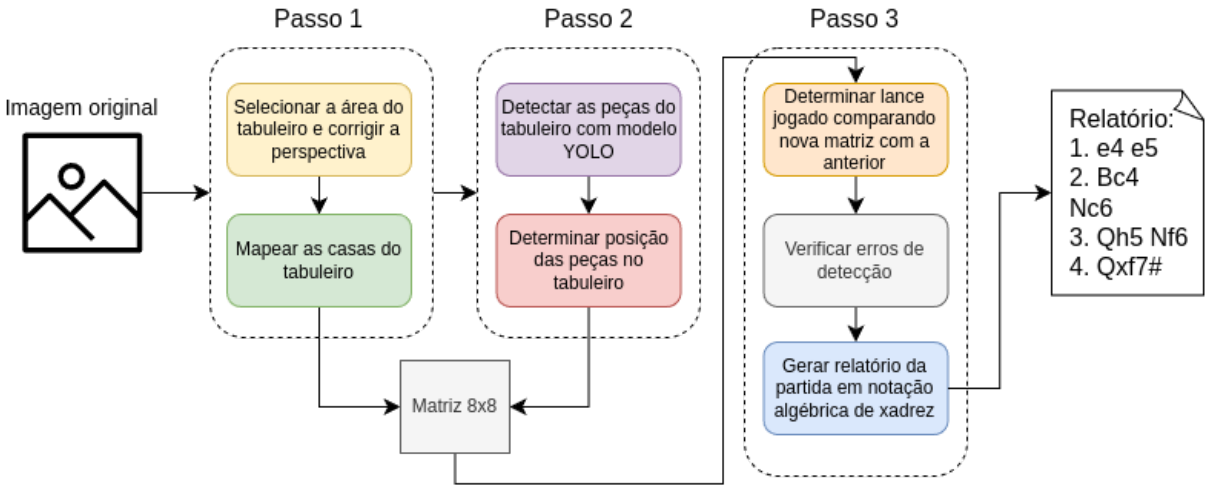
[1] https://docs.roboflow.com/api-reference/install-python-package
[2] https://python-chess.readthedocs.io/en/latest/

4 DESENVOLVIMENTO

4.1 Fluxo de funcionamento do algoritmo

O funcionamento do algoritmo consiste em etapas principais que permitem identificar os lances realizados a partir das imagens recebidas, conforme ilustrado na Figura 33.

Figura 33 – Fluxo geral de funcionamento do algoritmo.



Fonte: Elaborada pelo autor.

Inicialmente, a imagem original passa por técnicas de processamento para isolar a área do tabuleiro e aplicar uma transformação de perspectiva, gerando uma nova imagem com o ângulo devidamente ajustado.

Na sequência, é feito o mapeamento do tabuleiro, no qual a imagem ajustada é processada novamente para localizar os vértices de cada casa do tabuleiro. Com essas informações, as posições das casas são armazenadas em uma matriz 8×8.

Após o mapeamento do tabuleiro, o modelo YOLO realiza a detecção dos objetos presentes na imagem, identificando suas classes e as coordenadas de suas caixas delimitadoras. Então, determina-se a posição das peças através da comparação das coordenadas do centro da área de sua caixa delimitadora com as coordenadas das extremidades das casas do tabuleiro. Desta forma, a classe de cada peça detectada é registrada na matriz na posição correspondente à sua casa no tabuleiro.

Quando uma nova imagem é carregada, o processo se repete, e a matriz gerada é comparada com a matriz anterior. As diferenças entre ambas indicam os movimentos realizados.

Para mitigar problemas relacionados a erros na detecção ou classificação de objetos, utiliza-se uma biblioteca que valida os lances com base no estado do tabuleiro. Caso o lance identificado pelo algoritmo não seja legal, ele pode indicar uma falha de detecção ou um erro na classificação. Nessa situação, o algoritmo pode executar rotinas corretivas, como solicitar ao usuário uma nova foto do tabuleiro.

Ao final da partida, todos os lances identificados são concatenados para gerar um relatório completo, que é então apresentado ao usuário na notação algébrica padrão.

4.2 Conjunto de Imagens

A construção do conjunto de imagens consiste não só em capturar fotos do tabuleiro, mas também em aplicar técnicas de processamento de imagens para selecionar apenas a área do tabuleiro, e anotar as peças presentes nas imagens.

4.2.1 Seleção do tabuleiro

A seleção do tabuleiro e a transformação de perspectiva são etapas fundamentais para minimizar a interferência de elementos externos na detecção e mapeamento do tabuleiro, além de garantir que variações no ângulo e na altura da câmera tenham impacto reduzido nos resultados do modelo YOLO. Esse processo consiste em uma sequência de técnicas de processamento de imagens, como ilustrado na Figura 34.

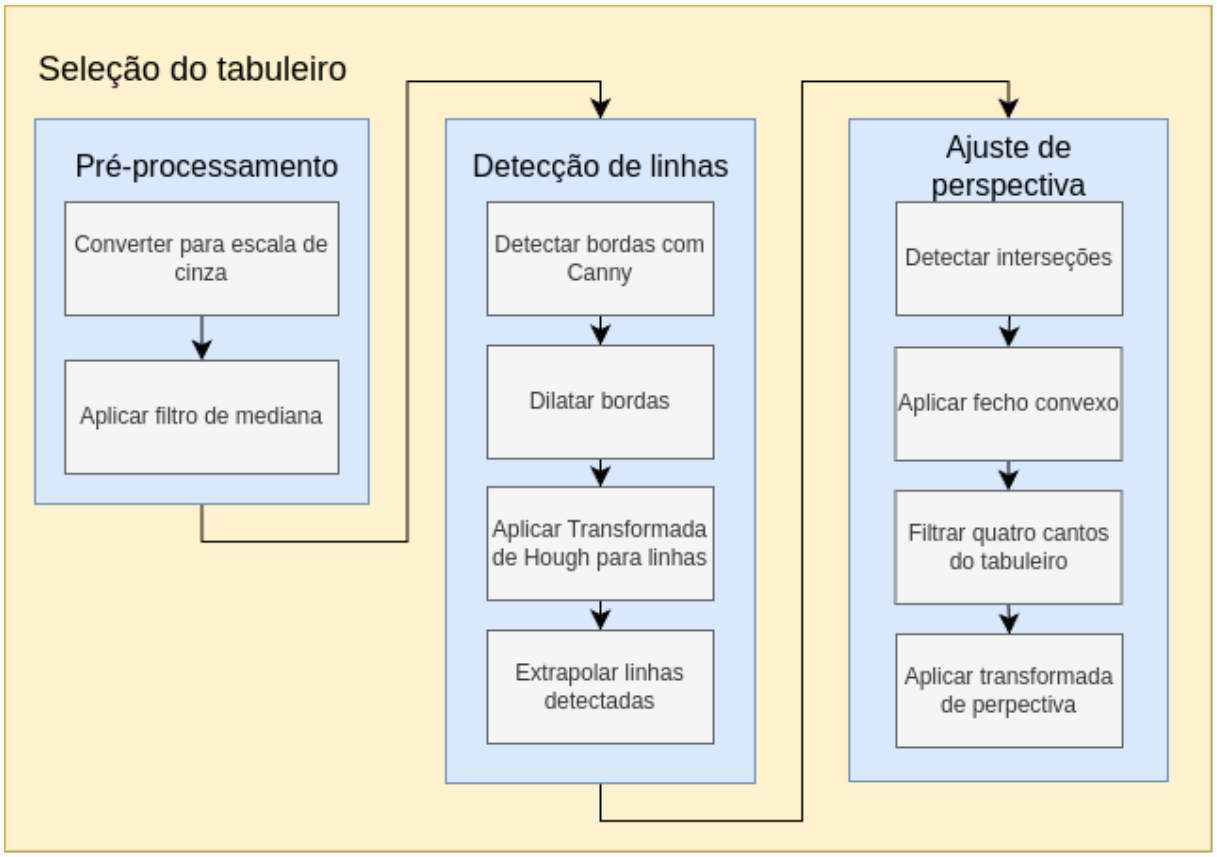
O primeiro passo é converter a imagem de RGB para escala de cinza, permitindo o uso de algoritmos de detecção de bordas, que operam exclusivamente com imagens monocromáticas. Em seguida, aplica-se um filtro de mediana para suavizar a imagem e reduzir ruídos causados por reflexos ou sombras.

Posteriormente, utiliza-se o detector de bordas de Canny para identificar contornos e bordas do tabuleiro. As bordas resultantes passam por uma transformação morfológica de dilatação para preencher pequenas descontinuidades. A transformada de Hough é então aplicada para detectar linhas contínuas com um comprimento mínimo de 200 *pixels*, evitando a detecção de bordas das peças de xadrez, e tolerando falhas de até 20 *pixels*.

Como as peças frequentemente obstruem a grade do tabuleiro, as linhas geradas pela transformada de Hough apresentam interrupções que o algoritmo não consegue corrigir sem introduzir falsos positivos. Para contornar isso, todas as linhas detectadas são extrapoladas, estendendo-se de uma extremidade da imagem à outra. Embora isso resulte em sobreposição de linhas e detecção de algumas linhas indesejadas, essas redundâncias não afetam a identificação dos quatro vértices extremos do tabuleiro.

O próximo passo é encontrar as interseções entre as linhas detectadas que formam ângulos próximos de 90 graus, com tolerância de até 10 graus. Essas interseções fornecem

Figura 34 – Sequência de técnicas usada para seleção do tabuleiro



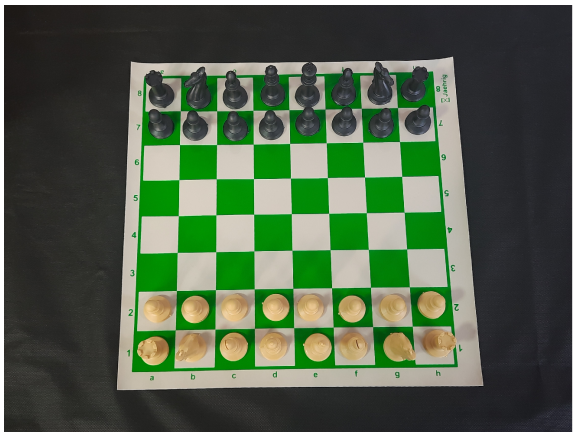
Fonte: Elaborada pelo autor.

uma lista de candidatos a vértices do tabuleiro.

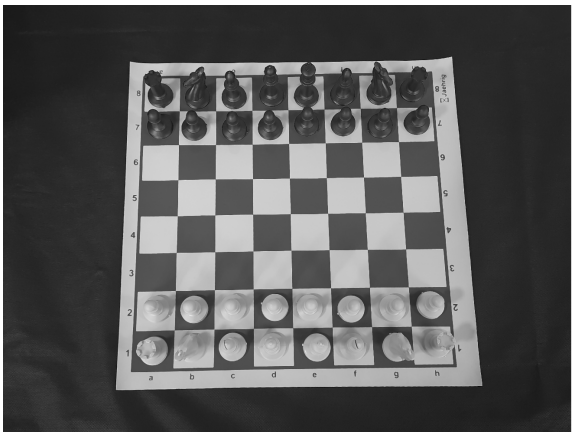
Para isolar os quatro pontos extremos, aplica-se o fecho convexo, que filtra os pontos pertencentes ao maior polígono detectado, correspondente ao contorno do tabuleiro. Entre esses pontos, os vértices mais extremos são determinados com base na soma e na diferença das coordenadas x e y, correspondendo às posições superior esquerda, superior direita, inferior esquerda e inferior direita. Por fim, os vértices extraídos permitem aplicar uma transformação de perspectiva, resultando em uma imagem na qual o tabuleiro é visualizado diretamente de cima.

A Figura 35 mostra exemplos de cada etapa descrita e apresenta os resultados intermediários, desde a conversão para escala de cinza até a obtenção da imagem final com a perspectiva corrigida. Algumas etapas foram omitidas pois não apresentam resultados visíveis, como o filtro de mediana.

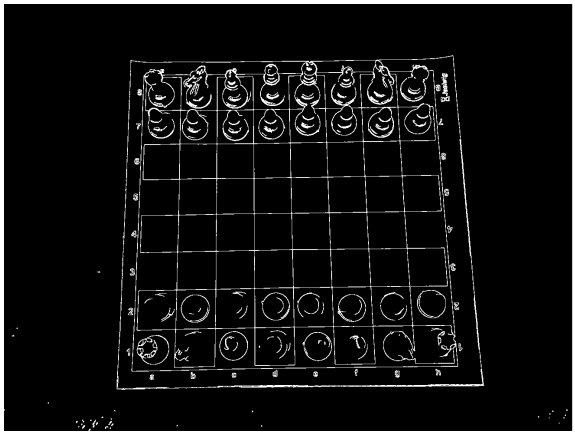
Figura 35 – Resultados intermediários das técnicas para seleção do tabuleiro



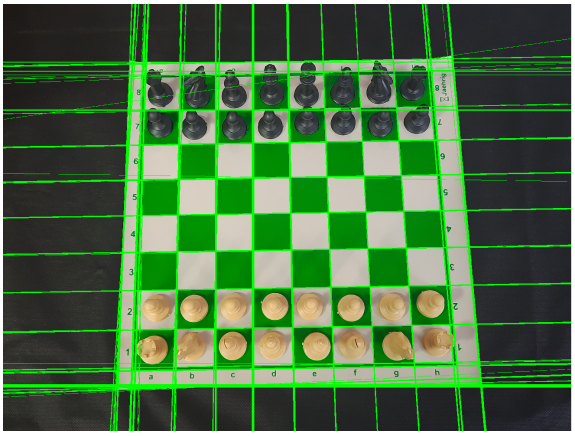
(a) Imagem original



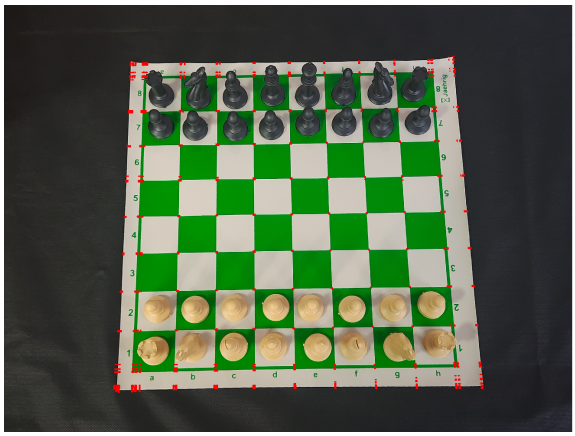
(b) Escala de cinza



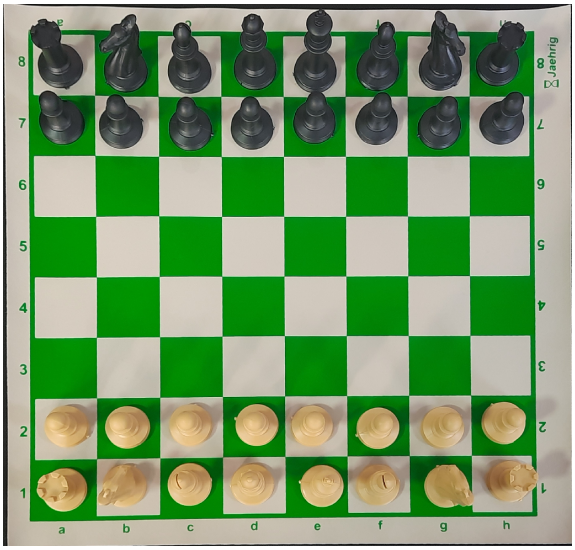
(c) Bordas detectadas



(d) Linhas extrapoladas



(e) Interseções



(f) Correção de perspetiva

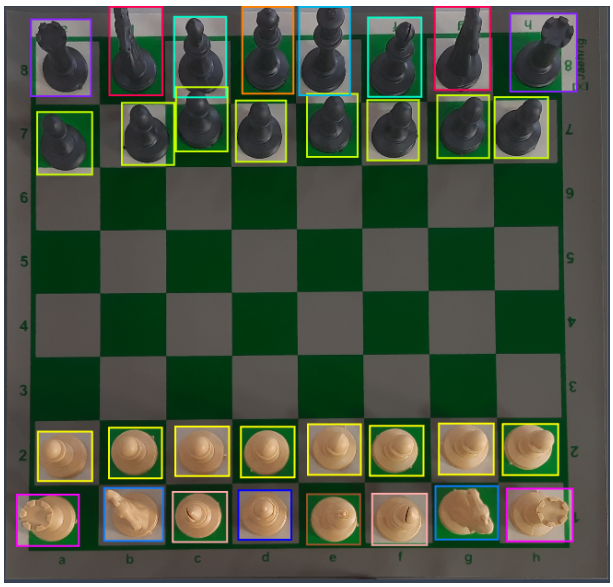
Fonte: Elaborada pelo autor.

4.2.2 Anotação das imagens

A anotação das imagens foi feita utilizando a plataforma Roboflow. Para cada imagem, foi necessário desenhar a caixa delimitadora de cada objeto manualmente, conforme mostrado na Figura 36, em que cada classe de objeto possui uma cor diferente e corresponde a um tipo de peça.

Após ter uma parte do conjunto de dados já anotada, foi possível gerar um modelo treinado com este conjunto reduzido para ajudar a detectar e anotar as demais imagens, o que tornou o processo mais rápido.

Figura 36 – Imagem anotada na plataforma Roboflow



Fonte: Elaborada pelo autor.

4.2.3 Distribuição de classes

No Roboflow, a divisão das imagens em subconjuntos de treinamento, teste e validação foi configurada automaticamente, utilizando proporções de 60%, 20% e 20%, respectivamente. Contudo, devido à variação na quantidade de peças de cada classe em cada imagem, a divisão exata não foi possível.

Foram anotadas 358 imagens, totalizando 7297 peças de xadrez em diferentes ângulos e posições no tabuleiro. A distribuição de classes no conjunto de dados e a separação em subconjuntos pode ser vista no Tabela 1.

Após essa separação, o Roboflow possibilitou expandir o conjunto de treino em até 7 vezes, gerando variações de uma mesma imagem com base em parâmetros configuráveis. Para este trabalho, foram aplicadas variações no brilho e na saturação entre -20% e $+20\%$, além de espelhamentos verticais e horizontais em algumas imagens. Essas modificações

Tabela 1 – Conjunto de dados construído para treinamento e teste

Peça	**Treino**	**Validação**	**Teste**
Peão branco	828	344	420
Torre branca	268	95	118
Cavalo branco	286	85	105
Bispo branco	241	76	103
Dama branca	184	68	69
Rei branco	172	69	72
Peão preto	847	360	418
Torre preta	293	117	127
Cavalo preto	282	85	107
Bispo preto	274	86	104
Dama preta	159	53	64
Rei preto	175	69	72
Total	3834	1507	1779

são realizadas automaticamente pela plataforma, aumentando o conjunto de treino sem necessidade de novas anotações e tornando o modelo mais robusto a pequenas variações.

4.3 Treinamento do modelo *YOLO*

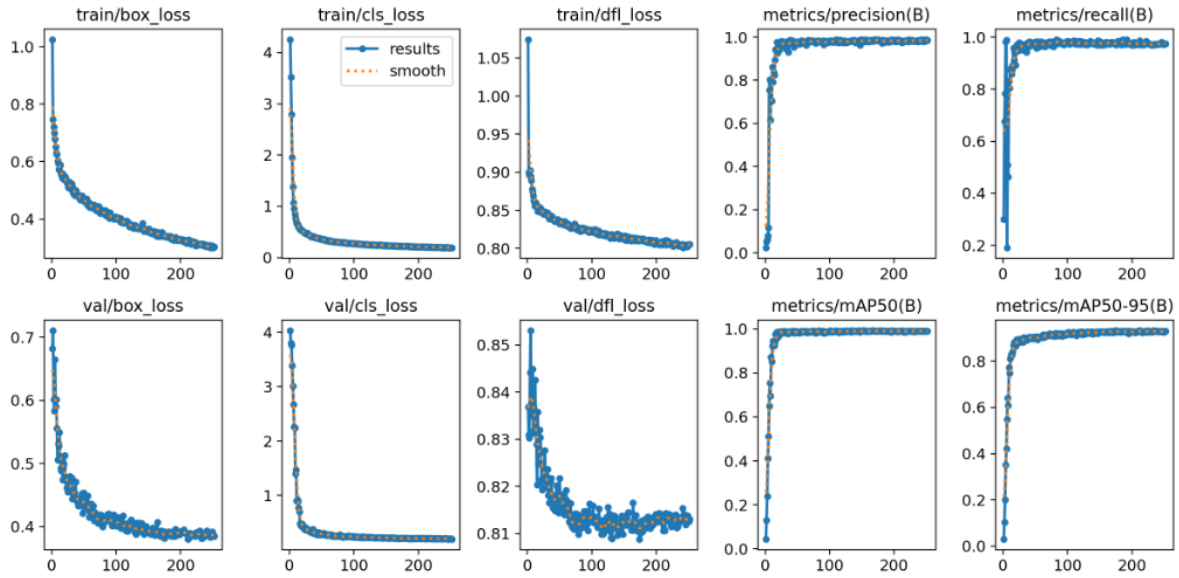
O treinamento do modelo de visão computacional YOLOv11 foi realizado utilizando a plataforma Roboflow, que facilita o treinamento e versionamento de modelos, disponibilizando gratuitamente o modelo YOLOv11 e seu próprio modelo Roboflow 3.0.

O processo de treinamento demorou cerca de 3 horas, com 250 gerações e obteve uma mAP de 99.1%, precisão de 97.3% e *recall* de 98.7% com o conjunto de testes utilizado. A mAP de cada classe pode ser vista no Tabela 2, tanto para o conjunto de validação quanto o de teste.

Tabela 2 – mAP de cada classe obtida no treinamento do modelo YOLOv11

Classe	**Peça**	**mAP validação**	**mAP teste**
black_bishop	Bispo preto	98%	99%
black_king	Rei preto	99%	99%
black_knight	Cavalo preto	99%	100%
black_pawn	Peão preto	99%	99%
black_queen	Dama preta	100%	100%
black_rook	Torre preta	99%	100%
white_bishop	Bispo branco	98%	100%
white_king	Rei branco	99%	100%
white_knight	Cavalo branco	99%	100%
white_pawn	Peão branco	99%	100%
white_queen	Dama branca	99%	100%
white_rook	Torre branca	99%	100%

Figura 37 – Estatísticas de treinamento do modelo YOLOv11



Fonte: Gerada pela plataforma Roboflow.

A Figura 37 mostra as estatísticas de treinamento do modelo fornecidas pela plataforma, em que o eixo x dos gráficos representa a quantidade de gerações. Percebe-se que a mAP do modelo aumenta rapidamente nas primeiras gerações e se estabiliza em torno de 99% próximo à 50ª geração. A precisão e o *recall* também atingem valores próximos ao seu máximo no mesmo período.

Pode-se ver também que a perda da caixa delimitadora (*box_loss*) continua diminuindo até o final do treinamento, indicando que o modelo continuou aprendendo e refinando suas previsões quanto à detecção dos objetos.

Por outro lado, vê-se que a perda de classificação inicia com um valor próximo de 4 e atinge seu mínimo e estabiliza abaixo de 0,5 entre a 50ª e a 100ª geração, o que sugere que o modelo alcançou um ponto onde aprendeu a classificar corretamente os objetos nos dados de treinamento e validação, sem melhorias significativas adicionais, possivelmente devido ao tamanho pequeno e pouca variedade do conjunto de dados.

A perda de distribuição focal também apresenta um comportamento semelhante, o que indica que o modelo aprendeu a lidar melhor com a distribuição de classes desbalanceada, penalizando mais as previsões incorretas de classes minoritárias.

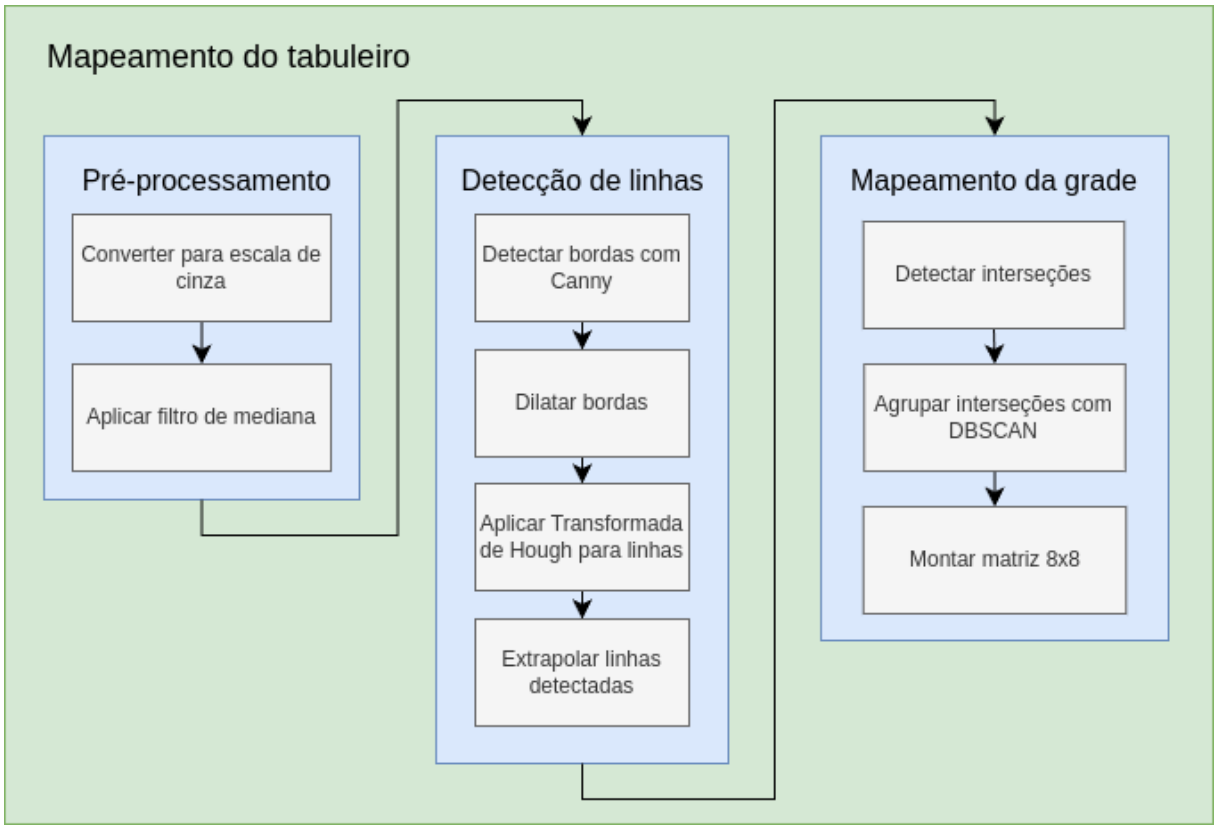
4.4 Algoritmos

Uma série de algoritmos foi desenvolvida a fim de extrair as posições das peças de xadrez e gerar o relatório da partida a partir das imagens originais com a perspectiva corrigida. Os algoritmos desenvolvidos consideram sempre que o tabuleiro está na posição normal, com a casa "a1" localizada no canto inferior esquerdo e a "h8" no canto superior direito. Além disso, devido ao uso do DBSCAN, o espaçamento entre as retas que compõem uma grade não pode ter grandes variações ao longo do tabuleiro, ou seja, o ângulo de captura da foto deve ser o mais próximo possível de 90º em relação ao tabuleiro.

4.4.1 Mapeamento das casas do tabuleiro

O mapeamento das casas do tabuleiro é necessário para determinar as coordenadas de cada casa do tabuleiro e poder descobrir em que casa cada peça está posicionada. Ela consiste inicialmente nas mesmas técnicas utilizadas para seleção do tabuleiro para pré-processamento da imagem e detecção de linhas, conforme a Figura 38. Porém, a partir da detecção de interseções, é preciso haver uma melhor filtragem dos pontos, uma vez que apenas 81 interseções compõem a grade do tabuleiro de xadrez.

Figura 38 – Sequência de técnicas usada para o mapeamento das casas do tabuleiro



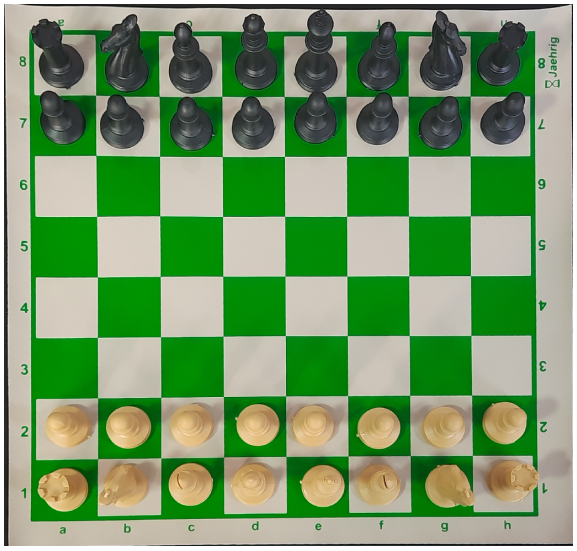
Fonte: Elaborada pelo autor.

Após repetir o mesmo processo utilizado na seleção do tabuleiro e detectar as interseções, têm-se milhares de pontos sobrepostos uns aos outros que devem ser agrupados em um só. Aqueles fora da grade do tabuleiro devem ser considerados anomalias e excluídos. Para este fim, aplica-se a técnica de clusterização de dados DBSCAN, com um raio definido empiricamente em 1/12 da largura do tabuleiro e uma quantidade mínima de 5 pontos dentro deste raio para configurar um *cluster*. O ponto central é definido como a mediana das coordenadas dos elementos do *cluster*, uma vez que a região em que ocorre a interseção das retas que compõem a grade do tabuleiro possui maior densidade de elementos e, por consequência, a mediana obtida é sempre exatamente sobre as interseções desejadas.

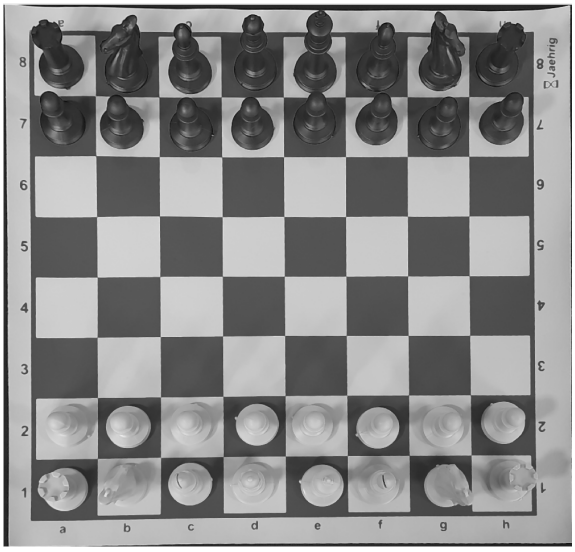
Por fim, as coordenadas dos vértices de cada casa do tabuleiro são armazenadas em uma matriz 8×8, de forma que possam ser obtidas facilmente em etapas posteriores e utilizadas para determinar a posição de uma peça no tabuleiro.

A Figura 39 mostra exemplos de cada etapa descrita para o mapeamento do tabuleiro e apresenta os resultados intermediários das técnicas utilizadas, desde a conversão para escala de cinza até a obtenção das interseções agrupadas com DBSCAN.

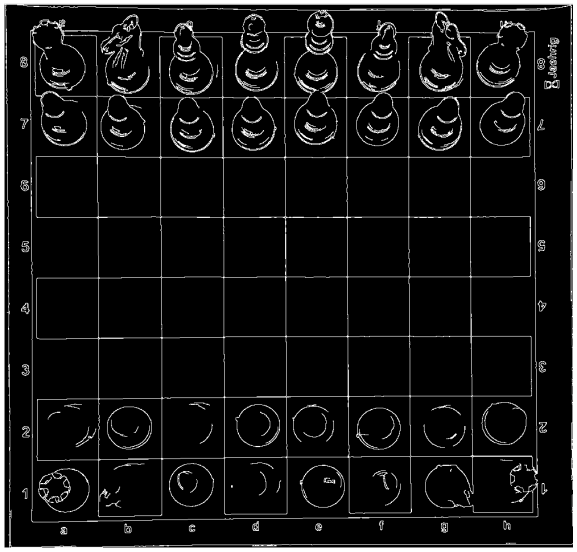
Figura 39 – Resultados intermediários das técnicas para mapeamento do tabuleiro



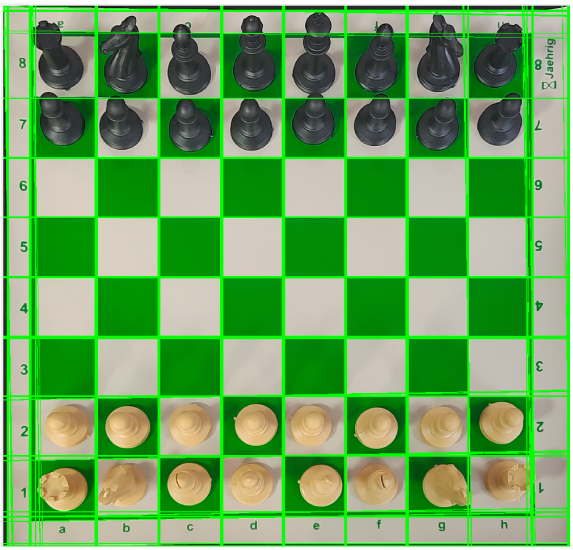
(a) Imagem original



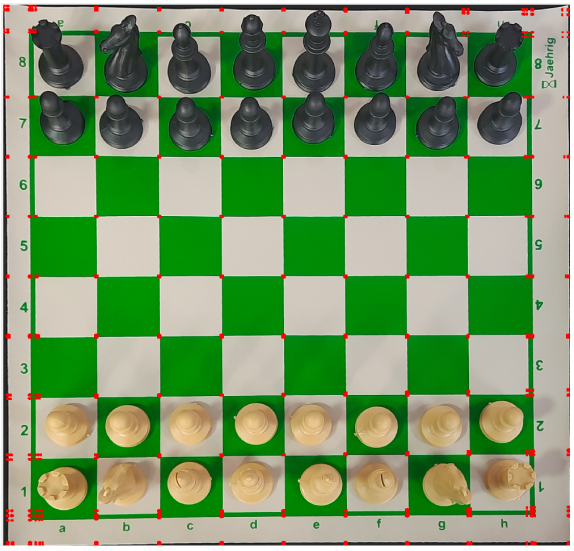
(b) Escala de cinza



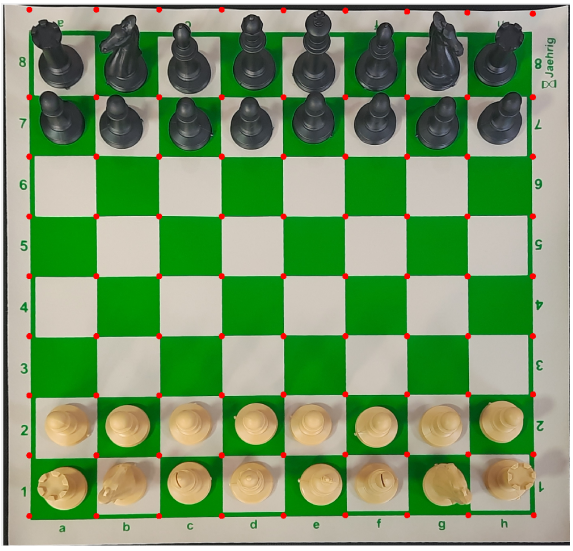
(c) Bordas detectadas



(d) Linhas extrapoladas



(e) Interseções



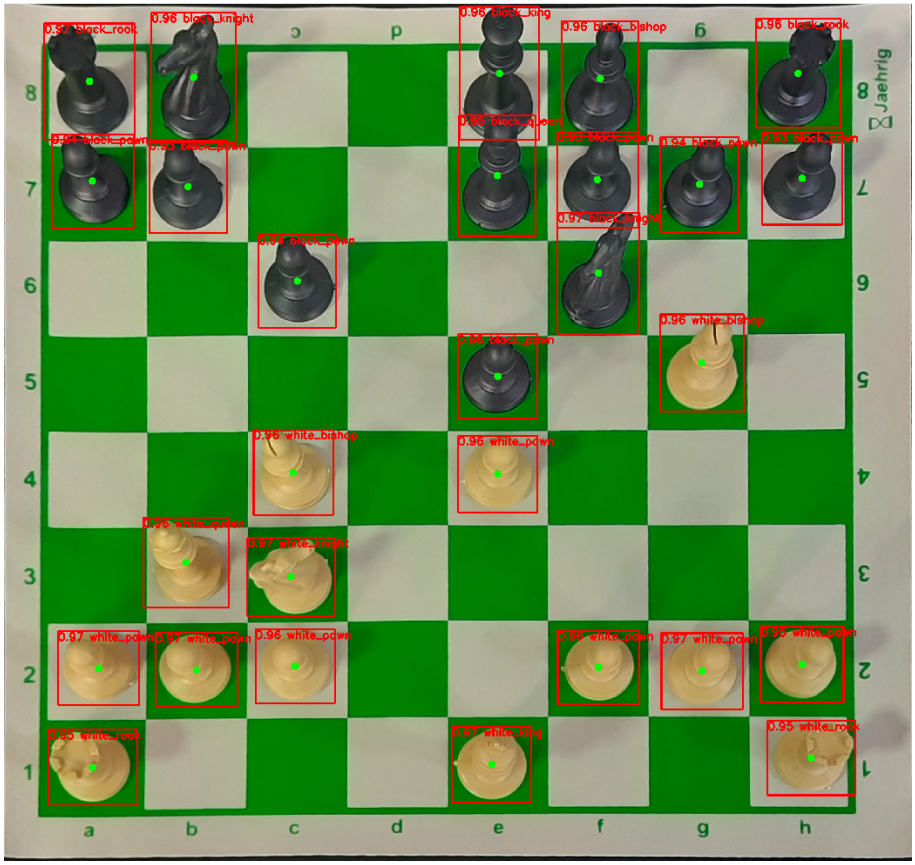
(f) Resultado do DBSCAN

Fonte: Elaborada pelo autor.

4.4.2 Detecção das peças

A detecção das peças é realizada por inferência via API do Roboflow, que retorna uma resposta em formato JSON, como a do Código 2.1, com as previsões do modelo YOLOv11. Cada previsão contém a classe da peça, a confiança da detecção e as coordenadas da caixa delimitadora da peça na imagem. A Figura 40 mostra um exemplo de detecção de peças em uma imagem, com as caixas delimitadoras desenhadas sobre as peças, o índice de confiança e a classe de cada peça.

Figura 40 – Peças detectadas pelo modelo YOLO



Fonte: Elaborada pelo autor.

4.4.3 Determinação da posição das peças no tabuleiro

Determinar a posição das peças no tabuleiro é um processo consideravelmente simples após o mapeamento das casas do tabuleiro. As predições do modelo YOLOv11 são utilizadas para obter as coordenadas das caixas delimitadoras das peças detectadas.

Para cada peça detectada, as coordenadas do centro de sua caixa delimitadora são comparadas com as coordenadas dos vértices das casas do tabuleiro, fileira por fileira, da casa ‘a1’ à ‘h8’. A casa que contém o centro da caixa delimitadora (representado por um ponto verde na Figura 40) é considerada a posição da peça.

4.4.4 Detecção de erros

Com o objetivo de identificar possíveis erros do modelo na detecção e classificação das peças, foi desenvolvido uma validação dos lances com base no estado do tabuleiro. Para este fim, foi utilizada a biblioteca *python-chess*, que permite a representação e manipulação de tabuleiros de xadrez em diferentes notações, validação de jogadas e determinação do lance realizado com base nos estados do tabuleiro antes e depois do lance.

A matriz com as peças é exportada para uma notação FEN, que é utilizada pela biblioteca *python-chess* para criar um objeto da classe 'Board()', que representa o tabuleiro de xadrez e possui diversos métodos e atributos úteis a respeito da partida.

Neste processo de criação do objeto, ele valida a quantidade de peças de cada tipo e sua posição. Se houver alguma peça a mais ou a menos, ou se a posição de alguma peça não corresponder à posição esperada, considera-se que houve um erro na detecção ou classificação das peças e é solicitada ao usuário uma nova foto do tabuleiro.

4.4.5 Identificação do lance e geração da notação algébrica

Dentre os atributos da classe 'Board()', destaca-se o *legal_moves*[1], que permite obter todos os lances legais possíveis dada a posição das peças no tabuleiro.

A identificação do lance é feita comparando dois estados do tabuleiro gerados a partir das matrizes de peças detectadas, antes e depois de um lance.

A biblioteca *python-chess* permite a obtenção do lance realizado na notação algébrica de xadrez a partir do estado inicial do tabuleiro e da lista de lances legais. Cada lance legal é testado até que se descubra qual originou o estado final em que o tabuleiro se encontra, e então é retornado como o lance realizado.

Ao fim da partida, identificado pela ausência de novas imagens na sequência, o algoritmo concatena todos os lances identificados e gera um relatório completo da partida, que é então apresentado ao usuário como visto na Figura 41. Os *logs* completos da execução do algoritmo podem ser vistos no Código A.1 do Apêndice A.

[1] https://python-chess.readthedocs.io/en/latest/core.html#chess.Board.legal_moves

Figura 41 – Relatório da partida gerado pelo algoritmo

```
1. e4 e5
2. Nf3 d6
3. d4 Bg4
4. dxe5 Bxf3
5. Qxf3 dxe5
6. Bc4 Nf6
7. Qb3 Qe7
8. Nc3 c6
9. Bg5 b5
10. Nxb5 cxb5
11. Bxb5+ Nbd7
12. O-O-O Rd8
13. Rxd7 Rxd7
14. Rd1 Qe6
15. Bxd7+ Nxd7
16. Qb8+ Nxb8
17. Rd8#
=======================================
```

Fonte: Elaborada pelo autor.

5 CONCLUSÕES

Este trabalho teve como objetivo desenvolver uma aplicação capaz de fazer o reconhecimento por imagem de lances em um jogo de xadrez utilizando técnicas de visão computacional e redes neurais convolucionais. Para isso, foi necessário aplicar técnicas de processamento de imagens, criar um conjunto de dados a partir de imagens de partidas de xadrez e treinar um modelo de detecção de peças de xadrez.

A primeira etapa do trabalho consistiu em capturar imagens de partidas de xadrez com um tabuleiro próprio. Depois, foi necessário aplicar uma série de técnicas de processamento de imagens para corrigir a sua perspectiva, a fim de garantir que o modelo usado trabalhasse com imagens do tabuleiro sempre na mesma perspectiva, próxima de 90 graus em relação ao tabuleiro, devido ao uso de técnicas como o DBSCAN.

Em seguida, foi feita na plataforma Roboflow a anotação manual das peças de xadrez presentes nas imagens. A partir de uma certa quantidade de imagens já anotadas, foi possível treinar um modelo inicial capaz de ajudar a rotular as imagens restantes. Então, foi treinado no Roboflow um modelo YOLOv11 para a identificação das peças no tabuleiro.

O modelo atingiu uma mAP de 99,1%, uma precisão de 97,3% e um *recall* de 98,7%. A perda de classificação obtida durante o treinamento sugere que o modelo alcançou um ponto de saturação, em que aprendeu a classificar corretamente os objetos presentes nas imagens, mas sem possibilidade de melhorias significativas, possivelmente devido ao tamanho pequeno e pouca variedade do conjunto de dados.

Por fim, com as informações das peças detectadas e classificadas pelo modelo obtidas por inferência via API do Roboflow, foi possível realizar cálculos para determinar a posição das peças no tabuleiro e, a partir disso, determinar os lances realizados ao longo da partida. Por último, a aplicação gera um relatório detalhado em notação algébrica de xadrez dos lances feitos pelos jogadores.

Para possibilitar a reprodutibilidade do trabalho, o código do projeto está disponível publicamente em um repositório do GitHub[1].

5.1 Trabalhos Futuros

O foco deste trabalho foi conseguir realizar o reconhecimento de uma partida de xadrez inteira, lance a lance, a partir de imagens capturadas de um tabuleiro real. No

[1] https://github.com/andreyadriano/chess-recognition

entanto, a solução ainda apresenta algumas limitações e pode ser aprimorada em diversos aspectos. Alguns possíveis trabalhos futuros incluem:

- Melhorias nas técnicas de processamento de imagens, que se mostraram o maior ponto de falha do algoritmo. Em específico a detecção de interseções da grade do tabuleiro feita anteriormente à transformação de perspectiva, que falha em até 30% das vezes;
- Treinamento local de um modelo para diminuir o tempo para detecção de peças, pois a inferência via API do Roboflow pode ter um tempo de resposta elevado, de 2 a 6 segundos por requisição;
- Testar outros modelos de detecção de objetos para verificar se há algum que se adapte melhor ao problema proposto;
- Correção de erros de detecção e classificação de peças através da comparação com a posição anterior das peças e do uso da segunda melhor predição dada pelo modelo;
- Validação do fim da partida por outros meios que não o fim da sequência de imagens, como a detecção de um xeque-mate, a contagem de movimentos ou informar à aplicação um empate ou desistência;
- Realizar o reconhecimento de peças a partir de videos, ao invés de imagens estáticas;
- Tornar possível fazer o reconhecimento da partida iniciando em qualquer posição do tabuleiro, ao invés de sempre iniciar da posição inicial das peças;
- Corrigir a posição do tabuleiro, caso as fotos sejam capturadas a partir das laterais do tabuleiro, por exemplo.
- Construção de uma API para integração com outras aplicações;
- Desenvolvimento de um aplicativo para dispositivos móveis, o qual permitiria a captura de imagens do tabuleiro e a análise imediata após cada lance, ao invés de uma sequência fixada de imagens.

REFERÊNCIAS

ANWAR, A. *What is Average Precision in Object Detection & Localization Algorithms and how to calculate it?* 2022. Disponível em: https://medium.com/towards-data-science/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330e Acesso em: 9 fev 2025. 42

BRIENZA, M. et al. Hri-based gaze-contingent eye tracking for autism spectrum disorder treatment: A preliminary study using a nao robot. In: . [S.l.: s.n.], 2023. 43

CANNY, J. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8, p. 679 – 698, 12 1986. 26

CHESS.COM. *Forsyth-Edwards Notation (FEN)*. 2020. Disponível em: https://www.chess.com/terms/fen-chess. Acesso em: 3 fev 2024. 23

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: SIMOUDIS, E.; HAN, J.; FAYYAD, U. M. (Ed.). AAAI Press, 1996. p. 226–231. ISBN 1-57735-004-9. Disponível em: http://dblp.uni-trier.de/db/conf/kdd/kdd96.html#EsterKSX96. 30

FIDE. *FIDE LAWS OF CHESS*. 2022. Disponível em: https://handbook.fide.com/chapter/E012023. Acesso em: 20 maio 2024. 14, 15, 20, 21, 22, 23

FLORES, T. *Median Filtering with Python and OpenCV*. 2019. Disponível em: https://medium.com/@florestony5454/median-filtering-with-python-and-opencv-2bce390be0d1. Acesso em: 19 fev 2025. 29

FORBES. *From Healthcare To Space: Top 10 Transformative Computer Vision Trends In 2024*. 2023. Disponível em: https://www.forbes.com/sites/bernardmarr/2023/09/26/from-healthcare-to-space-top-10-transformative-computer-vision-trends-in-2024/?sh=2138bcfd72c0. Acesso em: 5 abr 2024. 12

GLASSNER, A. *Deep Learning: A Visual Approach*. [S.l.]: No Starch Press, 2021. ISBN 9781718500723. 32, 33

GONZALEZ, R. C.; WOODS, f. e. R. E. *Digital Image Processing*. [S.l.]: Pearson, 2018. 25, 26, 27, 29

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Disponível em: http://www.deeplearningbook.org. 31, 33

HEATH, D.; ALLUM, D. The historical development of computer chess and its impact on artificial intelligence. *AAAI Technical Report WS-97-04*, p. 63–68, janeiro 1997. Disponível em: https://cdn.aaai.org/Workshops/1997/WS-97-04/WS97-04-013.pdf. 12

HOUGH, P. V. Method and means for recognizing complex patterns. 1962. 27

KIM, E. *Distribution Focal Loss (DFL) in YOLO colab*. 2024. Disponível em: https://medium.com/@elvenkim1/dual-focal-loss-dfl-in-yolo-colab-0c9ac722f917. Acesso em: 10 fev 2025. 43

LIU, R.; TANG, Y.; CHAN, P. A fast convex hull algorithm inspired by human visual perception. *Multimedia Tools and Applications*, v. 77, 12 2018. 29

MOHAMED, I. S. *Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques*. Dissertação (Mestrado), 2017. 34, 35

MURRAY, H. J. R. *A History of Chess: The Original 1913 Edition*. [S.l.]: Skyhorse, 2015. ISBN 9781632207708. 12

OPENCV. *Geometric Transformations of Images*. 2024. Disponível em: https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html. Acesso em: 6 agosto 2024. 30

OPENCV. *Hough Line Transform*. 2025. Disponível em: https://docs.opencv.org/4.x/d9/db0/tutorial_hough_lines.html. Acesso em: 19 fev 2025. 28

OVERLEAF. *Chess Notation*. 2024. Disponível em: https://www.overleaf.com/learn/latex/Chess_notation. Acesso em: 24 julho 2024. 21

QUEIROZ, J.; GOMES, H. Introdução ao processamento digital de imagens. *RITA*, v. 13, p. 11–42, 01 2006. 24, 25

REDMON, J. et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. Disponível em: https://arxiv.org/abs/1506.02640. 35, 36

ROSEBROCK, A. *Intersection over Union (IoU) for object detection*. 2016. Disponível em: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. Acesso em: 7 fev 2025. 41

SHAPIRO, L.; STOCKMAN, G. *Computer Vision*. [S.l.]: Prentice Hall, 2001. 23

SZELISKI, R. *Computer Vision: Algorithms and Applications 2nd Edition*. [S.l.]: Springer, 2022. 24, 26, 27, 31

TORRES, J. *What is DFL loss in YOLOv8?* 2024. Disponível em: https://yolov8.org/what-is-dfl-loss-in-yolov8/. Acesso em: 9 fev 2025. 43

ZHANG, X.; SHEN, X.; OUYANG, T. Extension of dbscan in online clustering: An approach based on three-layer granular models. *Applied Sciences*, v. 12, n. 19, 2022. Disponível em: https://www.mdpi.com/2076-3417/12/19/9402. 31

Apêndices

APÊNDICE A – LOGS DA APLICAÇÃO NA LINHA DE COMANDO

Código A.1 – Logs da aplicação na linha de comando

```
(.venv) andrey@pc:~/workspace/tcc/scripts/application$ python3 main.py
loading Roboflow workspace...
loading Roboflow project...
[debug] Loading image: ../test_images/0001.jpg
[debug] requested object detection to Roboflow server
[info] Initial state: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
[debug] Loading image: ../test_images/0002.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: e4
[info] New board state: rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq - 0 1
[debug] Loading image: ../test_images/0003.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: e5
[info] New board state: rnbqkbnr/pppp1ppp/8/4p3/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 0 2
[debug] Loading image: ../test_images/0004.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nf3
[info] New board state: rnbqkbnr/pppp1ppp/8/4p3/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2
[debug] Loading image: ../test_images/0005.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: d6
[info] New board state: rnbqkbnr/ppp2ppp/3p4/4p3/4P3/5N2/PPPP1PPP/RNBQKB1R w KQkq - 0 3
[debug] Loading image: ../test_images/0006.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: d4
[info] New board state: rnbqkbnr/ppp2ppp/3p4/4p3/3PP3/5N2/PPP2PPP/RNBQKB1R b KQkq - 0 3
[debug] Loading image: ../test_images/0007.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Bg4
[info] New board state: rn1qkbnr/ppp2ppp/3p4/4p3/3PP1b1/5N2/PPP2PPP/RNBQKB1R w KQkq - 1
    4
[debug] Loading image: ../test_images/0008.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: dxe5
[info] New board state: rn1qkbnr/ppp2ppp/3p4/4P3/4P1b1/5N2/PPP2PPP/RNBQKB1R b KQkq - 0 4
[debug] Loading image: ../test_images/0009.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Bxf3
[info] New board state: rn1qkbnr/ppp2ppp/3p4/4P3/4P3/5b2/PPP2PPP/RNBQKB1R w KQkq - 0 5
[debug] Loading image: ../test_images/0010.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Qxf3
[info] New board state: rn1qkbnr/ppp2ppp/3p4/4P3/4P3/5Q2/PPP2PPP/RNB1KB1R b KQkq - 0 5
[debug] Loading image: ../test_images/0011.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: dxe5
```

```
[info] New board state: rn1qkbnr/ppp2ppp/8/4p3/4P3/5Q2/PPP2PPP/RNB1KB1R w KQkq - 0 6
[debug] Loading image: ../test_images/0012.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Bc4
[info] New board state: rn1qkbnr/ppp2ppp/8/4p3/2B1P3/5Q2/PPP2PPP/RNB1K2R b KQkq - 1 6
[debug] Loading image: ../test_images/0013.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nf6
[info] New board state: rn1qkb1r/ppp2ppp/5n2/4p3/2B1P3/5Q2/PPP2PPP/RNB1K2R w KQkq - 2 7
[debug] Loading image: ../test_images/0014.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Qb3
[info] New board state: rn1qkb1r/ppp2ppp/5n2/4p3/2B1P3/1Q6/PPP2PPP/RNB1K2R b KQkq - 3 7
[debug] Loading image: ../test_images/0015.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Qe7
[info] New board state: rn2kb1r/ppp1qppp/5n2/4p3/2B1P3/1Q6/PPP2PPP/RNB1K2R w KQkq - 4 8
[debug] Loading image: ../test_images/0016.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nc3
[info] New board state: rn2kb1r/ppp1qppp/5n2/4p3/2B1P3/1QN5/PPP2PPP/R1B1K2R b KQkq - 5 8
[debug] Loading image: ../test_images/0017.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: c6
[info] New board state: rn2kb1r/pp2qppp/2p2n2/4p3/2B1P3/1QN5/PPP2PPP/R1B1K2R w KQkq - 0 9
[debug] Loading image: ../test_images/0018.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Bg5
[info] New board state: rn2kb1r/pp2qppp/2p2n2/4p1B1/2B1P3/1QN5/PPP2PPP/R3K2R b KQkq - 1 9
[debug] Loading image: ../test_images/0019.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: b5
[info] New board state: rn2kb1r/p3qppp/2p2n2/1p2p1B1/2B1P3/1QN5/PPP2PPP/R3K2R w KQkq - 0
    10
[debug] Loading image: ../test_images/0020.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nxb5
[info] New board state: rn2kb1r/p3qppp/2p2n2/1N2p1B1/2B1P3/1Q6/PPP2PPP/R3K2R b KQkq - 0
    10
[debug] Loading image: ../test_images/0021.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: cxb5
[info] New board state: rn2kb1r/p3qppp/5n2/1p2p1B1/2B1P3/1Q6/PPP2PPP/R3K2R w KQkq - 0 11
[debug] Loading image: ../test_images/0022.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Bxb5+
[info] New board state: rn2kb1r/p3qppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/R3K2R b KQkq - 0 11
[debug] Loading image: ../test_images/0023.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nbd7
[info] New board state: r3kb1r/p2nqppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/R3K2R w KQkq - 1 12
[debug] Loading image: ../test_images/0024.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: O-O-O
[info] New board state: r3kb1r/p2nqppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/2KR3R b kq - 2 12
[debug] Loading image: ../test_images/0025.jpg
[debug] requested object detection to Roboflow server
```

```
[info] Detected move: Rd8
[info] New board state: 3rkb1r/p2nqppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/2KR3R w k - 3 13
[debug] Loading image: ../test_images/0026.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Rxd7
[info] New board state: 3rkb1r/p2Rqppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/2K4R b k - 0 13
[debug] Loading image: ../test_images/0027.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Rxd7
[info] New board state: 4kb1r/p2rqppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/2K4R w k - 0 14
[debug] Loading image: ../test_images/0028.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Rd1
[info] New board state: 4kb1r/p2rqppp/5n2/1B2p1B1/4P3/1Q6/PPP2PPP/2KR4 b k - 1 14
[debug] Loading image: ../test_images/0029.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Qe6
[info] New board state: 4kb1r/p2r1ppp/4qn2/1B2p1B1/4P3/1Q6/PPP2PPP/2KR4 w k - 2 15
[debug] Loading image: ../test_images/0030.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Bxd7+
[info] New board state: 4kb1r/p2B1ppp/4qn2/4p1B1/4P3/1Q6/PPP2PPP/2KR4 b k - 0 15
[debug] Loading image: ../test_images/0031.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nxd7
[info] New board state: 4kb1r/p2n1ppp/4q3/4p1B1/4P3/1Q6/PPP2PPP/2KR4 w k - 0 16
[debug] Loading image: ../test_images/0032.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Qb8+
[info] New board state: 1Q2kb1r/p2n1ppp/4q3/4p1B1/4P3/8/PPP2PPP/2KR4 b k - 1 16
[debug] Loading image: ../test_images/0033.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Nxb8
[info] New board state: 1n2kb1r/p4ppp/4q3/4p1B1/4P3/8/PPP2PPP/2KR4 w k - 0 17
[debug] Loading image: ../test_images/0034.jpg
[debug] requested object detection to Roboflow server
[info] Detected move: Rd8#
[info] New board state: 1n1Rkb1r/p4ppp/4q3/4p1B1/4P3/8/PPP2PPP/2K5 b k - 1 17
========================================
======== GAME FINISHED! MOVES: =========
1. e4 e5
2. Nf3 d6
3. d4 Bg4
4. dxe5 Bxf3
5. Qxf3 dxe5
6. Bc4 Nf6
7. Qb3 Qe7
8. Nc3 c6
9. Bg5 b5
10. Nxb5 cxb5
11. Bxb5+ Nbd7
12. O-O-O Rd8
13. Rxd7 Rxd7
14. Rd1 Qe6
15. Bxd7+ Nxd7
16. Qb8+ Nxb8
17. Rd8#
```

```
158 ========================================
```