

Richardson Santiago Teles de Menezes

# **ChessPy: Ferramenta para Detecção Inteligente de Peças de Xadrez**

Natal - RN

Julho de 2022

Richardson Santiago Teles de Menezes

# **ChessPy: Ferramenta para Detecção Inteligente de Peças de Xadrez**

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Helton Maia

Universidade Federal do Rio Grande do Norte - UFRN

Departamento de Engenharia de Computação e Automação - DCA  
Engenharia de Computação

Natal - RN

Julho de 2022

Richardson Santiago Teles de Menezes

## **ChessPy: Ferramenta para Detecção Inteligente de Peças de Xadrez**

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Helton Maia

Trabalho aprovado. Natal - RN, 22 de Julho de 2022:

---

**Prof. Dr. Helton Maia Peixoto - Orientador**  
UFRN

---

**Prof. Dr. Rafael Marrocos Magalhães**  
UFPB

---

**Prof. Dr. Bruno Marques Ferreira da Silva**  
UFRN

---

**Prof. Dr. Francisco José Targino Vidal**  
UFRN

Natal - RN  
Julho de 2022

Universidade Federal do Rio Grande do Norte - UFRN  
Sistema de Bibliotecas - SISBI  
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Menezes, Richardson Santiago Teles de.  
ChessPy: ferramenta para detecção inteligente de peças de xadrez / Richardson Santiago Teles de Menezes. - 2022.  
54f.: il.

Monografia (Graduação) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Curso de Engenharia de Computação, Natal, 2022.

Orientador: Dr. Helton Maia Peixoto.

1. Xadrez - Monografia. 2. Aprendizado de máquina - Monografia. 3. Redes neurais convolucionais - Monografia. 4. Detecção de objetos - Monografia. I. Peixoto, Helton Maia.

RN/UF/BCZM

CDU 004

# AGRADECIMENTOS

À minha família por sempre acreditar em mim e apoiar minhas decisões. Sem vocês nada disso teria sido possível e todas as conquistas que eu obtiver também serão suas.

Aos meus grandes amigos, Luiz Felipe e Maurício, com os quais não apenas dividi um apartamento durante esses anos de graduação como também momentos inesquecíveis.

Ao meu orientador, Helton Maia, pela oportunidade de trabalharmos juntos ao longo da minha formação, pelo conhecimento transmitido e pelas lições que me acompanharão pelo resto da vida.

A todos os amigos que passaram a graduação ao meu lado, assistindo aulas, conversando nos corredores e sofrendo com os trabalhos.

Às incríveis pessoas que tive a oportunidade de conhecer no Laboratório de Automação e Robótica (LAR), onde passei a maior parte dos dias, conversando, produzindo e acima de tudo me divertindo. Todos vocês são incríveis e tem futuros brilhantes pela frente. Meu muitíssimo obrigado ao professor responsável, Orivaldo Vieira de Santana Júnior, pela oportunidade de participar deste laboratório.

*“Esse é só o começo do fim da nossa vida.”*

*- Los Hermanos, 2003*

# RESUMO

O xadrez é um dos domínios mais pesquisados na história da inteligência artificial. Os programas mais poderosos são construídos com base em técnicas de pesquisa sofisticadas, adaptações específicas de domínio e funções de avaliação artesanais refinadas por especialistas humanos ao longo de várias décadas. O principal objetivo deste trabalho consiste na construção de uma plataforma robusta para detecção de posições durante jogos de xadrez, para tanto lançou-se mão de técnicas clássicas de processamento digital de imagens, bem como de algoritmos do estado-da-arte de detecção de objetos. As imagens capturadas durante um jogo do tabuleiro de xadrez são analisadas para determinar a localização de cada quadrado que compõem o tabuleiro bem como a localização de cada peça em jogo, isso então é feito para cada turno, de modo que o sistema é capaz de acompanhar o jogo em sua totalidade.

**Palavras-chaves:** Xadrez, aprendizado de máquina, redes neurais convolucionais, detecção de objetos.

# ABSTRACT

Chess is one of the most researched domains in artificial intelligence history. The most powerful programs are based on sophisticated research techniques, domain-specific adaptations, and handcrafted assessment functions refined over decades by human experts. The main goal of this work is to create a robust platform for position detection during chess games by combining traditional digital image processing techniques with cutting-edge object detection algorithms. The images captured during a chess game are analyzed to determine the location of each square on the board as well as the location of each piece in play; this is then repeated for each turn so that the system can follow the game in its entirety.

**Keywords:** Chess, machine learning, convolutional neural networks, object detection.

# LISTA DE ILUSTRAÇÕES

Figura 1 – Tabuleiro de xadrez eletrônico produzido pela empresa DGT. Fonte: DGT. . . . .	14
Figura 2 – Exemplificação da notação algébrica. (a) Tabuleiro de xadrez com os rótulos empregados na notação algébrica; (b) Processo de identificação de uma casa no tabuleiro utilizando a notação algébrica; (c) Todas as casas do tabuleiro identificadas com a notação algébrica. Fonte: Adaptado de (OVERLEAF, 2022). . . . .	18
Figura 3 – Representação de posições utilizando a notação Forsyth-Edwards. (a) Posição inicial; (b) Posição após 1.e4; (c) Após a resposta das negras com 1...c5; (d) As brancas continuam com 2.Nf3. Fonte: Autor. . . . .	20
Figura 4 – Processo de amostragem dos pontos para construção de um sistema linear e obtenção dos coeficientes da matriz de transformação de perspectiva. Fonte: Autor. . . . .	21
Figura 5 – Processamento realizado para a transformada de Hough. (a) Ilustração dos parâmetros que descrevem uma reta em coordenadas polares; (b) Curva dos parâmetros de todas as retas que passam pelo ponto $(x_0, y_0)$ ; (c) Intersecção dos parâmetros de retas que passam por diferentes pontos. Fonte: Autor. . . . .	25
Figura 6 – A rede YOLO modela o problema de detecção de objetos como uma regressão (REDMON et al., 2015). Assim, ele divide a imagem de entrada em uma grade $S \times S$ e para cada célula da grade prevê $B$ caixas delimitadoras, pontuações de confiança para essas caixas e probabilidades de classe $C$ . Essas previsões são codificadas como um tensor de dimensão $S \times S \times (B * 5 + C)$ . Fonte: Adaptado de (REDMON et al., 2015). . . . .	29
Figura 7 – Diagrama de atividades UML do sistema desenvolvido para detecção e classificação de peças de xadrez. Fonte: Autor. . . . .	30
Figura 8 – Configuração experimental. (a) Representação 3D da configuração experimental para gravação de partidas de xadrez, bem como a posição da câmera para aquisição de vídeo; (b) <i>Frame</i> de exemplo mostrando a vista superior do tabuleiro de xadrez gravada pela câmera durante as partidas que compõe o conjunto de dados. Fonte: Autor. . . . .	32
Figura 9 – Exemplo de cálculo da métrica de <i>Intersection Over Union</i> para exemplos de caixas delimitadoras. Fonte: Autor. . . . .	34

Figura 10 – Exemplo de frame utilizado para realizar o posicionamento da câmera, tabuleiro e peças. Fonte: Autor. . . . .	36
Figura 11 – Processo de seleção do tabuleiro. (a) Seleção manual dos quatro cantos do tabuleiro, que deve ser iniciada no canto superior esquerdo e seguir o sentido horário; (b) Resultado do processo de eliminação dos arredores do tabuleiro que fornece apenas a região que contém os objetos de interesse. Fonte: Autor. . . . .	38
Figura 12 – Aplicação da transformação de perspectiva na região de interesse. (a) Região de interesse selecionada pelo usuário; (b) Imagem resultante da aplicação da transformação de perspectiva. Fonte: Autor. . . . .	39
Figura 13 – Aplicação do algoritmo de detecção de bordas. (a) Imagem resultante da transformação de perspectiva aplicada a região de interesse selecionada; (b) Resultado da aplicação da metodologia de detecção de bordas proposta. Fonte: Autor. . . . .	40
Figura 14 – Aplicação da transformada de Hough. (a) Imagem resultante da aplicação da metodologia de detecção de borda; (b) Todas as linhas detectadas pela transformada de Hough na imagem de entrada; (c) Vértices dos quadrados que compõem as casas do tabuleiro resultante das intersecções das linhas detectadas; (d) Mapeamento da localização de todas as casas do tabuleiro de acordo com sua notação algébrica. Fonte: Autor. . . . .	41
Figura 15 – Exemplos da saída do modelo de detecção de objeto treinado. Fonte: Autor. . . . .	43
Figura 16 – Visualização da saída das camadas convolucionais. (a)-(c) Referem-se as saídas de camadas convolucionais iniciais do extrator de características; (d)-(i) São as representações de saídas das camadas intermediárias do modelo; (j)-(l) Ilustram as saídas das camadas finais do extrator de características. Fonte: Autor. . . . .	44
Figura 17 – Minimização da função de perda durante o processo de treinamento das arquiteturas <i>YOLO Full</i> e <i>Tiny</i> . Fonte: Autor. . . . .	45
Figura 18 – O tempo de processamento em GPU utilizado pelos modelos. (a) Tempo necessário para treinamento de ambos os modelos por 8000 épocas; (b) A quantidade de tempo requerida para classificação de uma única imagem. Fonte: Autor. . . . .	48
Figura 19 – Interface desenvolvida. (a) Tela inicial da interface, aguardando o usuário carregar um vídeo para análise; (b) Interface durante a análise do vídeo de uma partida. Fonte: Autor. . . . .	49

# LISTA DE TABELAS

Tabela 1 – Nomes e letras representativas para cada uma das peças em um jogo de xadrez.	18
Tabela 2 – Conjunto de dados de imagens construído para treinamento e teste do algoritmo de detecção de objetos.	32
Tabela 3 – Métricas de classificação no conjunto de dados de teste para os modelos avaliados.	45
Tabela 4 – Precisão média para as detecções de cada classe para os modelos avaliados.	46
Tabela 5 – Matriz de confusão para os modelos analisados.	47

# LISTA DE ABREVIATURAS E SIGLAS

GPU	<i>Graphics Processing Unit</i>
SVM	<i>Support Vector Machine</i>
CNN	<i>Convolutional Neural Network</i>
FEN	<i>Forsyth-Edwards Notation</i>
YOLO	<i>You Only Look Once</i>
UML	<i>Unified Modeling Language</i>
AP	<i>Average Precision</i>
mAP	<i>Mean Average Precision</i>
IoU	<i>Intersection Over Union</i>
GUI	<i>Graphical User Interface</i>

# SUMÁRIO

1	INTRODUÇÃO . . . . .	12
1.1	Motivação . . . . .	13
1.2	Objetivos . . . . .	14
1.3	Estrutura do Trabalho . . . . .	15
2	FUNDAMENTAÇÃO TEÓRICA . . . . .	16
2.1	Xadrez . . . . .	16
2.1.1	Notação Algébrica . . . . .	17
2.1.2	Notação das Peças . . . . .	18
2.1.3	Notação Forsyth-Edwards . . . . .	19
2.2	Técnicas de Processamento de Imagens . . . . .	20
2.2.1	Transformação de Perspectiva . . . . .	20
2.2.2	Detecção de Bordas . . . . .	22
2.2.3	Morfologia Matemática . . . . .	23
2.2.4	Transformada de Hough . . . . .	24
2.3	Redes Neurais Convolucionais . . . . .	26
2.4	You Only Look Once . . . . .	27
3	METODOLOGIA . . . . .	30
3.1	Desenvolvimento Computacional . . . . .	30
3.2	Conjunto de dados . . . . .	31
3.3	Métricas . . . . .	33
3.3.1	Average Precision . . . . .	33
3.3.2	Intersection Over Union . . . . .	34
4	RESULTADOS . . . . .	36
4.1	Seleção do Tabuleiro . . . . .	37
4.2	Detecção das Casas do Tabuleiro . . . . .	39
4.3	Detecção das Peças . . . . .	42
4.4	Interface Gráfica . . . . .	48
5	CONCLUSÃO . . . . .	51
	REFERÊNCIAS . . . . .	52

# 1 INTRODUÇÃO

A área de visão computacional engloba alguns desafios fascinantes, como classificar imagens e identificar objetos, ambos sob o guarda-chuva do reconhecimento de objetos. Nos últimos anos, houve um desenvolvimento científico significativo nessas áreas, devido principalmente aos avanços em redes neurais convolucionais, técnicas de aprendizado profundo e aumento do poder de processamento paralelo fornecido pelas unidades de processamento gráfico (GPUs, do acrônimo em inglês para *graphics processing units*).

O problema de classificação de imagens envolve atribuir um rótulo de um conjunto fixo de categorias a uma imagem de entrada. Esse é um dos principais problemas da visão computacional porque, apesar de sua simplicidade, possui uma ampla gama de aplicações práticas e múltiplos usos, como rotular imagens de câncer de pele (ESTEVA et al., 2017), usar imagens de alta resolução para detectar desastres naturais como enchentes, vulcões e secas severas rastreando os impactos causados (JAYARAMAN; CHANDRASEKHAR; RAO, 1997; LEONARD et al., 2014; KOGAN, 1997).

As características, comumente referenciadas pelo termo inglês *features*, utilizadas para alimentar os algoritmos de classificação de imagens têm um impacto significativo em seu desempenho (SRINIVAS et al., 2017). Isso significa que o avanço das técnicas para classificação de imagens, baseadas em aprendizado de máquina tem sido fortemente dependente da engenharia de seleção de *features* essenciais das imagens que compõem a base de dados. Como resultado, obter essas *features* tornou-se uma tarefa árdua, aumentando a complexidade e o custo computacional das abordagens. Para classificação de imagens, extração de *features* e seleção de algoritmo e aprendizado, classicamente demanda duas etapas independentes, e isso foi amplamente desenvolvido e aprimorado utilizando máquinas de vetor de suporte (SVMs, do acrônimo em inglês para *support vector machines*).

Quando considerada a abordagem de aprendizado supervisionado, o algoritmo SVM é frequentemente usado para tarefas como classificação, regressão e detecção de *outliers* (MENEZES et al., 2018). O aspecto mais atraente desse algoritmo é que seu mecanismo de aprendizado para várias classes de interesse é mais fácil de ser matematicamente analisado do que as soluções providas por arquiteturas com redes neurais tradicionais, permitindo que alterações complexas com efeitos conhecidos sejam feitas nas principais características do algoritmo (OSKOEI; GAN; HU, 2009). Em suma, uma SVM mapeia os dados de treinamento para um espaço vetorial de dimensão mais alta e constrói um hiperplano ótimo de separação, resultando em um limite de separação não-linear no espaço de entrada (HEARST et al., 1998).

Arquiteturas de aprendizado profundo com camadas especializadas para automati-

zar o processo de filtragem e extração de *features* agora são usadas pelos algoritmos mais robustos de classificação e detecção de objetos. Algoritmos de aprendizado de máquina, como regressão linear, máquinas de vetor de suporte e árvores de decisão, têm processos de aprendizado particulares, porém todos seguem as mesmas etapas básicas: fazer uma predição, receber uma correção e ajustar o mecanismo de precisão com base no retorno recebido, processo esse que se assemelha à forma como os humanos aprendem. O aprendizado profundo introduziu uma abordagem revolucionária para o problema, com o objetivo de superar deficiências anteriores por meio do aprendizado de abstração dos dados utilizando um paradigma de descrição estratificada baseado em uma transformação não linear (PAN; DONG; WU, 2018). A capacidade do aprendizado profundo de aprender a extração de *features* em grandes conjuntos de dados é a vantagem que trouxe essa abordagem para os holofotes.

A utilização extensiva de redes neurais convolucionais (ConvNet ou CNN, ambas acrônimos para expressão inglesa *convolutional neural networks*) fornecem a capacidade de aprender a etapa de extração de *features* presente em algoritmos de aprendizado profundo. Nesse contexto, a convolução é um tipo especializado de operação linear que pode ser pensada como a aplicação de um filtro a uma determinada entrada. A repetida aplicação de filtros a uma entrada resulta em um mapa de características, que pode indicar as localizações e a força de uma *feature* detectada na entrada por meio do ajuste dos parâmetros do filtro convolucionar. A rede pode se ajustar para reduzir o erro e assim aprender os melhores parâmetros para extrair informações relevantes da base de dados. Diversos detectores de objetos baseados em redes neurais profundas foram propostos nos últimos anos (DENG; HINTON; KINGSBURY, 2013; KRIEGESKORTE, 2015).

## 1.1 Motivação

O objetivo deste trabalho foi desenvolver uma aplicação capaz de determinar as posições de cada peça em um tabuleiro de xadrez físico, permitindo que um computador grave automaticamente um jogo de xadrez anotando as posições das peças. Uma imagem do tabuleiro de xadrez é analisada para determinar a localização de cada casa do tabuleiro e peça. Isso é repetido a cada turno, permitindo que o sistema acompanhe todo o jogo.

Existem no mercado tabuleiros e peças de xadrez desenvolvidas com sensores especializados para detecção e transmissão de jogos de xadrez para o computador, com a empresa *Digital Game Technology* (DGT) sendo uma empresa proeminente no ramo, produzindo tabuleiros eletrônicos de xadrez entre outros produtos. O lançamento do *Live-Chess* pela DGT, um *software* para torneios de xadrez, foi outra contribuição significativa feita pela empresa. Esse *software* torna a transmissão de torneios de xadrez online simples

e acessível aos clubes e organizações menores de xadrez através da utilização do serviço de nuvem gratuito *LiveChessCloud*. A Figura 1 ilustra um modelo de tabuleiro eletrônico produzido pela empresa.



Figura 1 – Tabuleiro de xadrez eletrônico produzido pela empresa DGT. Fonte: DGT.

Embora soluções personalizadas de tabuleiros de xadrez eletrônico sejam comuns em torneios de xadrez de alto nível, elas ainda são proibitivamente caras para organizadores de torneios de médio e pequeno porte principalmente para aqueles localizados em países em desenvolvimento. Esse tipo de solução normalmente custa cerca de US\$ 900, aproximadamente R\$4 805.91 pela cotação de julho de 2022, por tabuleiro.

Assim, esse trabalho fornece uma solução alternativa que pode ser adaptada para funcionar com outros conjuntos de xadrez, podendo também ser utilizada para transmissão ao vivo da posição detectada. Para uso da plataforma proposta, necessita-se apenas de uma câmera de baixa resolução, posicionada acima do tabuleiro para aquisição das imagens que serão processadas.

## 1.2 Objetivos

O principal objetivo deste trabalho consiste na construção de uma plataforma robusta para detecção de posições durante jogos de xadrez, para tanto lançou-se mão de técnicas clássicas de processamento digital de imagens, bem como de algoritmos do estado-da-arte de detecção de objetos.

Para alcançar o objetivo geral proposto, o presente trabalho tem os seguintes objetivos específicos:

- Isolar o tabuleiro de xadrez que será utilizado para o jogo do restante dos objetos presentes no *frame* capturado pela câmera;
- Encontrar dentro da imagem isolada a posição de cada casa que compõe um tabuleiro de xadrez e atribuí-las a sua respectiva notação;
- Detectar todas as peças de xadrez presentes no tabuleiro;
- Combinar a detecção das peças com o mapeamento das casa para construir a posição atual do jogo;

### 1.3 Estrutura do Trabalho

Esse trabalho apresenta uma introdução sobre o tema a ser abordado, mostrando os fatores que motivaram seu desenvolvimento, além da justificativa e seus objetivos principais. Em sequência, o Capítulo 2 apresenta a fundamentação teórica por traz das ferramentas utilizadas no desenvolvimento do projeto. O Capítulo 3, por sua vez, explica a metodologia aplicada para utilização do ferramental teórico para solução do problema abordado. No Capítulo 4 os resultados obtidos com a metodologia proposta são apresentados e analisados. Finalmente, o Capítulo 5 apresentará as principais conclusões e contribuições providas por esse trabalho.

## 2 FUNDAMENTAÇÃO TEÓRICA

### 2.1 Xadrez

O xadrez é um esporte intelectual, jogado entre duas pessoas, ou equipes, que dispõe de forças iguais, tanto em quantidade como em qualidade, denominadas peças que possuem cores diferentes, classicamente brancas e pretas (D'AGOSTINI, 2002). As peças se movimentam segundo regras convencionadas, e o jogo tem como objetivo colocar o Rei adversário em uma posição especial denominada mate, na qual o Rei está sendo atacado e não possui movimentos legais.

Pesquisas computacionais envolvendo xadrez são tão antigas quanto a própria ciência da computação. Para analisar e jogar o jogo de xadrez, Charles Babbage, Alan Turing, Claude Shannon e John von Neumann desenvolveram *hardware*, algoritmos e teorias. O xadrez, posteriormente se tornou o grande desafio para uma nova geração de pesquisadores na área da inteligência artificial, culminando em programas de xadrez que atingiram um nível sobre-humano (SILVER et al., 2017; CAMPBELL; JR; HSU, 2002; HSU, 2002).

Shannon, foi um dos primeiros a publicar um artigo sobre xadrez computacional. Ele deu uma palestra sobre o tema nos Laboratórios Bell em 1949 e publicou seu famoso artigo “*Programming a Computer for Playing Chess*” na *Philosophical Magazine* em 1950 (SHANNON, 1950). Esse artigo descreve três tipos diferentes de estratégias: Para o tipo A, toma proveito da velocidade de cálculos dos computadores para uma abordagem de força bruta; O tipo B, lança mão de uma busca seletiva; e o tipo C, descreve uma possibilidade da máquina apresentar um raciocínio similar aos mestres de xadrez humanos. Um design de funções de avaliação com fatores de peso, uma busca de estabilidade, uma representação de tabuleiro bidimensional e uma consideração filosófica de jogar um jogo razoavelmente brilhante também foram incluídos nesse trabalho seminal.

O computador *Deep Blue*, construído pela IBM com a proposta de dominar o jogo de xadrez, derrotou o campeão mundial de xadrez humano Garry Kasparov em 1997, estabelecendo um marco significativo na inteligência artificial (CAMPBELL; JR; HSU, 2002). Esses programas avaliam posições de xadrez combinando recursos construídos manualmente, com pesos cuidadosamente ajustados por fortes jogadores e programadores humanos visado o emprego de algoritmos de busca de alto desempenho que expandem uma vasta árvore de pesquisa empregando uma infinidade de heurísticas inteligentes e adaptações específicas de domínio.

O xadrez, e especialmente o xadrez computacional, permaneceu assim no centro das

atenções. Apesar da derrota do campeão mundial humano, a competição entre humanos e computadores na comparação de suas forças de jogo continuou.

Técnicas de aprendizado profundo por reforço, foram aplicadas ao xadrez no outono de 2017. A empresa DeepMind criou o programa AlphaZero (SILVER et al., 2017) que começou apenas conhecendo os movimentos das peças e as regras do jogo, e foi treinando utilizando as técnicas mais recentes de aprendizado por reforço, jogando contra versões anteriores do próprio programa para que sem dados de jogos jogados por humanos pudesse aprender a dominar o jogo.

A equipe da DeepMind decidiu que, após treinamento suficiente, o programa era forte o suficiente para competir com o programa de xadrez mais forte do mundo até o momento. Stockfish (ROMSTAD et al., 2022) foi escolhido pela equipe dado sua classificação de Elo em 2017 estimada como superior a 3400, em comparação, o elo humano mais alto da época era 2837. O Elo é um método estatístico utilizado para se calcular a força relativa entre jogadores de xadrez.

Foi acordado então um embate de 100 partidas entre os programas AlphaZero e Stockfish. No decorrer do evento a nova abordagem do time da DeepMind se mostrou superior vencendo de forma convincente o evento com 28 vitórias, 0 derrotas, e 72 empates, uma vitória por 64-36, em termos de xadrez, a favor do desafiante AlphaZero.

### 2.1.1 Notação Algébrica

Um tabuleiro de xadrez é uma grade de  $8 \times 8$  quadrados que pode ser descrita como 8 linhas horizontais e 8 colunas verticais. Existem várias convenções para identificar cada quadrado em um tabuleiro de xadrez, mas a notação algébrica é o padrão internacional aceito atualmente.

As linhas e colunas no sistema de notação algébrica são rotulados da seguinte forma: as 8 linhas são rotuladas com números de 1 a 8, e as 8 colunas são rotuladas com letras de *a* a *h*. A Figura 2(a) mostra um tabuleiro de xadrez com os rótulos da notação algébrica representando as linhas e colunas.

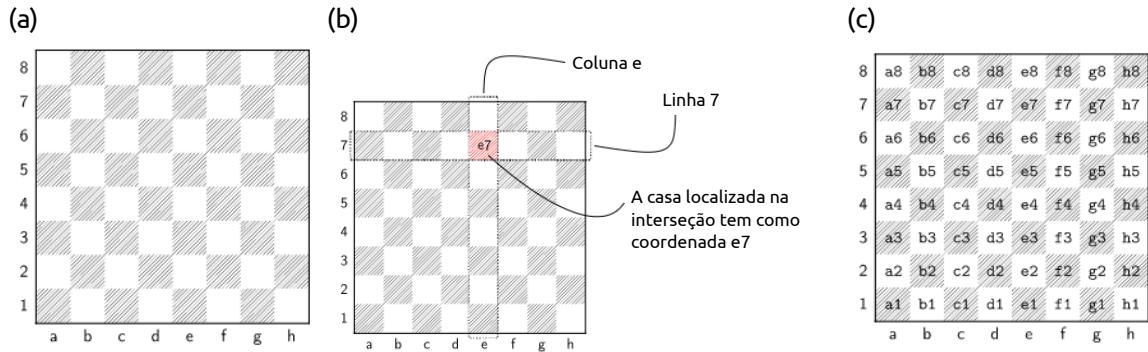


Figura 2 – Exemplificação da notação algébrica. (a) Tabuleiro de xadrez com os rótulos empregados na notação algébrica; (b) Processo de identificação de uma casa no tabuleiro utilizando a notação algébrica; (c) Todas as casas do tabuleiro identificadas com a notação algébrica. Fonte: Adaptado de (OVERLEAF, 2022).

Como cada casa no tabuleiro de xadrez está na interseção de uma coluna e linha específica, os valores correspondentes podem ser usados para identificar exclusivamente cada casa para uso na descrição da localização e movimentação das peças de xadrez.

A Figura 2(b) mostra o processo de identificação da casa em e7 utilizando a notação algébrica, enquanto a Figura 2(c) mostra um tabuleiro de xadrez que lista a localização de todas as 64 casas.

### 2.1.2 Notação das Peças

Cada tipo de peça de xadrez é identificada por uma letra maiúscula, que varia de acordo com o idioma utilizado. Para fins de padronização com outras plataformas de xadrez, esse trabalho empregará a notação inglesa, que utiliza as letras descritas na Tabela 1.

Tabela 1 – Nomes e letras representativas para cada uma das peças em um jogo de xadrez.

Nome Português	Nome Inglês	Letra Representativa	Símbolo Representativo
Rei	King	K	♔ ♚
Rainha ou Dama	Queen	Q	♕ ♛
Torre	Rook	R	♖ ♖
Bispo	Bishop	B	♗ ♘
Cavalo	Knight	N	♘ ♞
Peão	Pawn	P	♙ ♜

Os peões na notação algébrica padrão são distinguidos pela ausência de uma letra

maiúscula e não pela sua presença, no entanto, essa convenção nem sempre é seguida com a letra maiúscula P podendo ser usada para identificar os peões. Letras maiúsculas e minúsculas são usadas para diferenciar as cores de cada peça na notação Forsyth-Edwards (FEN).

### 2.1.3 Notação Forsyth-Edwards

As posições em um jogo de xadrez podem ser anotada de diversas maneiras, dentre elas se destaca a notação Forsyth-Edwards. Esta notação denota uma posição de xadrez como uma string ASCII de uma linha. O FEN é baseado em um sistema desenvolvido no século XIX pelo escocês David Forsyth. Como parte da *Portable Game Notation*, Steven Edwards especificou o padrão FEN para aplicações de xadrez que utilizam computadores (EDWARDS et al., 1994).

Essa notação consiste em se registrar as peças existentes em cada horizontal, iniciando-se na primeira horizontal do lado preto a partir da casa angular branca (casa a8) em direção a última horizontal preta, ou a primeira das brancas. As casas vazias são representadas por um número, as peças brancas por sua inicial maiúscula e as peças pretas por sua inicial minúscula com as horizontais sendo separadas por uma barra.

A Figura 3 ilustra exemplos de posições descritas por suas respectivas especificações FEN após alguns movimentos da defesa siciliana. A Figura 3(a) mostra a descrição da posição inicial onde apenas quatro linhas possuem as letras que representam as peças enquanto as outras linhas são representadas pelo número 8 indicando que estão vazias. Já as Figuras 3(b)-(d) mostram como a representação FEN se adapta a descrição de linhas parcialmente preenchida por peças.

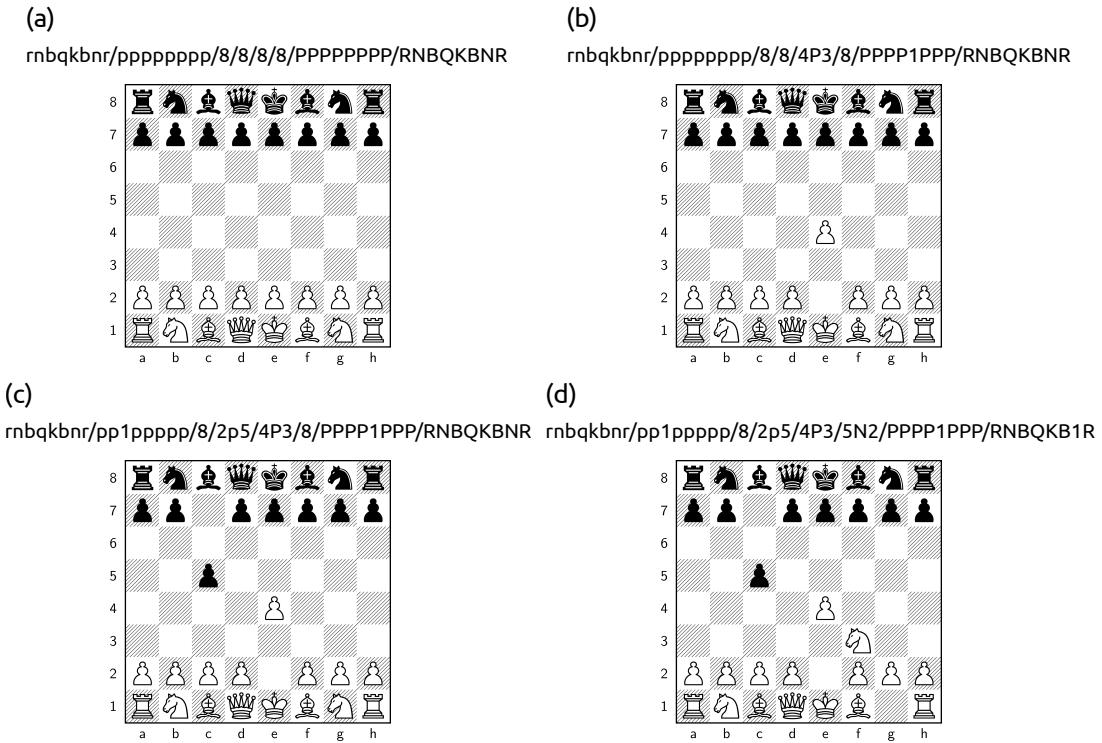


Figura 3 – Representação de posições utilizando a notação Forsyth-Edwards. (a) Posição inicial; (b) Posição após 1.e4; (c) Após a resposta das negras com 1...c5; (d) As brancas continuam com 2.Nf3. Fonte: Autor.

## 2.2 Técnicas de Processamento de Imagens

Essa seção apresenta as técnicas de processamento de imagens utilizadas no desenvolvimento desse trabalho.

### 2.2.1 Transformação de Perspectiva

Uma transformação de perspectiva está associada com uma mudança no ponto de vista de uma imagem. Esse tipo de transformação não preserva paralelismo, comprimento ou ângulo porém a colinearidade e incidência são preservadas, sendo assim, linhas retas permanecerão retas mesmo após a transformação. As transformações de perspectiva podem ser expressas pela Equação 2.1.

$$\begin{bmatrix} t_i x' \\ t_i y' \\ t_i \end{bmatrix} = \underbrace{\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}}_M \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.1)$$

Onde,  $t_i$  é um fator de escala,  $(x', y')$  são as coordenadas do ponto transformadas

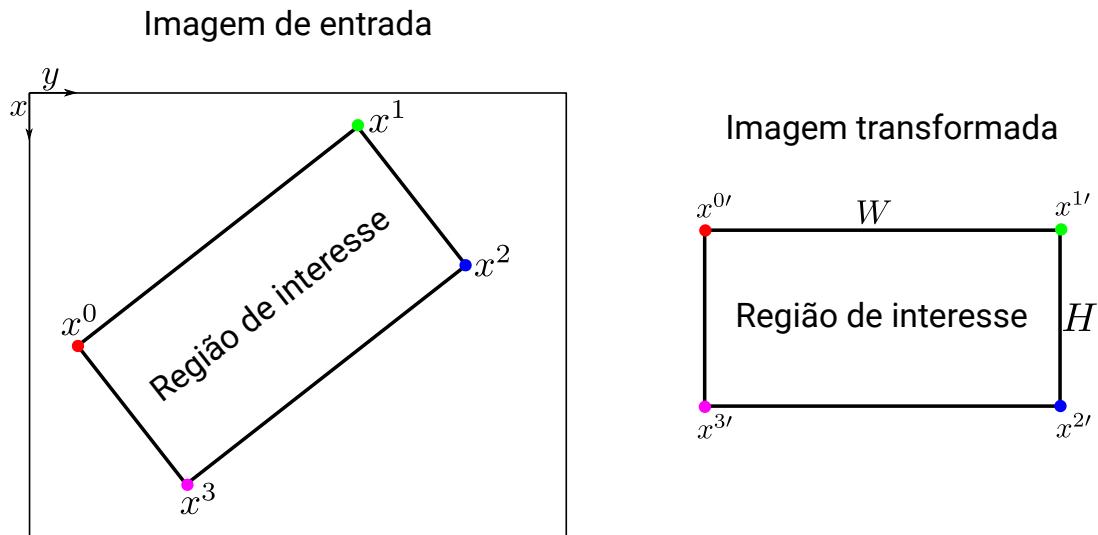


Figura 4 – Processo de amostragem dos pontos para construção de um sistema linear e obtenção dos coeficientes da matriz de transformação de perspectiva. Fonte: Autor.

e  $(x, y)$  as coordenadas originais de entrada. A matriz de transformação  $M$  pode ser vista como a seguinte combinação:

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \rightarrow \text{Define as transformações afins como rotação, escala, etc.}$$

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \rightarrow \text{Define o vetor de translação.}$$

$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} \rightarrow \text{Define o vetor de projeção.}$$

Para definir as 8 constantes que constituem a matriz de transformação  $M$  é necessário selecionar 4 pontos na imagem de entrada e fornecer o mapeamento desejado desses pontos na imagem de saída, assim um sistema linear de 8 equações com 8 incógnitas pode ser construída e resolvida através de diversos métodos clássicos já estabelecidos na literatura (ANTON; RORRES, 2013). A Figura 4 ilustra o processo de amostragem e mapeando dos pontos.

Com os pontos amostrados o sistema linear mostrado na Equação 2.2 pode ser construído e resolvido utilizando um dos diversos métodos para solução de sistemas do tipo  $A \cdot x = b$ .

$$\begin{bmatrix} x_x^0 & x_y^0 & 1 & 0 & 0 & 0 & -x_x^0 \cdot x_x^{0'} & -x_y^0 \cdot x_x^{0'} \\ x_x^1 & x_y^1 & 1 & 0 & 0 & 0 & -x_x^1 \cdot x_x^{1'} & -x_y^1 \cdot x_x^{1'} \\ x_x^2 & x_y^2 & 1 & 0 & 0 & 0 & -x_x^2 \cdot x_x^{2'} & -x_y^2 \cdot x_x^{2'} \\ x_x^3 & x_y^3 & 1 & 0 & 0 & 0 & -x_x^3 \cdot x_x^{3'} & -x_y^3 \cdot x_x^{3'} \\ 0 & 0 & 0 & x_x^0 & x_y^0 & 1 & -x_x^0 \cdot x_y^{0'} & -x_y^0 \cdot x_y^{0'} \\ 0 & 0 & 0 & x_x^1 & x_y^1 & 1 & -x_x^1 \cdot x_y^{1'} & -x_y^1 \cdot x_y^{1'} \\ 0 & 0 & 0 & x_x^2 & x_y^2 & 1 & -x_x^2 \cdot x_y^{2'} & -x_y^2 \cdot x_y^{2'} \\ 0 & 0 & 0 & x_x^3 & x_y^3 & 1 & -x_x^3 \cdot x_y^{3'} & -x_y^3 \cdot x_y^{3'} \end{bmatrix} \cdot \begin{bmatrix} M_{00} \\ M_{01} \\ M_{02} \\ M_{10} \\ M_{11} \\ M_{12} \\ M_{20} \\ M_{21} \end{bmatrix} = \begin{bmatrix} x_x^{0'} \\ x_x^{1'} \\ x_x^{2'} \\ x_x^{3'} \\ x_y^{0'} \\ x_y^{1'} \\ x_y^{2'} \\ x_y^{3'} \end{bmatrix} \quad (2.2)$$

A solução desse sistema fornece os coeficientes necessários para a construção da matriz de transformação de perspectiva  $M$ .

## 2.2.2 Detecção de Bordas

As informações estruturais e de forma reduzem drasticamente a quantidade de dados a serem processados. A extração de descritores com base na detecção de bordas, como o algoritmo de Canny, é uma alternativa de grande proeminência na literatura (SALMAN et al., 2017).

O método de processamento de imagens *Canny Edge Detection* (CANNY, 1986) pode ser aplicado para detectar bordas em uma imagem enquanto suprime ruídos. Para suspensão de ruídos são utilizados filtros Gaussianos, comumente com tamanho de *kernel*  $3 \times 3$ .

A aplicação desse método envolve converter a imagem para tons de cinza, realizar um desfoque gaussiano, determinar os gradientes de intensidade e calcular a magnitude,  $|G|$ , e o ângulo  $\angle G$ , do gradiente de acordo com as Equações 2.3 e 2.4, respectivamente (CANNY, 1986).

$$|G| = \sqrt{g_x^2 + g_y^2} \quad (2.3)$$

$$\angle G = \arctan(g_y/g_x) \quad (2.4)$$

Onde,  $g_x$  e  $g_y$  são as derivadas no ponto considerado. O método também realiza uma supressão não máxima para diminuir as bordas produzidas pela magnitude da imagem

(CANNY, 1986). Além disso, pode-se utilizar uma limiarização adaptativa com histerese para eliminar as listras dos contornos das bordas.

### 2.2.3 Morfologia Matemática

Na natureza, o termo morfologia refere-se ao estudo da forma e estrutura de plantas e animais. A morfologia matemática se refere ao conjunto de ferramentas matemáticas baseadas na teoria dos conjuntos através da qual as formas e estruturas em imagens digitais são estudadas (HARALICK; STERNBERG; ZHUANG, 1987; ZANA; KLEIN, 2001; TRAHANIAS, 1993; YU-QIAN et al., 2006).

Em imagens binárias, os conjuntos são membros do espaço  $\mathbb{Z}^2$  onde as coordenadas são a posição  $(x, y)$  de cada *pixel* na imagem. Para imagens em tons de cinza os conjuntos são membros do espaço  $\mathbb{Z}^3$ , onde além das coordenadas dos *pixels* uma dimensão extra precisa ser adicionada para dar codificar a cor do *pixel* em tom de cinza.

As operações em morfologia matemática são baseadas em pequenos conjuntos chamados elementos estruturantes que são sub-imagens usadas para pesquisar em uma imagem por propriedades de interesse. Algumas operações de conjunto podem ser realizadas sobre uma imagem de entrada com esses elementos, sendo as principais a erosão e a dilatação.

A erosão é a operação morfológica pela qual a fronteira de um conjunto é desgastada. A Equação 2.5 define formalmente a erosão do conjunto  $A$ , pertencente ao espaço Euclidiano  $E$ , pelo elemento estruturante  $B$ .

$$A \ominus B = \{ z \mid B_z \subseteq A \} \quad (2.5)$$

onde,  $B_z$  é a translação do elemento  $B$  pelo vetor  $z$ , ou seja,  $B_z = \{ b + z \mid b \in B \}, \forall z \in E$ . Por exemplo, a erosão de um quadrado de lado 10 centrado na origem, por um disco de raio 2, também centrado na origem, produz um quadrado de lado 6 centrado na origem.

Por outro lado, a dilatação é a operação morfológica pela qual a fronteira de um conjunto é expandida, formalmente definida pela Equação 2.6.

$$A \oplus B = \{ z \mid \hat{B}_z \cap A \neq \emptyset \} \quad (2.6)$$

onde,  $\hat{B}$  denota o simétrico de  $B$ , isto é,  $\hat{B} = \{ x \in E \mid -x \in B \}$  e  $\hat{B}_z$  é a translação do elemento pelo vetor  $z$ . A dilatação de um quadrado de lado 10, centrado na origem, por um disco de raio 2, também centrado na origem, é um quadrado de lado 14, com cantos arredondados, centrado na origem. Onde o raio dos cantos arredondados é 2.

Através da combinação dessas operações elementais, podem ser definidas duas operações de grande utilidade, nomeadamente a abertura e o fechamento.

A operação de abertura é utilizada para suavizar contornos, romper canais e eliminar pequenas saliências. A principal aplicação da operação de abertura é remover pequenas regiões de erro causado por ruídos no processo de segmentação. Esta operação é definida pela Equação 2.7.

$$A \circ B = (A \ominus B) \oplus B \quad (2.7)$$

A operação de fechamento é usada para suavizar contornos, mas tende a mesclar descontinuidades estreitas, eliminar pequenos orifícios e preencher lacunas em um contorno. A operação é definida pela Equação 2.8.

$$A \cdot B = (A \oplus B) \ominus B \quad (2.8)$$

## 2.2.4 Transformada de Hough

Embora a aproximação de curvas por polinômios muitas vezes possa levar a extração bem-sucedida de linhas, as linhas no mundo real às vezes são divididas em componentes desconectados ou compostas por múltiplos segmentos colineares. Em muitos casos, é desejável agrupar tais segmentos colineares em linhas estendidas que englobem todos os segmentos.

A transformada de Hough, nomeada assim em homenagem ao seu criador (HOUGH, 1962), é uma técnica bem conhecida por ter arestas “votando” para localizações de linhas plausíveis (DUDA; HART, 1972; BALLARD, 1981; ILLINGWORTH; KITTLER, 1988). Em sua formulação original, cada ponto de borda vota para todas as linhas possíveis que passam por ele, e as linhas correspondentes a valores altos de acumulador ou *bin* são examinadas como possíveis ajustes de linha.

Uma linha pode ser expressa de duas maneiras usando duas variáveis. No sistema de coordenadas cartesianas, usando os parâmetros  $m$  e  $b$ , ou no sistema de coordenadas polares, usando a distância do ponto de origem  $r$  e o ângulo com o eixo das abscissas  $\theta$  como parâmetros como mostrado na Figura 5(a).

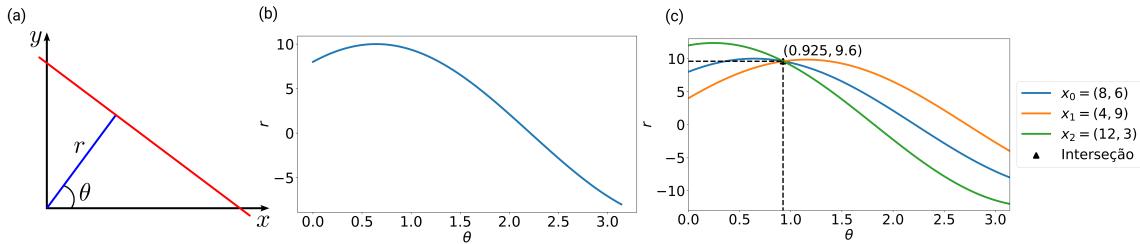


Figura 5 – Processamento realizado para a transformada de Hough. (a) Ilustração dos parâmetros que descrevem uma reta em coordenadas polares; (b) Curva dos parâmetros de todas as retas que passam pelo ponto  $(x_0, y_0)$ ; (c) Intersecção dos parâmetros de retas que passam por diferentes pontos. Fonte: Autor.

Para os cálculos da transformada de Hough as linhas são expressas no sistema Polar de coordenadas. Como resultado, uma equação de linha pode ser escrita como mostrado na Equação 2.9.

$$y = -\frac{\cos \theta}{\sin \theta}x + \frac{r}{\sin \theta} \quad \rightarrow \quad r = x \cos \theta + y \sin \theta \quad (2.9)$$

Em geral, para cada ponto  $(x_0, y_0)$ , pode-se definir a família de retas que passa por aquele ponto como:

$$r_\theta = x_0 \cos \theta + y_0 \sin \theta$$

De tal sorte que cada par  $(r_\theta, \theta)$  representa uma linha que passa por  $(x_0, y_0)$ . Se para um dado  $(x_0, y_0)$  for traçado a família de linhas que o atravessa, obterá-se uma senoide. Por exemplo, para  $x_0 = 8$  e  $y_0 = 6$  o gráfico no plano  $\theta - r$  mostrado na Figura 5(b) pode ser obtido considerando  $r > 0$  e  $0 \leq \theta \leq \pi$ .

O mesmo traçado pode ser feito para múltiplos pontos de uma imagem. Se as curvas de dois pontos distintos se cruzam no plano  $\theta - r$ , isso significará que ambos os pontos pertencem à mesma linha. O traçado da família de linhas para três pontos pode ser visto na Figura 5(c), todos os traçados se intersectam no ponto  $(0.925, 9.6)$  de forma que essas coordenadas são os parâmetros  $(\theta, r)$  da linha em que  $(x_0, y_0)$ ,  $(x_1, y_1)$  e  $(x_2, y_2)$  se encontram.

Em geral, uma linha pode ser detectada calculando o número de interseções entre as curvas. Quanto mais curvas se cruzarem, mais pontos existem na linha representada por essa intersecção. Pode-se definir um número mínimo de interseções necessárias para detectar uma linha.

A Transformada Hough realiza o processo de detecção de linhas mantendo o controle de onde as curvas de cada ponto na imagem se cruzam. Se o número de interseções

exceder um determinado limite, ele é declarado como uma linha com os parâmetros do ponto de interseção  $(\theta, r)$ .

## 2.3 Redes Neurais Convolucionais

O conceito fundamental introduzido pelas redes convolucionais (CNNs, acrônimo do inglês *Convolutional Neural Networks*) é utilizar uma cascata de camadas convolucionais para reduzir a dimensão espacial de uma imagem de entrada e combinar padrões locais para gerar *features* que se tornam mais abstratas à medida que os dados progridem na rede. Essa cascata é chamada de codificador, pois os *pixels* de entrada são codificados em *features* mais abstratas (MENEZES; MAGALHAES; MAIA, 2019).

Nesse trabalho,  $f_i^{in}$  denota o  $i$ -ésimo mapa de *features* de entrada,  $f_j^{out}$  denota o  $j$ -ésimo mapa de *features* de saída, e  $b_j$  denota o  $j$ -ésimo termo de *bias* para o mapa de saída. Para a camada de convolução,  $n_{in}$  e  $n_{out}$  representam o número de *features* de entrada e saída, respectivamente. Para a camada totalmente conectada,  $n_{in}$  e  $n_{out}$  são o comprimento do vetor de *features* de entrada e saída.

A camada convolucional recebe uma série de mapas de *features* como entrada e os convolui com *kernels* convolucionais para obter os mapas de recursos de saída. Uma camada que aplica uma função de ativação não linear a cada elemento nos mapas de *features* de saída geralmente é anexada às camadas convolucionais. Uma camada convolucional pode ser expressa formalmente através da Equação 2.10.

$$f_i^{out} = \sum_{j=1}^{n_{in}} f_j^{in} \otimes g_{i,j} + b_j \quad (1 \leq i \leq n_{out}), \quad (2.10)$$

Onde,  $g_{i,j}$  é o *kernel* convolucional aplicado ao  $j$ -ésimo mapa de *features* de entrada para formar o  $i$ -ésimo mapa de *features* de saída.

Os mapas de *features* resultantes das etapas convolucionais possuem um problema, eles são sensíveis à localização das *features* na imagem de entrada. Uma abordagem proposta para lidar com essa sensibilidade é reduzir a resolução do mapa de *features*. Isso resultará em mapas amostrados mais resistentes a mudanças na posição da *feature* na imagem de entrada, o que é conhecido por “invariância de tradução local”. Uma camada de *pooling* pode ser utilizada para atingir essa invariância.

A camada de *pooling* também conhecida como camada de sub-amostragem, que recebe um filtro  $n \times n$  e um passo de comprimento  $n$ , para então aplicá-lo ao vetor de entrada e produzir os valores máximos ou médios de cada subárea. A ideia dessa camada é que uma vez que sabemos que uma *feature* específica é uma entrada original, sua localização exata não é tão crucial quanto sua localização relativa a outras *features*, tal

invariância pode ser fornecida por essa camada como visto em (SCHERER; MÜLLER; BEHNKE, 2010; ZHOU et al., 2016). O *Max-pooling* pode ser expresso como visto na Equação 2.11.

$$f_{i,j}^{out} = \max_{p \times p} \begin{pmatrix} f_{m,n}^{in} & \cdots & f_{m,n+p-1}^{in} \\ \vdots & & \vdots \\ f_{m+p-1,n}^{in} & \cdots & f_{m+p-1,n+p-1}^{in} \end{pmatrix}, \quad (2.11)$$

Onde,  $p$  é o tamanho do *kernel* de *pooling*. Essa sub-amostragem não-linear não apenas reduz o tamanho do mapa de *features* que reduzirá as computações necessárias nas próximas camadas, como também fornece uma forma de invariância a translações.

Em uma CNN, o codificador é frequentemente seguido por camadas totalmente conectadas, como em uma configuração clássica de *multi-layer perceptron*. Essas camadas aplicam a transformação linear ao vetor de *features* de entrada mostrada na Equação 2.12.

$$f^{out} = W \cdot f^{in} + b, \quad (2.12)$$

Onde,  $f^{in}$  é o vetor de *features* de entrada resultante de convoluções anteriores,  $W$  é uma matriz de transformação  $n_{out} \times n_{in}$ ,  $b$  é o termo de *bias*, e  $f^{out}$  é a saída com as probabilidades das classes.

Essas camadas recebem *features* abstratas como entrada, processando-as com a passagem pelas camadas da rede e gera previsões estatísticas globais sobre a presença ou ausência de objetos de interesse na imagem de entrada (CIRESAN et al., 2011).

## 2.4 You Only Look Once

*You Only Look Once* (YOLO) (REDMON; FARHADI, 2018) é um algoritmo do estado-da-arte para detecção de objetos que visa aplicações em tempo real e, ao contrário de alguns concorrentes, não é um classificador tradicional reaproveitado como um detector de objetos.

O YOLO funciona dividindo a imagem de entrada em uma grade de células  $S \times S$ , onde cada célula é responsável por cinco previsões de caixas delimitadores que descrevem o retângulo ao redor de um objeto de interesse. Ele também gera uma pontuação de confiança, que é uma medida de certeza de inclusão do objeto dentro da área retangular proposta, sendo assim, a pontuação não tem relação com o tipo de objeto presente no área, apenas com o formato da caixa.

Para cada caixa delimitadora, também é previsto uma classe funcionando como um classificador regular, resultando em uma distribuição de probabilidade sobre todas as classes possíveis. A pontuação de confiança para a caixa delimitadora e a previsão de classe são combinadas em uma pontuação final que descreve a probabilidade de cada caixa incluir um tipo específico de objeto. Dada escolhas de *design*, a maioria das caixas terão pontuações de confiança baixas, portanto, apenas caixas cuja pontuação final estiver além de um limite serão mantidas.

A Equação 2.13 indica a função de perda minimizada pela etapa de treinamento no algoritmo YOLO.

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] + \sum_{i=0}^{s^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \sum_{i=0}^{s^2} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (2.13)$$

onde  $1_i^{obj}$  indica se um objeto aparece na célula  $i$  e  $1_{ij}^{obj}$  indica o  $j$ -ésimo preditor da caixa delimitadora na célula  $i$  responsável por essa predição;  $x, y, w, h$  e  $C$  denotam as coordenadas que representam o centro da caixa em relação aos limites da célula da grade com as previsões de largura e altura sendo relativas à imagem inteira. Finalmente,  $C$  denota a IoU da previsão de confiança entre a caixa prevista e qualquer caixa *ground truth*.

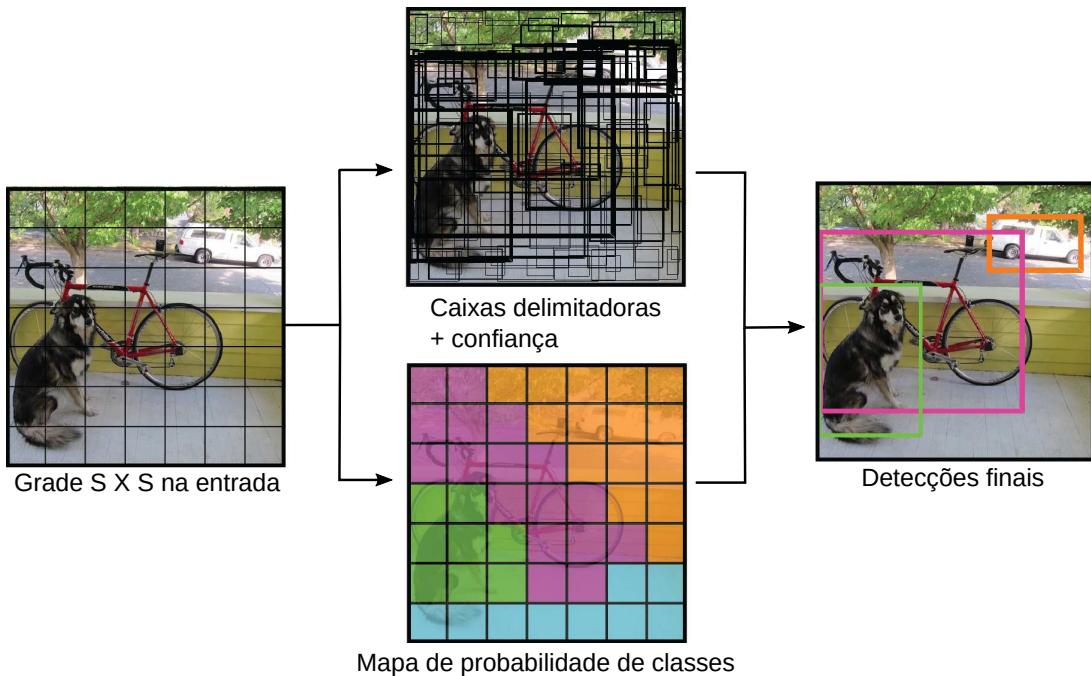


Figura 6 – A rede YOLO modela o problema de detecção de objetos como uma regressão (REDMON et al., 2015). Assim, ele divide a imagem de entrada em uma grade  $S \times S$  e para cada célula da grade prevê  $B$  caixas delimitadoras, pontuações de confiança para essas caixas e probabilidades de classe  $C$ . Essas previsões são codificadas como um tensor de dimensão  $S \times S \times (B * 5 + C)$ . Fonte: Adaptado de (REDMON et al., 2015).

A Figura 6 descreve como a rede YOLO processa uma imagem. Inicialmente, a entrada passa por uma CNN produzindo as caixas delimitadoras com suas respectivas pontuações de confiança e gerando o mapa de probabilidade de classe. Finalmente, os resultados das etapas anteriores são combinados para formar as previsões finais.

# 3 METODOLOGIA

## 3.1 Desenvolvimento Computacional

Os passos necessários para o funcionamento do sistema proposto nesse trabalho estão descritos no diagrama UML mostrado na Figura 7.

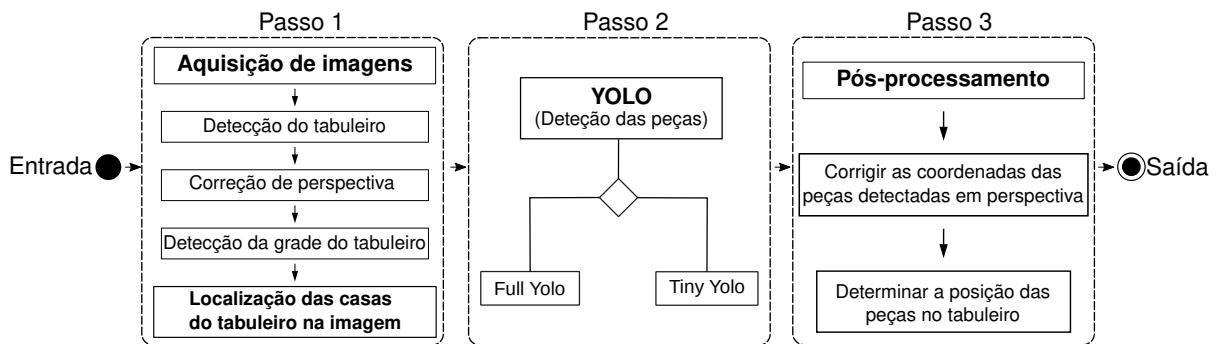


Figura 7 – Diagrama de atividades UML do sistema desenvolvido para detecção e classificação de peças de xadrez. Fonte: Autor.

A primeira etapa envolve a aquisição de imagens a partir de uma fonte de vídeo, podendo também ser um arquivo de vídeo ou até mesmo uma transmissão ao vivo de uma câmera. Na sequência, técnicas tradicionais de processamento de imagens são aplicadas para detecção do tabuleiro, correção de perspectiva e detecção da grade do tabuleiro bem como técnicas de filtragem de ruído e equalização de histograma, são aplicadas para melhorar a qualidade da imagem para as etapas subsequentes.

O resultado do primeiro passo é a criação do mapeamento de todas as casas presentes no tabuleiro para as coordenadas no espaço da imagem. Para tanto, foi adotado a representação dos quadrados que compõem as casas por duas posições, a superior esquerda e a inferior direita, permitindo assim, definir se um novo ponto amostral na imagem se encontra dentro da área delimitada por aquela casa.

O objetivo do segundo passo é a detecção das peças presentes no tabuleiro. Para isso, o *frame* original capturado pela câmera é processado pela rede YOLO, podendo aqui seguir dois caminhos distintos. O caminho de processamento da *Full YOLO* utiliza a versão mais poderosa do algoritmo YOLO para detecção de objetos fornecendo previsões mais precisas, no entanto, a execução dessa rede exige mais de poder computacional para que as previsões da rede sejam executadas em tempo real. Já a arquitetura *Tiny YOLO* do detector de objetos é uma versão da mais simplificada da rede que em alguns cenários atinge performance comparável à sua contrapartida mais poderosa, portanto, essa versão

é ideal para processamento de vídeo em tempo real. Assim, como resultado do passo dois, tem-se os pesos resultantes do treinamento de ambas as redes, *Yolo Full* e *Tiny*, ajustados para realizar a detecção das peças de xadrez presentes no tabuleiro. Também nesta etapa, o usuário realiza a escolha de qual versão utilizar, dependendo do que melhor se adequar a sua necessidade.

Por fim, a terceira etapa realiza algumas operações de pós-processamento nos resultados obtidos dos passos anteriores. Inicialmente, combina-se os resultados das predições das localizações das peças detectadas com o mapeamento das casas do tabuleiro, para esse fim, as coordenadas das detecções são transformadas para o espaço com correção de perspectiva e então é verificado à qual casa cada peça pertence. Deste modo, ao final do terceiro passo, tem-se um mapeamento completo de quais peças estão presentes no tabuleiro, e em qual casa se encontram, possibilitando a criação de uma representação bidimensional do estado atual do jogo.

### 3.2 Conjunto de dados

Para a aplicação do algoritmo de detecção de objetos, considerando o problema abordado por este trabalho, se fez necessário a construção de um conjunto de dados personalizado e adequado à detecção das peças de xadrez em um tabuleiro.

Com objetivo de validar o desenvolvimento computacional proposto neste trabalho, foram analisados vídeos de partidas de xadrez. A configuração experimental exibida na Figura 8, exemplifica a configuração utilizada para a realização das gravações.

Uma visualização tridimensional da configuração experimental utilizada para aquisição dos vídeos em partidas de xadrez, pode ser vista na Figura 8(a). Nas sessões de gravação, a câmera foi posicionada a uma altura de 61cm para gravar jogos em um tabuleiro com dimensões 47cm x 48cm, onde cada casa do tabuleiro é um quadrado com 5.3cm de lado.

*Frames* dos vídeos experimentais das partidas de xadrez foram capturados com o objetivo de validar a ferramenta computacional desenvolvida neste trabalho. A Figura 8(b) mostra a captura feita pela configuração proposta de um tabuleiro de xadrez, pronto para o início de uma partida. A câmera acoplada à configuração fornece uma visão panorâmica do tabuleiro.

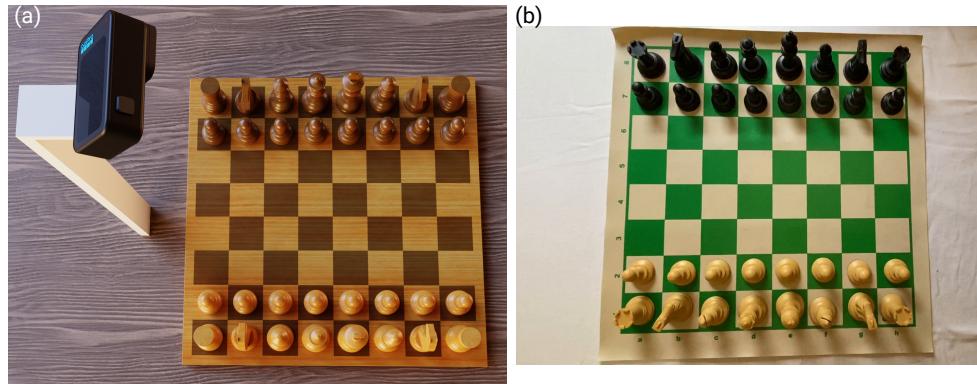


Figura 8 – Configuração experimental. (a) Representação 3D da configuração experimental para gravação de partidas de xadrez, bem como a posição da câmera para aquisição de vídeo; (b) *Frame* de exemplo mostrando a vista superior do tabuleiro de xadrez gravada pela câmera durante as partidas que compõe o conjunto de dados. Fonte: Autor.

A configuração experimental utilizada nesse estudo foi baseada em imagens capturadas utilizando um *smartphone* iPhone 11. A câmera foi configurada para capturar imagens RGB a uma taxa de 30 quadros por segundo em resolução de  $1920 \times 1080$  pixels.

O conjunto de dados utilizado para o treinamento do algoritmo de detecção de objetos é apresentado na Tabela 2. Esse sendo composto por 6317 imagens manualmente anotadas, com 5054 representando 80% separadas para serem utilizadas exclusivamente no processo de treinamento do algoritmo de detecção de objetos e 1263 representando 20% para o conjunto de teste, distribuídas em 12 classes que representam cada peça de xadrez de ambas as cores.

Tabela 2 – Conjunto de dados de imagens construído para treinamento e teste do algoritmo de detecção de objetos.

Cor	Peça	Exemplos de Treino	Exemplos de Teste
Branco	Peão	28 789	7 234
	Cavalo	5 881	1 328
	Bispo	4 884	1 144
	Torre	8 254	1 994
	Rainha	3 473	883
	Rei	4 925	1 222
Preto	Peão	28 299	7 000
	Cavalo	4 902	1 344
	Bispo	5 541	1 219
	Torre	8 799	1 946
	Rainha	3 423	874
	Rei	4 931	1 226
<b>Total</b>		112 101	27 414

### 3.3 Métricas

Essa seção apresenta as métricas que serão utilizadas para avaliar a qualidade das previsões realizadas pelo algoritmo de detecção de objetos.

#### 3.3.1 Average Precision

A precisão média interpolada (AP, do acrônimo em inglês para *average precision*) (SALTON; MCGILL, 1986) foi usada para avaliar tanto a classificação quanto a detecção de objetos neste trabalho.

A curva de precisão/recall é calculada a partir da saída classificada de um método para uma determinada tarefa e classe. O *recall* é definido como a proporção de verdadeiros-positivos identificados. A precisão é a porcentagem de todos os exemplos classificados corretamente como pertencentes à classe positiva. Altos valores para precisão indicam uma baixa taxa de falso-positivos, enquanto altos valores de recall implicam em uma baixa taxa de falso-negativos. Precisão,  $P$ , e recall,  $R$  são definidos pelas equações 3.1 e 3.2, respectivamente:

$$P = \frac{V_P}{V_P + F_P} \quad (3.1)$$

$$R = \frac{V_P}{V_P + F_N} \quad (3.2)$$

onde,  $V_P$  é o número de verdadeiros positivos,  $F_P$  é o número de falso positivos, e  $F_N$  é número de falso negativos.

Conforme mostrado na Equação 3.3, a AP resume a forma da curva de precisão/recall e é definida como a precisão média em um conjunto de onze níveis de recall igualmente espaçados.

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} p_{interp}(r) \quad (3.3)$$

A precisão em cada nível de recall  $r$ , é interpolada usando a precisão máxima medida para um método com um recall maior que  $r$ :

$$p_{interp}(r) = \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r}) \quad (3.4)$$

onde  $p(\tilde{r})$  é a precisão medida no recall  $\tilde{r}$ .

O objetivo de interpolar a curva de precisão/recall dessa maneira é reduzir o impacto das oscilações nessa curva causadas por pequenas variações na classificação de um exemplo. Deve-se notar que um método deve ter elevados níveis de precisão em todos os níveis de recall para receber uma pontuação alta, isso penaliza métodos que predizem apenas um subconjunto de exemplos com alta precisão.

Para avaliar o desempenho de modelos com múltiplas classes é empregado uma média dos valores de AP obtidos para cada classe, criando-se assim a métrica mAP, do acrônimo em inglês para *mean average precision*.

### 3.3.2 Intersection Over Union

Além de classificar corretamente à qual classe um objeto pertence, algoritmos para detecção de objetos também precisam localizar esses objetos na imagem, sendo assim, se faz necessário uma métrica para avaliar quão próximo a previsão da região que engloba o objeto está daquela que era esperada.

As detecções do algoritmo são atribuídas a objetos *ground-truth* e julgadas medindo a sobreposição da caixa delimitadora. A área de sobreposição  $a_0$  entre a caixa delimitadora prevista  $B_p$  e a caixa delimitadora esperada  $B_{gt}$  devem exceder um valor predeterminado de hiperparâmetro pela Equação 3.5 para ser considerada uma detecção correta.

$$IoU = a_0 = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \quad (3.5)$$

onde,  $B_p \cap B_{gt}$  denota a intersecção da caixa predita e esperada, e  $B_p \cup B_{gt}$  denota a união.

A Figura 9 ilustra exemplos de pontuações provenientes da métrica de Intersection Over Union, boas e ruins.

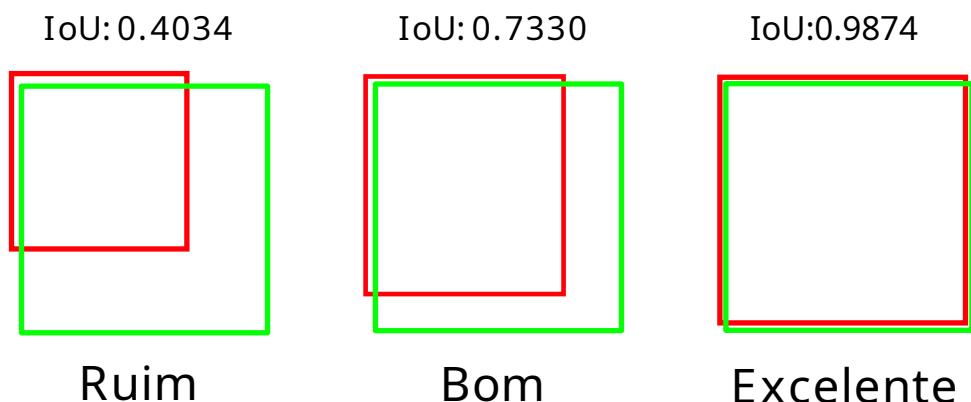


Figura 9 – Exemplo de cálculo da métrica de *Intersection Over Union* para exemplos de caixas delimitadoras. Fonte: Autor.

Uma correspondência perfeita entre as coordenadas  $(x, y)$  das caixas delimitadoras previstas e verdadeiras não é realista devido a parâmetros variáveis do modelo. Para contornar essa situação, o modelo de detecção deve prever caixas que estejam o mais próximo possível das *groundtruth*, e valores altos da métrica *Intersection Over Union* são responsáveis por isso.

## 4 RESULTADOS

Localizar com precisão as peças de xadrez em um tabuleiro é uma tarefa difícil, e exige-se uma computação moderna e eficiente. Neste trabalho, todo o desenvolvimento computacional foi realizado utilizando as linguagens de programação C e Python, com o auxílio de bibliotecas como Open-Source Computer Vision (OpenCV) e TensorFlow.

O *software* produzido nesse trabalho foi projetado para funcionar mais adequadamente com imagens nas quais o tabuleiro é o maior objeto em cena, e a câmera consegue capturar todo o tabuleiro, podendo estar posicionada em um ângulo pequeno. A Figura 10, ilustra um exemplo de imagem referente ao posicionamento da cena. Esta imagem também será utilizada ao longo desta seção, demonstrando o método para obtenção das posições durante o jogo.



Figura 10 – Exemplo de frame utilizado para realizar o posicionamento da câmera, tabuleiro e peças. Fonte: Autor.

A apresentação dos resultados obtidos nesse trabalho será dividida em quatro partes: Primeiramente, vai ser apresentado o processo de seleção do tabuleiro, que tem como objetivo, isolar o tabuleiro de xadrez do restante da imagem ao seu redor, depois, ajustar a imagem para que o tabuleiro fique perfeitamente quadrado. Em seguida, na segunda parte, será discutido a metodologia para detecção do posicionamento das 64 casas dentro do tabuleiro. A terceira parte, tratará do modelo de detecção de objetos, treinado para reconhecer e localizar as peças de xadrez. Por fim, na quarta parte, será apresentada a interface gráfica desenvolvida para uma utilização amigável da ferramenta.

## 4.1 Seleção do Tabuleiro

O objetivo dessa etapa é extrair o tabuleiro de xadrez da imagem capturada, depois, mudar a perspectiva da câmera para que a imagem aparente ser totalmente quadrada, como se a câmera estivesse posicionada diretamente acima do tabuleiro.

Esse passo se faz necessário por diversos motivos. As imagens de um jogo de xadrez com as quais o computador estará trabalhando incluiriam quaisquer outros objetos não relacionados que estivessem em cena, que durante etapas mais sensíveis como a detecção de objetos, podem “distrair” o modelo das peças de xadrez. Além disso, para analisar os *frames* de um jogo em tempo real, o computador deve processar o mínimo de *pixels* possível, fazendo com que a remoção de partes desnecessárias da imagem economize em tempo de processamento.

A perspectiva da imagem precisa ser corrigida, pelo fato do tabuleiro aparentar ter formato trapezoidal ao invés de quadrado – visão superior perfeitamente alinhada entre câmera e o cenário. Com a correção realizada, a grade dos quadrados do tabuleiro pode ser localizada posteriormente.

Para se detectar adequadamente o tabuleiro, seus quatro cantos precisam ser localizados. Esses pontos são utilizados para separar o objeto de interesse da imagem ao seu redor, e serão utilizados para mudar a perspectiva da imagem. Para realização desse processo, foi criada uma interface na qual o usuário seleciona os cantos diretamente na imagem, iniciando no canto superior esquerdo e seguindo em sentido horário para selecionar os demais cantos.

A Figura 11(a) ilustra o processo de seleção dos cantos do tabuleiro e destaca a ordem na qual precisam ser escolhidos. Na Figura 11(b) pode ser visto o resultado da operação de remoção dos arredores do tabuleiro, deixando em evidência apenas os objetos de interesse para os passos futuro do sistema. Nessa imagem, também fica claro a distorção causada pela angulação da câmera, que faz com que o tabuleiro aparente ter um formato trapezoidal.

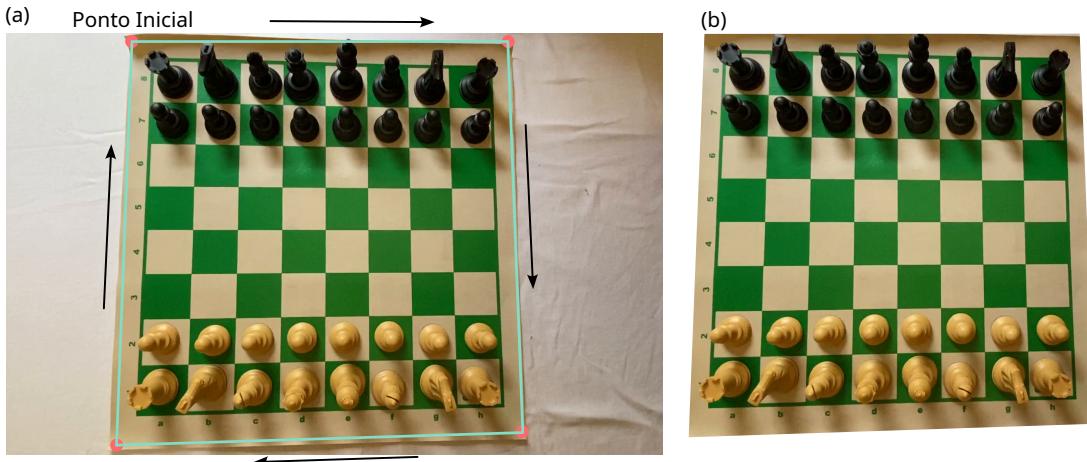


Figura 11 – Processo de seleção do tabuleiro. (a) Seleção manual dos quatro cantos do tabuleiro, que deve ser iniciada no canto superior esquerdo e seguir o sentido horário; (b) Resultado do processo de eliminação dos arredores do tabuleiro que fornece apenas a região que contém os objetos de interesse. Fonte: Autor.

Após selecionar os quatro cantos do tabuleiro e remover o restante da imagem, o próximo passo é retificar a perspectiva da imagem para que o tabuleiro fique completamente quadrado, pois o tabuleiro adquiriu um formato trapezoidal devido a angulação da câmera. Embora essa angulação seja desejada para determinar posteriormente o tipo de peça, o computador não possui a capacidade que os humanos têm de entender naturalmente que o tabuleiro é quadrado a partir desse ângulo.

A perspectiva de uma imagem pode ser alterada utilizando uma transformação de perspectiva, conforme descrito na Seção 2.2.1. De posse dos cantos atuais do tabuleiro de xadrez, deseja-se que após a transformação o tabuleiro de saída seja um quadrado apenas grande o suficiente para identificar a localização de cada casa do tabuleiro. Sendo assim, a saída foi definida como uma matriz  $400 \times 400$  com os cantos da imagem original sendo mapeados a partir do canto superior esquerdo e seguindo o sentido horário respectivamente para:  $[(0, 0), (400, 0), (400, 400), (0, 400)]$ .

Uma vez calculada a matriz de transformação de perspectiva, pode-se utilizá-la para “deformar” a região de interesse que contém o tabuleiro, fazendo com que esta imagem se torne quadrada e todas as casas do tabuleiro tenham aproximadamente o mesmo tamanho. A Figura 12 ilustra o resultado da aplicação da matriz de transformação de perspectiva sob região previamente selecionada.

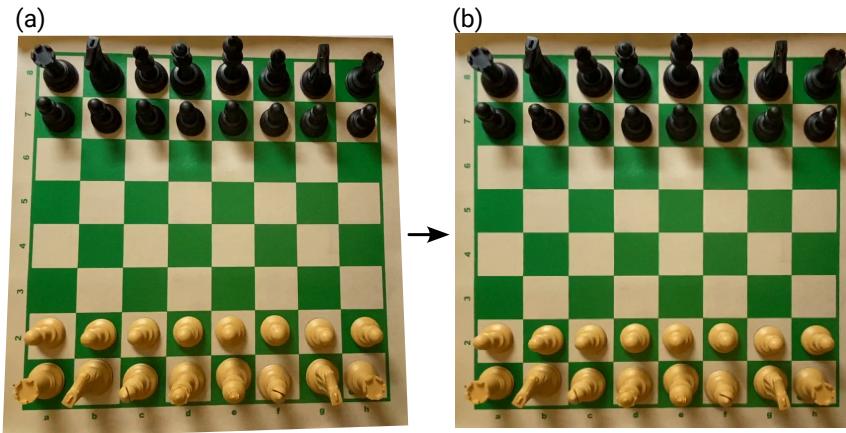


Figura 12 – Aplicação da transformação de perspectiva na região de interesse. (a) Região de interesse selecionada pelo usuário; (b) Imagem resultante da aplicação da transformação de perspectiva. Fonte: Autor.

Encontrar a grade que define cada casa do tabuleiro será possível graças a essa transformação, pois as linhas que dividem o tabuleiro, agora são perfeitamente horizontais e verticais. No entanto, à medida que o ângulo da câmera em relação ao plano do tabuleiro aumenta, as peças ficarão cada vez mais distorcidas, dificultando seu reconhecimento na imagem transformada. Para contornar esse problema, as peças serão localizadas na imagem original e suas coordenadas serão mapeadas para a grade de coordenadas da imagem distorcida, usando a matriz de transformação de perspectiva calculada. Assim, as coordenadas das peças podem ser transformadas para localizá-las dentro da grade que será detectada na imagem transformada.

## 4.2 Detecção das Casas do Tabuleiro

O principal objetivo da detecção das casas do tabuleiro é gerar um conjunto de vértices para cada uma das 64 casas do tabuleiro. Isso permitirá que as coordenadas de cada uma das peças detectadas seja mapeada para a posição correta do tabuleiro.

Inicialmente, o algoritmo de detecção de bordas discutido na Seção 2.2.1 é aplicado na imagem de saída da transformação de perspectiva, convertida para escala de cinza com o propósito de criar um mapa de todas as bordas presentes na imagem. Dois limites também são aplicados para suprimir *pixels* de borda não máximos, ou seja, aqueles *pixels* que possuem um gradiente alto, mas não maiores que seus vizinhos.

Algumas arestas podem ficar desconectadas, devido a ruídos nos gradientes após a detecção das bordas; para resolver esse problema, as operações morfológicas descritas na Seção 2.2.3 são aplicadas à saída do algoritmo. A Figura 13(b) mostra o resultado da aplicação dessa metodologia para detecção de bordas.

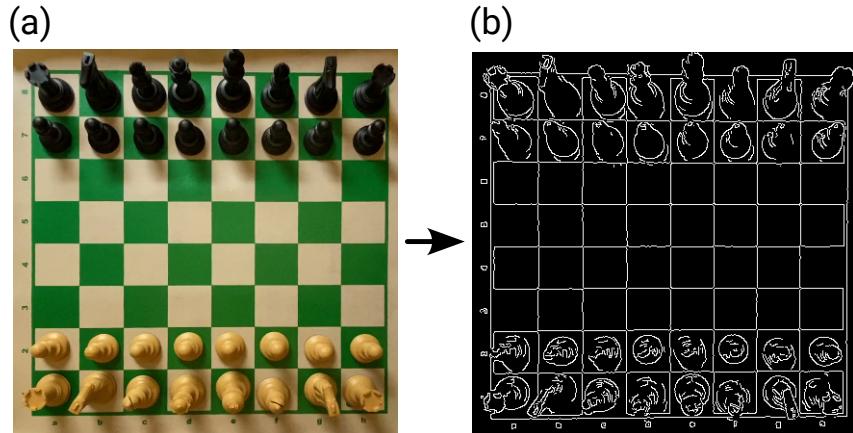


Figura 13 – Aplicação do algoritmo de detecção de bordas. (a) Imagem resultante da transformação de perspectiva aplicada a região de interesse selecionada; (b) Resultado da aplicação da metodologia de detecção de bordas proposta. Fonte: Autor.

O método conhecido como transformada de Hough, descrito na Seção 2.2.4 pode ser utilizado para encontrar as linhas ao longo de cada aresta. Permitindo que esse conjunto de bordas possa ser transformado na grade que constitui o tabuleiro de xadrez.

Alguns parâmetros de relevância foram utilizados na implementação dessa transformada. O parâmetro de comprimento mínimo de linha que especifica a extensão mínima que uma linha detectada deve ter para que seja considerada foi definido com um valor relativamente alto, já que as linhas de interesse devem percorrer toda a extensão do tabuleiro. Além disso, o intervalo máximo entre linhas foi definido para que as peças que atrapalhassem o fluxo das linhas não impedissem a sua detecção pelo método, tornando esse abordagem mais resiliente a objetos entre a câmera e o tabuleiro.

A transformada de Hough encontra as linhas do tabuleiro razoavelmente bem. Algumas linhas não são tão longas quanto deveriam ou foram encontradas em um ângulo, mas as linhas que compõem a grade do tabuleiro são consistentemente encontradas. A saída pura da transformada de Hough aplicada a detecção de bordas está ilustrada na Figura 14(b).

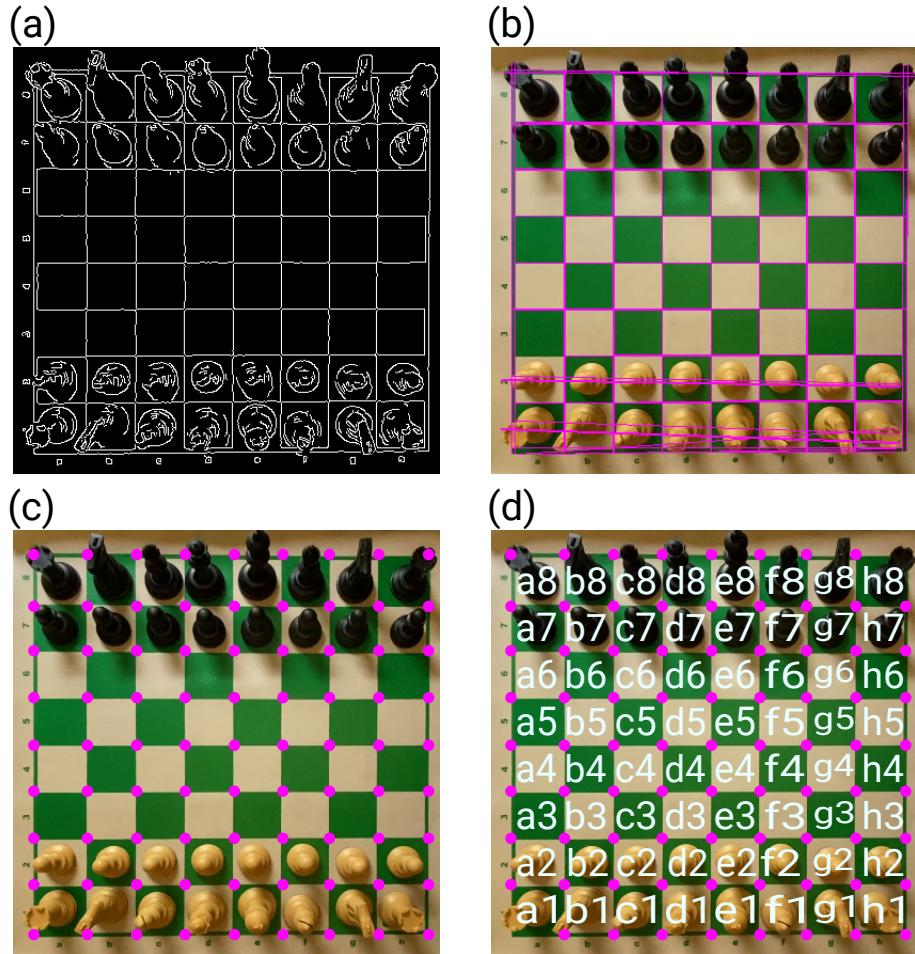


Figura 14 – Aplicação da transformada de Hough. (a) Imagem resultante da aplicação da metodologia de detecção de borda; (b) Todas as linhas detectadas pela transformada de Hough na imagem de entrada; (c) Vértices dos quadrados que compõem as casas do tabuleiro resultante das intersecções das linhas detectadas; (d) Mapeamento da localização de todas as casas do tabuleiro de acordo com sua notação algébrica. Fonte: Autor.

Como pode ser visto, várias linhas são frequentemente encontradas próximas umas das outras ao longo de uma mesma borda. Isso pode ser eliminado ao remover as linhas que estejam a uma distância pequena até as outras. Quaisquer linhas detectadas, que não estiverem próximas de serem verticais ou horizontais, também podem ser descartadas, dado que não podem fazer parte da grade do tabuleiro. As linhas restantes são levemente deslocadas para serem perfeitamente verticais ou horizontais, conforme necessário, e podem ser alongadas para que percorram o comprimento da imagem. A partir disso, os vértices de todos os quadrados na imagem podem ser determinados encontrando os pontos de interseção de todas as linhas. A Figura 14(c) mostra os vértices encontrados utilizando esse método.

Para finalizar esse passo, a lista de vértices encontrada pode ser processada para gerar um dicionário de localização das casas no tabuleiro. Para isso, os vértices encontrados

são ordenados de forma em que sua distribuição na lista de vértices, corresponda com sua distribuição no tabuleiro, assim, os primeiros 9 vértices na lista corresponderão aos pontos encontrados para a primeira linha e assim sucessivamente. Em seguida, percorre-se essa lista ordenada guardando os pontos superiores esquerdo e inferior direito de cada casa em sequência, começando pela casa  $a8$  e terminando na casa  $h1$ . Como somente dois pontos são necessários para a definição de um quadrado, dois pontos da lista de vértices detectados se tornam redundantes e podem ser descartados. A Figura 14(d) ilustra o resultado do processo de construção do dicionário de localização das casas.

### 4.3 Detecção das Peças

O objetivo da detecção de peças é pesquisar em uma imagem por peças de xadrez e retornar quais peças estão presentes, bem como sua localização. O modelo de detecção de objetos YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020) foi escolhido por ser bastante preciso e rápido para modelos baseados em redes neurais. Esse modelo é capaz de detectar objetos em tempo real depois de treinado, mesmo em *hardwares* de baixo custo.

Para detecção das peças de xadrez, a abordagem discutida neste trabalho utilizou duas versões da rede YOLO. Os resultados desta pesquisa foram obtidos tendo como base a análise de 6317 imagens, organizadas de acordo com o conjunto de dados descrito na Seção 3.2.

A primeira versão do modelo para detecção de objetos, foi treinada utilizando a rede do tipo *YOLO Full*, que emprega a arquitetura convolucional Darknet-53 (REDMON; FARHADI, 2018) com 53 camadas convolucionais. A partir de um modelo pré-treinado, que utilizou o conjunto de dados Imagenet (RUSSAKOVSKY et al., 2015), esse modelo foi treinado utilizando a técnica de transferência de aprendizado, conforme descrito em (REDMON et al., 2015), obtendo assim, máxima vantagem do processo de otimização e ajuste dos pesos. O modelo consiste em um segmento com camadas convolucionais e conexões residuais em um total de  $65.29 \cdot 10^9$  operações em ponto flutuante. Cada modelo requer 235MB de espaço de armazenamento. O modelo levou 11.59 horas para ser treinado.

A rede *YOLO Tiny*, uma alternativa de arquitetura menor e mais rápida, também foi treinada. Para acelerar o processo de detecção de objetos, essa versão “tiny” usa apenas um subconjunto dos recursos da *Darknet-53*, 23 camadas convolucionais, resultando em  $9.67 \cdot 10^9$  operações de ponto flutuante, quase sete vezes menos que sua contraparte maior. Cada modelo precisa de apenas 34 MB de espaço de armazenamento. A rede foi treinada usando o método descrito em (REDMON et al., 2015), o envolveu o ajuste fine de um modelo pré-treinado no conjunto de dados Imagenet (RUSSAKOVSKY et al., 2015). O treinamento do modelo levou 2.37 horas. A Figura 15 ilustra alguns dos resultados de

deteção de peças de xadrez no conjunto de dados construído.

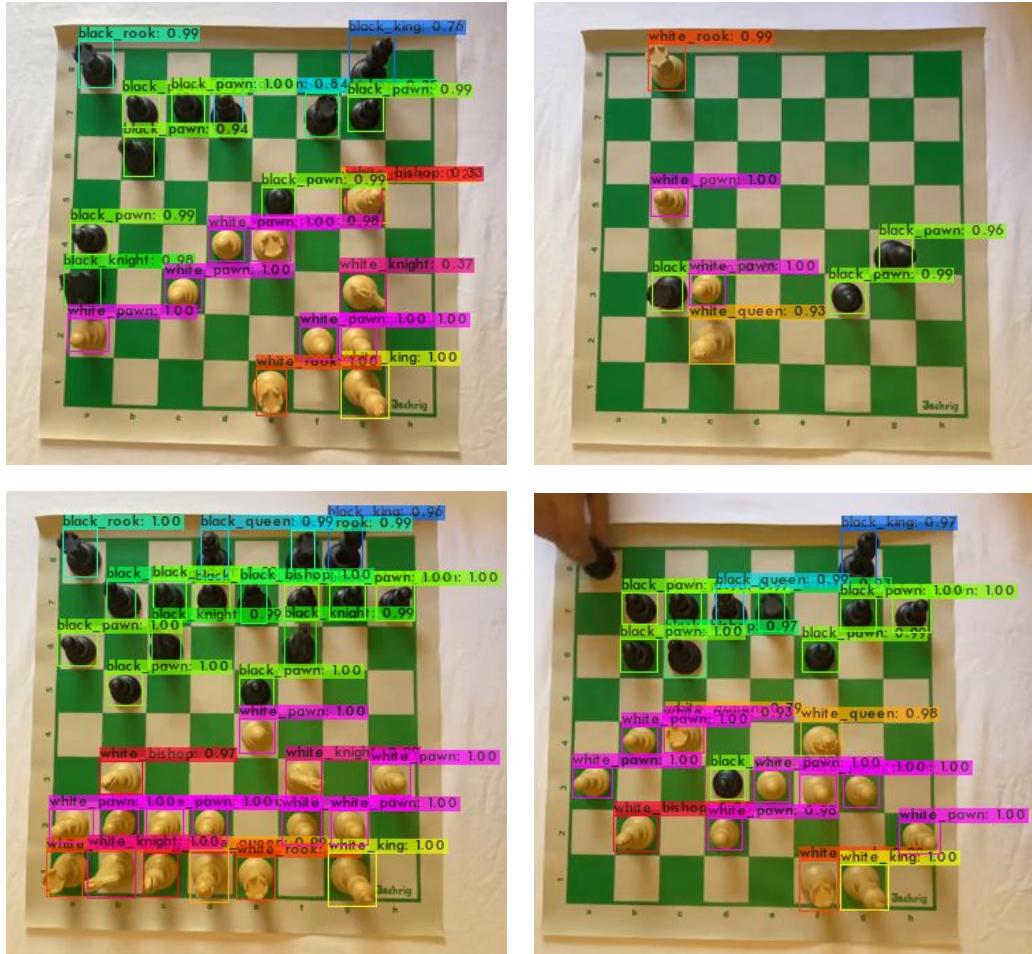


Figura 15 – Exemplos da saída do modelo de detecção de objeto treinado. Fonte: Autor.

O método proposto pela arquitetura *YOLO* é mais rápido que outras abordagens para detecção de objetos, em parte porque prevê a localização dos objetos na imagem como um todo, ou invés de aplicar a classificação de objetos em regiões diferentes, como fazem outras redes (GIRSHICK et al., 2013; GIRSHICK, 2015; REN et al., 2015). A *YOLO* divide a imagem em uma grade de  $S \times S$  células e procura objetos dentro de cada célula simultaneamente. Como por design apenas dois objetos podem estar na mesma célula, essa abordagem encontra dificuldades em lidar com objetos próximos ou sobrepostos. No entanto, o modelo prevê bem objetos em posições similares, semelhante a como as peças de xadrez estão sempre distribuídas de formas similares no tabuleiro.

A *YOLOv4* como um detector de objetos de estágio único se baseia no modelo *YOLO* original (REDMON et al., 2015) e no detector *YOLOv3* (REDMON; FARHADI, 2018) que o antecedeu. A nova versão é composta principalmente de duas partes, a primeira delas sendo denominada de espinha dorsal e a segunda de cabeça. A espinha dorsal se trata do extrator de características modelado como uma rede convolucional, já a ca-

beça se trata do parte do modelo responsável pelas predições das classes de objetos e suas caixas delimitadoras.

Para extraír propriedades relevantes das imagens, o modelo faz uso das redes neurais convolucionais, utilizando filtros que são aplicados às imagens para extração de características, que se tornam cada vez mais abstratas à medida que os dados progridem por uma série de etapas convolucionais. A Figura 16 mostra o resultado do processo de convolução de alguns filtros em estágios diferentes da rede.

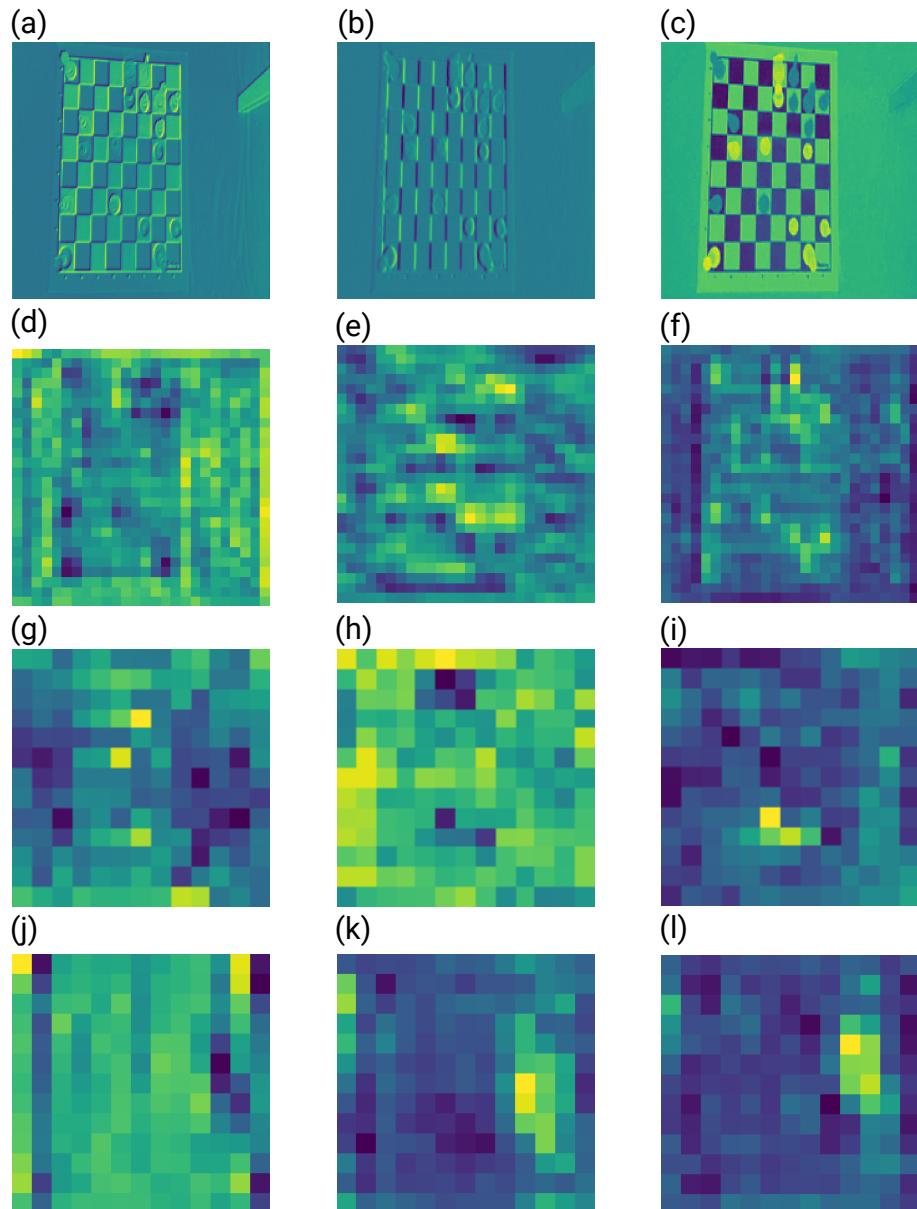


Figura 16 – Visualização da saída das camadas convolucionais. (a)-(c) Referem-se as saídas de camadas convolucionais iniciais do extrator de características; (d)-(i) São as representações de saídas das camadas intermediárias do modelo; (j)-(l) Ilustram as saídas das camadas finais do extrator de características. Fonte: Autor.

A Figura 17 ilustra o processo de otimização da função de perda para ambas as arquiteturas de rede durante a etapa de treinamento, esse processo deixou com sucesso os mínimos locais e se aproximou de zero no final das 8000 épocas alocadas. A análise da figura também revela a maior eficiência na otimização dos pesos da arquitetura *Tiny*, que desde do início do processo de treinamento obteve valores menores para a função de perda e atingiu valores cerca de dez vezes menores ao final do processo de otimização.

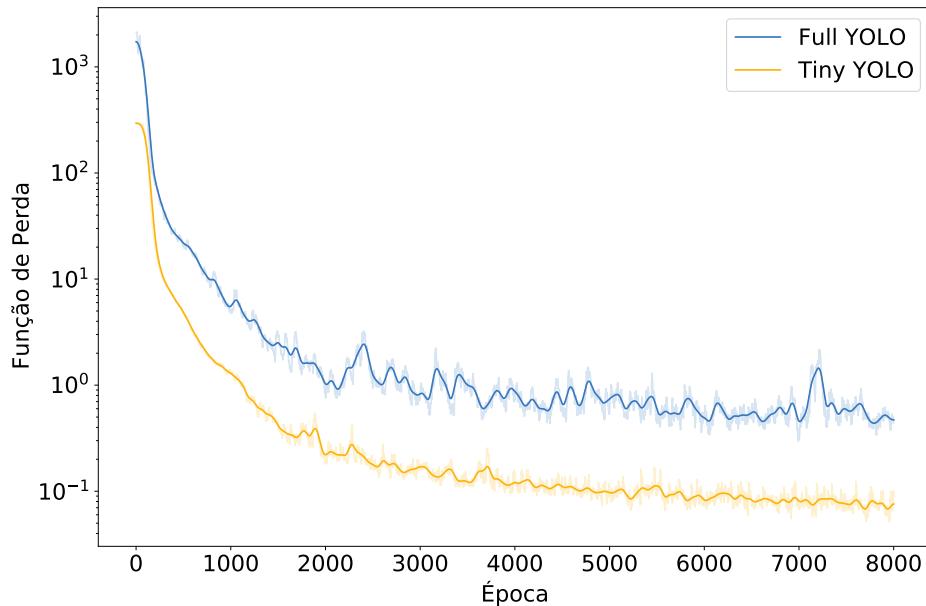


Figura 17 – Minimização da função de perda durante o processo de treinamento das arquiteturas *YOLO Full* e *Tiny*. Fonte: Autor.

A Tabela 3 apresenta os resultados da aplicação das métricas avaliativas de classificação, obtidas na etapa final do modelo. Foram utilizadas as arquiteturas, *Full* e *Tiny*. Essas métricas servem para avaliar a qualidade da saída dos classificadores. A precisão é definida como a proporção de observações positivas previstas corretamente para todas as observações marcadas como positiva. A razão de observações positivas corretamente previstas para todas as observações é definida como *recall*. Como o F1-Score é a média ponderada de precisão e recall, o valor máximo atribuído igual a 1, indicando precisão e *recall* perfeitos.

Tabela 3 – Métricas de classificação no conjunto de dados de teste para os modelos avaliados.

Modelo	Precisão (%)	Recall (%)	F1-Score
Full YOLO	87	90	0.88
Tiny YOLO	89	89	0.89

Ainda na Tabela 3, os valores de precisão, *recall* e F1-Score para a rede *Full* foram todos acima de 80%, indicando uma boa qualidade das previsões de classe e sucesso na

etapa de treinamento com boa generalização, enquanto a rede *Tiny* teve um desempenho ligeiramente melhor apesar seu menor tamanho, com precisão, *recall* e F1-Score todos iguais a 0.89.

A Tabela 4 contém resultados adicionais, relacionadas ao desempenho de detecção de peças. Primeiramente, é mostrado a precisão média para cada classe, como discutido na Seção 3.3.1. As classes correspondem a cada tipo de peça de xadrez em ambas as cores clássicas, essas sendo preto e branco. Por fim, os valores de mAP para cada modelo são mostrados na parte inferior da tabela.

Tabela 4 – Precisão média para as detecções de cada classe para os modelos avaliados.

<b>Objeto</b>	<b>Precisão Média (AP, %)</b>	
	<b>Modelo</b>	<b>Full YOLO</b>
Peão Branco	90.88	90.90
Cavalo Branco	84.97	84.11
Bispo Branco	74.09	58.88
Torre Branca	84.67	84.72
Rainha Branca	72.33	74.28
Rei Branco	86.70	84.81
Peão Preto	90.68	90.89
Cavalo Preto	90.13	89.61
Bispo Preto	67.47	70.10
Torre Preta	88.18	87.64
Rainha Preta	90.57	89.67
Rei Preto	90.84	90.86
<b>mAP</b>	<b>84.29</b>	<b>83.04</b>

Conforme mostrado na Tabela 4, a rede *Full* possui valores estatísticos ligeiramente melhores para mAP, com a diferença para sua contraparte *Tiny* sendo de 1.25%. O tamanho menor da rede *Tiny* mostra suas vantagens aqui. Embora possua uma menor capacidade de abstração, essa arquitetura foi capaz de superar sua contraparte maior em diversas classes. Também vale a pena notar que ambos os modelos tiveram um desempenho não muito satisfatório na detecção dos Bispos, em ambas as cores, com os melhores resultados em torno de 70%. Isso pode ser atribuído às enormes semelhanças entre os Bispos e os Peões, sendo que o último possui mais exemplos para apreender, dado que sua presença em um tabuleiro de xadrez é mais abundante.

Com relação a métrica de *intersection over union* apresentada na Seção 3.3.2, a arquitetura *Full* apresentou uma IoU média de 71.47%, alcançando uma generalização competente para detecção de um objeto. Já a contrapartida menor do modelo, nomeadamente de rede *Tiny*, obteve um valor médio de 77.80% para a métrica de IoU, mostrando mais uma vez a competência desse modelo para o problema apresentado.

A proporção de detecções corretas e equivocadas produzida a partir das previsões da rede no conjunto de dados de teste é exibida na Tabela 5, isto finaliza a análise dos resultados para o algoritmo de detecção de objetos. O detector cometeu relativamente poucos erros de previsão, conforme as altas pontuações nas métricas apresentadas anteriormente, classificando corretamente a maioria das amostras de teste.

Com os resultados indicados na Tabela 5, fica novamente evidente a dificuldade para correta classificação das peças Bispo. A arquitetura *Tiny* obteve maior sucesso na classificação de tal peça. Já no caso das Rainhas, peças que também se apresentaram problemáticas, a rede *Full* apresentou os melhores resultados de classificação para ambas as cores desta peça.

Tabela 5 – Matriz de confusão para os modelos analisados.

Objeto	Modelo			
	Full YOLO		Tiny YOLO	
	Verdadeiro Positivo	Falso Positivo	Verdadeiro Positivo	Falso Positivo
Peão Branco	7230	719	7226	846
Cavalo Branco	1059	222	1065	247
Bispo Branco	818	462	619	281
Torre Branca	1563	115	1490	17
Rainha Branca	578	270	523	166
Rei Branco	1052	201	1033	66
Peão Preto	6970	609	6990	480
Cavalo Preto	1121	40	1190	80
Bispo Preto	871	821	705	300
Torre Preta	1734	379	1718	512
Rainha Preta	705	5	741	24
Rei Preto	1053	6	1079	2

A Figura 18 compara a eficiência da utilização da arquitetura *Tiny* com a *Full* em termos de tempo de treinamento e predição. O treinamento com a arquitetura menor levou cerca de 2.4 horas, o que foi aproximadamente cinco vezes mais rápido que a versão *Full*. Já a Figura 18(b) mostra um gráfico de caixa do tempo gasto em ambas as arquiteturas na classificação de uma única imagem. O menor tamanho da versão *Tiny* tem uma tradução direta no seu tempo de execução, com valores de média e desvio padrão de  $3.08 \pm 0.05\ ms$  contra  $20.54 \pm 0.09\ ms$  para a versão *Full*.

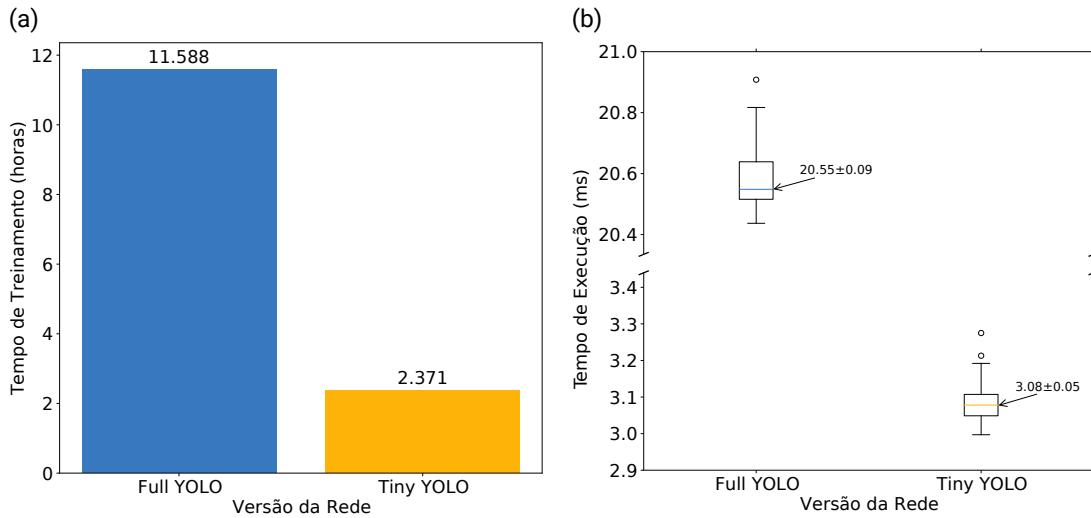


Figura 18 – O tempo de processamento em GPU utilizado pelos modelos. (a) Tempo necessário para treinamento de ambos os modelos por 8000 épocas; (b) A quantidade de tempo requerida para classificação de uma única imagem. Fonte: Autor.

#### 4.4 Interface Gráfica

Uma interface gráfica de usuário (GUI do acrônimo em inglês para *graphical user interface*), conforme mostrado na Figura 19, foi criada para facilitar o acesso às ferramentas apresentadas. Por meio desta, o usuário pode carregar o vídeo e visualizar os *frames* à medida que são processados, além de alterar alguns parâmetros e opções em tempo real.

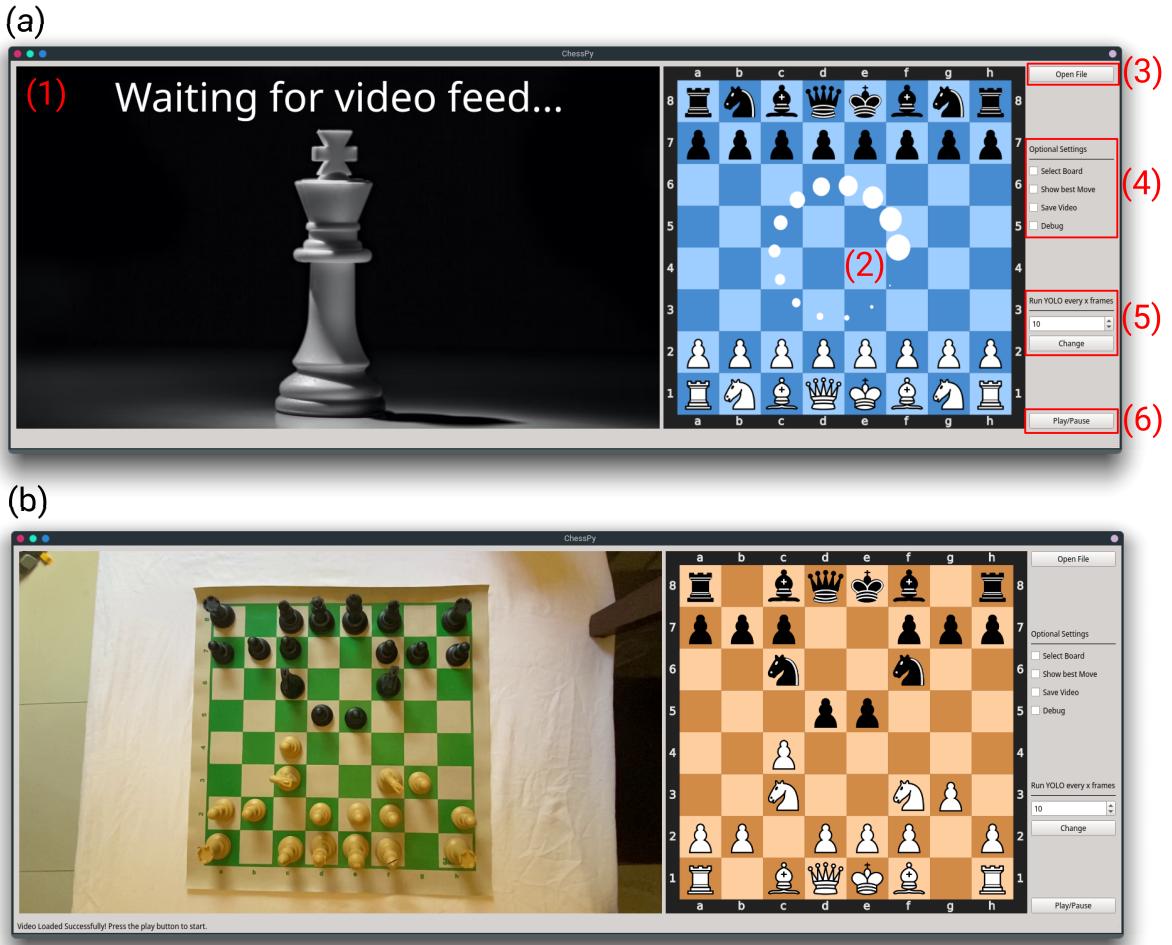


Figura 19 – Interface desenvolvida. (a) Tela inicial da interface, aguardando o usuário carregar um vídeo para análise; (b) Interface durante a análise do vídeo de uma partida. Fonte: Autor.

A Figura 19(a) destaca as partes que compõem a GUI. A maior porção mostrada em (1) está dedicada para a exibição do vídeo que está sendo processado, onde os quadros deste vão sendo exibidos de acordo com que são processados pela *pipeline* de detecção. Em (2) está o tabuleiro que incorpora todos os dados colhidos com a *pipeline* de detecção, juntando as detecções das peças feitas pela rede YOLO com suas devidas posições em notação algébrica no tabuleiro para formar a sequência FEN que representa a posição e esta pode ser processada para criação da imagem representativa da posição atual do jogo no *frame* processado.

Na lateral da interface encontram-se os controladores da aplicação. O botão destacado em (3) é responsável pela abertura de uma janela na qual o usuário selecionará o arquivo de vídeo a ser processado pelo algoritmo. A região destacada em (3) contém as caixas de seleção que controlam as configurações opcionais do sistema, primeiramente tem-se a caixa de seleção para que a interface de seleção do tabuleiro seja mostrada assim que um arquivo for carregado, em sequência encontra-se a caixa de seleção para alterar

a exibição dos melhores movimentos na posição atual do jogo para ambos os jogadores calculados por uma *engine* de xadrez. Depois tem-se a caixa de seleção para salvamento de um arquivo de vídeo contendo o resultado da detecção de objetos em cada *frame* do vídeo selecionado. Por fim, é disponibilizado uma opção de *debug*, na qual uma nova janela será criada mostrando em tempo real tanto as detecções das casas do tabuleiro como o resultado da detecção de peças.

A região destacada em (5) trata do controle da frequência de execução do algoritmo de detecção de peças. A depender do ritmo de jogo retratado no arquivo de vídeo o usuário pode mudar a frequência com a qual o *frame* é processado dado que para ritmos de jogos mais lentos a mesma posição persistirá por diversos frames fazendo com que a execução constante do detector de objetos seja redundante dado que a posição não terá sido alterada. Por fim, em (6) está destacado o botão que controla a reprodução do vídeo sendo processado, o que permite a reprodução do mesmo seja pausada e retomada sempre que o usuário deseje.

A Figura 19(b) ilustra a GUI sendo utilizada para processamento de um arquivo de vídeo, com o detector de objetos sendo utilizado a cada dez *frames*. Pode-se perceber o sucesso da abordagem proposta com a posição no vídeo sem detectada perfeitamente pela *pipeline* de processamento.

## 5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento computacional de uma ferramenta para acompanhamento de jogos de xadrez em tempo real, por meio de técnicas do estado-da-arte em detecção de objetos as peças presentes no tabuleiro puderam ser encontradas e classificadas bem como sua posição dentro do tabuleiro pôde ser determinada por meio da utilização de técnicas clássicas de processamento digital de imagens.

A utilização da ferramenta desenvolvida neste trabalho permite que organizadores de torneio de xadrez de médio e pequeno porte registrem e transmitam os jogos do evento de forma simples e com um custo menor do que os oferecidos pela utilização de tabuleiros eletrônicos, já que para a solução aqui proposta é necessário apenas a utilização de uma câmera de baixo custo.

Como proposta de trabalhos futuros pode-se elencar a construção de um conjunto de dados com estilos de peças e tabuleiros mais variados, dado a grande variedade de estilos de peças de xadrez disponíveis no mercado, assim o sistema atenderia imediatamente uma variedade maior de casos de uso. Além disso, pode-se desenvolver uma interface mais simplificada para que os jogos detectados pelo sistema possam ser transmitidos ao vivo para as plataformas online de acompanhamento de torneios.

# REFERÊNCIAS

- ANTON, H.; RORRES, C. *Elementary linear algebra: applications version.* [S.l.]: John Wiley & Sons, 2013.
- BALLARD, D. H. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, Elsevier, v. 13, n. 2, p. 111–122, 1981.
- BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection.* 2020.
- CAMPBELL, M.; JR, A. J. H.; HSU, F.-h. Deep blue. *Artificial intelligence*, Elsevier, v. 134, n. 1-2, p. 57–83, 2002.
- CANNY, J. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, Ieee, n. 6, p. 679–698, 1986.
- CIRESAN, D. C. et al. *Flexible, High Performance Convolutional Neural Networks for Image Classification.* 2011.
- D'AGOSTINI, O. *Xadrez Basico.* [S.l.]: EDIOURO, 2002. ISBN 9788500108402.
- DENG, L.; HINTON, G.; KINGSBURY, B. New types of deep neural network learning for speech recognition and related applications: An overview. In: IEEE. *2013 IEEE international conference on acoustics, speech and signal processing.* [S.l.], 2013. p. 8599–8603.
- DUDA, R.; HART, P. <sup>a</sup>use of the hough transform to detect lines and curves in pictures, <sup>o</sup> comm. ACM, 1972.
- EDWARDS, S. J. et al. Standard portable game notation specification and implementation guide. *Online]. Dosegljivo: http://www. saremba. de/chessgml/standards/pgn/pgncomplete. htm*, 1994.
- ESTEVA, A. et al. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, Nature Publishing Group, v. 542, n. 7639, p. 115–118, 2017.
- GIRSHICK, R. Fast r-cnn. arxiv 2015. *arXiv preprint arXiv:1504.08083*, 2015.
- GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. corr, abs/1311.2524. *arXiv preprint arXiv:1311.2524*, 2013.
- HARALICK, R. M.; STERNBERG, S. R.; ZHUANG, X. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, n. 4, p. 532–550, 1987.
- HEARST, M. A. et al. Support vector machines. *IEEE Intelligent Systems and their applications*, IEEE, v. 13, n. 4, p. 18–28, 1998.
- HOUGH, P. V. *Method and means for recognizing complex patterns.* [S.l.]: Google Patents, 1962. US Patent 3,069,654.

- HSU, F.-H. *Behind Deep Blue: Building the computer that defeated the world chess champion.* [S.l.]: Princeton University Press, 2002.
- ILLINGWORTH, J.; KITTLER, J. A survey of the hough transform. *Computer vision, graphics, and image processing*, Elsevier, v. 44, n. 1, p. 87–116, 1988.
- JAYARAMAN, V.; CHANDRASEKHAR, M.; RAO, U. Managing the natural disasters from space technology inputs. *Acta Astronautica*, Elsevier, v. 40, n. 2-8, p. 291–325, 1997.
- KOGAN, F. N. Global drought watch from space. *Bulletin of the American Meteorological Society*, American Meteorological Society, v. 78, n. 4, p. 621–636, 1997.
- KRIEGESKORTE, N. Deep neural networks: a new framework for modelling biological vision and brain information processing. *biorxiv*, Cold Spring Harbor Laboratory, p. 029876, 2015.
- LEONARD, M. et al. A compound event framework for understanding extreme impacts. *Wiley Interdisciplinary Reviews: Climate Change*, Wiley Online Library, v. 5, n. 1, p. 113–128, 2014.
- MENEZES, R. S. T. D.; MAGALHAES, R. M.; MAIA, H. Object recognition using convolutional neural networks. In: *Recent Trends in Artificial Neural Networks-from Training to Prediction.* [S.l.]: IntechOpen, 2019.
- MENEZES, R. S. T. de et al. Classification of mice head orientation using support vector machine and histogram of oriented gradients features. In: IEEE. *2018 International Joint Conference on Neural Networks (IJCNN).* [S.l.], 2018. p. 1–6.
- OSKOEI, M. A.; GAN, J. Q.; HU, H. Adaptive schemes applied to online svm for bci data classification. In: IEEE. *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society.* [S.l.], 2009. p. 2600–2603.
- OVERLEAF. *Chess notation.* 2022. Disponível em: <[https://www.overleaf.com/learn-latex/Chess\\_notation](https://www.overleaf.com/learn-latex/Chess_notation)>.
- PAN, W. D.; DONG, Y.; WU, D. Classification of malaria-infected cells using deep convolutional neural networks. *Machine learning: advanced techniques and emerging applications*, BoD–Books on Demand, v. 159, 2018.
- REDMON, J. et al. You only look once: Unified, real-time object detection. arxiv 2015. *arXiv preprint arXiv:1506.02640*, 2015.
- REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, v. 28, 2015.
- ROMSTAD, T. et al. *Stockfish 15: UCI Chess Engine.* 2022. Disponível em: <<https://github.com/official-stockfish/Stockfish>>.
- RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, Springer, v. 115, n. 3, p. 211–252, 2015.

- SALMAN, A. et al. Leaf classification and identification using canny edge detector and svm classifier. In: IEEE. *Inventive Systems and Control (ICISC), 2017 International Conference on*. [S.l.], 2017. p. 1–4.
- SALTON, G.; MCGILL, M. *Introduction to modern information retrieval*. New York, NY: McGrawHill. [S.l.]: Inc, 1986.
- SCHERER, D.; MÜLLER, A.; BEHNKE, S. Evaluation of pooling operations in convolutional architectures for object recognition. In: *Artificial Neural Networks-ICANN 2010*. [S.l.]: Springer, 2010. p. 92–101.
- SHANNON, C. E. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 41, n. 314, p. 256–275, 1950.
- SILVER, D. et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- SRINIVAS, S. et al. An introduction to deep convolutional neural nets for computer vision. In: *Deep Learning for Medical Image Analysis*. [S.l.]: Elsevier, 2017. p. 25–52.
- TRAHANIAS, P. An approach to qrs complex detection using mathematical morphology. *IEEE Transactions on Biomedical Engineering*, IEEE, v. 40, n. 2, p. 201–205, 1993.
- YU-QIAN, Z. et al. Medical images edge detection based on mathematical morphology. In: IEEE. *2005 IEEE engineering in medicine and biology 27th annual conference*. [S.l.], 2006. p. 6492–6495.
- ZANA, F.; KLEIN, J.-C. Segmentation of vessel-like patterns using mathematical morphology and curvature evaluation. *IEEE transactions on image processing*, IEEE, v. 10, n. 7, p. 1010–1019, 2001.
- ZHOU, B. et al. Learning deep features for discriminative localization. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016.