

**UNIVERSIDADE FEDERAL DE JUIZ DE FORA**  
**FACULDADE DE ENGENHARIA**  
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**Nayara Maria Sperandio Rocha**

**Sistema de Visão Computacional Aplicado em Análises de Jogos de Tênis**

Juiz de Fora

2021

Nayara Maria Sperandio Rocha

**Sistema de Visão Computacional Aplicado em Análises de Jogos de Tênis**

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas de Energia Elétrica.

Orientador: D.Sc. André Luís Marques Marcato

Coorientadora: D.Sc. Milena Faria Pinto

Juiz de Fora

2021

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF  
com os dados fornecidos pelo(a) autor(a)

Rocha, Nayara Maria Sperandio .

Sistema de Visão Computacional Aplicado em Análises de Jogos de  
Tênis / Nayara Maria Sperandio Rocha. – 2021.

123 f. : il.

Orientador: André Luís Marques Marcato

Coorientadora: Milena Faria Pinto

Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Faculdade  
de Engenharia. Programa de Pós-Graduação em Engenharia Elétrica, 2021.

1. Análise de partida de tênis. 2. Processamento de Imagens. 3. Visão  
Computacional. 4. Detecção de Linhas. 5. Ponto de inflexão da bola. I.  
Marcato, André Luís Marques, orient. II. Pinto, Milena Faria, coorient. III.  
Título.

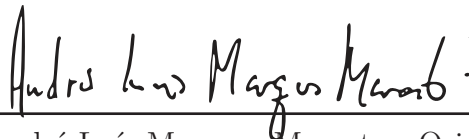
Nayara Maria Sperandio Rocha

Sistema de Visão Computacional Aplicado em Análises de Jogos de Tênis

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas de Energia Elétrica.

Aprovada em 26 de Março de 2021

BANCA EXAMINADORA



---

D.Sc. André Luís Marques Marcato - Orientador  
Universidade Federal de Juiz de Fora (UFJF)



---

D.Sc. Milena Faria Pinto - Coorientadora  
Centro Federal de Educação Tecnológica Celso Suckow  
da Fonseca (CEFET-RJ)



---

D.Sc. Paulo Lilles Jorge Drews Junior  
Universidade Federal do Rio Grande (FURG)



---

D.Sc. Alexandre Bessa dos Santos  
Universidade Federal de Juiz de Fora (UFJF)

Dedico este trabalho aos meus pais, Gilson e Simone,  
pelo apoio constante e infinito.

## AGRADECIMENTOS

Agradeço primeiramente à Deus por tudo o que tenho, por todas as realizações que alcancei, por ter iluminado meu caminho, por ter me dado saúde para percorrer toda essa jornada, pelo Seu amor tão presente em cada momento de minha vida.

Aos meus pais, Gilson de Melo Rocha e Simone Aparecida Sperandio Rocha, pelo apoio infindável durante toda a minha vida. Não houve um momento sequer em que vocês não me incentivaram a ser cada vez melhor. Muito obrigada por nunca me deixarem desistir por mais difícil que o objetivo possa parecer, devo a vocês todas as minhas vitórias. Essa conquista não seria possível sem vocês ao meu lado.

Ao meu orientador Professor D.Sc. André Luís Marques Marcato e coorientadora Professora D.Sc. Milena Faria Pinto por aceitarem conduzir o meu trabalho de pesquisa, pelo apoio desde o início do desenvolvimento do trabalho e por estarem ao meu lado durante todo o mestrado, sempre me incentivando a continuar crescendo na área acadêmica.

Ao meu professor do curso técnico de informática, Professor D.Sc. Dário Barros de Oliveira (in memoriam), que despertou meu interesse pela engenharia, me inspirando desde a primeira aula e sempre acreditando no meu potencial. Muito obrigada por ter passado pela minha vida de forma tão marcante e única.

Aos meus amigos, pela amizade incrível cheia de reciprocidade que construímos durante todo esse tempo. Vocês tornaram o caminho até aqui muito mais leve e prazeroso. Muito obrigada por torcerem tanto por mim.

À FAPEMIG pela valiosa concessão da bolsa de estudos que viabilizou o desenvolvimento do projeto.

À Universidade Federal de Juiz de Fora e o seu corpo docente que demonstrou estar comprometido com a qualidade e excelência do ensino.

Ao espaço PQ Tennis Center por permitir a realização dos testes práticos envolvidos neste trabalho em suas dependências.

*"Everything you want's a dream away  
Under this pressure, under this weight  
We are diamonds taking shape".*

Coldplay - Adventure of a Lifetime

## RESUMO

O campo de inteligência artificial tem desempenhado um papel muito importante no desenvolvimento tecnológico atual, fazendo com que as máquinas tenham uma percepção de mundo de forma semelhante à dos humanos e sejam capazes de realizar diversas tarefas difíceis com boa precisão. Neste sentido, uma parte significativa da inteligência artificial lida com sistemas que realizam ações que necessitam de um sistema de visão computacional, que age como um sensor fornecendo informações de alto nível sobre um determinado ambiente. Uma área de pesquisa dentro do campo de visão computacional estuda como reconstruir e compreender uma cena 3D a partir das propriedades presentes em imagens 2D. Os resultados dessa linha de pesquisa se tornam muito valiosos devido ao fato de que alguns lances podem ser muito difíceis de serem analisados somente assistindo fisicamente a uma partida. Assim, ferramentas de visão computacional estão sendo cada vez mais utilizadas em aplicações esportivas, onde as decisões técnicas em um determinado lance podem ser decisivas para a partida. Um esporte que vem sendo altamente explorado em termos de desenvolvimento de pesquisas é o tênis pela sua crescente disseminação e relevância financeira, sendo responsável por movimentar anualmente R\$1,8 bilhão só no Brasil. Dessa forma, este trabalho de dissertação de mestrado objetiva o desenvolvimento de um sistema baseado em visão computacional para análise de jogos de tênis. O sistema implementado captura vídeos durante o jogo de tênis por meio de câmeras instaladas na quadra e processa as imagens obtidas, aplicando métodos de aprendizado de máquina e operações morfológicas, a fim de localizar a posição da bola, as linhas da quadra e a localização dos jogadores após o processamento. Além disso, o algoritmo determina o momento do quique da bola durante o jogo e analisa se este ocorreu dentro ou fora do campo. A partir dos resultados obtidos, o sistema demonstrou robustez e confiabilidade. Esses dados são disponibilizados aos jogadores e juízes tanto na quadra durante a realização do jogo quanto por meio de um aplicativo Android, também desenvolvido neste trabalho. O aplicativo tem por objetivo permitir que todos os dados resultantes do processamento sejam acessado a partir de dispositivos móveis, fornecendo os resultados de forma rápida e acessível ao usuário.

**Palavras-chave:** Análise de partida de tênis, Processamento de Imagens, Visão Computacional, Detecção de Linhas, Ponto de inflexão da bola.



## ABSTRACT

The field of artificial intelligence has played a very important role in current technological development, making machines have a perception of the world similar to humans and are able to perform many difficult tasks with good precision. In this sense, a significant part of artificial intelligence deals with systems that perform actions that require a computer vision system, which acts as a sensor providing high-level information about a given environment. A research area within the field of computational vision studies how to reconstruct and understand a 3D scene from the properties present in 2D images. The results of this line of research become very valuable due to the fact that some moves can be very difficult to be analyzed just by physically watching a game. Thus, computer vision tools are being increasingly used in sports applications, where technical decisions in a given move can be decisive for the match. A sport that has been highly explored in terms of research development is tennis due to its growing dissemination and financial relevance, being responsible for annual turnover of R\$1.8 billion in Brazil alone. Thus, this master's dissertation work aims to develop a system based on computer vision for analyzing tennis games. The implemented system captures videos during the tennis game through cameras installed on the court and processes the images obtained, applying machine learning methods and morphological operations, in order to locate the position of the ball, the lines of the court and the location of the players after processing. In addition, the algorithm determines the moment the ball bounces during the game and analyzes whether it occurred in or out of the field. From the results obtained, the system demonstrated robustness and reliability. These data are made available to players and judges both on the court during the game and through an Android application, also developed in this work. The application aims to allow all data resulting from processing to be accessed from mobile devices, providing the results quickly and accessible to the user.

**Keywords:** Tennis match analysis, Image Processing, Computer Vision, Line Detection, Ball Point Inflection.

## LISTA DE ILUSTRAÇÕES

Figura 1	– Exemplo de aplicação da função <i>cv2.inRange</i> . (a) Imagem original de entrada. (b) Imagem obtida como saída da função <i>cv2.inRange</i> . . . . .	27
Figura 2	– Exemplo de aplicação da função <i>cv2.bitwise_and</i> . (a) Imagem original de entrada. (b) Imagem obtida como saída da função <i>cv2.bitwise_and</i> após aplicar a imagem da Figura 1b como máscara para reconhecimento da roupa laranja. . . . .	28
Figura 3	– Exemplo de aplicação da função <i>cv2.threshold</i> . (a) Imagem original de entrada. (b) Imagem processada pelo filtro <i>cv2.threshold</i> após ser convertida em escala de cinza. . . . .	28
Figura 4	– Exemplo de aplicação da função <i>cv2.erode</i> . (a) Imagem processada pelo filtro <i>cv2.threshold</i> após ser convertida em escala de cinza. (b) Imagem processada pela função <i>cv2.erode</i> após passar pelo filtro <i>cv2.threshold</i> . . . . .	29
Figura 5	– Exemplo de aplicação da função <i>cv2.dilate</i> . (a) Imagem processada pelo filtro <i>cv2.threshold</i> após ser convertida em escala de cinza. (b) Imagem processada pela função <i>cv2.dilate</i> após passar pelo filtro <i>cv2.threshold</i> . . . . .	30
Figura 6	– Exemplo de aplicação da função <i>cv2.Canny</i> . (a) Imagem processada pelo filtro convertida em escala de cinza. (b) Imagem processada pela função <i>cv2.Canny</i> após passar ser convertida em escala de cinza. . . . .	32
Figura 7	– Exemplo de aplicação da função <i>cv2.findContours</i> . (a) Imagem original de entrada. (b) Imagem original com os contornos obtidos após processamento da imagem. . . . .	34
Figura 8	– Exemplo de aplicação da função <i>cv2.pointPolygonTest</i> . (a) Ponto fora do contorno. (b) Ponto com posição pertencente ao contorno. (c) Ponto fora do contorno. . . . .	35
Figura 9	– Representações de uma reta. (a) Forma linear. (b) Forma paramétrica. . . . .	36
Figura 10	– Exemplo de aplicação da função <i>cv2.HoughLines</i> . (a) Imagem original de entrada. (b) Imagem original com as linhas obtidas após processamento da imagem. . . . .	37
Figura 11	– Representações de um círculo com centro em $(x_c, y_c)$ e raio R. . . . .	38
Figura 12	– Exemplo de aplicação da função <i>cv2.HoughCircles</i> . (a) Imagem original de entrada. (b) Imagem original com os círculos obtidos após processamento da imagem. . . . .	38
Figura 13	– Exemplo de conjunto de dados com duas dimensões. . . . .	40
Figura 14	– Exemplo de conjunto de dados com duas dimensões classificados através da Máquina de Vetor de Suporte. . . . .	40
Figura 15	– Exemplo de extração de gradientes de um imagem. (a) Imagem original. (b) Imagem após obtenção dos gradientes. . . . .	41

Figura 16 – Exemplo de histograma obtido na análise dos gradientes de uma região da imagem. . . . .	41
Figura 17 – Esquemático de um perceptron. . . . .	42
Figura 18 – Esquemático de um rede Multilayer Perceptron. . . . .	43
Figura 19 – Rede Neural Convolutacional. . . . .	44
Figura 20 – Achatamento de uma matriz de imagem 3x3 em um vetor 9x1. . . . .	45
Figura 21 – Exemplo da imagem de entrada e de um filtro de aprendizagem da CNN. (a) Imagem de entrada. (b)Filtro. . . . .	46
Figura 22 – Exemplo da convolução entre o filtro de aprendizagem com a imagem de entrada. . . . .	47
Figura 23 – Resultado da convolução entre o filtro de aprendizagem com a imagem de entrada. . . . .	47
Figura 24 – Exemplo de convolução entre imagem e filtro. . . . .	48
Figura 25 – Diferentes mapas de características obtidos de uma mesma entrada. . . . .	48
Figura 26 – Conexões entre camadas de uma CNN. . . . .	49
Figura 27 – Camada com dimensão reduzida devido ao <i>stride</i> . . . . .	50
Figura 28 – Exemplo de pooling usando operador Max. . . . .	52
Figura 29 – Exemplo de pooling usando pooling médio e max. . . . .	53
Figura 30 – Exemplo de camadas totalmente conectada. . . . .	54
Figura 31 – Representação da rede convolutacional conectada a uma rede deconvolucional. . . . .	55
Figura 32 – Exemplo da operação feita na camada de <i>unpooling</i> quando os locais de reconstrução da camada anterior são guardados. . . . .	56
Figura 33 – Representação gráfica da função ReLU. . . . .	57
Figura 34 – Exemplo da aplicação da função ReLU. . . . .	58
Figura 35 – Exemplo da aplicação da camada de transposição. (a) Convolução entre matriz de entrada $x$ e filtro $w$ com <i>zero padding</i> igual a 1 e <i>stride</i> igual a 2. (a) Deconvolução da matriz $x$ e filtro $w$ com <i>padding</i> igual a 1 e <i>stride</i> igual a 1. . . . .	59
Figura 36 – Visão lateral da posição das câmeras para aquisição das imagens de entrada para o SVM. . . . .	60
Figura 37 – Posicionamento da câmera em relação à quadra de tênis para método de detecção da posição da bola utilizando CNN, detecção das linhas da quadra e do jogador. . . . .	61
Figura 38 – Arquitetura da rede TrackNet. . . . .	63
Figura 39 – Bola se mistura com marcações do campo. . . . .	63
Figura 40 – Bola oclusa pelo jogador. . . . .	64
Figura 41 – Saída da rede TrackNet. . . . .	65
Figura 42 – Representação 2D e 3D de uma gaussiana. . . . .	65

Figura 43 – Quique da bola. . . . .	67
Figura 44 – Deslocamento da posição estimada da bola. . . . .	67
Figura 45 – Fluxograma do método de detecção da quadra. . . . .	68
Figura 46 – Posicionamento da câmera em relação à quadra de tênis. . . . .	70
Figura 47 – Linhas detectadas a partir das imagens capturadas pela câmera. . . . .	70
Figura 48 – Pontos de interseção obtidos a partir das linhas reconhecidas. . . . .	71
Figura 49 – Linhas detectadas a partir das imagens capturadas pela câmera. . . . .	71
Figura 50 – Componentes de um retângulo. . . . .	72
Figura 51 – Projeção do ponto da quadra através da utilização do ponto de simetria. . . . .	73
Figura 52 – Pontos da quadra projetados através da utilização do ponto de simetria. . . . .	73
Figura 53 – Projeção do ponto da quadra através da utilização do ponto de simetria e de reta definida através de outros pontos projetados. . . . .	74
Figura 54 – Visão da quadra do ponto de vista da câmera com as diagonais e o ponto central representados . . . . .	74
Figura 55 – Fluxograma do método de detecção do jogador. . . . .	75
Figura 56 – Representação da detecção dos jogadores durante a partida de tênis . . . . .	76
Figura 57 – Rede de Petri do aplicativo desenvolvido. . . . .	78
Figura 58 – Exemplo ligação entre <i>roscore</i> e nós para estabelecer comunicação. . . . .	80
Figura 59 – Exemplo de comunicação feita entre servidor (nó 2) e cliente (nó 1). . . . .	80
Figura 60 – Dispositivo Odroid XU4. . . . .	81
Figura 61 – Esquema de conexão entre os dispositivos necessário para funcionamento do aplicativo Android. . . . .	82
Figura 62 – Trajetoria da bola detectada através de operações do OpenCV. (a) Antes do toque no solo. (b) Depois do toque no solo. . . . .	84
Figura 63 – Trajetoria da bola detectada através do SVM. (a) Antes do toque no solo. (b) Depois do toque no solo. . . . .	85
Figura 64 – Trajetória da bola detectada com quique próximo à linha de fundo da quadra. (a) OpenCV. (b) SVM. . . . .	85
Figura 65 – Movimento da bola para o primeiro cenário. . . . .	86
Figura 66 – Movimento da bola para o segundo cenário. . . . .	86
Figura 67 – Reconhecimento errado da posição da bola. . . . .	87
Figura 68 – Exemplo de distorção apresentada pela bola em imagens capturadas durante jogo de tênis. . . . .	88
Figura 69 – Gráfico com exemplo de detecções da posição da bola. . . . .	93
Figura 70 – Exemplos de detecção da posição da bola. . . . .	94
Figura 71 – Exemplo de detecção correta do quique. . . . .	96
Figura 72 – Exemplos de erro na detecção do quique. . . . .	97
Figura 73 – Exemplo de detecção errada do quique em momento em que jogador rebate a bola. . . . .	99

Figura 74 – Exemplo de quique da bola de difícil detecção para posicionamento da câ- mera. . . . .	100
Figura 75 – Exemplos de erro na detecção das linhas da quadra. . . . .	101
Figura 76 – Exemplo de detecção correta das linhas da quadra. . . . .	102
Figura 77 – Exemplo de detecção das linhas da quadra para aplicação prática. . . .	102
Figura 78 – Representação real e detectada das linhas da quadra de tênis durante experi- mento. (a) Quadra real. (b) Quadra detectada . . . . .	104
Figura 79 – Gráfico que permite compara representação real e detectada das linhas da quadra de tênis . . . . .	105
Figura 80 – Exemplo de detecção correta dos jogadores. . . . .	106
Figura 81 – Imagem capturada pela câmera instalada na quadra com jogador detec- tado. . . . .	107
Figura 82 – Tela do aplicativo Android que permite o usuário iniciar o reconhecimento do campo. . . . .	108
Figura 83 – Tela do aplicativo Android mostrada ao usuário enquanto o aplicativo não recebe confirmação que o campo foi reconhecido. . . . .	109
Figura 84 – Tela do aplicativo Android com a imagem da quadra reconhecida com a opção "Não"selecionada. . . . .	109
Figura 85 – Tela do aplicativo Android responsável por iniciar gravação do jogo. . .	110
Figura 86 – Tela do aplicativo Android onde é possível selecionar de qual campo deseja-se visualizar o vídeo do jogo. . . . .	111
Figura 87 – Tela do aplicativo Android mostrada ao usuário enquanto o aplicativo não recebe do computador o vídeo gravado pela câmera selecionada na tela da Figura 86. . . . .	111
Figura 88 – Tela do aplicativo Android onde é possível selecionar intervalo do vídeo que deverá ser processado. . . . .	112
Figura 89 – Tela do aplicativo Android mostrada ao usuário enquanto o aplicativo não recebe do computador o vídeo processado contendo o trecho selecionado na tela da Figura 88. . . . .	112
Figura 90 – Tela do aplicativo Android que reproduz o vídeo processado. . . . .	113

## LISTA DE TABELAS

Tabela 1 – Matriz de confusão . . . . .	89
Tabela 2 – Matriz de confusão da rede considerando como corretas as detecções que apresentam apenas uma bola detectada dentro dos parâmetros especificados por imagem. . . . .	91
Tabela 3 – Métricas da rede considerando como corretas as detecções que apresentam apenas uma bola detectada dentro dos parâmetros especificados por imagem. . . . .	91
Tabela 4 – Matriz de confusão da rede considerando como corretas todas as detecções que apresentam pelo menos uma bola dentro dos parâmetros especificados detectada por imagem. . . . .	92
Tabela 5 – Métricas da rede considerando como corretas todas as detecções que apresentam pelo menos uma bola dentro dos parâmetros especificados detectada por imagem. . . . .	92
Tabela 6 – Comparação dos valores das métricas da rede TrackNet original e modificada. . . . .	93
Tabela 7 – Classificação do quique das bolas no vídeo analisado . . . . .	98
Tabela 8 – Comparação entre valores reais e detectados para os pontos que compõem as linhas da quadra . . . . .	103

## LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence (Inteligência Artificial)
BGR	Blue, Green, Red (Azul, verde, vermelho)
BSD	Berkeley Software Distribution
CNN	Convolutional Neural Network (Redes neurais convolucionais)
DNN	Deep Neural Network (Redes neurais convolucionais)
FN	Falsos Negativos
FP	Falsos Positivos
GUI	Interface Gráfica do Usuário
HSV	Hue, Saturation and Brightness (Matiz, saturação e brilho)
I/O	Input/Output (Entrada/Saída)
IHC	Interação Homem-Computador
MLP	Multilayer Perceptron (Perceptron Multicamadas)
OpenCV	Open Source Computer Vision Library
PC	Personal Computer (Computador pessoal)
PCA	Principal Component Analysis (Análise do Componente Principal)
PR	Pattern Recognition (Reconhecimento de padrões)
R-CNN	Region Based Convolutional Neural Networks (Redes Neurais Convolucionais Baseadas em Região)
ReLU	Rectified Linear Units (Unidade Linear Retificada)
STL	Standard Template Library (Biblioteca de Modelos Padrão)
SVM	Support Vector Machine (Máquina de Vetores de Suporte)
USB	Universal Serial Bus (Porta Serial Universal)
VN	Verdadeiros Negativos
VP	Verdadeiros Positivos

## SUMÁRIO

<b>1</b>	<b>Introdução</b>	<b>16</b>
1.1	Objetivos Gerais e Específicos	21
1.2	Contribuições	21
1.3	Publicações Relacionadas	23
1.4	Organização	24
<b>2</b>	<b>Fundamentação Teórica</b>	<b>25</b>
2.1	Reconhecimento de Padrões	25
<b>2.1.1</b>	<b>Extrator de Características</b>	<b>26</b>
<i>2.1.1.1</i>	<i>Biblioteca OpenCV</i>	<i>26</i>
<b>2.1.2</b>	<b>Classificação Baseada em Máquina de Vetor de Suporte</b>	<b>39</b>
<b>2.1.3</b>	<b>Classificação Baseada em Redes Neurais</b>	<b>42</b>
<i>2.1.3.1</i>	<i>Redes Neurais Convolucionais</i>	<i>43</i>
2.1.3.1.1	Camadas Convolucionais	45
2.1.3.1.2	Camadas de Pooling	51
2.1.3.1.3	Camada Totalmente Conectada	53
<i>2.1.3.2</i>	<i>Redes Deconvolucionais</i>	<i>55</i>
2.1.3.2.1	Unpooling	55
2.1.3.2.2	ReLU	57
2.1.3.2.3	Camada de Transposição Convolucional ou Convolução Transposta	58
<b>3</b>	<b>Metodologia Proposta</b>	<b>60</b>
3.1	Detecção da Bola	60
<b>3.1.1</b>	<b>Detecção Através da Utilização da Máquina de Vetor de Suporte</b>	<b>61</b>
<b>3.1.2</b>	<b>Detecção Através da Utilização de Redes Neurais Convolucionais</b>	<b>62</b>
3.2	Detecção do Quique da Bola	66
3.3	Detecção da Quadra	68
3.4	Detecção dos Jogadores	75
3.5	Interface Gráfica	76
<b>4</b>	<b>Resultados e Discussões</b>	<b>83</b>
4.1	Detecção da Bola	84
<b>4.1.1</b>	<b>Detecção Através da Utilização da Máquina de Vetor de Suporte</b>	<b>84</b>
<b>4.1.2</b>	<b>Detecção Através da Utilização de Redes Neurais Convolucionais</b>	<b>88</b>
4.2	Detecção do Quique	96
4.3	Detecção da Quadra	100
4.4	Detecção dos Jogadores	106
4.5	Descrição do Ambiente de Teste	107
<b>5</b>	<b>Conclusões e Trabalhos Futuros</b>	<b>114</b>
5.1	Conclusões	114



5.2	Trabalhos Futuros . . . . .	115
	<b>REFERÊNCIAS</b> . . . . .	<b>117</b>

## 1 Introdução

As análises no setor esportivo sempre foram de grande importância para a implementação de técnicas de treinamento e supervisão dos jogadores de forma a atingir melhores resultados. Por muito tempo os dados utilizados por analistas eram obtidos através de uma análise tardia de vídeos gravados ou a partir de anotações manuais de eventos pertinentes para um futuro estudo, tal como em (DANIEL; CHEN, 2003). A coleta desses dados de forma manual pode apresentar resultados inexatos pois a obtenção dos dados requer concentração por parte dos especialistas e alguns eventos podem acabar sendo ignorados. Outro aspecto a se destacar é a subjetividade de alguns parâmetros relevantes que não são possíveis de se determinar a olho nu, como por exemplo a velocidade da bola.

A partir deste cenário surgiram inúmeras pesquisas voltadas para o desenvolvimento de novas tecnologias que apoiassem todo o setor esportivo. Os avanços das tecnologias aplicadas no esportes tornaram-se fundamentais ao treinamento esportivo de elite, fazendo com que grandes equipes profissionais possuam um departamento de análise ou um especialista em análise dedicado a fornecer todas as informações necessárias (STEINBERG, April 2015). Esta demanda por análises abriu oportunidade para o surgimento de empresas responsáveis por fornecer serviços de consultoria para as equipes e até mesmo para traders em apostas, como por exemplo a ATASS, STATS e SmartOdds.

Dentre os métodos aplicados estão o rastreamento de sinais biomédicos, a gravação de vídeo para a verificação de erros e a implementação de sistemas automáticos que são capazes de extrair informações relevantes, além de toda a gama de acessórios e objetos utilizados como ferramentas para se implementar essa análise. A partir de sistemas inteligentes é possível rastrear e analisar os movimentos do atleta durante a prática do esporte, o que permite a adoção de melhorias de acordo com o que foi analisado para o aperfeiçoamento de sua performance (KAMBLE; KESKAR; BHURCHANDI, 2019).

Note que todos esses dados podem servir de indicadores para melhorar o desenvolvimento da equipe e o desempenho pessoal do atleta, sendo uma forma de auxiliar no desenvolvimento de novas estratégias e por consequência melhorar a performance dos jogadores durante a partida.

A busca por melhores desempenhos por parte da equipes também é alimentada pela grande quantia de dinheiro que vem sendo investida no ramo esportivo. Segundo um estudo publicado pela empresa Sports Value, estima-se que o mercado de esportes global movimenta um valor em torno de US\$ 756 bilhões por ano, considerando todas as diferentes modalidades (VALUE, 2020). Esse valor considera os salários dos jogadores, investimento em publicidade, bilheteria, patrocínios, vendas de jogadores, apostas, etc.

Um exemplo da aplicação desse tipo de tecnologia pode ser visto no clube de futebol

S. L. Benfica, que utiliza quase uma dúzia de sensores em cada um dos seus jogadores de forma a coletar dados que permitam observar o desempenho do jogador durante a partida. Utilizando esses dados em análises por meio de plataformas como o Microsoft Azure Cloud, a equipe é capaz de direcionar o jogador com treinamentos personalizados para melhorar sua saúde e desempenho. Desta forma, o jogador que inicialmente entrou na equipe como um jovem talento com muito a ser aperfeiçoado é vendido para outros times por valores 10 ou 20 vezes o valor original. (ANTHONY, 2017)

Nos últimos anos, técnicas de aprendizado de máquina (SALEH, 2018) têm sido utilizadas em análises esportivas, tal como visto em (BUNKER; THABTAH, 2019) e (HAAREN et al., 2013). Essas técnicas têm como objetivo o fornecimento de uma perspectiva estatística para o desempenho do jogador ou fazendo previsões com base em registros históricos. Durante a última década os métodos baseados em aprendizado profundo (ou do inglês, Deep Neural Network - DNN) têm levado continuamente o desempenho da representação visual a um estágio superior (AGGARWAL, 2018), sendo utilizadas especialmente na detecção e classificação de objetos (JIANG et al., 2019). Em relação ao contexto do mundo dos esportes, o trabalho de (CASTRO; CANOSA, 2019) apresenta uma abordagem híbrida baseada em DNN e técnicas tradicionais de rastreamento de objetos para rastrear um jogador de futebol individual em uma sequência de vídeo.

A aplicação de inteligência artificial nos esportes tem se tornado cada vez mais frequente por causa da crescente necessidade de se implementar aplicações que analisassem automaticamente partidas para um resultado final preciso, determinassem se uma jogada duvidosa foi válida, gerasse dados estatísticos mais confiáveis e de rápido acesso. A Inteligência Artificial (do inglês, *Artificial Intelligence* - AI) (NORVIG; RUSSELL, 1994) tem desempenhado um papel muito importante no desenvolvimento tecnológico atual. Um dos grandes objetivos deste campo de pesquisa é fazer com que as máquinas tenham uma percepção de mundo de forma semelhante à dos humanos, e com isso, serem capazes de realizar diversas tarefas complexas de forma rápida e eficaz. (ALATAS, 2019) Diversos trabalhos publicados propõem soluções que aplicam visão computacional visando o processamento de jogos com o objetivo de auxiliar os competidores, treinadores e públicos, assim como (CHEN; LITTLE, 2017), (MANAFIFARD; EBADI; MOGHADDAM, 2017), (CANT et al., 2020), (LIN; YU; HUANG, 2020) e (GOMEZ-GONZALEZ et al., 2019). Dentre eles é possível citar (YANG et al., 2011) e (THOMAS et al., 2017) que fazem uma revisão do estado da arte em algoritmos de rastreamento visual. Dentre os tópicos abordados destacam-se a análise de como os jogadores e o grupo de jogadores se movem, a detecção da posição dos jogadores em um determinado momento, a identificação das principais fases do jogo para estatísticas, entre outros.

E assim, dentro deste contexto, uma parte de grande importância no campo de AI é a visão computacional (CIPOLLA; FARINELLA; BATTIATO, 2012). A visão computacional tem como objetivo o processamento de imagens e vídeos por determinados

modelos a fim de reproduzir a visão humana usando *softwares e hardwares*. Uma das áreas deste campo de pesquisa tem como finalidade reconstruir e compreender uma cena 3D a partir das propriedades presentes em imagens 2D. A visão computacional é utilizada em diversas áreas de pesquisa, tais como em inspeção de grandes estruturas (PINTO et al., 2020), navegação autônoma (COELHO et al., 2018), vigilância (PINTO et al., 2017), determinação de possíveis alterações estruturais (MELO et al., 2020), entre outros.

A visão computacional desempenha um papel fundamental no mundo dos esportes. Por exemplo, algumas das áreas de aplicação atuais mais conhecidas são as análises de esportes para transmissão a fim de mostrar tanto a posição dos jogadores quanto da bola, permitindo que as jogabilidades e técnicas sejam explorados em detalhes por um apresentador de TV (THOMAS, 2011). O mesmo se aplica às vias de transmissão esportiva de jogos ao vivo, onde os comentaristas se utilizam do histórico das equipes e dos jogadores para trazer informações interessantes ao espectador. Tal como detalhado em (THOMAS et al., 2017), no setor esportivo, os movimentos acontecem de forma muito rápida e tornam a análise de movimentos e estatísticas importantes difíceis de serem feitas sem auxílio computacional. Sendo assim, as ferramentas de visão computacional vêm sendo cada vez mais utilizadas em aplicações esportivas, onde as decisões técnicas tomadas pelos juízes em um lance podem ser decisivas para a partida. Os resultados dessa linha de pesquisa se tornam muito valiosos pois alguns lances podem ser muito difíceis de se analisar apenas assistindo fisicamente a uma partida uma vez que, dependendo do esporte, a velocidade das bolas podem chegar a mais de 200 Km/h, tornando-se praticamente impossível determinar qual fora sua trajetória exata (THOMAS et al., 2017).

A integração dos sistemas de visão computacional e eletrônica com o mundo dos esportes tem resultado em avanços nas formas em como o esporte e o esportista são analisados. Essa integração se faz relevante não somente no âmbito da arbitragem, mas também na geração de estatísticas. Por exemplo, em (STEIN et al., 2017), os autores aplicaram técnicas avançadas de análise de trajetória e movimento para gerar medidas analíticas relevantes para melhorar a análise de uma equipe em jogo.

O rastreamento tem sido foco de estudo da área de visão computacional nos últimos anos, sendo considerado como uma das tarefas essenciais dessa área de estudo, como em (SILVEIRA, 2013), (MACIEL et al., 2019) e (WU; LIM; YANG, 2013). Em (LIU et al., 2009) é feito um reconhecimento dos jogadores utilizando recursos Haar (VIOLA; JONES, 2004) aplicados em cascata. Para realizar o treinamento usaram jogadores em poses diferentes como amostras positivas e o plano de fundo como um amostra negativa. Como feito em (TEACHABARIKITI; CHALIDABHONGSE; THAMMANO, 2010), existem pesquisas cujo objetivo é a detecção e classificação de objetos. No trabalho são utilizadas técnicas de visão computacional para detectar jogadores e suas posições durante a partida. Existem trabalhos que exploram a obtenção resultados de uma forma mais complexa, como é o caso de (MESSELODI et al., 2019) que implementa um sistema baseado em visão

computacional que é capaz de detectar a posição 3D da bola e dos jogadores e fornece mais informações como velocidade da bola e a estimativa da região de contato da bola na quadra. Em (ZHANG; SONG; SONG, 2007), o objetivo foi rastrear o movimento de uma bola de pingue-pongue utilizando uma metodologia de medição visual e com isso prever seu ponto de impacto. Duas câmeras de alta velocidade são usadas para determinar a posição 3D da bola e estimar sua trajetória.

Devido à capacidade de extração de características e capacidade de generalização, as redes CNN superaram métodos tradicionais, conforme visto nos trabalhos de (TRAN et al., 2015) e (SIMONYAN; ZISSERMAN, 2014a). Recentemente, métodos baseados em aprendizado profundo (GOODFELLOW; BENGIO; COURVILLE, 2016) ou modelos baseados em redes neurais estão sendo amplamente usados em tarefas complexas de visão computacional. Como exemplo a ser citado, (LE et al., 2011) utilizou uma CNN tridimensional e uma Máquina de Vetores de Suporte (Support Vector Machine - SVM) (VAPNIK, 1963) para reconhecer ações humanas em vídeos. Além disso, (TRAN et al., 2015) projetou uma CNN para extrair recursos de vídeo que foram posteriormente enviados a um SVM para reconhecimento de ações. Em (YAN et al., 2014), os autores extraíram recursos dos vídeos para treinar um modelo para detecção dos destaques que aparecem nos vídeos. Similarmente, em (OTANI et al., 2017), os autores utilizaram uma CNN para treinar um modelo para encontrar destaques em vídeos.

Um esporte que vem sendo altamente explorado em termos de desenvolvimento de pesquisas é o tênis. O esporte está entre os 10 mais praticados e disseminados no Brasil, contando com cerca de 2 milhões de praticantes, 370 torneios por ano e 33.675 jogadores registrados na Confederação Brasileira de Tênis (CBT). Além disso, o tênis é reponsável por movimentar anualmente R\$1,8 bilhão no Brasil, considerando valores relativos à prática do tênis, gastos para a realização dos torneios, patrocínios aos atletas e gastos da mídia. (BRASIL, 2020)

Diversas pesquisas estão sendo realizadas com foco nesse esporte devido à sua relevância a nível global e pela grande quantidade de dados históricos disponíveis que permitem a construção de modelos estatísticos para prever os resultados das partidas. Outro fatores que colaboram para a grande quantidade de estudos no tênis são as características intrínsecas à realização do jogo, por exemplo:

- Espaço físico de realização do jogo compreendido em uma área limitada;
- Sistema de pontuação faz com que os resultados possíveis sejam facilmente mapeados;
- Jogadores ficam posicionados na quadra em espaços definidos, tornando mais simples a detecção de suas interações.

Dentre as tecnologias utilizadas no contexto do tênis pode-se destacar os projetos desenvolvidos em (KOS; KRAMBERGER, 2017) para detectar e gravar o movimento e informações biométricas dos tenistas durante a partida; em (RAYMOND; MADAR; MONTOYE, 2019) com propostas de relógios inteligentes que permitem acessar dados como velocidade da batida na bola, encontrar outros jogadores, comparar resultados e registrar os pontos de cada jogo; em (MYERS et al., 2019) com o desenvolvimento de raquetes que permitem a integração com sensores e reconhece as interações da raquete com a bola durante uma partida, a velocidade que a bola bateu e o efeito e a velocidade que a bola pegou;

Existem também vários trabalhos que utilizam de visão computacional em aplicações voltadas para a captura de dados e detecção de parâmetros do jogo de tênis. Em (MAO, 2006) implementam-se diversas técnicas que tem por base a subtração de fundo com reconhecimento de cor para realizar o rastreamento automático em tempo real da bola de tênis e compara os métodos criados com os resultados obtidos a partir de classificadores Haar (VIOLA; JONES, 2004). Um dos mais famosos e largamente difundidos é o sistema Hawk-eye desenvolvido em (OWENS; HARRIS; STENNETT, 2003). Este sistema é a plataforma de rastreamento mais utilizada em competições profissionais de tênis e é oficialmente utilizada na decisão de jogadas controversas, mostrando o caminho mais provável da bola de acordo com a trajetória capturada pelas várias câmeras de alta performance. Seu sistema complexo e altamente sofisticado apresenta um alto custo de implementação devido às várias câmeras de alta velocidade e ao computador com alto poder de processamento que é necessário para implementá-lo. Isso tem motivado outros estudos na tentativa de criar sistemas que apresentem alta precisão com baixo custo de implementação, como (RENÒ et al., 2018) que utiliza redes neurais convolucionais para obter as posição da bola. Já (HUANG et al., 2019) utiliza um sistema baseado em aprendizagem profunda com a aplicação de redes neurais convolucionais e deconvolucionais para rastrear objetos pequenos e de alta velocidade. Os autores utilizaram vídeos de tênis e badminton para testar a eficiência do sistema. Os resultados obtidos por este trabalho são bastante precisos e possuem qualidade tão boa quanto ao sistema Hawk-eye (OWENS; HARRIS; STENNETT, 2003). É possível notar que diferentemente deste trabalho de mestrado, os autores somente se preocuparam em fazer o rastreamento da bola. Desta forma, no presente trabalho, as entradas da rede são modificadas para utilizar imagens gravadas durante o jogo. Assim, o usuário será capaz de selecionar o período no qual deseja processar as informações pertinentes ao jogo. Além de serem fornecidas as informações sobre a posição da bola, o sistema desenvolvido neste trabalho fornece também informações sobre a posição dos jogadores e das linhas da quadra, o que facilita a identificação de pontuações no jogo.

## 1.1 Objetivos Gerais e Específicos

O principal objetivo deste trabalho é produzir um sistema capaz de detectar vários aspectos importantes durante uma partida de tênis por meio de ferramentas de visão computacional, de modo a prover assistência ao árbitro ou técnico durante partidas de tênis com anotações de dados sobre jogadas. Dentre os parâmetros detectados, podem-se citar a posição da bola de tênis durante a partida, os momentos em que a mesma quica na quadra, as linhas que compõe a quadra, o reconhecimento do quique da bola para saber se a mesma tocou dentro ou fora da quadra, e por fim, detectar os jogadores em campo. Para disponibilizar tais resultados de uma forma iterativa e dinâmica aos usuários, desenvolveu-se neste trabalho um aplicativo no ambiente Android Studio responsável por compilar todos esses resultados e permitir o acesso aos dados processados em dispositivos móveis.

## 1.2 Contribuições

Outras contribuições deste trabalho de pesquisa podem ser resumidas da seguinte maneira:

- **Comparação de duas técnicas de classificação para detecção da posição da bola de tênis**

No trabalho exposto em (HUANG et al., 2019) foi desenvolvido uma rede de deep learning baseada em CNN chamada TrackNet, que tem por objetivo detectar o posicionamento de bolas pequenas que atingem altas velocidades em vídeos de esportes. Esta rede neural é formada pela junção de duas redes, convolucional e deconvolucional. As treze primeiras camadas são referentes às camadas convolucionais da rede neural convolucional e as 12 últimas camadas são referentes às camadas de transposição convolucional da rede deconvolucional. A rede utiliza camadas de upsampling que possuem a função de recuperar a perda de informação gerada pelas camadas de pooling. Seus resultados de rastreamento e detecção da posição da bola durante o jogo de tênis têm alta qualidade, permitindo uma detecção comparável aos resultados profissionais fornecidos pelo sistema Hawk-eye (OWENS; HARRIS; STENNETT, 2003). No trabalho em que foi publicada, a rede mostrou em seus experimentos uma precisão de 99.8%, chegando a encontrar a posição de bolas de difícil detecção como as bolas que estavam borradas na imagem e até mesmo bolas oclusas. A entrada é composta pela concatenação de três frames consecutivos de um vídeo e a saída é um mapa de calor que é dimensionado em uma distribuição gaussiana 2D no centro da bola de tênis.

O presente trabalho faz uma diversificação dos métodos utilizados com o propósito de encontrar os melhores resultados na detecção e classificação do quique da bola.

Inicialmente, a determinação da posição de toque da bola é realizada através da aplicação de filtros de cores e operações morfológicas na imagem, atreladas à utilização de técnicas como Máquina de Vetores de Suporte (Support Vector Machine - SVM). Com esses processamentos, foi possível determinar a posição da linha e fazer o rastreamento da posição da bola.

Com o objetivo de melhorar o desempenho dos resultados obtidos, foi implementada a rede TrackNet (HUANG et al., 2019), com adaptações em relação à forma como os dados são obtidos antes da entrada da rede, onde ao invés da entrada da rede ser o vídeo de um jogo inteiro gravado anteriormente, usaremos imagens gravadas durante a realização do jogo, onde o usuário poderá selecionar o intervalo que deseja processar e este será mostrado logo após a sua efetiva realização, fornecendo aos jogadores as informações sobre a posição da bola no instante desejado.

- **Desenvolvimento de um algoritmo computacional de detecção das linhas do campo**

A detecção das linhas do campo de tênis apresentam alguns desafios para sua correta detecção. A quadra que será objeto de pesquisa deste trabalho é composta por saibro, um material que é tradicionalmente preparado com uma mistura de areia, pedra e argila, resultando numa coloração alaranjada. Apesar de proporcionar jogos mais lentos devido à sua capacidade de absorção de impactos, a quadra de saibro apresenta um inconveniente. Durante a partida de tênis, o saibro se desprende do solo, depositando em cima das linhas delimitantes do campo. Essa característica não afeta de nenhuma forma o andamento do jogo, porém em um contexto de detecção das linhas através de visão computacional, esse aspecto não permite que os mesmos métodos simples de detecção utilizadas em outras quadras de piso duro ou grama sejam utilizadas com o saibro. O pó depositado nas linhas torna mais difícil a detecção das linhas que estão mais longe da câmera uma vez que elas não possuem uma boa visibilidade.

O presente trabalho consegue ultrapassar esse impedimento através da criação de um método de uma detecção que não necessita detectar todas as linhas do campo, mas somente as linhas do campo que estiver mais próximo à câmera. Detectados os pontos que compõem as linhas do campo mais próximo, o passo seguinte é a projeção dos pontos da outra quadra através dos pontos já obtidos e assim a quadra estará detectada da forma desejada. Esse tipo de detecção é uma forma robusta de detecção e pode ser utilizado não somente para o saibro, mas também em qualquer outro tipo de campo. A detecção completa do campo permite que análises sejam feitas, como por exemplo a detecção do momento em que a bola toca o chão da quadra e se houve pontuação por parte de algum dos jogadores.



- **Desenvolvimento de um algoritmo computacional de detecção dos jogadores em quadra**

O método de reconhecimento dos jogadores utilizado é realizado através de operações morfológicas na imagem original de forma a ressaltar a posição do jogador.

Na literatura a maioria das técnicas utilizadas realizam o reconhecimento através de processos de deep learning. A vantagem da abordagem proposta neste trabalho é a obtenção de resultados excelentes de detecção com menor custo computacional e tempo de processamento se comparados ao que seria necessário caso houvesse a adoção de sistemas mais robustos de reconhecimento.

- **Desenvolvimento de um aplicativo para ser utilizado em dispositivos móveis para analisar partidas de tênis**

Foi desenvolvido um aplicativo Android que compila todos os métodos desenvolvidos no presente projeto, de forma a proporcionar aos jogadores e aos juízes uma plataforma interativa e compatível com dispositivos móveis que possibilita assim um fácil acesso aos resultados do processamento do jogo de tênis.

No aplicativo, o usuário poderá acionar o reconhecimento das linhas da quadra, verificar o resultado deste processamento e, caso não fique satisfeito com o resultado apresentado, poderá solicitar o reprocessamento da quadra para todos os limites estejam de acordo com o desejado. No passo seguinte o usuário deverá indicar o início da jogada para que o tempo de jogo comece a ser computado e para que as câmeras retornem a gravação apenas do período de interesse da partida. Quando for necessário a verificação de algum momento do jogo, o usuário poderá escolher a câmera da qual deseja acessar a gravação e em seguida o vídeo da câmera selecionada é mostrado ao usuário. Logo depois, uma tela permite que seja feita a seleção do intervalo de jogo em que ocorreu o quique da bola e que será feita a verificação. Esse seguimento do vídeo é processado e mostrado logo em seguida, detalhando as posições da bola, do campo e do jogador no momento desejado. Após a visualização dos dados, o usuário pode optar por visualizar o lance através de uma outra câmera ou voltar ao menu para começar uma nova jogada.

### 1.3 Publicações Relacionadas

Este trabalho de pesquisa resultou até o presente momento em um artigo publicado em uma revista internacional, conforme, listado abaixo:

- "Low-cost Trajectory-Based Ball Detection for Impact Indication and Recording", Aurelio G. Melo, Milena F. Pinto, Andre L. M. Marcato, Iago Z. Biundini, e Nayara M. S. Rocha, J Control Autom Electr Syst (2021).

A principal contribuição deste trabalho no desenvolvimento do artigo foi com relação à utilização dos métodos do SVM e OpenCV para a detecção da bola de tênis.

#### 1.4 Organização

Além deste capítulo introdutório, essa dissertação de mestrado é dividida em mais quatro capítulos.

O Capítulo 2 apresenta uma revisão teórica das técnicas utilizadas neste trabalho, bem como os trabalhos relacionados, destacando o estado da arte em sistema de visão aplicados em esportes.

O Capítulo 3 detalha a metodologia proposta e seus fundamentos matemáticos. Neste capítulo é explicado também toda a parte de detecção da quadra, dos jogadores e da bola, e reconhecimento se houve ou não pontuação.

O Capítulo 4 mostra os experimentos com uma discussão apropriada dos resultados, demonstrando a eficácia e robustez do sistema proposto.

E por fim, as observações finais até o presente momento e trabalhos futuros para o algoritmo desenvolvido são apresentados no Capítulo 5.

Sugere-se ao leitor que possua fundamentação teórica nos tópicos relacionados à processamento de imagem e *deep learning* que avance diretamente para o Capítulo 3, desconsiderando o Capítulo 2 (fundamentação teórica). Ao leitor que desejar entender as teorias que fundamentam as técnicas de processamento aqui utilizadas, sugere-se que este prossiga com a leitura do Capítulo 2.

## 2 Fundamentação Teórica

Nas próximas seções serão abordados com mais profundidade conceitos relevantes para a construção dos processos de detecção propostos neste trabalho.

A seção sobre a biblioteca OpenCV detalha o funcionamento das manipulações morfológicas usadas durante as etapas de detecção da quadra e do jogador.

A seção sobre a classificação baseada em Máquina de Vetor de Suporte detalha o funcionamento deste método que será utilizado como uma das alternativas para a detecção da posição da bola de tênis.

A seção com detalhamento em redes neurais convolucionais e deconvolucionais serão fundamentais para o entendimento do funcionamento e de como é composta a rede TrackNet (HUANG et al., 2019) que será utilizada neste trabalho.

### 2.1 Reconhecimento de Padrões

O reconhecimento de padrões (do inglês *pattern recognition* - PR) é responsável por realizar a classificação e descrição de objetos. Sendo assim, o reconhecimento de sinais permite a aquisição de informações de alto nível referentes a um determinado registro, variando de acordo com cada aplicação, ou seja, essas informações podem ser imagens, sinais em forma de ondas, ou qualquer tipo de medida que precisa ser classificada (SCHALKOFF, 2007). PR é um pré-requisito para um comportamento inteligente, e é caracterizado como um processo de redução, mapeamento ou rotulação das informações. Portanto, é utilizado nas mais distintas áreas. Como exemplo, na análise de imagens, o reconhecimento de determinados padrões pode ser utilizado nas finanças (FRANÇOIS-LAVET et al., 2018), biologia (CHEN et al., 2018), agricultura (PATRÍCIO; RIEDER, 2018), automação industrial (GRILO; FIGUEIREDO, 2018), acessibilidade (HUANG et al., 2018), entre outros.

Basicamente, um sistema completo de PR consiste de sensores capazes de registrar as informações do ambiente, além de um mecanismo capaz de determinar quais informações serão extraídas dessas aquisições. De uma forma abrangente, um projeto de PR envolve:

- Extração de características dos objetos a classificar (ou a descrever);
- Seleção das características mais discriminativas;
- Construção de um classificador (ou descritor).

### 2.1.1 Extrator de Características

Segundo (SOUZA; PISTORI, 2005), a visão computacional é uma área de pesquisa que permite realizar a extração de características que representam um aspecto importante da imagem, tendo algum parâmetro específico como norte na busca dessa informação, com o objetivo principal de discriminar e identificar essa característica. Uma ferramenta muito utilizada na extração de características e identificação de parâmetros é a biblioteca OpenCV que será utilizada no presente trabalho.

#### 2.1.1.1 Biblioteca OpenCV

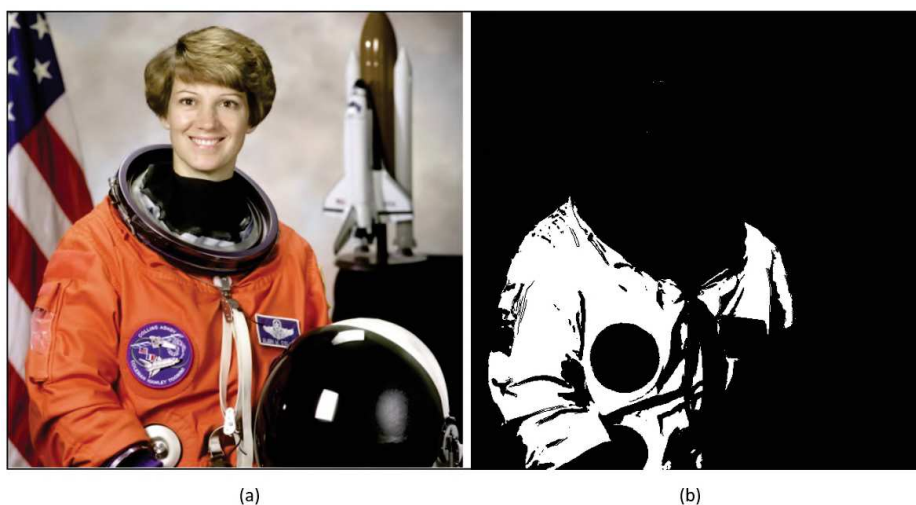
A biblioteca OpenCV, acrônimo para *Open Source Computer Vision Library*, é uma ferramenta gratuita de visão computacional que permite realizar análise, interpretação e processamento de imagens (BRADSKI; KAEHLER, 2012). Esta biblioteca foi criada em 1999 por Garry Bradski com o objetivo de acelerar a visão computacional e a inteligência artificial, criando uma estrutura simples para auxiliar toda a comunidade de visão computacional (REAL-TIME. . . , ).

O OpenCV é uma biblioteca multiplataforma para a criação de aplicações na área de visão computacional que possui livre utilização em trabalhos acadêmicos e comerciais, desde que siga o modelo de licença BSD Intel (ALBUQUERQUE, 2016). Essa biblioteca contém mais de 500 funções que atendem a diversos tipos de aplicações, como por exemplo: Interação Homem-Computador (IHC), identificação de objetos, reconhecimento de face, inspeção de produtos de fábrica, processamento de Imagens e Video I/O, imagens médicas, rastreamento, reconhecimento de movimentos, segurança, Interface Gráfica do Usuário (do inglês, graphical user interface - GUI), calibração de câmera, visão estéreo, robótica, entre outros (KAEHLER; BRADSKI, 2016). Dentre as operações existentes, este trabalho destaca abaixo algumas dentre as que foram mais pertinentes durante o desenvolvimento deste trabalho de pesquisa. Para demonstrar uma aplicação prática das funções abaixo foram utilizadas as imagens retiradas do livro (KAPUR, 2017) e alguns dos filtros serão aplicados pela própria autora:

- ***Extração de região a partir de limiares de cor***

A função *cv2.inRange* é utilizada para destacar uma região que está dentro dos limiares de cor que é escolhido pelo usuário. Ela possui três parâmetros de entrada: uma imagem de entrada e dois arranjos de uma linha e três colunas. O primeiro arranjo se refere à cor HSV que representa o limite inferior e o segundo arranjo se refere à cor HSV do limite superior da região do limiar. Um pixel será definido como 255 se estiver dentro dos limites especificados e, caso contrário, será definido como 0. A saída retornada pela função é uma imagem preto e branca dentro dos limiares estabelecidos. (VILLAN, 2019)

Figura 1 – Exemplo de aplicação da função *cv2.inRange*. (a) Imagem original de entrada. (b) Imagem obtida como saída da função *cv2.inRange*.



Fonte: próprio autor.

O exemplo abaixo mostra a aplicação da Figura 1 (a) na função *cv2.inRange*. Os parâmetros são as cores correspondentes aos tons máximo e mínimo na escala BGR, de forma a reconhecer a roupa laranja na imagem original. Os arranjos que contém o limite inferior e superior são  $[30,63,184]$  e  $[110,147,255]$ , respectivamente. O resultado da função para estes parâmetros é apresentado na Figura 1 (b).

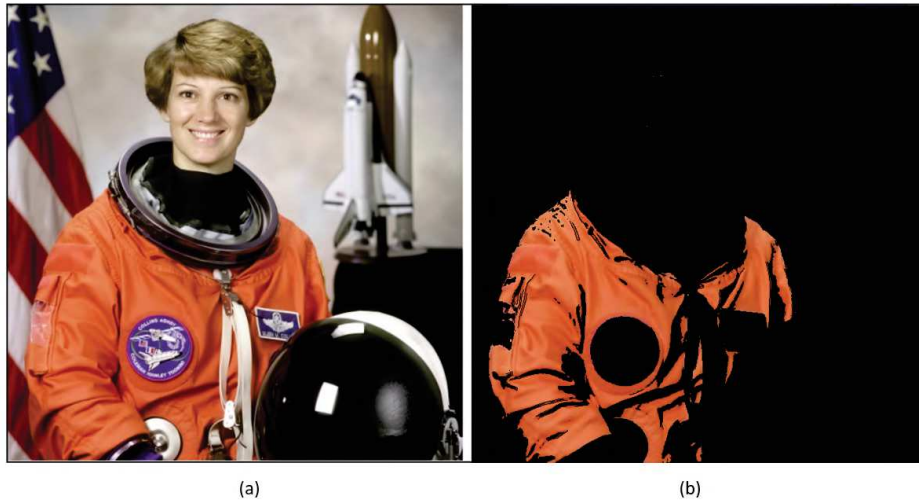
- ***Conjunção bit a bit de elementos da matriz de entrada***

Algumas operações são capazes de serem executadas em nível de bit usando operadores bit a bit. Essas operações podem manipular os valores para comparações e cálculos. As operações bit a bit incluem AND, OR, NOT e XOR. (VILLAN, 2019)

A função *cv2.bitwise\_and* fará uma operação bit a bit entre duas imagens de entrada e uma máscara binária. A função calculará a conjunção de duas imagens de entrada e, em seguida, aplicará uma máscara binária no resultado dessa conjunção. Desta forma, a função retornará uma imagem que possui os pixels resultante da combinação das duas imagens apenas na parte onde seus pixels correspondentes na máscara binária forem brancos. Assim, é possível extrair o intervalo de cores necessário.

No exemplo abaixo, a Figura 2 (a) foi tomada como parâmetro para as duas imagens de entrada, pois nesse momento, não estamos interessados na conjunção de duas imagens diferentes, mas sim na aplicação da máscara na imagem original. Assim, a imagem original foi usada duas vezes como parâmetro, juntamente com a máscara da Figura 1 (b). O resultado da aplicação da função para estes parâmetros é apresentado na Figura 2 (b).

Figura 2 – Exemplo de aplicação da função *cv2.bitwise\_and*. (a) Imagem original de entrada. (b) Imagem obtida como saída da função *cv2.bitwise\_and* após aplicar a imagem da Figura 1b como máscara para reconhecimento da roupa laranja.

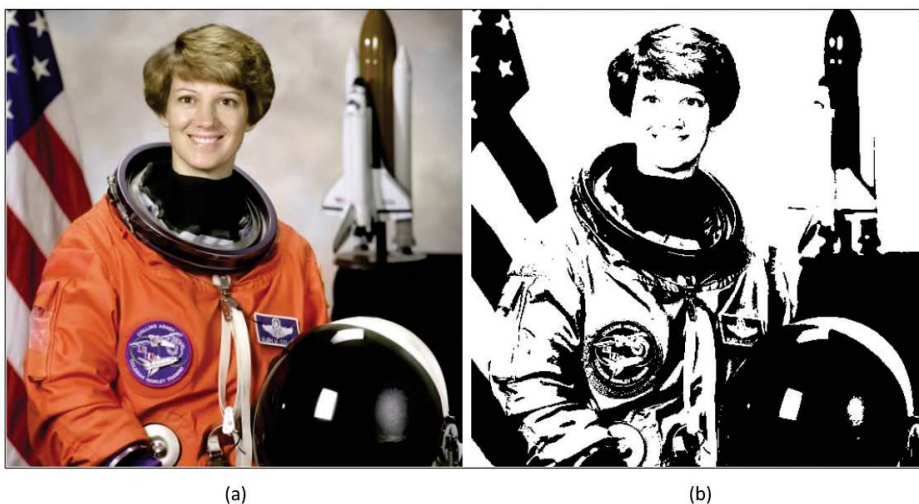


Fonte: próprio autor.

- ***Aplicação de limiar na imagem***

A função *cv2.threshold* aplica um valor limite para classificar as intensidades de pixel na imagem em tons de cinza. Caso o valor do pixel seja maior de que este limiar, ele será configurado como branco e, caso contrário, como preto. Essa descrição funciona quando o plano de fundo da imagem é uniforme. Quando isso não acontece, são usados dois parâmetros de entrada da função, que definem limites máximo e mínimo para a função ao invés de definir apenas um limite global (KAPUR, 2017).

Figura 3 – Exemplo de aplicação da função *cv2.threshold*. (a) Imagem original de entrada. (b) Imagem processada pelo filtro *cv2.threshold* após ser convertida em escala de cinza.



Fonte: (KAPUR, 2017)

No exemplo da imagem acima, a Figura 3 (a) foi convertida para uma escala de cinza e todos os pixels que estão acima de 120, são definidos como brancos e os outros são definidos como pretos. O resultado é apresentado na Figura 3 (b).

- **Erosão**

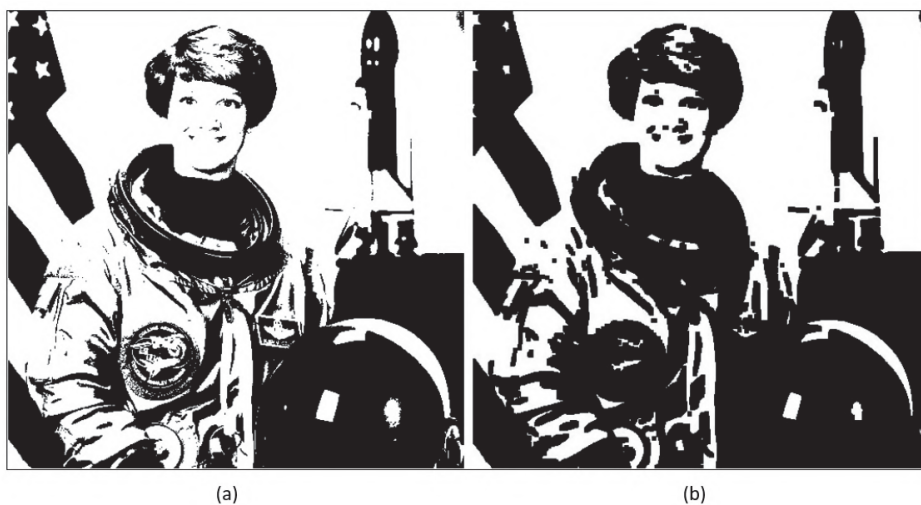
A ideia básica da função *cv2.erode* pode ser baseada na erosão do solo, ou seja, ela desgasta os limites do objeto em primeiro plano, e o kernel desliza pela imagem (como na convolução 2D). Um pixel na imagem original (1 ou 0) será considerado 1 apenas se todos os pixels sob o kernel forem 1, caso contrário, ele é corroído (reduzido a zero). A entrada da função é dada por uma imagem, um kernel e o número de iterações. A operação de erosão é utilizada para eliminar ruídos de uma imagem. O objetivo é reduzir os ruídos e não afetar as regiões maiores que contêm conteúdo visualmente significativo (KAEHLER; BRADSKI, 2016).

Durante a execução da função *cv2.erode*, o valor de um ponto  $p$  é definido a partir do valor mínimo de todos os pontos cobertos pelo kernel quando estão alinhados em  $p$ . A Equação 2.1 determina o valor dos pixels da imagem.

$$\text{erosão}(x, y) = \min_{(i,j) \in \text{kernel}} \text{src}(x + i, y + j) \quad (2.1)$$

No exemplo da imagem abaixo, a Figura 4 (a) obtida na saída da função *cv2.threshold* é utilizada como entrada da função *cv2.erode* juntamente com um kernel formado por uma matriz 5x5 com todos os elementos da matriz valendo 1. Figura 4 (b) apresenta o resultado de uma iteração com esse processamento.

Figura 4 – Exemplo de aplicação da função *cv2.erode*. (a) Imagem processada pelo filtro *cv2.threshold* após ser convertida em escala de cinza. (b) Imagem processada pela função *cv2.erode* após passar pelo filtro *cv2.threshold*.



- **Dilatação**

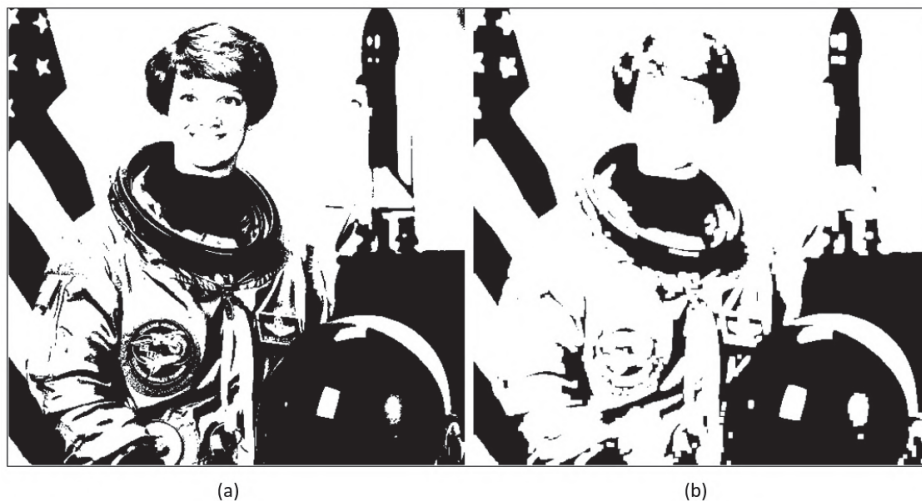
Assim como na função *cv2.erode*, a entrada da função *cv2.dilate* é dada por uma imagem, um kernel e o número de iterações. A operação de dilatação é usada para encontrar componentes conectados, ou seja, grandes regiões discretas de cor ou intensidade de pixels semelhantes, em casos onde há uma grande região que pode ser dividida em vários componentes como resultado de ruído, sombras ou algum outro efeito semelhante. Uma pequena dilatação fará com que esses componentes se unam novamente (KAEHLER; BRADSKI, 2016).

Para o operador *cv2.dilate*, a equação tem a mesma forma da função *cv2.erode*, exceto pelo fato de que max é considerado em vez de min. Assim:

$$\text{dilatação}(x, y) = \max_{(i,j) \in \text{kernel}} \text{src}(x + i, y + j) \quad (2.2)$$

No exemplo da imagem abaixo, a Figura 5 (a) obtida na saída da função *cv2.threshold* é utilizada como entrada da função *cv2.dilate* juntamente com um kernel formado por uma matriz 5x5 com todos os elementos da matriz valendo 1. A Figura 5 (b) apresenta o resultado de uma iteração com esse processamento.

Figura 5 – Exemplo de aplicação da função *cv2.dilate*. (a) Imagem processada pelo filtro *cv2.threshold* após ser convertida em escala de cinza. (b) Imagem processada pela função *cv2.dilate* após passar pelo filtro *cv2.threshold*.



Fonte: (KAPUR, 2017)

- **Detecção de bordas**

Um conceito muito importante no processamento de imagens é a detecção de bordas. Elas permitem saber mais sobre os limites dos objetos em uma imagem e a forma como estes estão estruturados. As bordas são definidas como sendo as regiões onde há uma mudança significativa nos valores dos pixels quando a imagem é percorrida,



ou seja, quando uma imagem é processada e seus pixels são percorridos a borda será evidenciada pela brusca mudança nos valores armazenados pelos pixels (KAPUR, 2017). Uma das técnicas utilizadas para detecção de bordas é o método de Canny desenvolvido por (CANNY, 1986). Como apresentado em (LAGANIERE, 2017) e (DEY, 2018), a detecção é feita após a imagem passar pelas seguintes etapas:

– *Redução de ruído*

O processo de detecção de bordas é susceptível a ruídos presentes na imagem pois se estas apresentarem muito ruído, produzirão bordas falsas, o que não é desejável. Assim, a primeira etapa é a aplicação de um filtro gaussiano que removerá o ruído. Esse filtro utiliza um kernel gaussiano de dimensão (5,5) que fará convolução com a imagem.

– *Cálculo do Gradiente de Intensidade da Imagem*

A imagem suavizada é filtrada novamente com um kernel Sobel, que é uma operação conjunta de suavização e diferenciação Gaussiana na qual pode-se escolher uma direção a ser tomada. Para a detecção de Canny são tomadas as duas direções, horizontal e vertical, e para cada operação se obtém duas imagens, uma com a primeira derivada na direção horizontal ( $G_x$ ) e a outra com a primeira derivada na direção vertical ( $G_y$ ). Assim, é possível obter a magnitude  $|G|$  a partir da Função 2.3 ou da Função 2.4. A função de  $|G|$  adotada será configurada pelo usuário. Já a direção do gradiente dos pixels  $G$  é dada por através de  $\theta$ , que é definido pela Equação 2.5.

$$|G| = \sqrt{G_x^2 + G_y^2} \quad (2.3)$$

$$|G| = |G_x| + |G_y| \quad (2.4)$$

$$\theta = \tan^{-1} \left( \frac{G_y}{G_x} \right) \quad (2.5)$$

A direção do gradiente será sempre perpendicular às arestas.

– *Supressão não máxima*

A próxima etapa faz uma varredura na imagem para remover pixels que podem não pertencer à borda, verificando se o pixel é um máximo local da sua vizinhança na direção do gradiente, caso contrário, é suprimido (ou seja, configurado em zero).

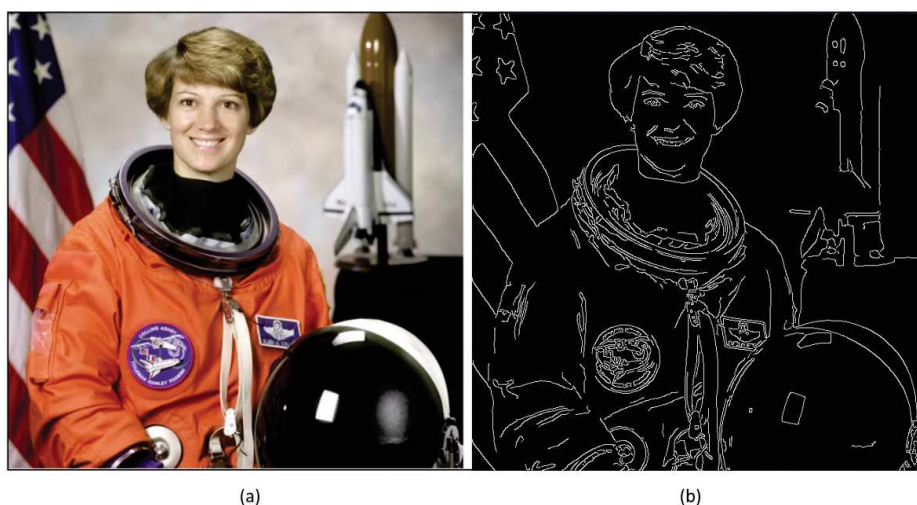
– *Limiar de histerese*

A etapa anterior resulta em uma imagem com todas as arestas da imagem, sem nenhum parâmetro para definir quais delas são bordas realmente.

Para decidir quais delas são bordas serão utilizados dois limiares (máximo e mínimo), denominados de `minVal` e `maxVal`. As arestas que possuem o módulo do gradiente com valor maior que `maxVal` são determinadas como bordas. Já aquelas com valor abaixo de `minVal` não são bordas e são descartadas. As arestas que tiverem módulo do gradiente situado entre esses dois limites serão classificadas como bordas ou não de acordo com sua conectividade. Se elas estiverem conectadas em pixels de uma borda com módulo do gradiente acima do `maxVal`, as mesmas serão consideradas parte das bordas. Caso contrário, serão descartadas.

A função `cv2.Canny` aplica o método de detecção de Canny, recebendo como parâmetros de entrada a imagem, os valores de limite mínimo e máximo que definirão o valor do módulo máximo e mínimo das arestas na etapa de histerese, o tamanho do kernel Sobel usado para encontrar o gradiente de intensidade da imagem (possui valor padrão igual a 3) e por último uma flag que especifica a equação que deve ser usada para encontrar a magnitude do gradiente. Se essa flag for setada como `TRUE`, a função utilizada será a Equação 2.3, caso contrário, usará a Função 2.4. O valor da flag é definida por default como `FALSE`.

Figura 6 – Exemplo de aplicação da função `cv2.Canny`. (a) Imagem processada pelo filtro convertida em escala de cinza. (b) Imagem processada pela função `cv2.Canny` após passar ser convertida em escala de cinza.



Fonte: próprio autor.

No exemplo da imagem acima, a Figura 6 (a) foi convertida para uma escala de cinza e em seguida é utilizada como entrada da função `cv2.Canny`. O limite mínimo

foi setado em 100, o limite máximo foi setado em 200 e os valores para o tamanho do kernel Sobel e a flag que especifica a equação da magnitude do gradiente foram mantidas com seus valores padrão. O resultado é apresentado na Figura 6 (b).

- ***Detecção de contornos***

Apesar do detector de bordas do Canny ser usado para encontrar as bordas que separam diferentes segmentos em uma imagem, esta função não agrega nenhum tipo de informação a essas bordas, ou seja, não as trata como entidades individuais. Dessa forma, a próxima etapa é tratar as bordas obtidas como contornos. Contornos são listas de pontos que representam curvas em uma imagem e são representados no OpenCV por objetos de modelo de vetor no estilo STL, em que cada entrada no vetor codifica informações sobre a localização do próximo ponto na curva. A função `cv2.findContours` é capaz de realizar a detecção dos contornos presentes em uma imagem segundo a hierarquia com que se apresentam na imagem.

Mesmo sua representação sendo normalmente uma sequência de pontos 2D (vetor `cv.Point` ou `cv.Point2f`), essa representação pode ser feita de diversas formas. A função `cv2.findContours` recebe por parâmetro uma imagem binária, a hierarquia e o método, conforme apresentado por (KAEHLER; BRADSKI, 2016). A hierarquia será a saída que descreve a estrutura em árvore dos contornos e diz ao OpenCV como você gostaria que os contornos fossem extraídos. Existem 4 parâmetros possíveis para esse argumento:

- `cv2.RETR_EXTERNAL`: Recupera só os contornos externos extremos;
- `cv2.RETR_LIST`: Recupera todos os contornos e os coloca na lista;
- `cv2.RETR_CCOMP`: Recupera todos os contornos e os organiza em uma hierarquia de dois níveis. Os limites do nível superior são os limites externos, e os limites do segundo nível são os limites dos furos;
- `cv2.RETR_TREE`: Recupera todos os contornos e reconstrói a hierarquia completa de contornos aninhados.

O próximo argumento é o método, ou seja, como os contornos são representados. Existem 3 parâmetros possíveis para esse argumento:

- `cv2.CHAIN_APPROX_NONE`: Transforma todos os pontos do contorno detectado em vértices do contorno. Esta operação produz uma grande quantidade de pontos, não reduzindo o número de vértices retornados.
- `cv2.CHAIN_APPROX_SIMPLE`: Recupera todos os contornos e os coloca na lista. Ele remove os pontos redundantes e comprime o contorno, economizando memória.

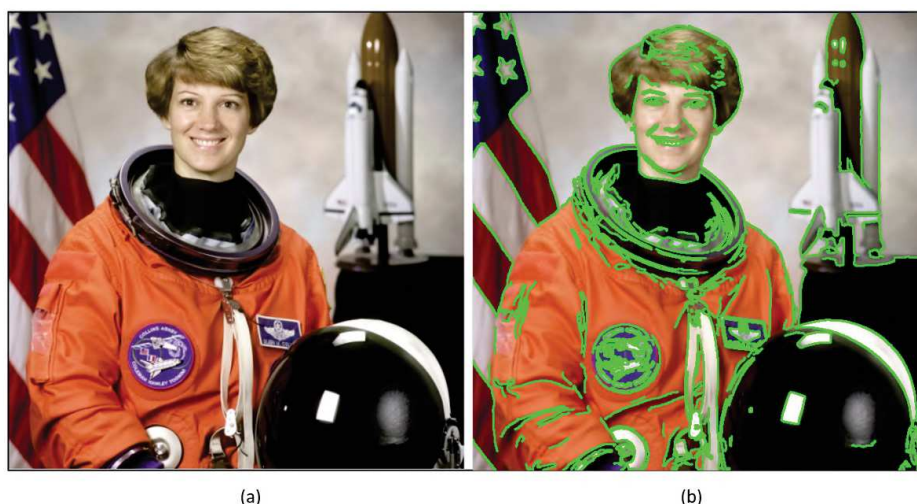
- `cv2.CHAIN_APPROX_TC89_L1`: Recupera todos os contornos e os organiza em uma hierarquia de dois níveis. Os limites do nível superior são os limites externos e os limites do segundo nível são os limites dos furos.

A função retornará na saída uma lista com os contornos detectados segundo os parâmetros de entrada escolhidos pelo usuário com suas respectivas hierarquias.

No exemplo da imagem abaixo, a Figura 7 (a) foi convertida para uma escala de cinza, e em seguida, utilizada como entrada da função `cv2.threshold`. E assim, a função `cv2.findContours` recebe como parâmetros de entrada a imagem processada por `cv2.threshold`, `cv2.CHAIN_APPROX_SIMPLE` como método de representação dos contornos e o argumento `cv2.RETR_TREE` como hierarquia.

O resultado apresentado na Figura 7 (b) mostra a imagem original com seus os contornos encontrados.

Figura 7 – Exemplo de aplicação da função `cv2.findContours`. (a) Imagem original de entrada. (b) Imagem original com os contornos obtidos após processamento da imagem.



Fonte: próprio autor.

- ***Medição da menor distância entre ponto e contorno***

Para ocasiões onde se faz necessário a verificação da distância entre um ponto e um determinado contorno ou quando se deseja saber se o ponto está contido em um contorno específico, a função `cv2.pointPolygonTest` fornece os dados desejados. O usuário informa um contorno, um ponto e uma flag booleana como argumentos de entrada da função e tem como retorno a distância medida.

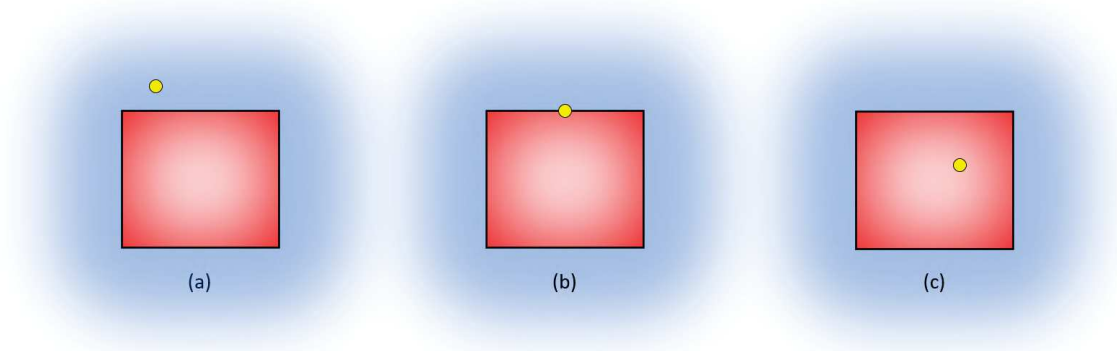
A distância informada pela função pode ser positiva, negativa ou igual a zero, que corresponde, respectivamente, a dentro do contorno, fora do contorno ou em uma posição pertencente ao contorno. A flag booleana informada na entrada da função determina a função retorna a distância exata ou um valor correspondente à posição

do ponto em relação ao contorno (+1; 0; -1). O sinal do indicador segue o mesmo sinal da distância calculada entre o ponto e o contorno (SPIZHEVOY; RYBNIKOV, 2018).

A imagem abaixo mostra um exemplo para cada situação possível de saída da função, sendo as Figuras 8 (a), (b) e (c) a representação de um ponto que está na região externa do contorno, um ponto que pertence ao contorno (faz parte dele) e um ponto que está na região interna do contorno, respectivamente.

A região pintada em azul simboliza a zona que, caso o ponto em análise esteja presente nessa área, retorna valores negativos de distância quando a função é chamada. A região pintada em vermelho simboliza a região que retorna valores positivos de distância. Caso o ponto esteja presentes na linha preta do contorno, a função retornará o valor zero para a função.

Figura 8 – Exemplo de aplicação da função *cv2.pointPolygonTest*. (a) Ponto fora do contorno. (b) Ponto com posição pertencente ao contorno. (c) Ponto fora do contorno.



Fonte: próprio autor.

- **Detecção de Linhas**

A função *cv2.HoughLines* faz a detecção de linhas através da transformada de Hough (DUDA; HART, 1972), que é uma técnica de extração de características que tem por objetivo buscar instâncias de objetos de uma determinada forma se ela puder ser representada de forma matemática, mesmo se estiver quebrada ou um pouco distorcida.

Dentro da função, um array ou acumulador 2D é criado para armazenar os valores dos parâmetros que descrevem as retas identificadas.

Uma linha pode ser representada em sua forma linear como

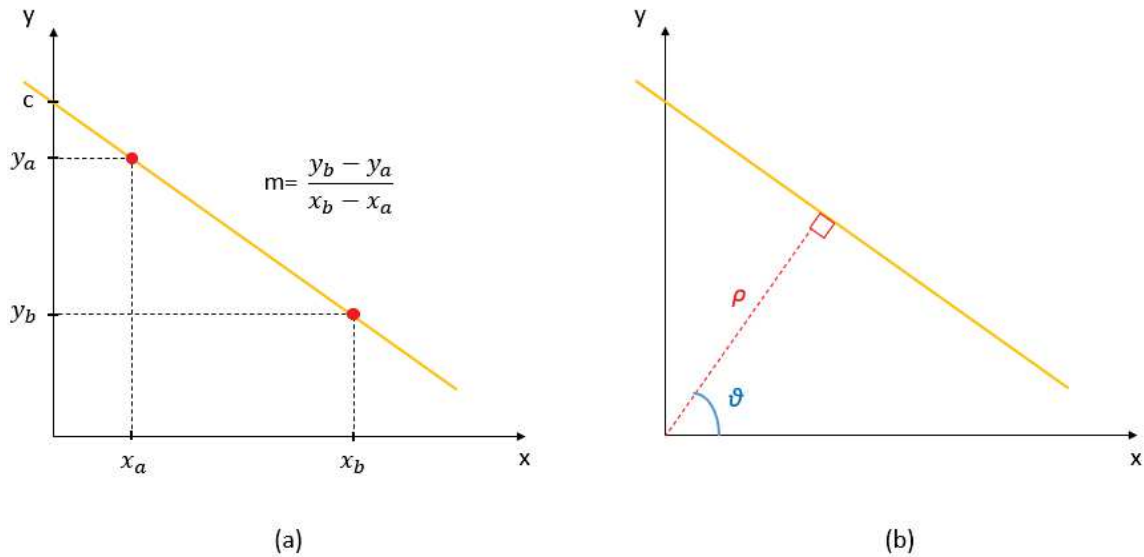
$$y = mx + c \quad (2.6)$$

ou em uma forma paramétrica como

$$\rho = x \cos(\theta) + y \sin(\theta) \quad (2.7)$$

, onde  $\rho$  é a distância perpendicular da origem (0,0) à linha e  $\theta$  é o ângulo formado por esta linha perpendicular e o eixo horizontal medido no sentido anti-horário. Ambas as formas descritas estão representadas na Figura 9.

Figura 9 – Representações de uma reta. (a) Forma linear. (b) Forma paramétrica.



Fonte: próprio autor.

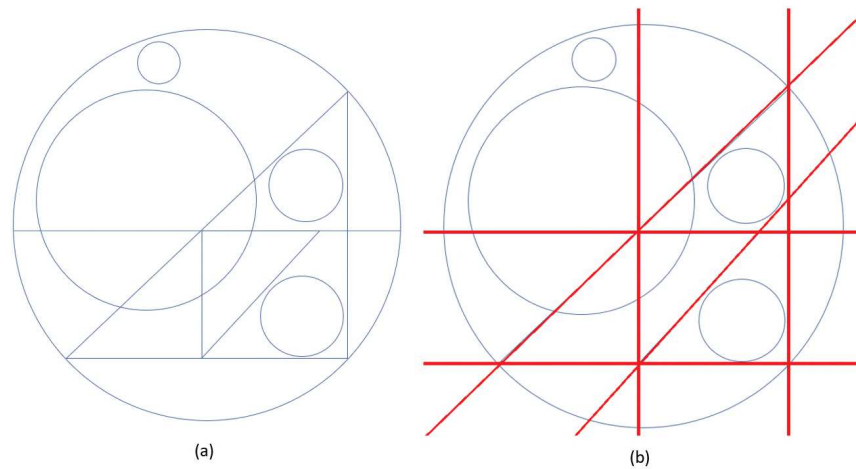
Para cada valor de  $\rho$  e  $\theta$ , a função calcula o número de pixels diferentes de zero na imagem de entrada que estão próximos da linha correspondente e incrementa a matriz na posição  $(\rho, \theta)$ . Assim, cada pixel diferente de zero contabilizará um voto à linha que está sendo analisada caso o pixel pertença à esta reta. As linhas mais prováveis correspondem aos valores dos parâmetros que obtiveram os maiores votos, ou seja, os máximos locais em um histograma 2D.

A entrada da função é composta por uma imagem binária, a resolução da distância em pixels da grade de Hough, a resolução angular em radianos da grade de Hough e o número mínimo de pixels para que o conjunto de pixels constitua uma linha.

A função retorna um array de  $(\rho, \theta)$  que representam cada linha que foi encontrada com o método de detecção de linhas de Hough. (DEY, 2018)

A imagem abaixo mostra um exemplo da aplicação da função `cv2.HoughLines`, sendo a Figura 10 (a) a entrada da função e a Figura 10 (b) é a saída retornada, com as linhas detectadas representadas em vermelho.

Figura 10 – Exemplo de aplicação da função `cv2.HoughLines`. (a) Imagem original de entrada. (b) Imagem original com as linhas obtidas após processamento da imagem.



Fonte: próprio autor.

- **Detecção de círculos**

O OpenCV também possui uma função de detecção de círculos, chamada *HoughCircles*.

A forma geral de um círculo é dada por

$$(x - x_c)^2 + (y - y_c)^2 = R^2 \quad (2.8)$$

onde  $(x_c, y_c)$  é o centro do círculo e  $R$  é o raio. Essa equação pode ser reescrita na forma polar como

$$y = y_c \pm \sqrt{R^2 - (x - x_c)^2} \quad (2.9)$$

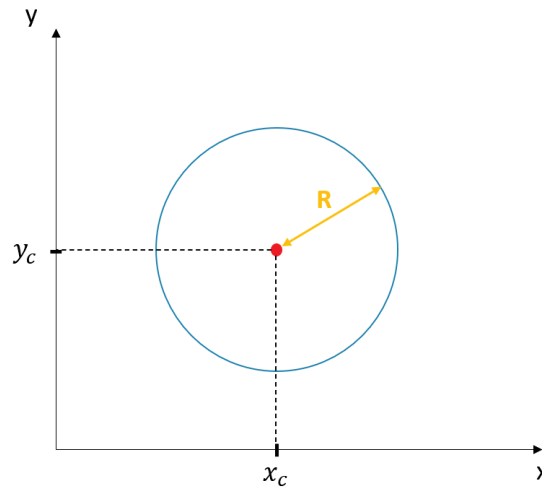
onde  $\theta \in [0, 2\pi]$ . A representação do círculo descrito é feita conforme Figura 11.

Como pode ser visto nas equações acima, é possível representar cada ponto do plano  $(x, y)$  no hiper-plano  $(x_c, y_c, R)$ , que também pode ser chamado de plano acumulador. Então os valores  $x$  e  $y$  podem ser calculados para um  $(R, x_c, y_c)$  específico e para cada valor  $\theta$ .

Todos os valores de  $(x, y)$  serão percorridos e aquele que pertencer ao círculo, o valor de 1 será adicionado à coordenada específica  $(R, x_c, y_c)$  no plano do acumulador. Após esse processo, o hiperplano acumulador terá um alto valor acumulado nos pontos correspondentes a um círculo, sendo então possível então extrair os parâmetros do círculo da imagem original a partir dos pico locais do plano acumulador. (CHITYALA; PUDIPEDDI, 2020)

A função recebe como entrada uma imagem binária, `cv2.HOUGH_GRADIENT` como método de detecção, a distância mínima entre os centros dos círculos a serem detectados, o limite mais alto do método Canny usado internamente, o limite para o

Figura 11 – Representações de um círculo com centro em  $(x_c, y_c)$  e raio R.



Fonte: próprio autor.

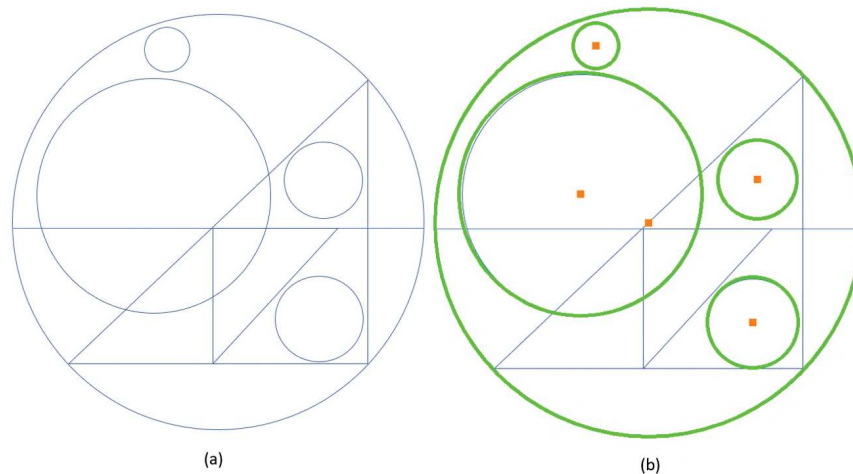
acumulador, os raios máximo e mínimo dos círculos a serem detectados e a razão inversa da resolução dada pela Equação 2.10.

$$\text{razão inversa da resolução} = \frac{\text{resolução da imagem}}{\text{resolução do acumulador}} \quad (2.10)$$

Como saída a função retorna um arranjo contendo os centros e raios dos círculos detectados. (ESCRIVA; LAGANIERE, 2019)

A imagem abaixo mostra um exemplo da aplicação da função `cv2.HoughCircles`, sendo a Figura 12 (a) a entrada da função e a Figura 12 (b) é a saída retornada, com os círculos detectados representados em verde.

Figura 12 – Exemplo de aplicação da função `cv2.HoughCircles`. (a) Imagem original de entrada. (b) Imagem original com os círculos obtidos após processamento da imagem.



Fonte: próprio autor.



### 2.1.2 Classificação Baseada em Máquina de Vetor de Suporte

A Máquina de Vetor de Suporte é um algoritmo de aprendizado supervisionado muito utilizado em aplicações relacionadas à classificação e regressão (VILLAN, 2019). O objetivo dessa técnica é encontrar um hiperplano, ou um conjunto de hiperplanos, com a capacidade de separar duas classes distintas, realizando ainda uma otimização para alcançar a maximização da margem entre as classes. A margem é determinada pela distância entre o hiperplano e as amostras das classes que estiverem mais próximas a ele. Essas amostras são chamadas de *vetores de suporte* e as outras pertencentes à classe são chamadas de *vetores de recursos* (AHMAD, 2020).

Para exemplificar o processo realizado, serão considerados um conjunto de dados que possua apenas duas classes. Como apresentado em (BONACCORSO, 2018), seja um conjunto de dados

$$X = \{x_1, x_2, x_3, \dots, x_n\} \text{ onde } x_i \in \mathfrak{R}^m, \quad (2.11)$$

ou seja, os itens podem ser representado no espaço n-dimensional. Cada item deste conjunto será classificado, tendo o valor 1 ou -1 atribuído de acordo com a classe a qual pertencer. Esse classificador pode ser representado por

$$Y = \{y_1, y_2, y_3, \dots, y_n\} \text{ onde } y_i \in \{-1, 1\}. \quad (2.12)$$

O objetivo é determinar o hiperplano que melhor separa as classes, descrito por

$$W^T \cdot X + b = 0 \text{ onde } W = (w_1, w_2, w_3, \dots, w_m)^T. \quad (2.13)$$

Com isso, o classificador  $Y$  será dado por

$$y_i = f(x_i) = \text{senal}(W \cdot x_i + b). \quad (2.14)$$

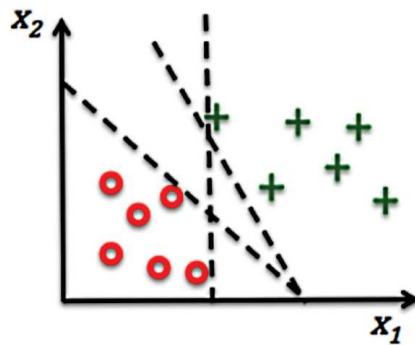
Fazendo uma normalização do conjunto de dados, é possível determinar os hiperplanos dos vetores de suporte das classes presentes nas margens de separação como

$$\begin{cases} W \cdot x_i + b = -1 \\ W \cdot x_i + b = 1 \end{cases} \quad (2.15)$$

A partir dessa determinação, o objetivo desse método será determinar o hiperplano de separação ideal, visando sempre maximizar a distância entre os hiperplanos dos vetores de suporte. Quando o SVM é treinado até encontrar o hiperplano e as margens para um conjunto de dados de teste ele será capaz de classificar novos dados de entrada utilizando os hiperplanos encontrados.

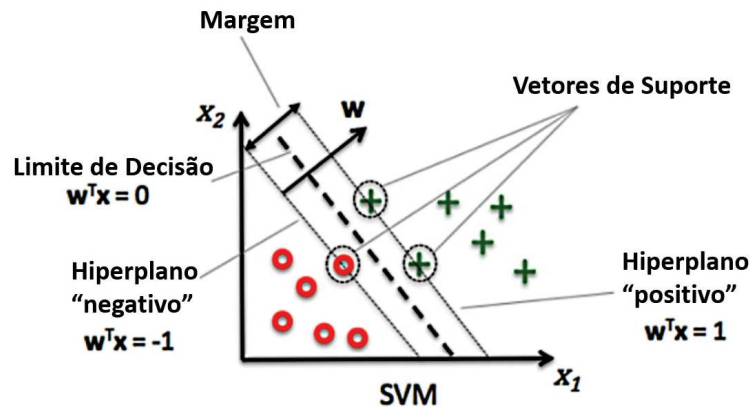
A Figura 13 apresenta um exemplo de um conjunto de dados com apenas duas dimensões que serão utilizados no treinamento do SVM. Note que as linhas tracejadas representam diferentes retas que podem separar as classes, mas não são as retas ideais.

Figura 13 – Exemplo de conjunto de dados com duas dimensões.



Fonte: (AHMAD, 2020).

Figura 14 – Exemplo de conjunto de dados com duas dimensões classificados através da Máquina de Vetor de Suporte.



Fonte: Adaptado de (AHMAD, 2020).

Figura 14 detalha o resultado do SVM treinado, com os hiperplanos ideais fazendo a separação das classes.

Uma das formas de extrair os vetores de recursos da imagem é através da aplicação de um método muito utilizado na detecção de objetos através de descritores, conhecido como Histograma de Gradientes Orientados (HOG). Este método foi popularizado por (DALAL; TRIGGS, 2005), onde era proposto a utilização do método na detecção de humanos em imagens.

O HOG tem por base uma análise das formas locais e da aparência dos objetos presentes na imagem, descrevendo-a a partir da distribuição das direções das bordas (GEVORGYAN; BEYELER; MAMIKONYAN, 2020). O processamento se inicia com a divisão da imagem em análise em pequenas regiões e calculando em seguida um conjunto de gradientes que descrevem cada região, onde cada gradiente representa a mudança de intensidade dos pixels da região em uma direção. (HOWSE; MINICHINO, 2020) Um exemplo dessa etapa pode ser visto na Figura 15.

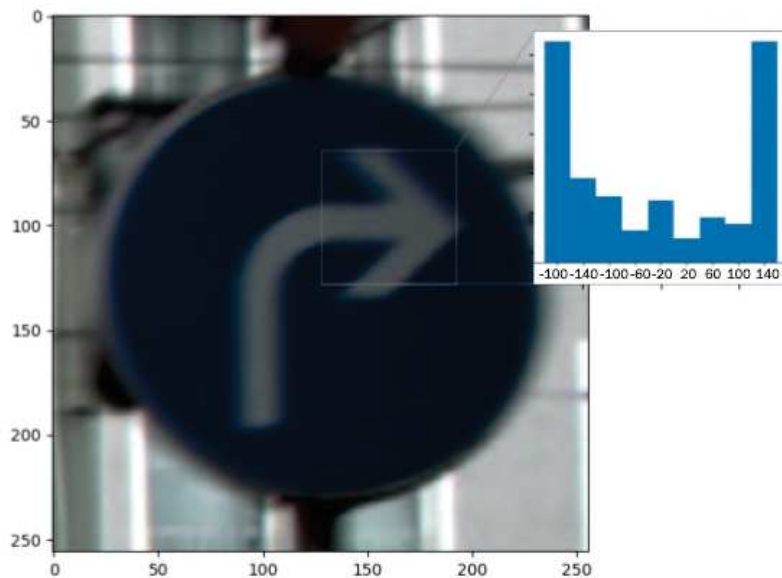
Figura 15 – Exemplo de extração de gradientes de um imagem. (a) Imagem original. (b) Imagem após obtenção dos gradientes.



Fonte: (VONDRICK et al., 2013).

Em seguida, um histograma é compilado para cada região a partir das direções de seus respectivos gradientes (GEVORGYAN; BEYELER; MAMIKONYAN, 2020). Figura 16 destaca o histograma correspondente à uma região da imagem mostrada. Os ângulos dos gradientes estão presentes no intervalo  $(-180, 180)$ , como mostrado no histograma.

Figura 16 – Exemplo de histograma obtido na análise dos gradientes de uma região da imagem.



Fonte: (GEVORGYAN; BEYELER; MAMIKONYAN, 2020).

O descritor resultante do processamento do HOG é obtido através da normalização dos blocos de gradientes e o achatamento dos dados obtidos em um vetor de descritor de recurso, chamados de descritores HOG (DEY, 2020). Para realizar o treinamento do SVM, são utilizadas imagens de treinamento positivo e negativo. As imagens positivas

possuem o objeto que deseja-se detectar, enquanto que as negativas são compostas por imagens que não possuem o objeto (DEY, 2020). Esse conjunto de imagens de treinamento é convertido em descritores de HOG e em seguida são utilizados como entrada para o modelo do SVM, que retornará a classificação correspondente a cada um dos recursos (GEVORGYAN; BEYELER; MAMIKONYAN, 2020).

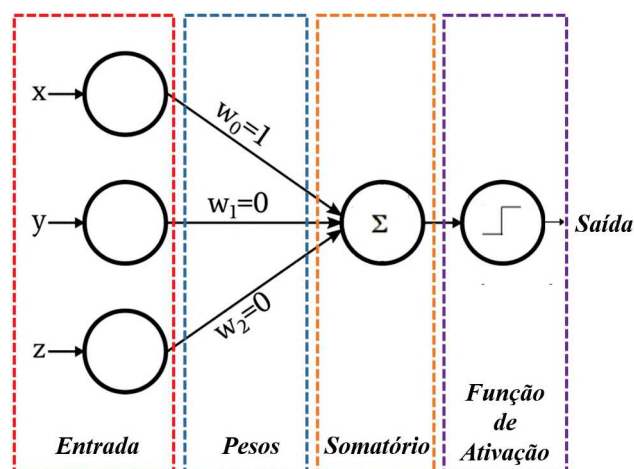
Após o treinamento do SVM, os dados a serem classificados são inicialmente processados pelo método de HOG e em seguida passam pelo SVM. Caso o classificador SVM detecte o objeto na imagem em qualquer escala, uma caixa delimitadora será retornada englobando o objeto de interesse (DEY, 2020).

### 2.1.3 Classificação Baseada em Redes Neurais

No decorrer das últimas décadas, diversos métodos foram propostos inspirados no comportamento da natureza. Como exemplo, algoritmos genéticos (HOLLAND, 1992), enxame de partículas (KENNEDY; EBERHART, 1995), algoritmo de morcegos (YANG, 2010), entre outros. Além disso, avanços profundos foram feitos no desenvolvimento de técnicas de inteligência artificial baseadas no comportamento cognitivo humano, tal como o desenvolvimento de algoritmo inspirado em redes neurais (VIANA et al., 2020).

As redes neurais, desenvolvidas por (MCCULLOCH; PITTS, 1943), foram propostas como uma forma de suporte às hipóteses e cálculos matemáticos, onde foram modeladas por meio de circuito elétrico equivalente. A unidade básica de uma rede neural é o perceptron. Este, possui um conjunto de entradas  $\{x_1, \dots, x_n\}$  que são multiplicadas por pesos  $\{w_1, \dots, w_n\}$ . Estes pesos são parâmetros ajustáveis. A saída do valor resultante da multiplicação é a entrada para uma função de ativação. A Figura 17 apresenta um esquemático de um perceptron.

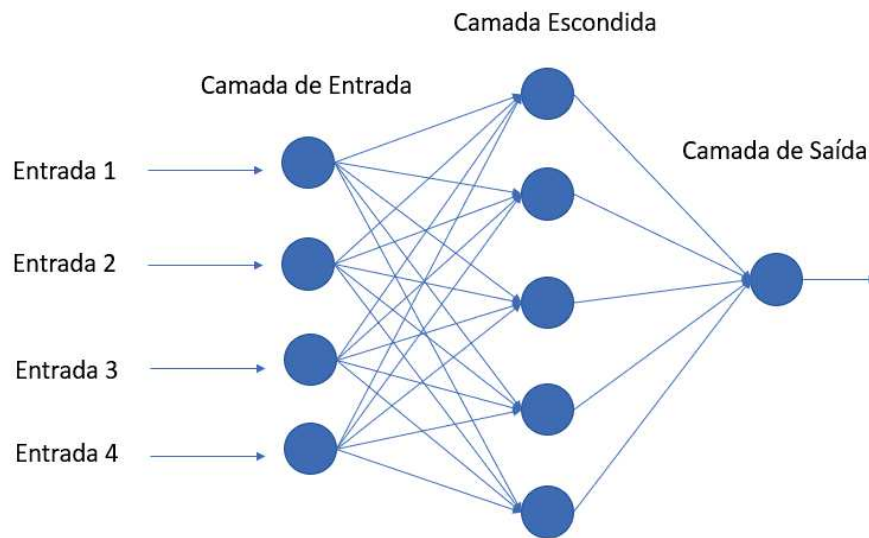
Figura 17 – Esquemático de um perceptron.



Fonte: próprio autor.

O perceptron funciona como um classificador linear devido ao fato de possuir um hiperplano de separação. Portanto, não consegue separar função não lineares, como um XOR. Uma forma de solucionar a classificação de funções não lineares, surge então o perceptron multicamadas (do inglês, Multilayer Perceptron - MLP) (ROJAS, 1996). O MLP consiste basicamente de três camadas: (i) camada de entrada; (ii) camada oculta; e (iii) camada de saída. Figura 18 apresenta um esquemático de uma rede MLP.

Figura 18 – Esquemático de um reade Multilayer Perceptron.



Fonte: próprio autor.

É importante ressaltar que o MLP é uma rede neural do tipo *feedforward*, no qual possui uma função de ativação não linear em cada um dos seus neurônios. Além disso, todas as camadas entre a entrada e a saída são denominadas de camadas ocultas, e quanto maior o número de neurônios, maior a complexidade da rede. Ademais, para cada exemplo de um conjunto de treino colocado na entrada, é possível estimar a classe por meio da saída, e comparando esse resultado com o resultado correto conhecido. A diferença desta comparação é ajustada por um algoritmo treinamento, tal como o *backpropagation* proposto por (WERBOS, 1974). O procedimento é repetido até o mínimo local da função de custo ser atingido.

### 2.1.3.1 Redes Neurais Convolucionais

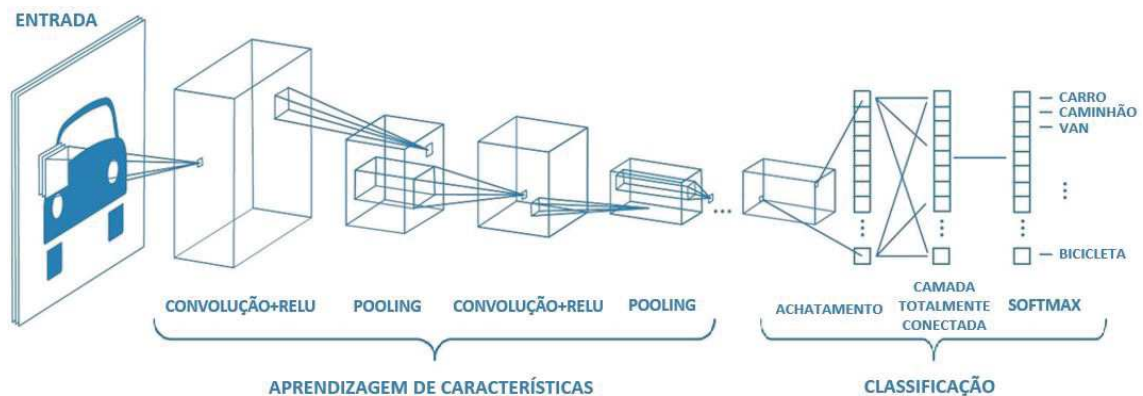
Redes neurais convolucionais (do inglês, *Convolutional Neural Networks - CNN*) são um tipo de tecnologia baseada em modelos de aprendizado profundo (do inglês, *Deep Neural Network - DNN*). São amplamente utilizadas no processamento de dados e tem obtido muito sucesso em aplicações práticas, como no reconhecimento de objetos (HE et al., 2016), análise de imagens (YAN et al., 2014), análise de sinais (COELHO et al., 2020), segmentação semântica (VIANA et al., 2020), entre outros.

O desenvolvimento da estrutura desse tipo de rede tem como base o trabalho desenvolvido em (HUBEL; WIESEL, 1962) e (HUBEL; WIESEL, 1977). Os autores demonstraram que o córtex visual de gatos é composto por um conjunto de células que são sensíveis a sub-regiões. E assim, os neurônios individuais reconhecem estímulos apenas em uma parte do campo visual, chamada de campo receptivo. Um grupo desses campos se sobrepõe, e, desta forma, conseguem cobrir toda a área visual. Este estudo mostrou que esse conjunto é organizado hierarquicamente e que as células são classificadas de acordo com sua resposta a estímulos que recebem, podendo ser simples, complexas e supercomplexas de acordo com o tipo de padrão que são capazes de reconhecer.

Esses estudos inspiraram trabalhos como (FUKUSHIMA, 1980), que foi evoluindo até chegar ao que hoje se conhece como *redes neurais convolucionais*. Outro importante trabalho desenvolvido foi o proposto por (LECUN et al., 1998) que propôs a arquitetura *LeNet-5* para o reconhecimento de palavras escritas a mão e números em cheques manuscritos. Essa estrutura introduziu dois novos blocos de construção ao que era utilizado em redes neurais artificiais: *as camadas convolucionais* e *as camadas de pool*.

As CNNs são redes de aprendizado profundo que tentam extrair características de um conjunto de dados e atribuem importância aos aspectos da imagem. Assim como no modelo biológico que inspirou o modelo, essa arquitetura é composta por uma organização hierárquica, onde as características obtidas em alto nível são baseadas na composição do aprendizado feito nas camadas de nível mais baixo. As primeiras camadas tendem a ter um aprendizado de características de baixo nível, ou seja, identificando padrões mais simples, enquanto as últimas camadas aprendem características de alto nível, ou seja, detectando padrões mais complicados ou conjuntos de padrões simples (MORA, 2017). Figura 19 ilustra uma arquitetura das redes CNNs e seus vários estágios.

Figura 19 – Rede Neural Convolutacional.



Fonte: Adaptado de (SAHA, 2018)

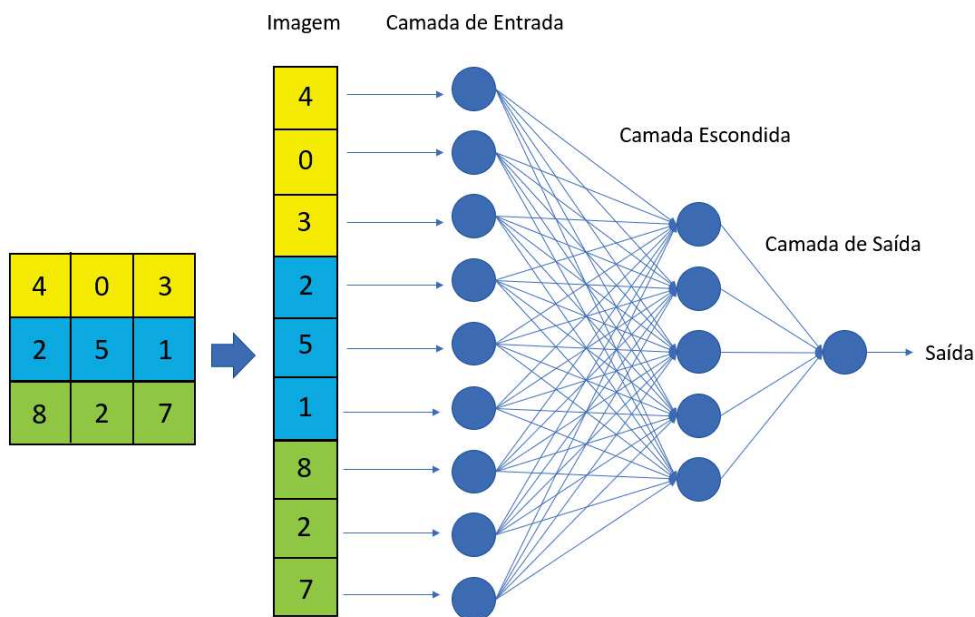
Como é possível observar, após a realização da convolução, há uma transformação da imagem de entrada em um mapa de características. A quantidade de mapas extraídos dependerá de quão profunda for a rede. A próxima etapa é a operação de *pooling*, que é um processo de subamostragem. Em seguida, há a aplicação de uma função de ativação da última camada convolucional (ACADEMY, 2019). A maioria dos modelos implementados apresentam uma estrutura formada por várias camadas intercaladas formando um sistema mais robusto. A inferência da imagem será feita a partir das características de alto nível, que servirão de entrada para a camada totalmente conectada responsável por gerar o resultado da classificação da imagem de entrada. A vantagem presente nesse tipo de arquitetura é a capacidade de reutilização dos pesos, sendo capaz então de ser treinada para entender melhor a sofisticação da imagem (FARIA, 2018).

A função da CNN é reduzir os parâmetros de processamento das imagens para uma forma mais fácil de processar sem perder atributos críticos para uma boa previsão.

#### 2.1.3.1.1 Camadas Convolucionais

Essa camada é o bloco de construção mais importante de uma CNN. Nas redes neurais artificiais multicamadas, as camadas eram compostas por uma longa linha de neurônios e para ter uma imagem como entrada da rede seria necessário achatar a imagem para uma dimensão antes de alimentá-las na rede, pois cada neurônio precisaria estar conectado a um pixel da imagem na primeira camada, como representado na Figura 20.

Figura 20 – Achatamento de uma matriz de imagem 3x3 em um vetor 9x1.



Fonte: próprio autor.

O diferencial das camadas convolucionais é que cada neurônio da primeira camada convolucional não precisa estar conectado a cada pixel da imagem, mas sim a todos os pixel presentes em seu campo receptivo, delimitado por um pequeno retângulo chamado de filtro. Todas os elementos presentes nessa região terão uma conexão com o neurônio oculto. Durante a fase de treinamento, cada conexão aprende um peso e o neurônio oculto aprende um bias. Neste caso, é como se cada neurônio estivesse aprendendo a observar e a absorver parâmetros para analisar seu campo receptivo local. Da mesma forma, os neurônios da segunda camada da rede estarão conectados apenas aos neurônios compreendidos dentro do filtro de tamanho definido. (GÉRON, 2018)

A camada convolucional realiza uma operação de convolução entre a entrada e um filtro de aprendizado que será capaz de identificar características específicas nos dados da entrada, onde o tipo de característica a ser encontrado dependerá diretamente dos valores definidos no filtro. Desta forma, é possível produzir resultados diferentes para uma mesma imagem de entrada mudando apenas o filtro aplicado, onde cada resultado obtido é chamado de mapa de características. Durante o treinamento da rede, os filtros vão sendo ajustados com pesos que irão destacar a presença de características importantes para o correto processamento da entrada e para que se atinja o objetivo desejado. Esse resultado, acrescido de termo de bias, é aplicado em uma função de ativação não linear, que pode ser a função Unidade Linear Retificada (do inglês, Rectified Linear Units - ReLU), tangente hiperbólica ou sigmoide. (ACADEMY, 2019)

Seja a imagem dada pela Figura 21 (a) e o filtro dado pela Figura 21 (b). O filtro irá varrer toda a imagem de entrada, começando pela esquerda e deslocando-se um pixel por vez para a direita. (BALSYS, 2019)

Figura 21 – Exemplo da imagem de entrada e de um filtro de aprendizagem da CNN. (a) Imagem de entrada. (b) Filtro.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

(a)

1	0	1
0	1	0
1	0	1

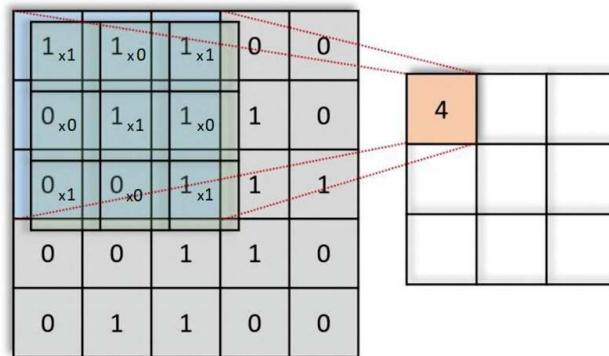
(b)

Fonte: (BALSYS, 2019).

O filtro multiplica seus valores com os valores sobrepostos da imagem enquanto desliza sobre ela e soma todos os valores obtidos de forma a gerar um único valor para cada sobreposição, como mostra a Figura 22. (BALSYS, 2019)



Figura 22 – Exemplo da convolução entre o filtro de aprendizagem com a imagem de entrada.

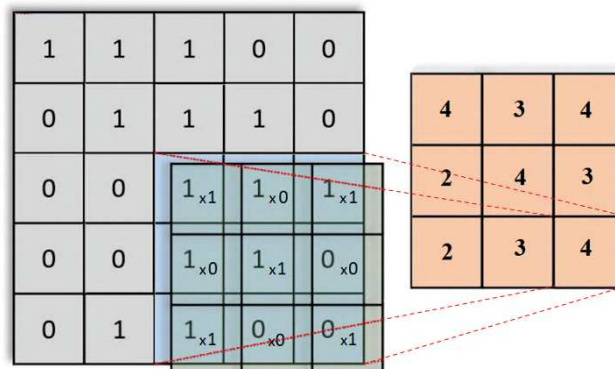


$$(1 \times 1 + 0 \times 1 + 1 \times 1) + (0 \times 0 + 1 \times 1 + 1 \times 0) + (1 \times 0 + 0 \times 0 + 1 \times 1) = 4$$

Fonte: (BALSYS, 2019).

Este processo se repete até que toda a imagem seja percorrida pelo filtro, resultando na Figura 23.

Figura 23 – Resultado da convolução entre o filtro de aprendizagem com a imagem de entrada.

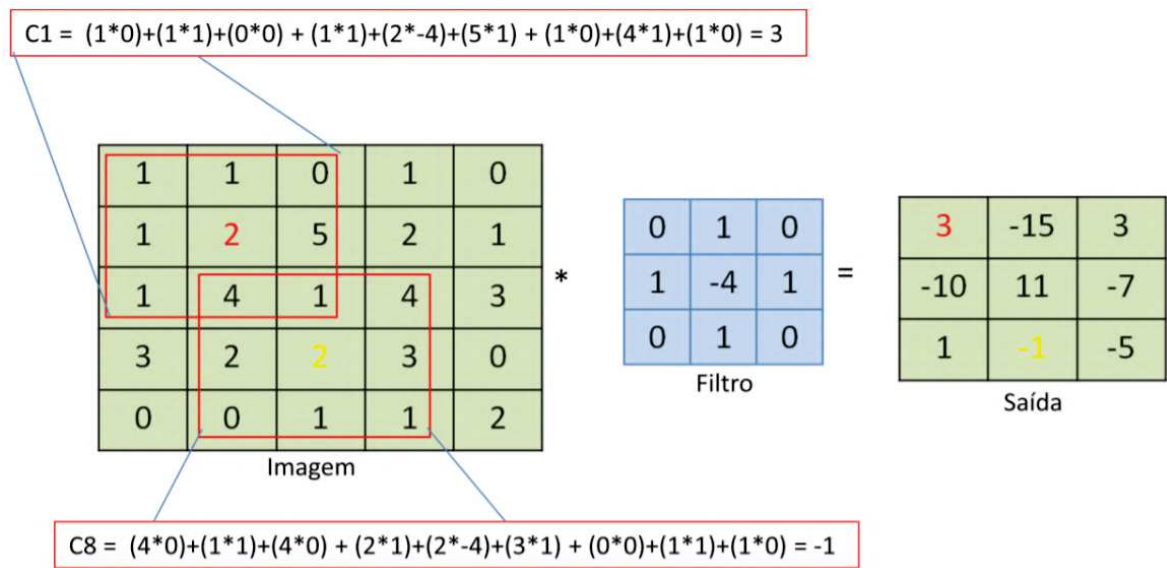


Fonte: Adaptado de (BALSYS, 2019).

Outro exemplo pode ser observado na Figura 24. A primeira camada convolucional captura os recursos de baixo nível, como orientação de gradiente, bordas e cores. Já nas camadas mais profundas, são obtidos recursos de alto nível, gerando uma rede capaz de compreender todos os aspectos desejados nas imagens do conjunto de dados (BALSYS, 2019).

Diversos filtros são utilizados em camadas convolucionais com o objetivo de extrair vários tipos de atributos diferentes da entrada, e com isso, seus mapas de características resultantes são empilhados, gerando uma matriz 3D para imagens 2D como está representado na Figura 25. Desta forma, os dados evoluirão na CNN na forma de um volume de dados. As camadas dos mapas de características vão aumentando sua profundidade a cada

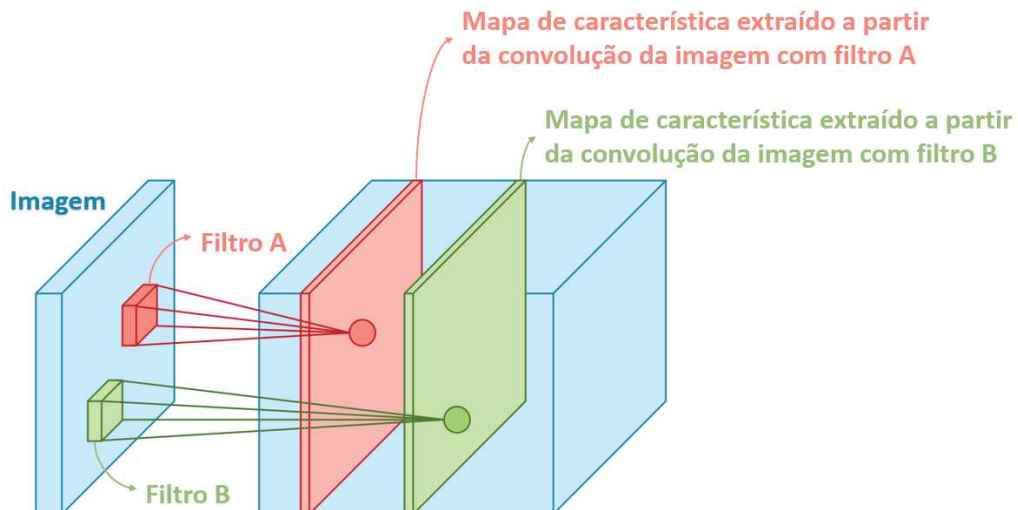
Figura 24 – Exemplo de convolução entre imagem e filtro.



Fonte: (FARIA, 2018)

camada convolucional que encontram. Outro aspecto a ser observado é que as camadas convolucionais também vão reduzindo sua largura e altura quanto maior for o nível das características extraídas pela rede (FARIA, 2018).

Figura 25 – Diferentes mapas de características obtidos de uma mesma entrada.



Fonte: Adaptado de (ACADEMY, 2019)

De acordo com (FARIA, 2018), as redes convolucionais exploram a correlação espacial dos pixels impondo um padrão de conectividade local esparsa entre os neurônios das camadas adjacentes. As conexões são locais no espaço, mas sempre se estendem ao longo de toda a profundidade do volume de entrada. Três hiperparâmetros controlam o tamanho do volume de saída da camada convolucional:

## 1. Profundidade

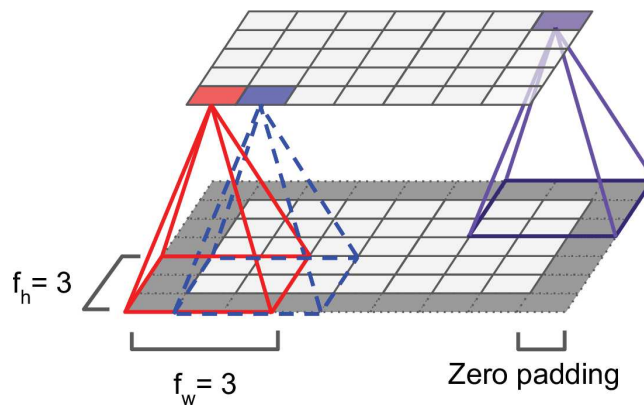
Esse hiperparâmetro controla o número de neurônios em uma camada que se conecta à mesma região do volume de entrada;

## 2. *Zero padding* (ou preenchimento com zeros)

Sejam  $f_h$  e  $f_w$  a altura e a largura do campo receptivo, respectivamente. Um neurônio localizado na linha  $i$ , coluna  $j$  de uma camada estará conectado às saídas dos neurônios na camada anterior posicionadas nas linhas  $i$  a  $i + f_h - 1$ , colunas  $j$  a  $j + f_w - 1$ . (GÉRON, 2018)

É comum que as imagens sejam preenchidas com zeros na borda do volume de entrada, pois dependendo do tamanho da imagem de entrada e dos filtros utilizados, esse processo expande a matriz de entrada, adicionando elementos de valor igual a zero na borda da matriz, o que permite que o filtro seja aplicado em todos os elementos da matriz original, e com isso, permitirá que mais informações da região das bordas da imagem sejam extraídas (ACADEMY, 2019). Dessa forma, a camada de neurônios terá a mesma altura e largura da camada anterior, conforme mostrado na Figura 26.

Figura 26 – Conexões entre camadas de uma CNN.



Fonte: (GÉRON, 2018)

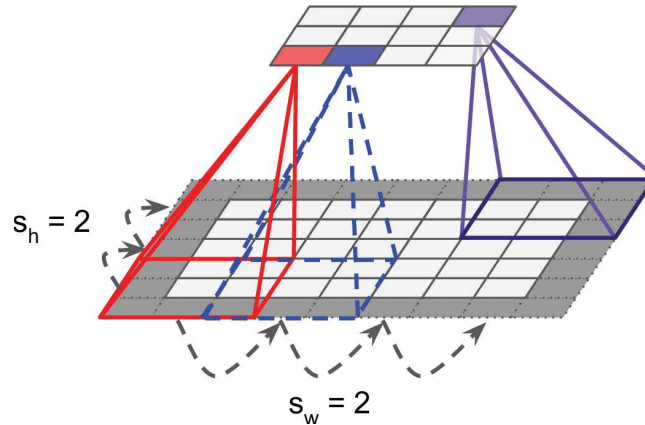
O *wide*, ou *narrow convolution*, é um hiperparâmetro diretamente relacionado com o *zero padding*. Esse hiperparâmetro determina qual é o comportamento do filtro nos elementos da borda da matriz, especificando como o mesmo será aplicado. Uma opção muito utilizada é o *zero padding*. Neste caso, o hiperparâmetro é conhecido como *wide convolution*, e quando não se usa o *padding*, o hiperparâmetro é conhecido como *narrow convolution* (FARIA, 2018).

## 3. *Stride*:

O *stride* é a distância entre dois campos receptivos consecutivos. Esse hiperparâmetro permite conectar uma camada de entrada a uma camada muito menor, como mostrado

na Figura 27, onde a camada de entrada com dimensão  $5 \times 7$  e *zero padding* é conectada a uma camada de dimensão  $3 \times 4$  através de um campo receptivo de dimensão  $3 \times 3$  e *stride* de 2. No exemplo, a passada utilizada foi a mesma em ambas as direções, mas não tem que ser assim necessariamente (GÉRON, 2018).

Figura 27 – Camada com dimensão reduzida devido ao *stride*.



Fonte: (GÉRON, 2018)

Sejam  $s_h$ ,  $s_w$ ,  $f_h$  e  $f_w$  os *stride* vertical e horizontal, a altura e a largura do campo receptivo, respectivamente. Um neurônio localizado na linha  $i$ , coluna  $j$  de uma camada estará conectado às saídas dos neurônios na camada anterior posicionadas nas linhas  $i \cdot s_h$  a  $i \cdot s_h + f_h - 1$ , colunas  $j \cdot s_w$  a  $j \cdot s_w + f_w - 1$  (GÉRON, 2018).

Em aplicações que fazem o reconhecimento de imagens, o valor do *stride* é usualmente igual a 1, fazendo com que o filtro se desloque pela imagem um pixel por passo. Nesse caso, a altura e largura da camada de saída é igual a entrada da camada (ACADEMY, 2019).

Segundo (GÉRON, 2018), a Equação 2.16 apresenta como calcular a saída de um neurônio em uma camada convolucional.

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_n-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{sendo} \begin{cases} i' = i \cdot s_h + u \\ j' = j \cdot s_w + v \end{cases} \quad (2.16)$$

onde:

- $k$  representa o mapa de características da camada convolucional  $l$
- $k'$  representa o mapa de características da camada convolucional  $l - 1$
- $u$  representa o índice das linhas do campo receptivo que cobre as posições do mapa de características  $k'$  da camada convolucional  $l - 1$
- $v$  representa o índice das colunas do campo receptivo que cobre as posições do mapa de características  $k'$  da camada convolucional  $l - 1$

- $i'$  representa o índice das linhas do mapa de características  $k'$  da camada convolucional  $l - 1$
- $j'$  representa o índice das colunas do mapa de características  $k'$  da camada convolucional  $l - 1$
- $i$  representa as linhas do mapa de características  $k$  da camada convolucional  $l$
- $j$  representa as colunas do mapa de características  $k$  da camada convolucional  $l$
- $s_h$  e  $s_W$ : *strides* verticais e horizontais
- $f_h$  e  $f_W$  : altura e largura do campo receptivo
- $f_n$ : número de mapas de características na camada anterior
- $b_k$ : termo de *bias* para o mapa de características  $k$  na camada  $l$
- $x_{i',j',k'}$ : saída do neurônio localizado na camada  $l - 1$ , linha  $i'$ , coluna  $j'$ , mapa de características  $k'$
- $w_{u,v,k',k}$ : peso da conexão entre um neurônio no mapa de características  $k$  da camada  $l$  com a sua entrada localizada na linha  $u$ , coluna  $v$  (em relação ao campo receptivo do neurônio) e o mapa de características  $k'$
- $z_{i,j,k}$ : saída do neurônio localizado na linha  $i$ , coluna  $j$  no mapa de características  $k$  da camada convolucional  $l$ )

#### 2.1.3.1.2 Camadas de Pooling

A camada de *pooling*, também conhecida como camada de sub-amostragem (*down-sampling*), é aplicada em seguida da camada convolucional, recebendo cada saída do mapa de características resultante dessa camada e preparando um mapa condensado. Seu objetivo é extrair a informação mais significativa dos dados fornecidos na saída da camada de convolução e fazer uma subamostragem, reduzindo por consequência a dimensionalidade da rede, o número de parâmetros, o custo operacional do sistema, e evitando o *overfitting*. Além disso, o *pooling* é capaz de extrair características dominantes que não variam com a rotação nem com a translação. De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), a camada de *pooling* muda os valores obtidos em uma região por uma estatística que resume as saídas mais próximas, o que gera uma diminuição no número total de neurônios da camada anterior. Desse modo, a quantidade de parâmetros a serem utilizados nas camadas posteriores também será reduzido, reduzindo também o tempo de treinamento. (FARIA, 2018) e (BALSYS, 2019)

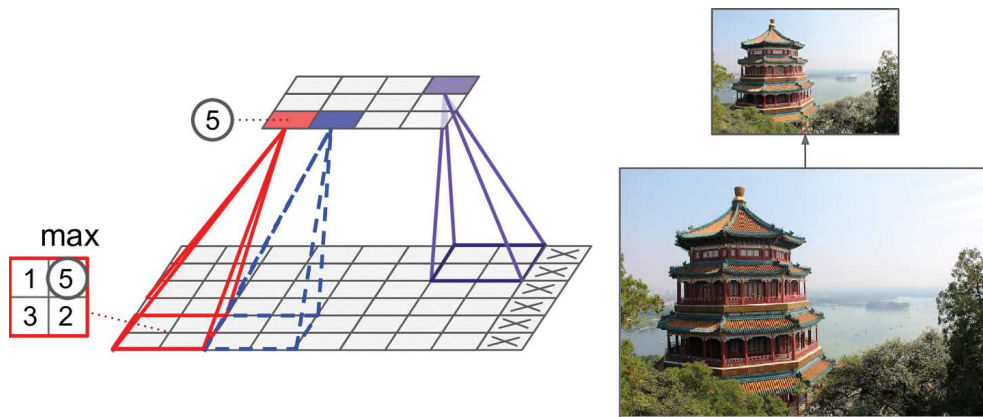
Como a camada convolucional geralmente resulta em mais de um mapa de características, a aplicação da função de *pooling* é aplicada em cada mapa de recursos individualmente. Na camada de *pooling*, assim como nas camadas convolucionais, cada neurônio está conectado nas saídas de um conjunto de neurônios da camada anterior

delimitados por um filtro. Para esta camada também é necessário definir o tamanho do filtro, o *stride* e a forma como as bordas são preenchidas, assim como antes. A diferença desta camada é que os filtros não possuem peso (GÉRON, 2018).

A função utilizada para determinar o comportamento da camada de *pooling* na região analisada também é um parâmetro que pode ser definido, onde dentre as opções, há o operador de max, que é o mais comum e consiste em manter apenas o maior valor de uma região, fazendo com que apenas as melhores características sejam armazenadas e eliminando valores desprezíveis (ACADEMY, 2019).

Figura 28 traz um exemplo da camada de *pooling* para uma aplicação onde o filtro tem dimensão 2x2, o *stride* igual a 2 e *narrow convolution* (GÉRON, 2018). A camada implementada neste exemplo é duas vezes menor em ambas as direções, fazendo com que 75% dos dados originais sejam deixados de lado.

Figura 28 – Exemplo de pooling usando operador Max.



Fonte: (GÉRON, 2018).

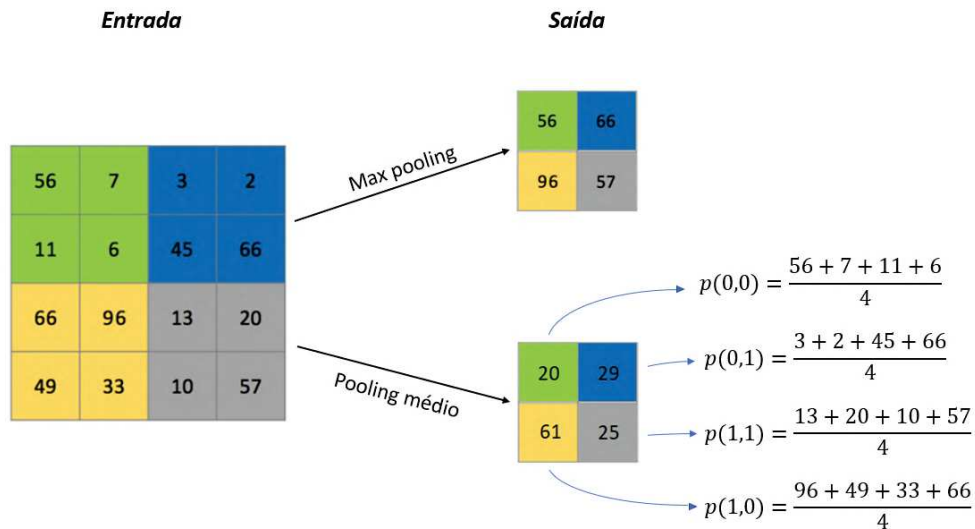
Existem outras alternativas para operadores, como por exemplo a média, a norma  $\ell_2$ , e a média ponderada. Um exemplo numérico da aplicação dos operadores médio e max é representado no exemplo da Figura 29. (GÉRON, 2018)

Pode-se observar no exemplo que o *max pooling* busca o maior valor na região compreendida no filtro e descarta os outros valores. Já o *pooling médio* calcula a soma de todos os valores na região compreendida no filtro e divide pelo número de componentes da região, onde  $p(i,j)$  representou o valor correspondente ao neurônio de saída da camada de pooling na linha  $i$ , coluna  $j$ . Desta forma, todos os componentes da região são levados em consideração na definição do valor final.

As equações 2.17 representam as equações da função de *maxpooling* e *pooling médio*, respectivamente.

$$\begin{cases} f_{MaxPooling}(X) = \max_{i,j} X(i, j) \\ f_{PoolingMédio}(X) = \frac{1}{n+m} \sum_{i=1}^n \sum_{j=1}^m X(i, j) \end{cases} \quad (2.17)$$

Figura 29 – Exemplo de pooling usando pooling médio e max.



Fonte: próprio autor.

onde  $X(i, j)$  é o conjunto de valores compreendidos pela região do filtro no momento em que o mesmo está percorrendo a imagem.

Um ponto importante na camada de *pooling* é que ela irá interferir apenas na dimensão da altura e largura dos mapas de características, mas não mudará a profundidade. Este é um componente importante das redes neurais convolucionais para detecção de objetos com base na arquitetura Fast R-CNN (GIRSHICK, 2015).

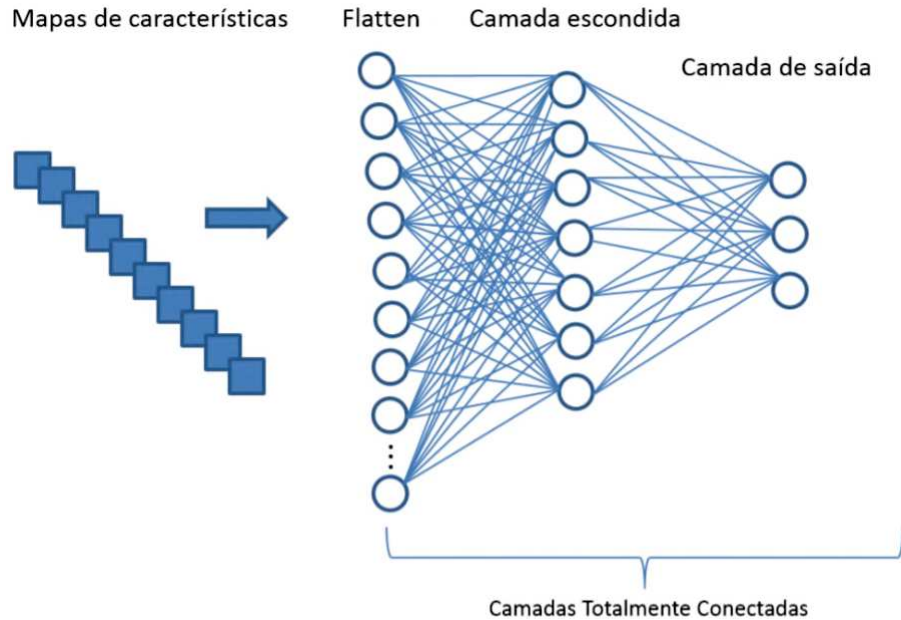
### 2.1.3.1.3 Camada Totalmente Conectada

As CNNs possuem camadas totalmente conectadas com o objetivo de usar as características de alto nível abstraídas das camadas anteriores para classificar o dado de entrada da rede em classes determinadas pelo conjunto de treinamento. Ao contrário das anteriores que possuíam uma conectividade esparsa onde os pesos eram conectados apenas à uma região específica da camada anterior, essa camada interconecta todos os neurônios provenientes da última camada de *pooling* com os neurônios subsequentes, formando a primeira camada totalmente conectada e adotando assim uma conectividade global a partir desta camada. Essa etapa da rede também possui uma última camada com um número de neurônios igual ao número de classes do modelo, chamada de camada de saída (FARIA, 2018).

A rede CNN é treinada através de um algoritmo de *feed-forward* e *backpropagation*, que tem por objetivo diminuir a diferença entre a saída desejada e a saída real da rede. O processo de treinamento é interrompido somente quando um critério pré-estabelecido é atingido, como por exemplo, chegar a um número máximo de iterações ou alcançar um limiar de erro (FARIA, 2018).

A Figura 30 mostra a estrutura das camadas totalmente conectadas, onde neste exemplo há apenas uma camada escondida, mas não é necessariamente sempre assim. O número de camadas é de escolha do desenvolvedor da rede.

Figura 30 – Exemplo de camadas totalmente conectada.



Fonte: (FARIA, 2018).

A camada de saída tem por fim uma técnica de classificação Softmax. Essa função transforma as saídas para cada classe em valores entre 0 e 1 e divide pela soma das saídas, gerando assim um vetor de probabilidades que representa a probabilidade de uma entrada estar em uma determinada classe. Juntos, os componentes deste vetor somam 1, fazendo com que o cálculo do valor de saída de um neurônio dependa do valor dos outros neurônios de saída. (ACADEMY, 2019) A Equação 2.18 mostra a equação do softmax.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{para } j = 1, \dots, K \quad (2.18)$$

onde:

- $i$  representa o índice do neurônio de saída  $\sigma$  que está sendo calculado
- $j$  representa os índices dos neurônios de um nível
- $z$  representa o vetor de neurônios de saída.

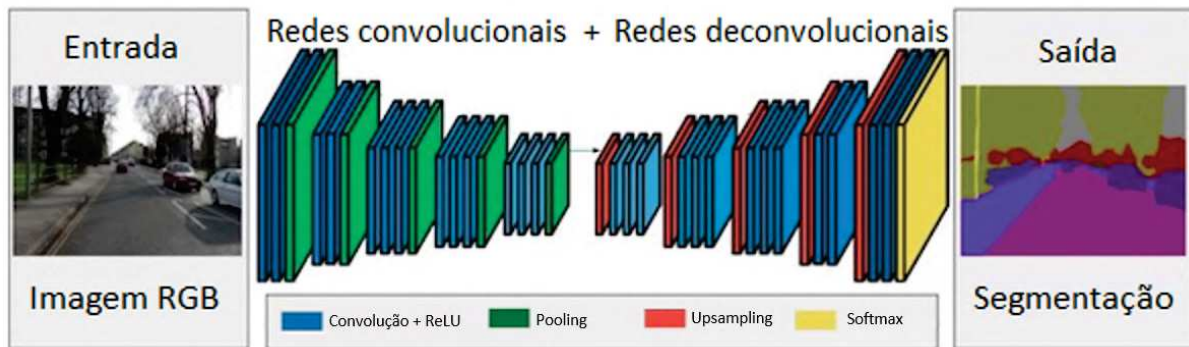
A classificação final da rede é obtida através da escolha do item que possui o valor que indicar a maior probabilidade (FARIA, 2018). O modelo da camada totalmente conectada é capaz de distinguir entre características dominantes e características de baixo nível nas imagens (BALSYS, 2019).



### 2.1.3.2 Redes Deconvolucionais

A operação de deconvolução é na verdade uma operação convolução transposta que também pode ser visto como um redimensionamento de imagem, seguida de uma operação de convolução (GALEONE, 2019). Uma rede deconvolucional possui as mesmas camadas, como por exemplo as camadas de convolução e pooling, mas ao contrário. Desta forma, os resultados obtidos pela rede não serão os mapas de características, mas sim uma imagem (com as mesmas dimensões da original) que destaca as posições da imagem original responsáveis por gerar o mapa de características obtido durante o processamento da rede convolucional (PROTAS, 2017). Os mapas passarão por camadas de *unpooling*, *ReLU* e convolução transposta para que a região que originou a ativação do mapa de características seja destacada. A Figura 31 mostra a representação da rede deconvolucional.

Figura 31 – Representação da rede convolucional conectada a uma rede deconvolucional.



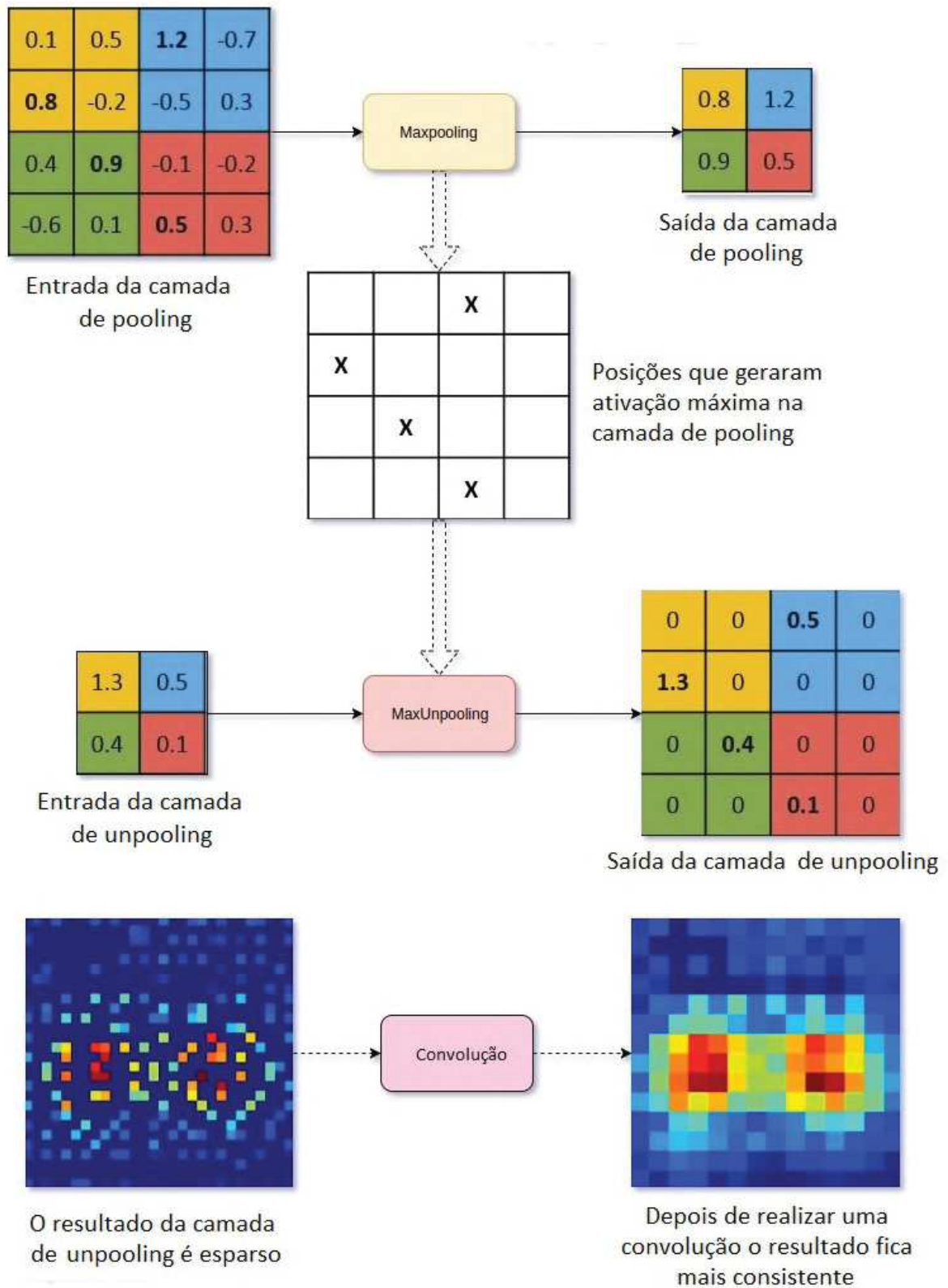
Fonte: Adaptado de (ZAFAR et al., 2018).

Uma desvantagem da utilização da rede deconvolucional é a sua complexidade de implementação pois é necessário fazer a adaptação de toda a estrutura da rede. Além disso, pode ser que a rede precise de uma grande quantidade de imagens na base de dados para conseguir definir as características responsáveis pela ativação de um mapa de característico específico. (GALEONE, 2019)

#### 2.1.3.2.1 Unpooling

O termo *upsampling* se refere a qualquer tipo de técnica que aumenta a resolução atual de uma imagem para uma resolução mais alta. A camada de *unpooling* é uma das técnicas que realiza um *upsampling*, onde esta realiza uma operação que tenta obter o resultado inverso da camada de maxpooling, recontruindo uma versão aproximada das ativações dos mapas de características da camada anterior em seus tamanhos originais, embora o resultado da operação dessa camada não seja completamente reversível. (PROTAS, 2017)

Figura 32 – Exemplo da operação feita na camada de *unpooling* quando os locais de reconstrução da camada anterior são guardados.



Na camada de *unpooling* a rede de deconvolução reconstrói uma versão do mapa de características da CNN da camada anterior. É possível atingir um resultado próximo à entrada da camada de maxpooling através do armazenamento de qual posição originou a ativação máxima de cada região (SHANMUGAMANI, 2018). Para isso, a camada superior é gerada realocando o valor resultado do maxpooling na sua devida posição, o que permite que a estrutura que criou o estímulo seja preservada, e o resto das posições são completados com zeros, como é mostrado na Figura 32.

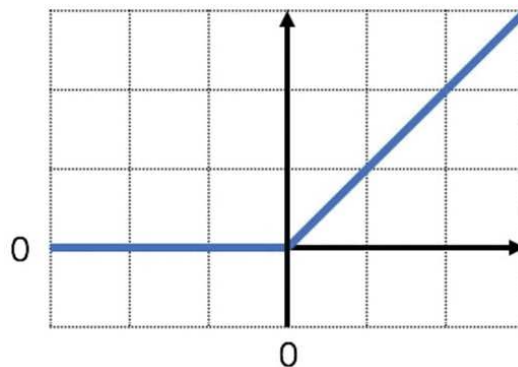
### 2.1.3.2.2 ReLU

As CNNs usam a não-linearidade da função de ativação Rectified Linear Unit (ReLU) para ter certeza de que a ativação dos mapas de características será positiva (ACADEMY, 2019). A função ReLU é dada pela Equação 2.19.

$$f(x) = \max(0, x) = \begin{cases} 0, & \text{se } x < 0 \\ x, & \text{se } x \geq 0 \end{cases} \quad (2.19)$$

Essa função considera apenas a parte não negativa dos valores originais de  $x$  e atribui o valor 0 para a porção negativa. Figura 33 mostra o gráfico da função ReLU.

Figura 33 – Representação gráfica da função ReLU.




Fonte: (LOY, 2019)

A vantagem da utilização dessa função é que ela não ativa todos os neurônios ao mesmo tempo pois os que possuem valor negativo ela não faz a ativação, o que torna a rede esparsa e eficiente. Essa função é mais eficiente do que a sigmóide e a tangente hiperbólica, uma vez que esta não utiliza expoentes. Da mesma forma, essa função será utilizada nas redes deconvolucionais para que as reconstruções sejam válidas e positivas em todas as camadas (ACADEMY, 2019). Figura 34 ilustra um exemplo do efeito da função ReLU em um mapa de características.

Figura 34 – Exemplo da aplicação da função ReLU.

1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80



1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80

Fonte: (BALSYS, 2019)

### 2.1.3.2.3 Camada de Transposição Convolutiva ou Convolução Transposta

Essa camada é utilizada em redes deconvolucionais, sendo a responsável por fazer a operação inversa de uma camada convolutiva. Há uma grande discussão com relação a nomenclatura da operação feita por esta camada, uma vez que o termo "deconvolução" gera a ideia de que através desta operação é possível obter a imagem original que foi utilizada como entrada para a operação de convolução. Apesar da operação de deconvolução ser considerada a operação inversa da convolução, sua função é reconstruir a imagem com as mesmas dimensões de antes, mas não vai reverter o processo com relação aos valores numéricos dos pixels que formam a imagem (GALEONE, 2019).

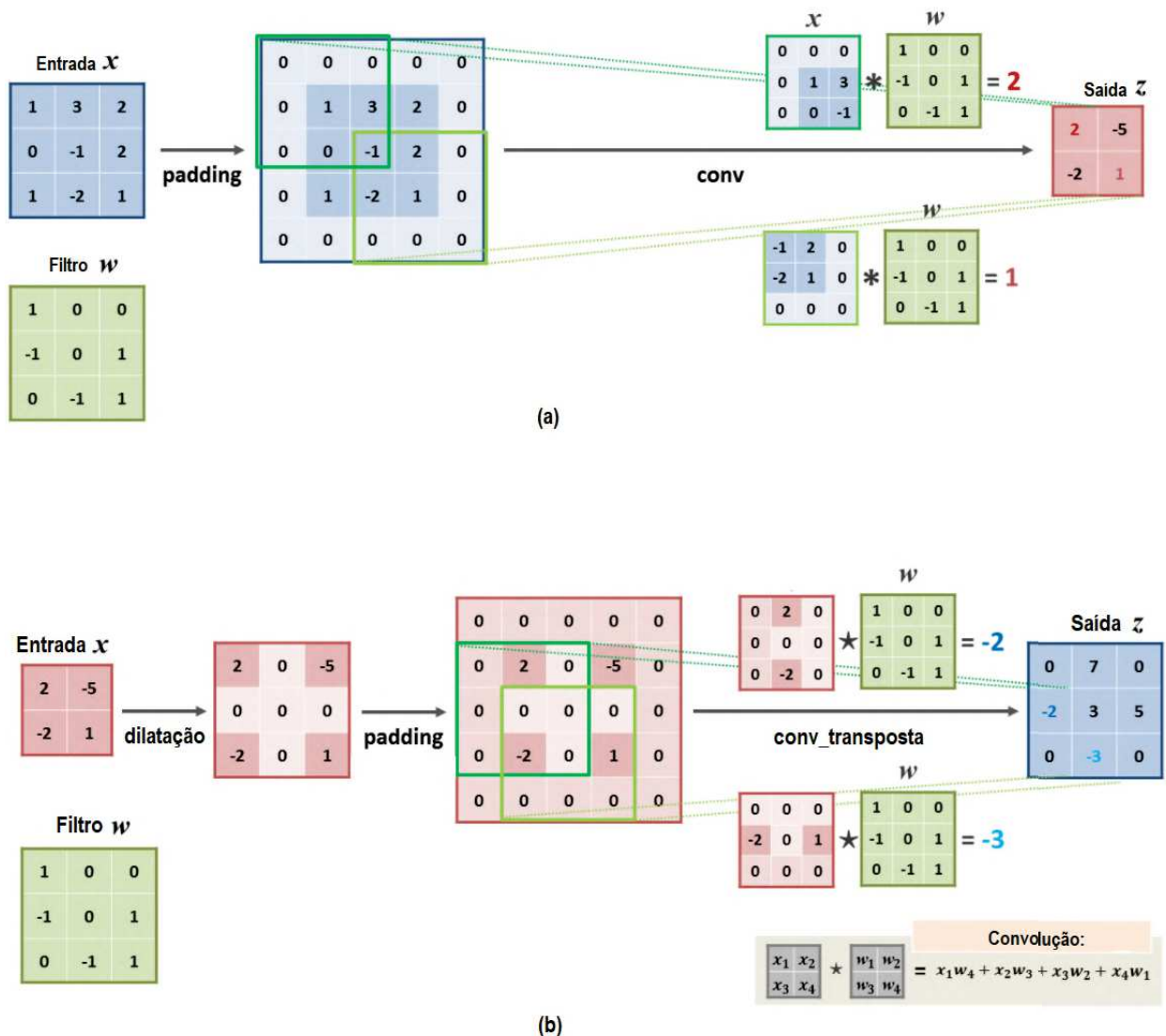
Para se obter um mapa de características como saída da rede, a camada de transposição convolutiva utiliza versões transpostas dos filtros usados na rede convolutiva e os aplica aos mapas de características processados pelas operações de unpooling e ReLU (GALEONE, 2019). Outra opção para obter esse resultado é redimensionar a entrada para a resolução desejada e adicionar uma convolução 2D com o mesmo preenchimento na parte superior da imagem redimensionada (PLANCHE; ANDRES, 2019).

A reconstrução em cada camada resulta em uma imagem semelhante a uma pequena parte da imagem original, com traços que possuem peso de acordo com a sua contribuição para a ativação do mapa de características quando ele foi obtido no processamento da CNN, mostrando quais partes da entrada foram relevantes para a rede identificar uma dada característica (PLANCHE; ANDRES, 2019). A partir de dados compactados em um espaço dimensional menor, essas camadas serão capazes de expandir a informação para o espaço dimensional original.

Seja um mapa de características resultante da convolução entre um mapa de entrada e um filtro com um *stride* definido. Se for aplicado no mapa a operação de deconvolução com o mesmo tamanho de filtro e *stride*, obtém-se como resultado dessa operação uma imagem com a mesma extensão espacial do mapa de características da entrada (GALEONE, 2019).

A Figura 35 é um exemplo da operação de deconvolução. Neste caso, dado a entrada  $x$   $3 \times 3$  e o filtro  $w$   $3 \times 3$  na Figura 35 (a), a entrada sofre primeiramente a convolução com o filtro para um dado valor de  $stride = 2$  e  $padding = 1$ , resultando no mapa de característica  $z$ . Pela propriedade citada anteriormente, e apresentada em (GALEONE, 2019), se o mesmo filtro e os mesmos valores de  $padding$  e  $stride$  forem utilizados ao fazer a convolução transposta como mostrado na Figura 35 (b), o resultado terá a mesma extensão espacial da imagem de entrada.

Figura 35 – Exemplo da aplicação da camada de transposição. (a) Convolução entre matriz de entrada  $x$  e filtro  $w$  com *zero padding* igual a 1 e *stride* igual a 2. (a) Deconvolução da matriz  $x$  e filtro  $w$  com *padding* igual a 1 e *stride* igual a 1.



Fonte: Adaptado de (PLANCHE; ANDRES, 2019)

### 3 Metodologia Proposta

Nas próximas subseções será abordado os detalhes da metodologia desenvolvida no presente projeto.

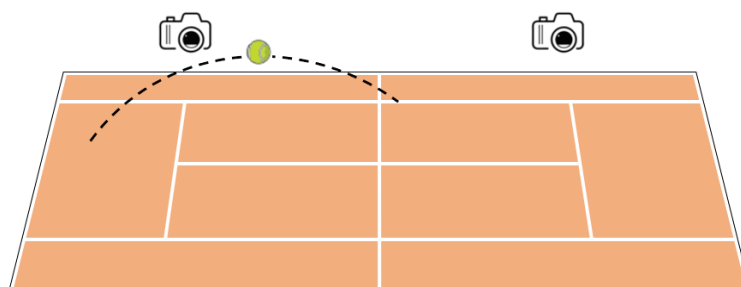
#### 3.1 Detecção da Bola

Duas metodologias são propostas em busca da correta detecção da posição da bola. A primeira é feita através da utilização da Máquina de Vetor de Suporte, onde a análise é feita em imagens focadas em apenas uma região da quadra. Já a segunda é feita através da utilização de redes neurais convolucionais, onde a entrada da rede é composta por imagens que capturam toda a quadra de tênis.

Para avaliar os resultados de ambos os métodos, forma propostas duas formas de instalação das câmeras para captura das imagens.

No experimento feito para o processamento através do SVM, foi utilizado o posicionamento das câmeras como mostrado na Figura 36. Esta configuração foi utilizada devido à característica de processamento do SVM de precisar de uma imagem mais focada na bola, tentando capturar suas características físicas bem definidas na imagem. Com as câmeras instaladas desta forma, a imagem capturada fornecerá uma visão local de uma região do campo, concentradas apenas um campo da quadra, o que limitará a região de interesse onde análise será feita. Não há impedimentos nesta forma de análise, uma vez que grande parte dos lances duvidosos (lances que não se tem certeza se a bola caiu dentro ou fora do campo) que se deseja analisar estão próximos à linha de fundo ou nas linhas laterais.

Figura 36 – Visão lateral da posição das câmeras para aquisição das imagens de entrada para o SVM.

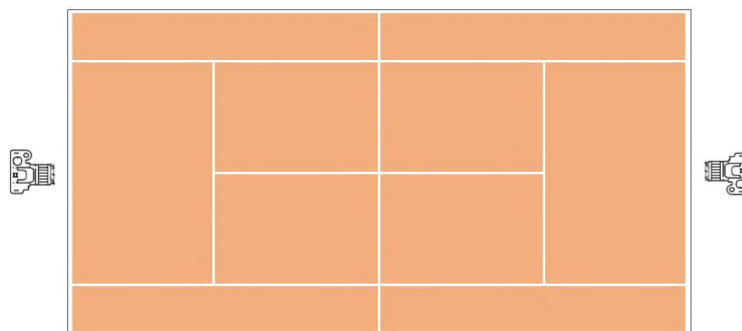


Fonte: próprio autor.

No capítulo seguinte, serão discutidas as vantagens e desvantagens desse tipo de processamento, bem como a viabilidade deste tipo de instalação para as demais detecções que deseja-se apresentar neste trabalho.

Já na segunda fase do segundo experimento, outra forma de posicionamento das câmeras foi adotada. A capacidade de detecção da rede TrackNet (HUANG et al., 2019) que será utilizada permite detecção da posição da bola de tênis em vídeos semelhantes aos gravados durante partidas oficiais de tênis. Além disso, essa perspectiva da partida permite análises complementares que possibilitam a extração de outros dados relevantes da partida que não se limitam apenas à posição da bola. Desta forma, para a análise da segunda fase de detecção da bola e em todas as detecções subsequentes será adotada a nova configuração do posicionamento das câmeras. A câmera é posicionada na área atrás dos jogadores seguindo a disposição mostrada na Figura 37. A lógica implementada segue o pressuposto de que a câmera esta compreendida na região entre os dois vértices externos da linha de fundo da quadra.

Figura 37 – Posicionamento da câmera em relação à quadra de tênis para método de detecção da posição da bola utilizando CNN, detecção das linhas da quadra e do jogador.



Fonte: próprio autor.

### 3.1.1 Detecção Através da Utilização da Máquina de Vetor de Suporte

O método funciona aplicando filtros de cores e transformações combinadas com Máquina de Vetor de Suporte (Support Vector Machine - SVM) para detectar com precisão as posições da linha e da bola. A posição da bola é rastreada junto com diferentes imagens. Inicialmente, o sistema inicializa todo o hardware. Em seguida, após a obtenção das imagens, o algoritmo aplica correção de cor e filtro à cor da bola. Uma detecção de círculo é introduzida para rastrear a posição da bola. Em paralelo, um SVM também é usado para detectar a posição da bola.

Quando comparado aos modelos computacionais usados no reconhecimento de padrões, o SVM apresenta algumas vantagens, como boa capacidade de generalização, algoritmo de treinamento eficiente, estrutura especificada e baixo número de parâmetros a serem ajustados. (MA; ZHAO; HAN, 2015)

Os dados obtidos pelos sensores podem ser ruidosos, errôneos e incompletos (BUONOCORE; JÚNIOR; NETO, 2013). Uma etapa que também interfere na qualidade dos dados é o método de processamento que foi escolhido para obter as informações (SVM

e Hough). Nesta etapa o presente trabalho estuda a combinação dos dados obtidos dos métodos SVM e OpenCV para a detecção da bola de tênis. Os erros das coordenadas da bola são o resultado de vários fatores, como calibração da câmera, baixo número de quadros por segundo (fps) e erros de medição.

A imagem filtrada é usada para estimar os descritores HOG. De acordo com (YANG; YANG, 2017), este tipo de descritor é comumente usado porque pode superar o borrão com seu histograma de células e baixo contraste por normalização de bloco. Esta combinação de HOG e SVM é uma abordagem clássica para classificação de imagens usada em muitas abordagens diferentes. Esses descritores são então compilados em uma única matriz e encaminhados para o algoritmo SVM.

O algoritmo SVM foi treinado de antemão usando um grupo de imagens de bolas de tênis e imagens de quadras de tênis compiladas de várias fontes em imagens do Google. Esse processo foi executado para garantir que o conjunto de treinamento e teste fosse completamente diferente dos dados de teste de validação e experimento. O processo de treinamento foi automatizado com o SVM sendo treinado, testado e verificado várias vezes automaticamente. No início de cada seção, o treinamento SVM começou usando conjuntos de treinamento e teste aleatórios com valores iniciais aleatórios. O modelo com o melhor resultado nos conjuntos de teste e validação foi selecionado após várias gerações, sendo exposto aos dados do experimento somente a finalização completa do treinamento.

### 3.1.2 Detecção Através da Utilização de Redes Neurais Convolucionais

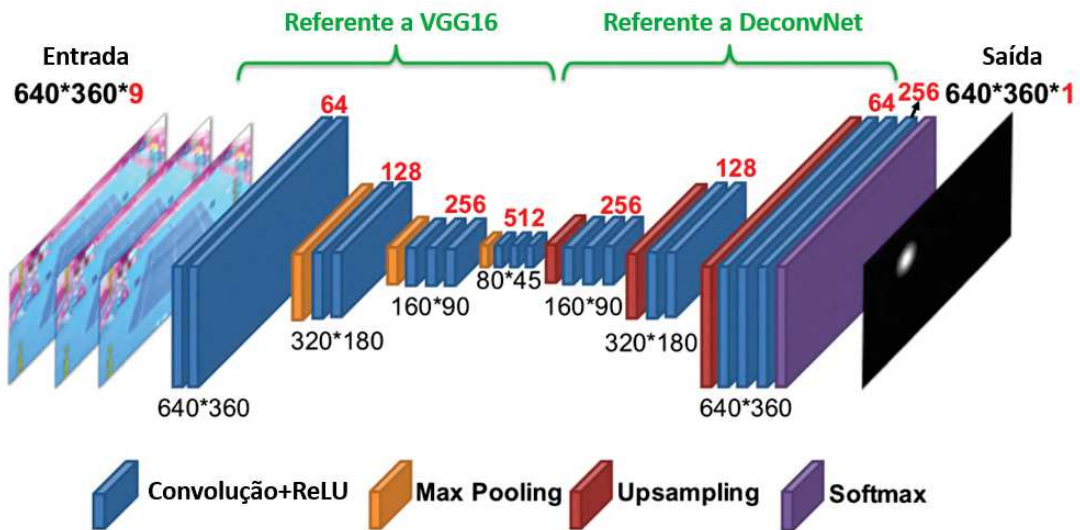
Para a detecção da posição da bola, utilizou-se a rede de aprendizagem profunda TrackNet proposta por (HUANG et al., 2019). Esta rede neural é formada pela junção de duas redes. As treze primeiras camadas são referentes às camadas da rede neural convolucional proposta em (SIMONYAN; ZISSERMAN, 2014b). Em seguida, a TrackNet é formada por camadas referentes à rede deconvolucional proposta por (NOH; HONG; HAN, 2015). A rede utiliza camadas de upsampling que possuem a função de recuperar a perda de informação gerada pelas camadas de pooling. A rede apresentou bons resultados de detecção em vídeos de competições esportivas de tênis e badminton (HUANG et al., 2019). Figura 38 ilustra a arquitetura da rede TrackNet.

Na fase de treinamento, a rede utiliza uma grande variedade em seu conjunto de dados, apresentando 16.118 frames em seu conjunto de dados. Esses dados foram coletados de 9 vídeos gravados em vários tipos de quadras (i.e., grama, saibro, quadra dura, etc), o que deu a rede a capacidade de reconhecer a posição da bola em diferentes cenários e, por consequência, aumentando sua robustez.

Devido à alta velocidade atingida pela bola, muitas vezes a captura da mesma nas imagens se apresenta borrada, o que torna difícil saber com precisão a sua real posição. Em outros momentos a bola fica oclusa ou se mistura com as marcações do campo, como



Figura 38 – Arquitetura da rede TrackNet.



Fonte: Adaptado de (HUANG et al., 2019)

mostrado nas Figuras 39 e 40. Uma das vantagens da rede utilizada é a sua capacidade de localizar a bola de tênis mesmo nesses momentos onde a posição da bola não é facilmente encontrada. Essa característica se deve ao modo como os frames do vídeo entram na rede, sendo cada entrada uma concatenação de três frames consecutivos, gerando assim um histórico para a rede e possibilitando que esta aprenda o padrão de trajetória da bola.

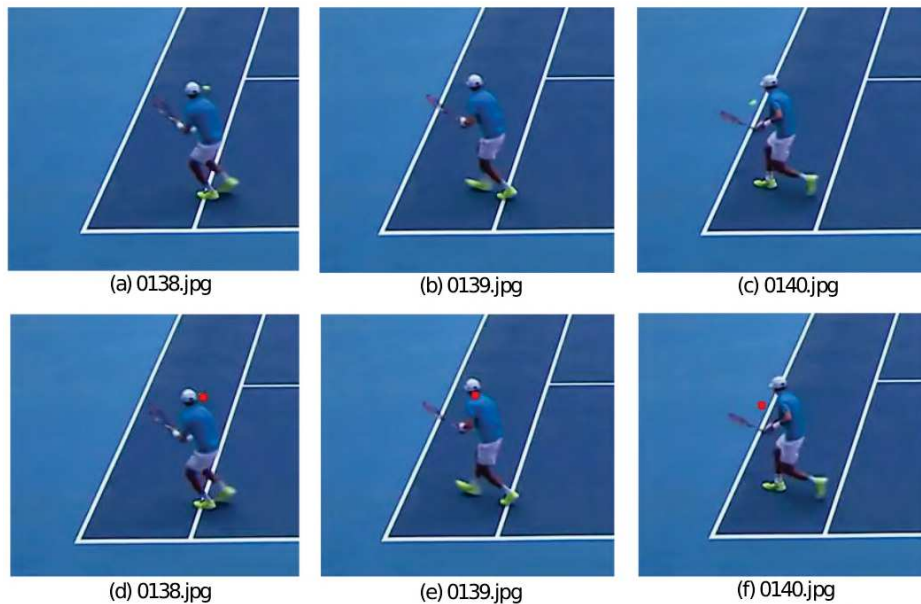
Figura 39 – Bola se mistura com marcações do campo.



Fonte: (HUANG et al., 2019)

Considerando o centro da bola  $(x_0, y_0)$ , tem-se na saída da rede um mapa de calor onde cada pixel possui valores contínuos no intervalo de  $[0,255]$ . O mapa de calor é

Figura 40 – Bola oclusa pelo jogador.



Fonte: (HUANG et al., 2019)

uma técnica de representação gráfica de dados que mostra os valores individuais contidos na matriz através de uma representação de cores. Por meio do mapa de calor, pode-se identificar visualmente a posição do objeto, uma vez que a localização da bola no espaço é distinguível através da variação na intensidade da cor na imagem. Esses valores são baseados em uma distribuição gaussiana 2D que se apresenta centralizada no centro da bola de tênis dada pela Equação 3.1.

$$G(x, y) = \left[ \left( \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}} \right) (2\pi \cdot \sigma^2 \cdot 255) \right] \quad (3.1)$$

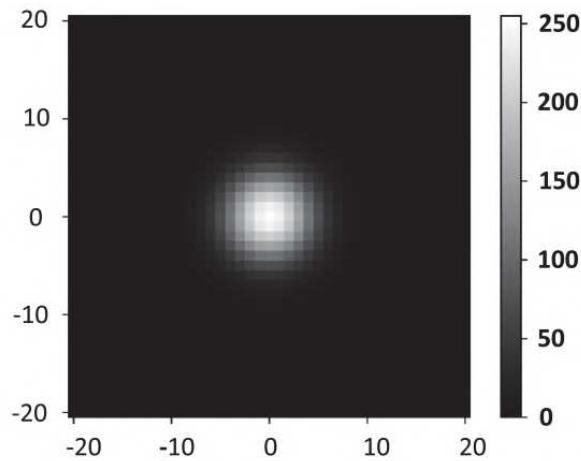
O primeiro termo da expressão acima é uma distribuição gaussiana centralizada em  $(x_0, y_0)$  com uma variância de  $\sigma^2$ . Já o segundo termo tem a função de configurar o valor resultante no intervalo  $[0, 255]$ .

Seja  $L(i, j, k)$  um arranjo onde  $0 \leq i < 640$ ,  $0 \leq j < 360$  e  $0 \leq k \leq 255$ . As dimensões  $i$  e  $j$  representam, respectivamente, a altura e comprimento da imagem. A dimensão  $k$  é uma profundidade equivalente ao valor dentro do intervalo  $[0, 255]$ . A camada softmax calcula a distribuição de probabilidade da profundidade de  $k$  e atribui o valor dentro da escala de 256 tons de cinza. Seja  $P(i, j, k)$  a probabilidade da profundidade  $k$  na posição  $(i, j)$ . Então a função da camada softmax dada por  $P(i, j, k)$  é dada pela Equação 3.2

$$P(i, j, k) = \frac{e^{L(i, j, k)}}{\sum_{l=0}^{255} e^{L(i, j, l)}} \quad (3.2)$$

Para gerar a saída da rede representada pelo mapa de calor, mostrado na Figura 41, é feito ainda o cálculo do valor de  $h(i, j)$  para cada pixel  $(i, j)$ .

Figura 41 – Saída da rede TrackNet.



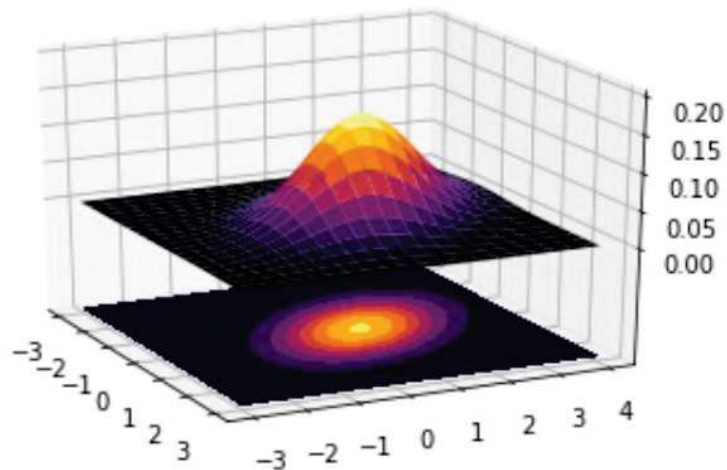
Fonte: (HUANG et al., 2019)

A função  $h(i, j)$  é dada por pela Equação 3.3.

$$h(i, j) = \arg \max_k P(i, j, k) \quad (3.3)$$

Basicamente, essas operações obtém a representação 2D de uma gaussiana 3D calculada a partir da posição da bola como mostrado no exemplo da Figura 42, sendo  $(x_0, y_0)$  o local onde a gaussiana está centralizada.

Figura 42 – Representação 2D e 3D de uma gaussiana.



Fonte: próprio autor.

Após a obtenção dessa saída, Huang et al. (HUANG et al., 2019) realizou um processamento de forma a identificar as coordenadas do centro da bola e, através da variância da distribuição, extrair o diâmetro da mesma. A imagem primeiramente é convertida em um mapa de calor preto e branco binário, onde através da função `cv2.threshold`, define-se

um limiar de valor  $t$ . Caso o valor do pixel ultrapasse o valor de  $t$ , o mesmo é redefinido com valor 255. Caso contrário, o pixel é configurado com o valor zero. Para esta aplicação, o artigo considerou o raio médio da bola como cerca de 5 pixels, o que implica em  $\sigma^2 = 10$ , correspondendo à aproximadamente  $G(x, y) = 128$ . Dessa forma, foi utilizado o valor do limiar  $t = 128$ . O segundo passo no processamento da saída da rede é a aplicação do Método do Gradiente de Hough para encontrar círculos na imagem binária preto e branca de saída da rede. Os autores consideraram as saídas que apresentaram apenas um círculo na imagem de saída da rede, caracterizando um reconhecimento correto da rede e sem falsos reconhecimentos no ambiente. No presente trabalho, foram incluídos os frames que apresentaram mais de um círculo no mapa de calor da saída da rede, porém que atendessem a exigência de distanciamento entre as localizações detectadas, isto é, para que um círculo seja considerado como uma correta localização da bola, o mesmo deve estar a uma distância menor do que 40 pixels da última bola detectada.

Essa medida tem por objetivo levar em consideração as localizações que foram detectadas corretamente mas que apresentaram detecções erradas em alguma parte da imagem causada por alguma característica do ambiente que se assemelhou com o que a rede compreende como uma características da bola. Essa é uma forma de alinhar a detecção com a realidade apresentada à rede, pois como muitas das vezes a bola se apresentará de forma distorcida ou borrada, abre margem para momentos em que o ambiente terá uma semelhança com as características da bola em locais onde ela não está realmente. Esse aspecto da rede a torna flexível na detecção da posição da bola, gerando rastreamentos com uma ótima precisão. Então, se em um mesmo frame a bola é detectada na posição correta mas apresenta outras detecções que não condizem com a real localização da bola, ao invés de desconsiderar essa saída da rede como foi feito em (HUANG et al., 2019), o presente trabalho considerará a localização correta dessa saída e descartará as incorretas. Os impactos nos resultados causados por esta adaptação serão apresentados no próximo capítulo.

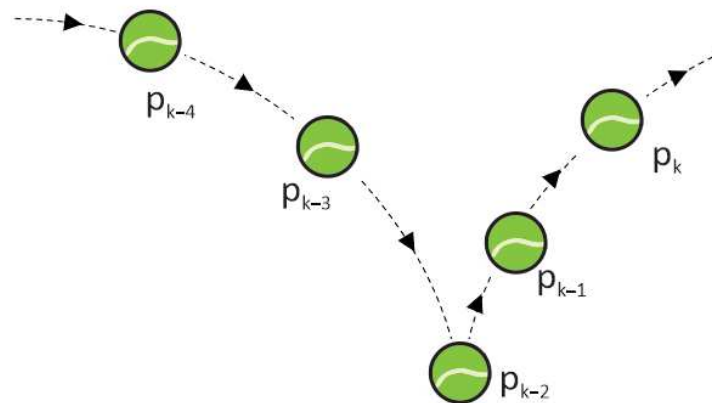
### 3.2 Detecção do Quique da Bola

A partir da detecção da posição da bola, é possível determinar mais dois parâmetros: *(i)* o momento em que a bola quica na quadra; e *(ii)* se ela quicou dentro ou fora da quadra. Para isso, foi feito o armazenamento dinâmico das últimas quinze posições da trajetória da bola e seus valores serão atualizados a cada nova detecção de posição. Essas posições são armazenadas desta forma para que quando for necessário indicar a posição atual da bola também seja possível imprimir suas quatorze últimas posições, gerando para o usuário um histórico do caminho percorrido pela bola. Esse histórico também permite determinar o momento em que a bola quica na quadra utilizando suas cinco últimas posições.

A detecção do quique é feita comparando a altura da bola nessas posições. Armazenam-

se os valores da diferença de altura das bolas e esses valores são atualizados a cada nova posição detectada. A diferença de altura entre uma posição e outra é positiva quando a bola está descendo e negativa quando está subindo. Com isso, é possível determinar o quique da bola através de uma janela deslizante de tamanho fixo de 4 posições, fazendo assim a comparação as quatro diferenças de altura provenientes das cinco últimas posições da bola, onde duas diferenças de altura positivas seguidas de duas diferenças de altura negativas caracterizam um quique, como mostrado na Figura 43

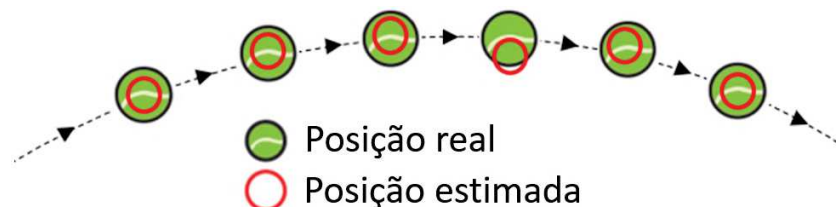
Figura 43 – Quique da bola.



Fonte: próprio autor

Na teoria essa comparação funcionaria apenas com uma janela deslizante que permitisse a comparação de três posições da bola e calculando apenas duas diferenças de altura, porém podem ocorrer situações como mostrado na Figura 44 onde a posição da bola estimada pela rede apresenta outliers que ficam um pouco deslocados em relação ao a sua real posição. Este evento causaria uma falsa detecção de quique na rede. Para corrigir esse problema foi adotado a abordagem com a comparação com cinco posições, o que torna a detecção do quique mais segura contra deslocamentos de posição.

Figura 44 – Deslocamento da posição estimada da bola.



Fonte: próprio autor

Sabendo a posição do local onde a bola quicou, é possível determinar se a mesma caiu dentro ou fora do campo. Assim, o algoritmo verifica se a coordenada (x,y) do quique

da bola situa-se dentro do contorno descrito pelos cantos externos da quadra previamente identificados utilizando a função *cv2.pointPolygonTest*.

A formulação matemática da curva que descreve as coordenada espaciais do movimento da bola de tênis mostrado nas Figuras 43 e 44 é dada por uma relação quadrática básica, mostrada na Equação 3.4. Essa equação é uma ótima aproximação para o movimento real da bola.

$$p_k = -a_k \cdot t^2 + b_k \cdot t + c_k \quad (3.4)$$

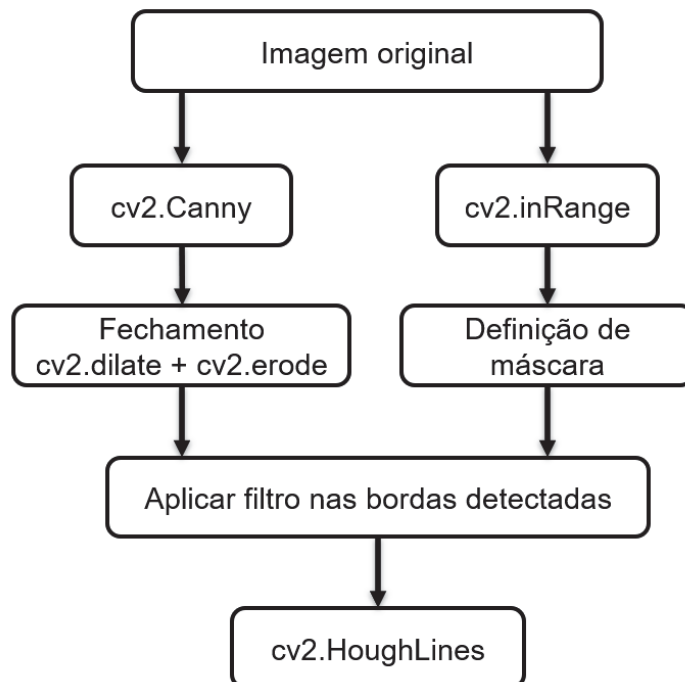
onde:

- $a_k$ ,  $b_k$  e  $c_k$  representam os coeficientes que caracterizam os diferentes arcos descritos pela bola durante seu deslocamento no jogo
- $p_k$  representa a altura da bola em função do tempo  $t$

### 3.3 Detecção da Quadra

O segundo passo deste trabalho é a detecção das linhas da quadra. Para isso, foi empregado o método mostrado no fluxograma da Figura 45, onde cada passo será detalhado a seguir.

Figura 45 – Fluxograma do método de detecção da quadra.



Fonte: próprio autor

Inicialmente aplica-se o método de detecção de contornos de Canny (KAEHLER; BRADSKI, 2016) na imagem. Este método destaca todos os contornos da imagem utilizando dois limiares previamente determinados e resulta em uma imagem preta com as bordas de valor dentro do intervalo dos limiares mínimo e máximo destacadas em branco. Esse processo gera um maior realce na quadra e nos contornos presentes na imagem. Porém, os cantos podem sofrer descontinuidade. Para corrigir esse problema, é necessário a aplicação de operações morfológicas para eliminar descontinuidades.

Essas operações morfológicas precisam de duas entradas: a imagem original e o elemento estruturante, também conhecido como kernel. O kernel responsável por decidir a natureza da operação. O método utilizado neste trabalho será o processo de fechamento, que é a combinação de dos operadores básicos de Dilatação seguido por de Erosão. Esse processo é útil para completar objetos que possuam pequenos orifícios.

Em seguida, implementou-se uma máscara com base na cor do saibro para selecionar a área do jogo como área de interesse. A função *cv2.inRange* foi utilizada para encontrar as cores dentro dos limites especificados e aplicar a máscara na imagem original. Com o resultado obtido, aplicou-se a função *cv2.findContours* para achar o maior contorno da região de interesse. Este é preenchido com a cor branca e armazenado como a máscara da região de interesse. Esse processo é necessário, pois se forem considerados como filtro apenas o resultado do reconhecimento da cor do saibro, as linhas do campo não serão inclusas na aplicação da máscara.

A máscara obtida é aplicada na imagem resultante do método de detecção de Canny. Assim, os resultado serão os contornos que estão dentro da região de interesse, o que elimina todos os outros contornos que poderiam interferir no processamento.

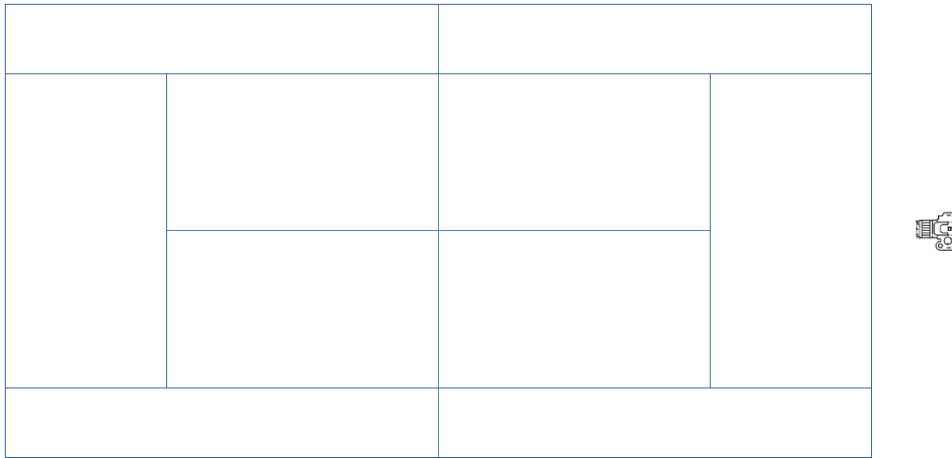
Com os contornos da região de interesse identificados, o próximo passo é a identificação das linhas do campo. O método de detecção de linhas de Hough foi utilizado para encontrar todas as linhas presentes na região de interesse com comprimento maior que 250 pixels.

Após a identificação das retas, busca-se os pontos de intersecção entre elas para definir todos os vértices que compõe a quadra e guardar suas coordenadas em um vetor.

O próximo passo no reconhecimento do campo é a aplicação de uma lógica para o reconhecimento dos vértices e conectá-los de forma a compor as linhas corretamente. A câmera é posicionada na área atrás dos jogadores seguindo a disposição mostrada na Figura 46. A lógica implementada segue o pressuposto de que a câmera esta compreendida na região entre os dois vértices externos da linha de fundo da quadra.

Esta disposição faz com que a imagem capturada mostre as linhas do campo do jogador à sua frente mais nítidas do que a do jogador do outro campo. Esta diferença proporciona um reconhecimento do campo mais próximo à câmera melhor do que o que está mais distante. Outro fator que interfere nesse reconhecimento é a composição do

Figura 46 – Posicionamento da câmera em relação à quadra de tênis.

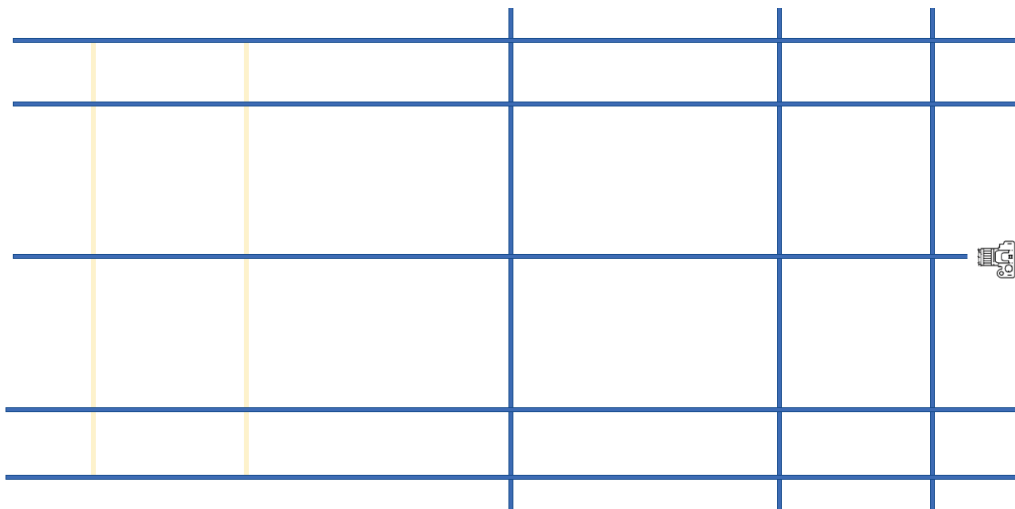


Fonte: próprio autor.

material da quadra. O pó do saibro se solta da quadra durante as jogadas e eventualmente fica depositado sobre a linha do campo. Esse tipo de problema não afeta o reconhecimento do campo que está próximo da câmera. Porém, influencia na detecção das linhas mais distantes, o que dificulta a definição de algumas linhas do campo. Outro fator que atrapalha na detecção é a faixa branca da rede que muitas vezes esconde uma das linhas, sendo mais um empecilho na detecção do outro campo. Isso acontece com todas as linhas da quadra, porém quanto mais distante da câmera, mais difícil é para o sistema de visão computacional detectar as linhas que estão com saibro depositado sobre elas.

Desta forma, as linhas detectadas corretamente pela câmera são as destacadas em azul na Figura 47 e as linhas com cor clara representam as linhas de difícil detecção pelos motivos explicados acima.

Figura 47 – Linhas detectadas a partir das imagens capturadas pela câmera.



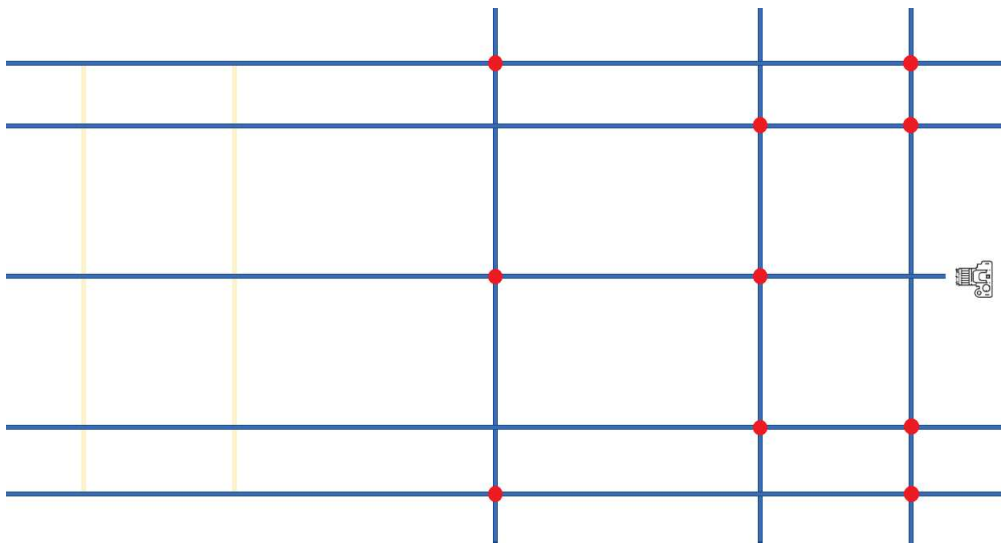
Fonte: próprio autor.



As linhas azuis ultrapassam os limites do campo pois o método de detecção utilizado, caso detecte uma linha de tamanho maior do que um valor definido anteriormente, desenha a linha estendendo seu comprimento. Com isso, é possível obter a interseção da reta de interesse com todas as outras detectadas.

A partir da detecção das linhas da imagem, é possível calcular e armazenar suas intersecções em uma lista. Em seguida, a lista é varrida de forma a encontrar os pontos principais que possibilitam a formação do campo próximo à câmera. Assim, os pontos são localizados conforme mostra na Figura 48.

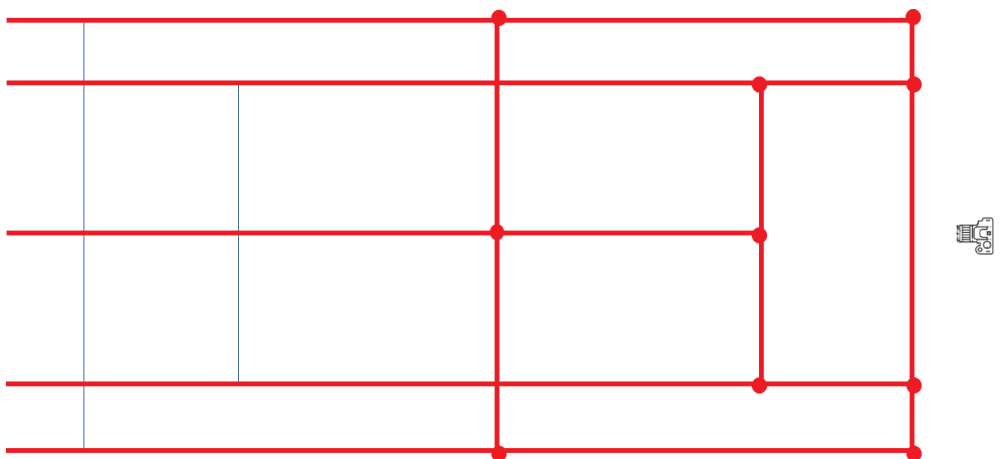
Figura 48 – Pontos de interseção obtidos a partir das linhas reconhecidas.



Fonte: próprio autor.

A partir destes pontos é possível determinar as linhas em vermelho representadas na Figura 49, que são as principais para a obtenção do campo próximo à câmera.

Figura 49 – Linhas detectadas a partir das imagens capturadas pela câmera.



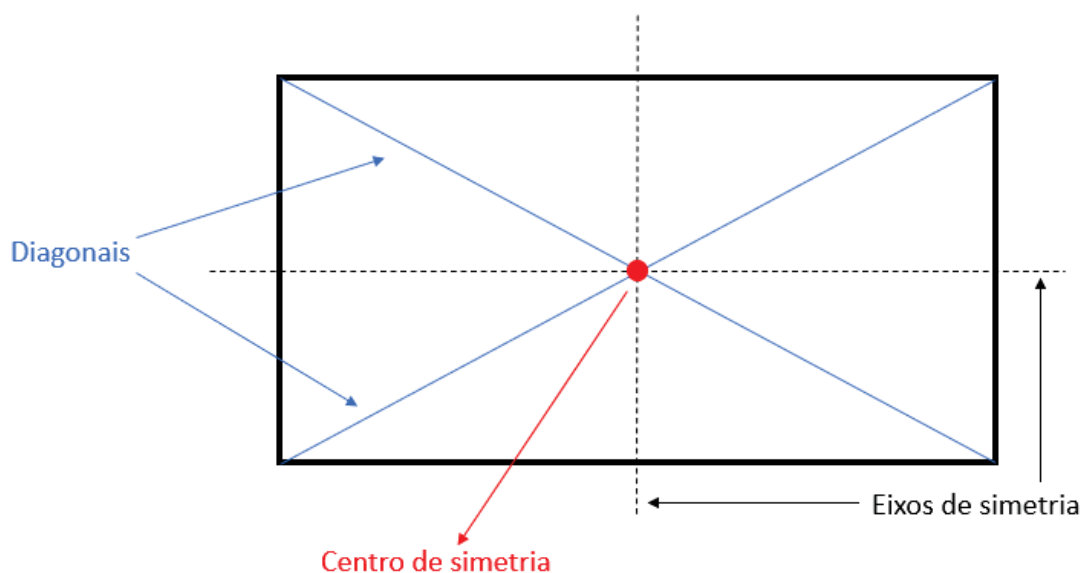
Fonte: próprio autor.

Como foi discutido anteriormente, a detecção das linhas do fundo do outro campo é de difícil detecção, o que não permite a determinação dos vértices do campo da mesma forma que foi desenvolvida no primeiro campo.

De forma a determinar os pontos do outro campo e solucionar esse problema na detecção das linhas, o presente trabalho propõe uma forma de determinar os pontos desejados utilizando os pontos vértices previamente obtidos pela intersecção das linhas reconhecidas do campo.

Sabendo todos os vértices e o ponto central da quadra, é possível assim realizar a projeção dos vértices do outro campo fazendo uso das propriedades matemáticas do retângulo. Para qualquer retângulo sabe-se que suas diagonais são iguais, ele possui dois eixos de simetria e que as diagonais se interceptam no ponto médio, sendo este um centro de simetria, como mostrado na Figura 50.

Figura 50 – Componentes de um retângulo.

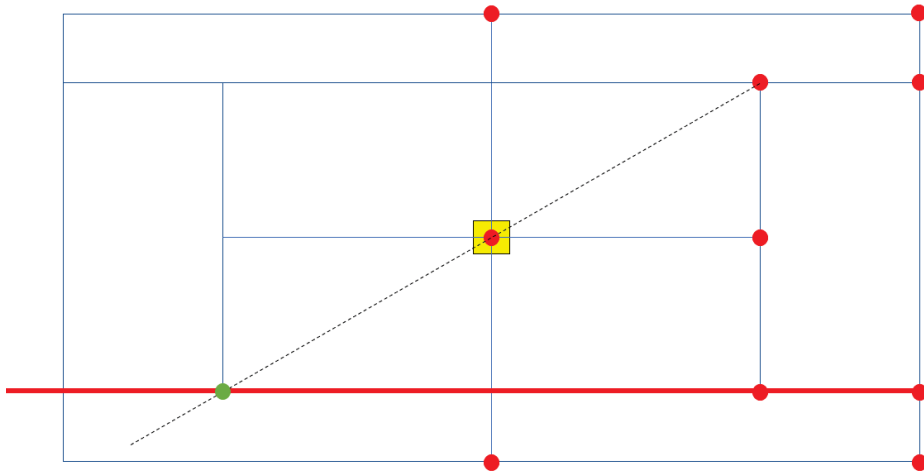


Fonte: próprio autor.

Aplicando tais propriedades na quadra de tênis, pode-se utilizar o centro da quadra como ponto de simetria. Como pode ser visto na Figura 46, os pontos que definem as linhas do campo podem ser descritos como os vértices de retângulos com o mesmo ponto central de simetria.

Desta forma, é possível usar o centro da quadra para determinar todos os pontos da quadra. Para isso, definiu-se uma reta descrita a partir do ponto de simetria e de um dos vértices de um dos retângulos. Note que para determinar o ponto simétrico que será projetado na outra quadra, é necessário determinar a intersecção dessa reta com a linha anteriormente detectada pelo método de Hough, tal como mostrado na Figura 51.

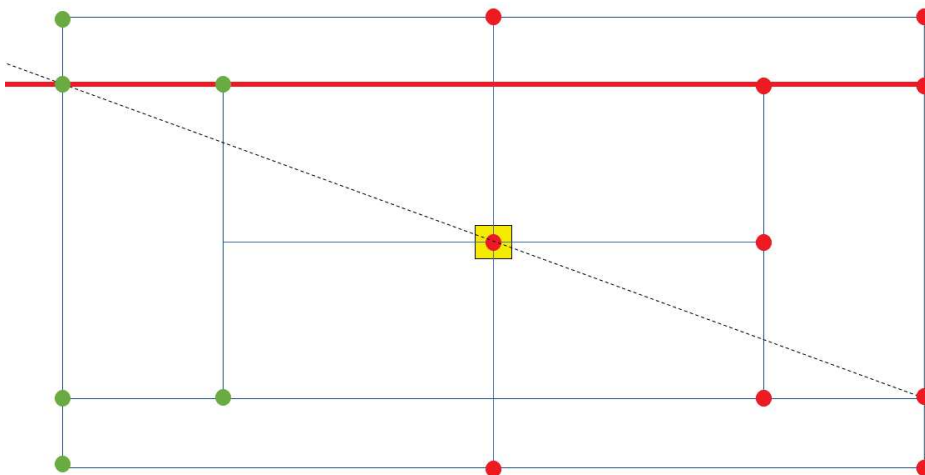
Figura 51 – Projeção do ponto da quadra através da utilização do ponto de simetria.



Fonte: próprio autor.

Repetindo esse processo para todos os pontos da quadra, obtém-se um resultado com o mostrado na Figura 52.

Figura 52 – Pontos da quadra projetados através da utilização do ponto de simetria.

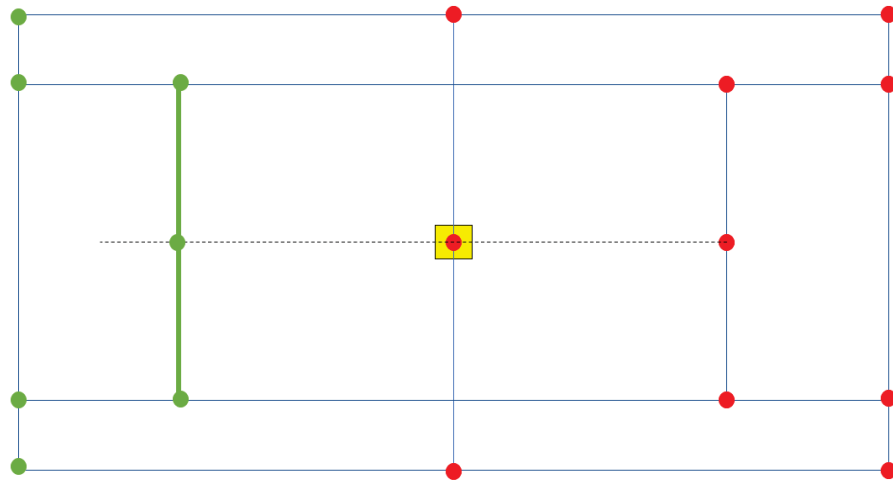


Fonte: próprio autor.

A Figura 53 ilustra a definição do ponto sendo feito de forma semelhante à descrita anteriormente. Contudo, ao invés de utilizar uma linha previamente detectada, neste ponto é necessário usar uma linha descrita por dois pontos que foram projetados pela simetria. Com todos os pontos determinados, é possível determinar todas as linhas importantes para que a quadra seja completamente reconhecida.

A imagem capturada pela câmera não mostrará um retângulo como representado, mas sim um quadrilátero como apresentado na Figura 54. Essa representação é consequência de um problema clássico para detectar retas em imagens, onde as linhas paralelas não

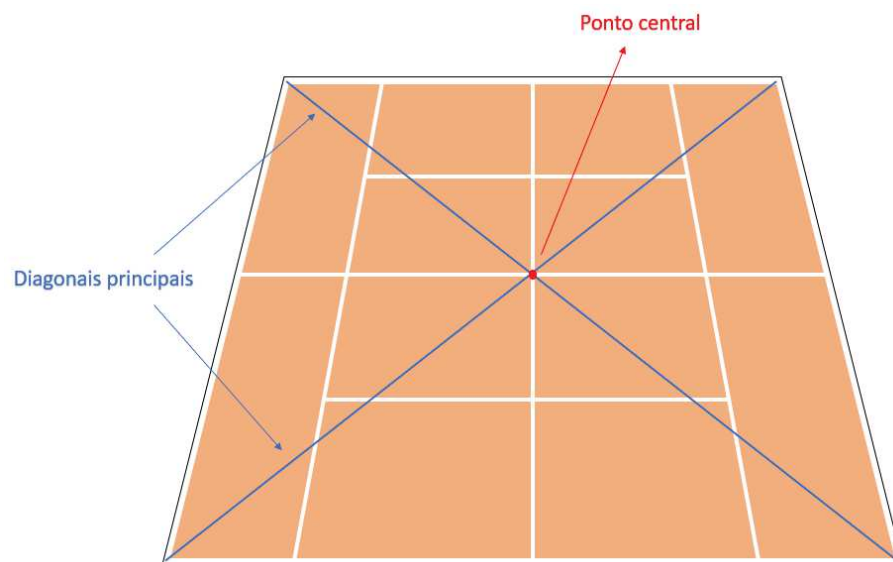
Figura 53 – Projeção do ponto da quadra através da utilização do ponto de simetria e de reta definida através de outros pontos projetados.



Fonte: próprio autor.

serão mais paralelas devido à projeção de perspectiva da câmera, ou seja, a imagem mostra a representação gráfica do objeto como aparece do ponto de vista da posição da câmera.

Figura 54 – Visão da quadra do ponto de vista da câmera com as diagonais e o ponto central representados



Fonte: próprio autor.

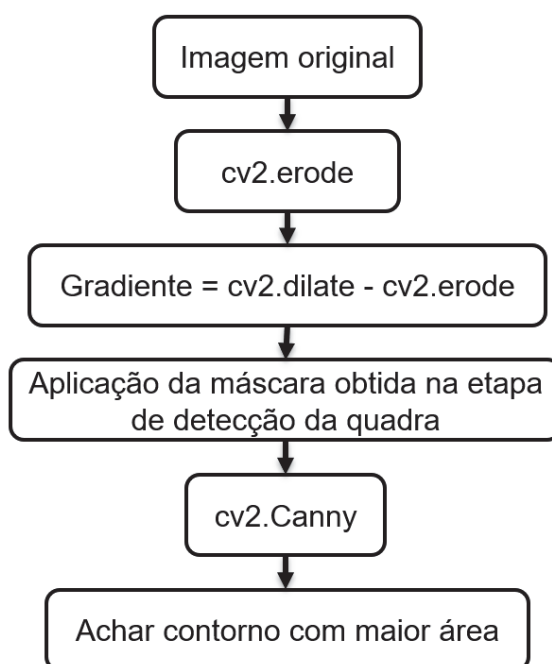
Essa lógica para a detecção das linhas da quadra é apresentada aqui devido às características físicas da quadra e para melhor entendimento do leitor de que o procedimento é verdadeiro para a quadra em questão. O quadrilátero que é capturado pelas câmeras pode ser processado desta forma pois o interesse não é a medida das retas diagonais, mas sim com a determinação do ponto de cruzamento entre as diagonais. E assim, seja

possível descrever uma reta que passa por este cruzamento e possui intersecção com as retas anteriormente detectadas pelo método de Hough, encontrando, portanto, os pontos do outro campo.

### 3.4 Detecção dos Jogadores

Para se realizar a detecção dos jogadores foi proposto o método de detecção descrito no fluxograma da Figura 55.

Figura 55 – Fluxograma do método de detecção do jogador.



Fonte: próprio autor

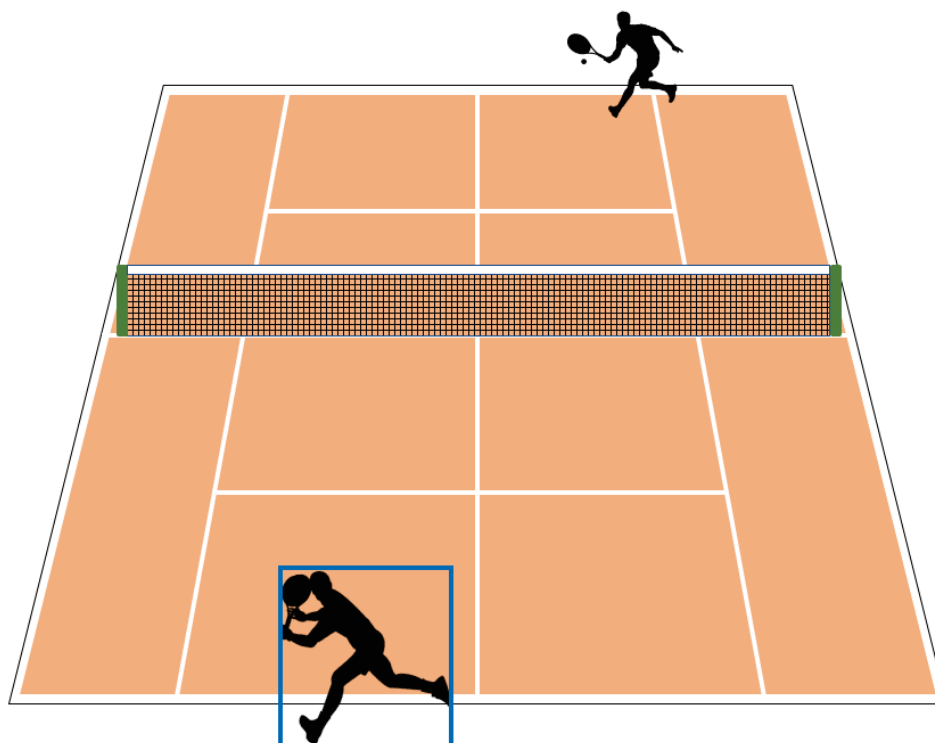
O primeiro passo do estágio de detecção do jogador dentro da quadra é a aplicação da função *cv2.erode* que desgasta os limites dos objetos em primeiro plano. Essa função será responsável por suavizar as linhas da quadra, tornando mais simples a detecção nas próximas etapas. Em seguida, aplicou-se a operação morfológica gradiente, que é equivalente à resultante da diferença entre a operação Dilatação e Erosão. O resultado obtido é uma imagem com o contorno dos objetos valorizados, mas sem a presença das linhas.

Nesta etapa, o método adotado para a detecção do jogador dependerá de alguns parâmetros detectados na etapa de reconhecimento da quadra de tênis. A máscara obtida no processamento das linhas da quadra de tênis é utilizada para que novamente a área do campo seja filtrada e componentes que podem gerar interferências na detecção sejam excluídos, como por exemplo, jogadores de outras quadras e juízes.

Assim, aplicou-se o método de detecção de Canny na imagem resultante para que a imagem tenha apenas os contornos segundo o limiar definido, assim como aplicado na etapa de detecção do campo. Os contornos detectados são analisados de forma a obter aquele que possua maior área, finalizando o processo de detecção do jogador.

No método desenvolvido no presente trabalho, o objetivo é encontrar o jogador presente no campo mais próximo à câmera, assim como é mostrado na Figura 56.

Figura 56 – Representação da detecção dos jogadores durante a partida de tênis



Fonte: próprio autor.

A escolha desta abordagem é explicada pelos problemas na detecção do adversário causadas pela região situada no fundo do campo, onde a grade e a coloração dessa área não criam uma boa diferenciação do jogador quando aplicado o método explicado acima. É importante ressaltar que como o método apresentou bons resultados para o que foi proposto, e devido a aplicação geral utilizar somente uma câmera de cada lado, não há problema em seguir com essa linha de detecção uma vez que ambos os jogadores serão detectados, porém em câmeras diferentes.

### 3.5 Interface Gráfica

Para fornecer uma plataforma que mostre ao usuário todos os resultados obtidos, foi implementada uma interface gráfica na plataforma Android Studio 4.1.1. Esta plataforma é responsável por apresentar de forma iterativa todos os processamentos feitos a partir dos

métodos discutidos anteriormente. E assim, permitir que tanto o juiz quanto os jogadores tenham acesso às informações do jogo através de um smartphone ou tablet.

É importante salientar que aplicativo servirá apenas como interface, não fazendo o processamento dos dados dentro no dispositivo. O real processamento será feito em um computador presente na mesma rede wireless que o dispositivo que estiver executando a aplicação. Este dispositivo será o responsável por ler a opção escolhida pelo usuário e enviar uma mensagem ao computador informando a opção selecionada. Uma vez enviada a mensagem, o computador acionará o processo referente a opção recebida, enviando uma resposta após a execução do processo.

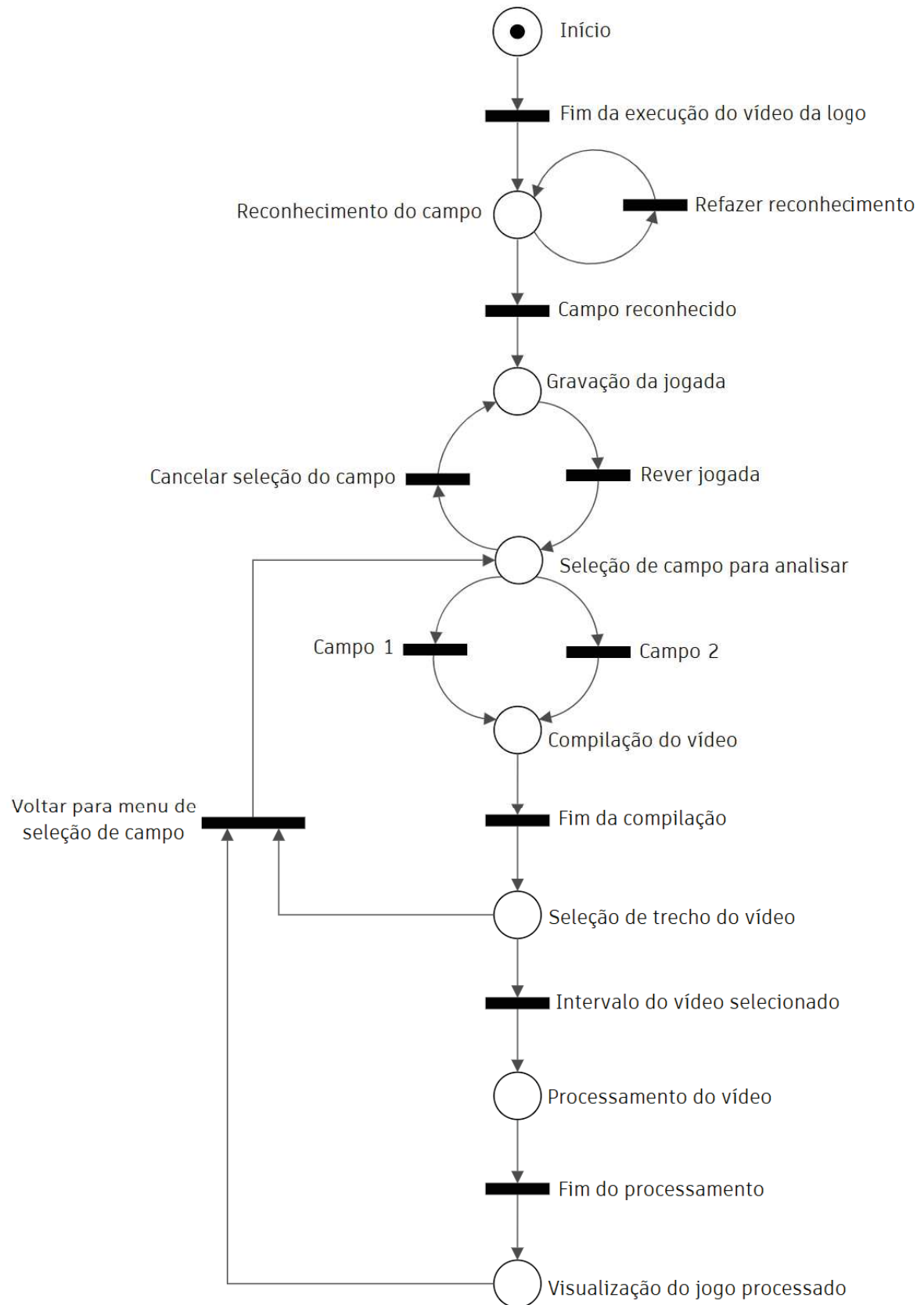
As mensagens são enviadas e recebidas entre o aplicativo Android e o computador através de sockets que armazenam dados, como por exemplo, a informação da opção de operação do usuário e arquivos de mídia que serão enviados para que o dispositivo móvel mostre o resultado do processamento ao usuário. A Figura 57 mostra a rede de Petri referente ao funcionamento do aplicativo. A representação aqui é feita com relação à interface do Android. Note que cada lugar (ou seja, círculo) na rede representa um estado onde o aplicativo pode estar, e cada transição representa uma decisão/ação que o usuário pode realizar mediante o atual estado onde se encontra. As etapas mais intrínsecas a cada estado serão abordadas mais detalhadamente no próximo capítulo.

O programa inicia com uma tela que mostrará um vídeo com a logo do projeto e, ao fim do vídeo, o usuário poderá iniciar o reconhecimento do campo e refazê-lo caso não esteja satisfeito com o resultado. Quando o campo for reconhecido, o jogo começará a ser gravado no momento que o usuário desejar. Caso haja um intervalo de tempo em que o jogador ou o juiz queira rever um lance, a gravação do jogo é interrompida e o usuário deverá escolher de qual câmera deseja rever a gravação. Em seguida, o vídeo da câmera será compilado a partir do momento em que o usuário deu início à gravação e o vídeo será mostrado na tela do dispositivo. O próximo passo é a seleção do trecho do vídeo que deseja ser processado para detecção dos componentes do jogo. O intervalo é processado e, ao final do processamento, o vídeo é mostrado para o usuário, que poderá voltar para a seleção dos campos e rever a mesma jogada através de outra câmera. Caso não queira rever o outro vídeo, o usuário poderá voltar à tela principal e gravar outra jogada.

Os comandos enviados pelo dispositivo móvel são recebidos pelo computador e em seguida são interpretados pelo *software* desenvolvido. De acordo com cada decisão do usuário, o programa desencadeará uma série de processos específicos àquela demanda solicitada. Para desencadear cada processo, utilizou-se *Robot Operating System - ROS*.

O sistema ROS é formado por diversos nós diferentes executados simultaneamente e que estabelecem comunicação entre si através da transmissão de mensagens. A partir dele é possível implementar uma arquitetura cujo processamento de dados seja feito em paralelo. Assim, esse problema pode ser resolvido criando nós dentro do ROS em um

Figura 57 – Rede de Petri do aplicativo desenvolvido.



Fonte: próprio autor.



computador capaz de realizar o processamento pesado e em cada um dos dispositivos usados na aplicação, onde eles se comunicarão através de mensagens. (QUIGLEY; GERKEY; SMART, 2015)

Para cada mensagem recebida do dispositivo móvel, o nó presente no computador solicitará dados aos nós necessários para a realização da operação. Nesta aplicação, desenvolveu-se:

- um nó para cada câmera, onde cada um destes é responsável por enviar as imagens captadas por suas respectivas câmeras;
- um nó para a detecção das linhas do campo;
- um nó responsável pela compilação das imagens da câmera em um vídeo e um nó para o processar o vídeo, detectando a bola e o jogador;
- um nó capaz de receber a informação do socket recebido do aplicativo e retransmitir para os nós;
- um nó responsável por enviar os dados processados para o aplicativo Android via socket, tal como o vídeo processado. O dispositivo móvel que estiver rodando o aplicativo deverá estar conectado na mesma rede Wi-fi que o computador.

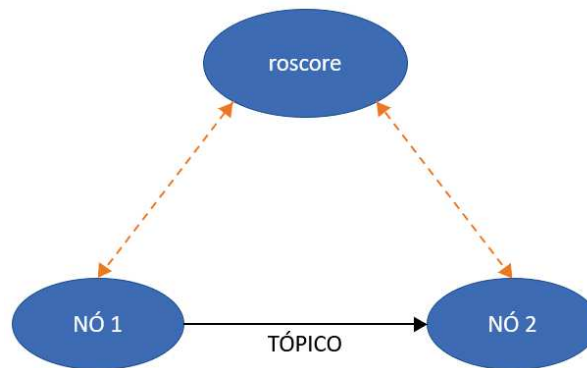
A comunicação entre os nós é feita através de tópicos, por onde os fluxos de mensagens são enviados. Mas essa comunicação não acontece de forma desordenada. Para fazer esse controle é feita a implementação de um *roscore*, que é um serviço que fornece informações de conexão aos nós para que eles sejam capazes de formar uma conexão ponto a ponto direta com outros nós e possam transmitir mensagens uns aos outros. Sem a inicialização do *roscore*, os nós não conseguem encontrar outros nós. Desta forma, todo sistema ROS necessita que o *roscore* seja executado. (QUIGLEY; GERKEY; SMART, 2015)

Para se comunicar, todos os nós devem se conectar ao *roscore* na inicialização, informando assim os detalhes dos fluxos de mensagens que ele é capaz de publicar e os fluxos nos quais ele deseja receber. Em seguida, o *roscore* fornece os endereços dos nós que publicam e os que assinam os tópicos de mensagens que o nó informou (QUIGLEY; GERKEY; SMART, 2015).

A Figura 58 mostra um sistema com dois nós e um serviço *roscore*, onde as linhas tracejadas indicam a transmissão das informações que os nós conseguem solicitar ao *roscore* sobre os outros nós do sistema e a linha preta indica o tópico estabelecido entre o nó 1 e 2 que permite a troca de mensagens.

Uma das formas de comunicação entre os nós pode ser feita através de um sistemas de transporte de mensagens conhecido como publicador/subscritor (publisher/subscriber).

Figura 58 – Exemplo ligação entre *roscore* e nós para estabelecer comunicação.



Fonte: próprio autor.

Nesse sistema, os nós enviam mensagens através da publicação das mesmas em um tópico específico. Para que um nó receba a mensagem publicada ele deve se inscrever neste mesmo tópico. Os nós podem se inscrever e mandar mensagens em vários tópicos simultaneamente, onde cada um dos tópicos recebem nomes individuais para que estes sejam identificados e comuniquem os nós corretamente (MOTA, 2020).

Esse sistema é o representado entre os nós da Figura 58, onde a comunicação é feita através do tópico. Neste caso, o nó 1 é o publicador e o nó 2 é o subscritor.

Outro sistemas de transporte de mensagens muito utilizado é o servidor/cliente (service/client). Esse modelo é utilizado para situações onde um nó disponibiliza um serviço e aguarda a informação ser requisitada por outro nó para somente então transmitir a resposta contendo o que lhe foi solicitado. Os serviço são identificados pelos nomes específicos a eles atribuídos, sendo solicitados através de uma mensagem de requisição (KOUUBÂA, 2019).

A Figura 59 mostra a conexão feita quando o sistemas de transporte de mensagens servidor/cliente é utilizado.

Figura 59 – Exemplo de comunicação feita entre servidor (nó 2) e cliente (nó 1).

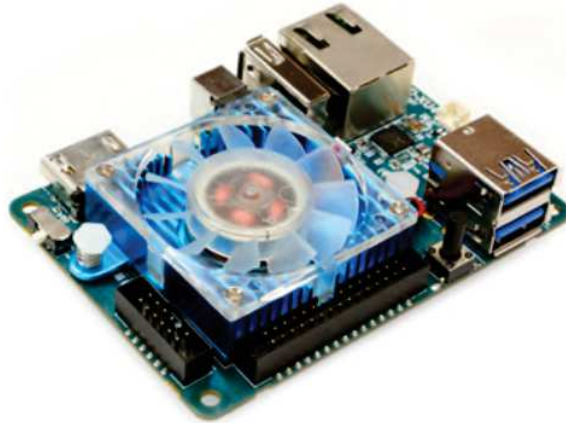


Fonte: próprio autor.

Para os testes, o computador foi conectado com as câmeras através de dispositivos Odroid XU4, mostrado na Figura 60. Este foi ligado às câmeras através de um cabo USB e ao computador através de um cabo de rede. O dispositivo Odroid é um *single board computer*, que possui oito núcleos rodando a 2 GHz e com 2 GB de RAM, e suporta

várias versões de um sistema operacional, tais como Ubuntu, Android e muitas outras versões do Linux. Esse sistema é ideal para aplicações ROS e de processamento de imagem, oferecendo boas velocidades de transferência de dados (MAHTANI et al., 2018).

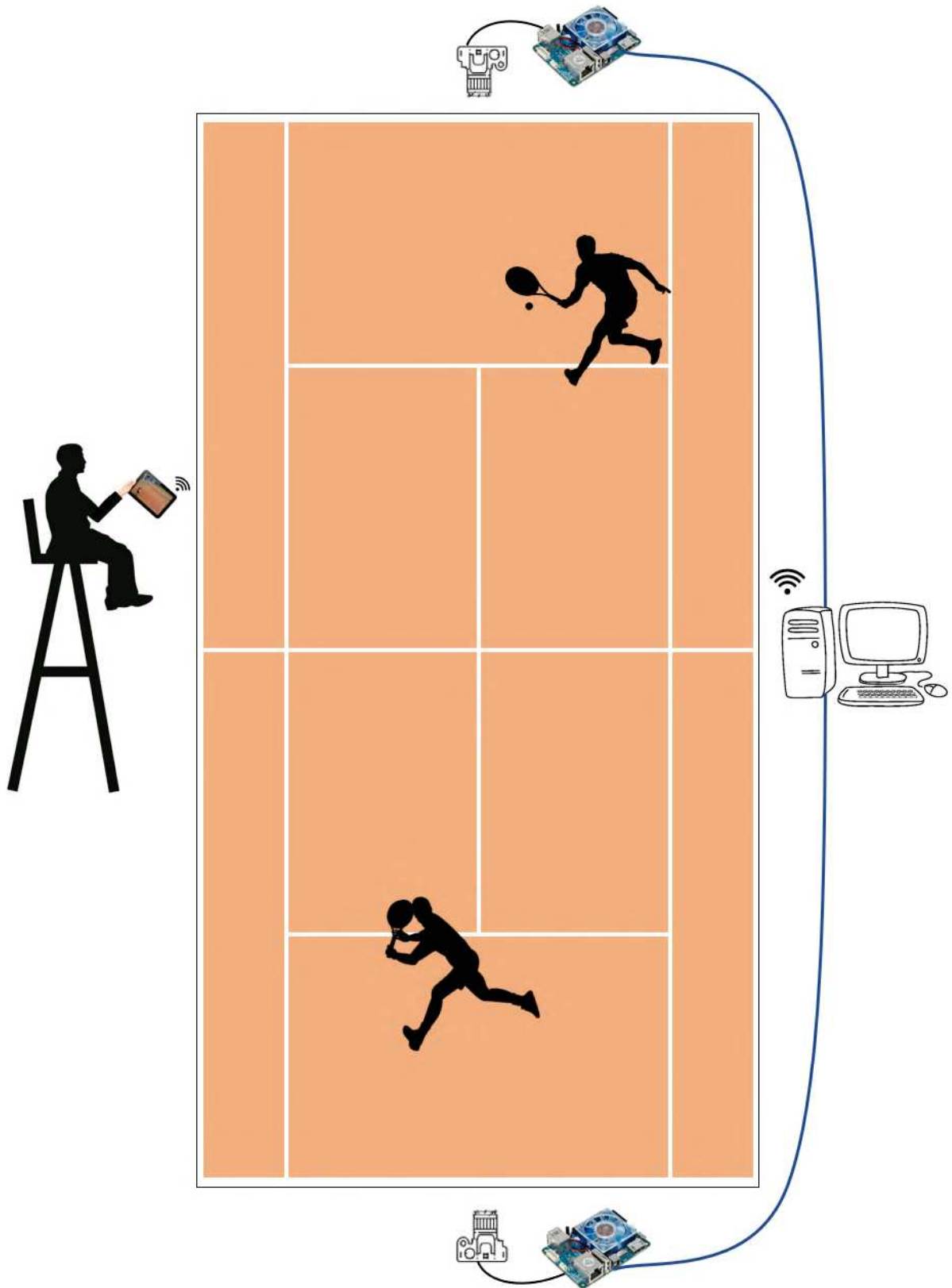
Figura 60 – Dispositivo Odroid XU4.



Fonte: (MAHTANI et al., 2018).

O esquema das conexões entre todos os dispositivos segue o esquema de conexão mostrado na Figura 61.

Figura 61 – Esquema de conexão entre os dispositivos necessário para funcionamento do aplicativo Android.



Fonte: próprio autor.

## 4 Resultados e Discussões

Neste capítulo serão abordadas as características construtivas que impactaram cada etapa de processamento do jogo de tênis, que são *(i)* detecção da posição da bola; *(ii)* o momento em que esta toca o chão da quadra; e *(iii)* detecção das linhas do campo e do jogador. Também serão feitas discussões quanto a adoção de cada método, expondo as análises que levaram a tais decisões. Ao final, será abordado o funcionamento do aplicativo que faz a integração de todas as etapas.

Para a obtenção dos resultados descritos neste capítulo foram utilizados os seguintes equipamentos:

- Desktop com processador Intel(R) Core™ i7-9750H, memória RAM de 16Gb, sistema operacional Ubuntu 18.04 LTS 64-bit e placas de vídeo NVIDIA(R) RTX™ 2060 com ROS Melodic Morenia instalado;
- Dois Odroid-XU4 com sistema operacional Ubuntu 18.04 LTS 32-bit e ROS Melodic Morenia instalados;
- Duas câmeras Intel ® RealSense™ D415 de resolução FHD 1920 × 1080, 30 fps;
- Dispositivo móvel com versão do Android mínima 9.0.

Como explicado anteriormente no Capítulo 3, para que o processamento do vídeo seja feito de maneira correta, a câmera deve se situar no fundo da quadra atrás do jogador, em uma posição compreendida entre as linhas laterais do campo. A distância da instalação da câmera em relação à quadra não segue nenhuma restrição, mas recomenda-se que o posicionamento da mesma englobe a maior área da quadra possível de acordo com as limitações físicas do ambiente em que o projeto for aplicado.

O tempo de aquisição do jogo não teve limitações, uma vez que o objetivo do aplicativo é gravar as jogadas e não o jogo inteiro de uma vez só. Desta forma, cada jogada deve levar por volta de 10 a 15 segundos apenas, não gerando nenhum problema no que se refere ao processamento do jogo ou ao funcionamento do aplicativo.

Como ferramentas de programação do aplicativo foi utilizada a plataforma Android Studio 4.1.1, onde foram desenvolvidos os códigos referentes ao aplicativo, utilizando as linguagens Java e XML. A parte de processamento de imagem e implementação de uma aplicação ROS foram implementadas utilizando a linguagem Python em sua versão 3.7.0.

As imagens processadas mostradas neste capítulo foram adquiridas através da implementação prática do projeto proposto, onde foi feita a aquisição das imagens de um jogo ao vivo por meio das câmeras instaladas e através do processamento de vídeos de jogos de tênis profissional disponível nos *links* abaixo:

- <<https://www.youtube.com/watch?v=EnPIQdtNbx8>>
- <<https://www.youtube.com/watch?v=Yil9PpIr1DA>>

#### 4.1 Detecção da Bola

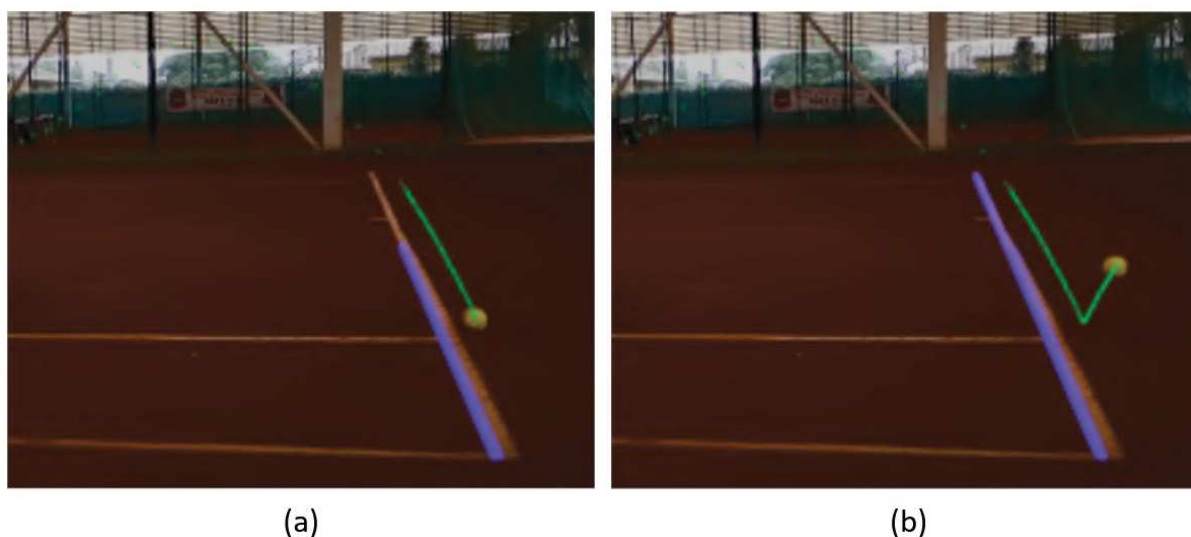
As subseções apresentadas a seguir exploram os resultados referentes à detecção da posição da bola, mostrando qual método foi definido como ideal para a aplicação prática do projeto em questão.

##### 4.1.1 Detecção Através da Utilização da Máquina de Vetor de Suporte

O setup da câmera foi montado e as imagens foram tiradas para serem processadas posteriormente. Os vídeos podem ser baixados em **GitHub**. A partir dos dados obtidos duas situações principais foram avaliadas.

A primeira situação é uma jogada regular onde a bola de tênis toca após a linha de base quando vem da outra área de serviço. As Figuras 62 e 63 mostram a trajetória detectada por ambos os algoritmos nesta situação.

Figura 62 – Trajetória da bola detectada através de operações do OpenCV. (a) Antes do toque no solo. (b) Depois do toque no solo.

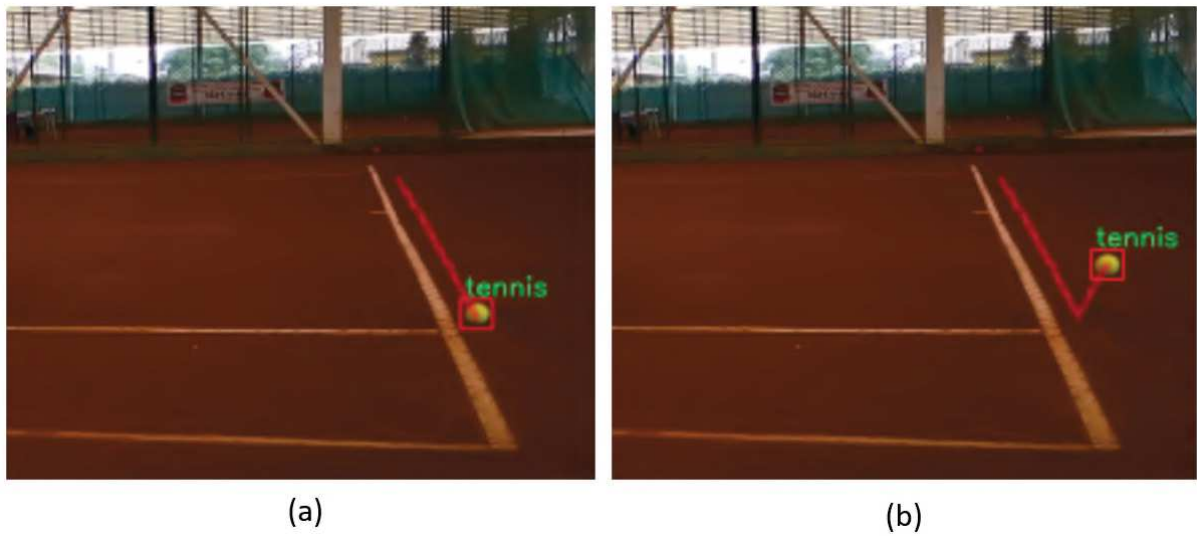


Fonte: próprio autor.

A linha destacada em azul pela transformação de Hough é a linha de base usada para levar em conta as pontuações. A imagem processada também mostra o caminho reconhecido para a bola de tênis para ambos os métodos de visão computacional.

A segunda situação investigada mostra a bola de tênis tocando logo após a linha de base, como mostrado na Figura 64. Essa situação é crítica pois a pontuação do jogo não é

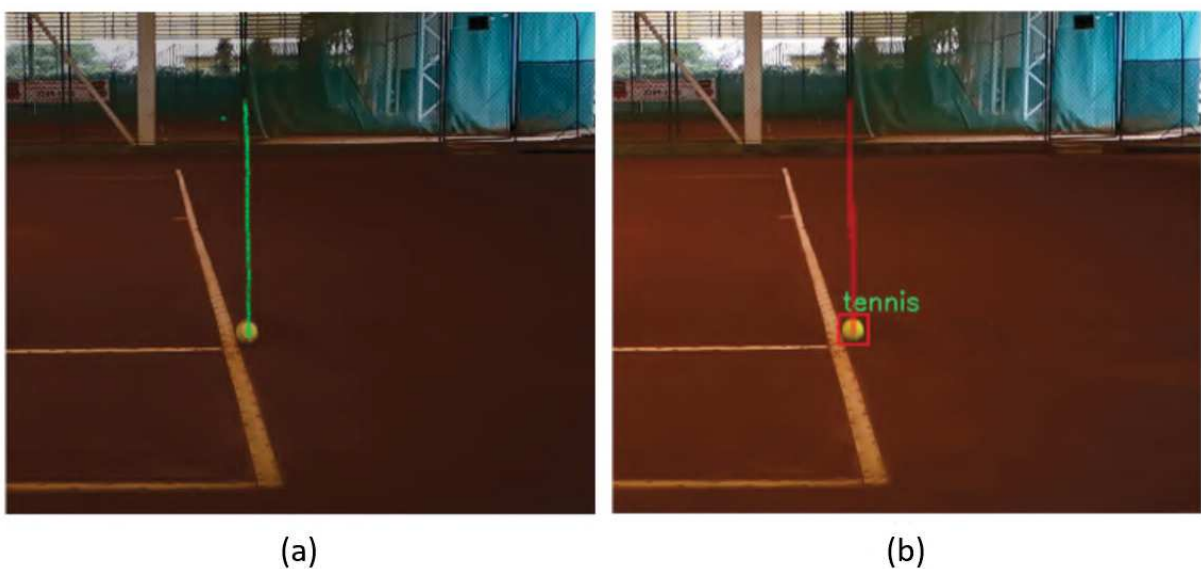
Figura 63 – Trajetória da bola detectada através do SVM. (a) Antes do toque no solo. (b) Depois do toque no solo.



Fonte: próprio autor.

clara para o juiz, tornando-se um caso de difícil avaliação sem a utilização de tecnologias como a proposta no presente trabalho. A bola foi lançada em uma determinada altura para que sua velocidade atingisse valores maiores. A Figura 64 (a) mostra a detecção do círculo OpenCV enquanto a Figura 64 (b) apresenta a operação SVM.

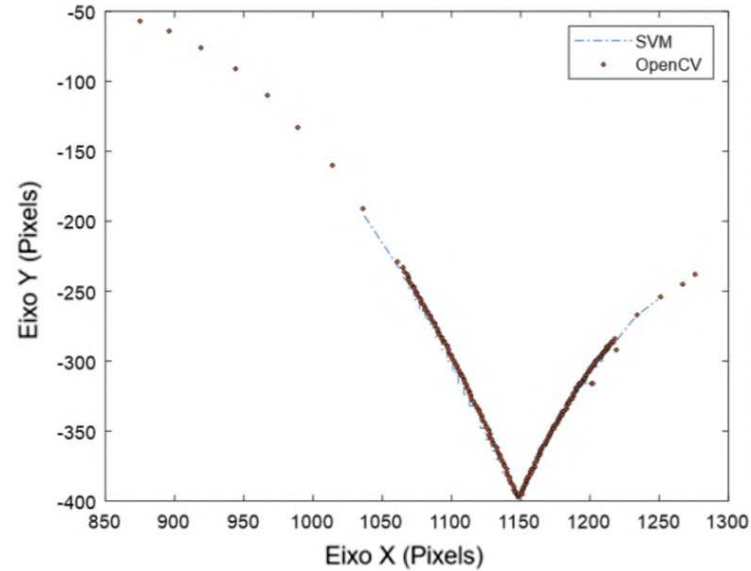
Figura 64 – Trajetória da bola detectada com quique próximo à linha de fundo da quadra. (a) OpenCV. (b) SVM.



Fonte: próprio autor.

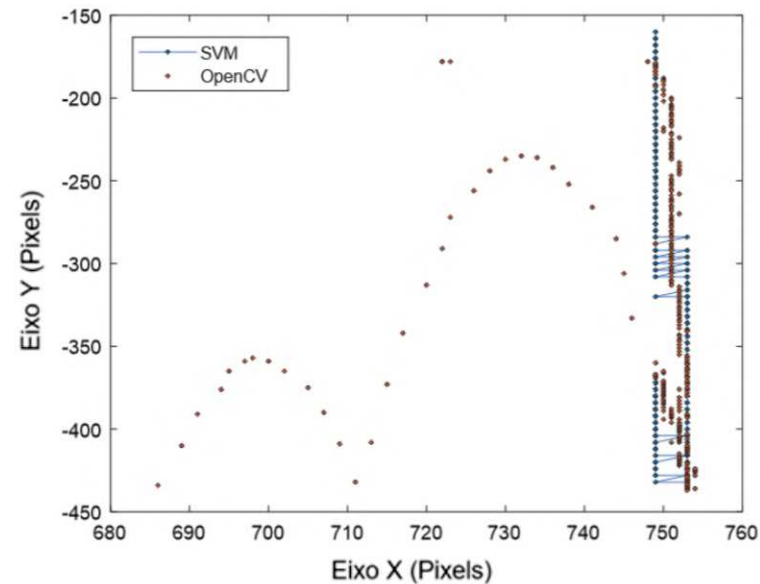
A Figura 65 e a Figura 66 detalham o caminho reconhecido por esses algoritmos no primeiro e no segundo cenário, respectivamente.

Figura 65 – Movimento da bola para o primeiro cenário.



Fonte: próprio autor.

Figura 66 – Movimento da bola para o segundo cenário.



Fonte: próprio autor.

A Figura 67 é um exemplo de erro de reconhecimento executado pelo algoritmo de detecção de círculo. Este erro também pode ser visto na Figura 66 como dois pontos isolados no topo do gráfico longe da trajetória real da bola.

Ambos os métodos de detecção apresentam resultados excelentes com relação à detecção da bola, porém isso se dá em situações de teste. Para a obtenção do processamento



Figura 67 – Reconhecimento errado da posição da bola.



Fonte: próprio autor.

mostrado, a bola está sendo gravada em câmera lenta e, além disso, nos vídeos capturados a bola atinge baixas velocidades em relação às atingidas em jogos reais pois para a gravação de teste as bolas foram jogadas manualmente. A obtenção das imagens foi feita desta forma pois os métodos utilizados necessitam que as características da bola se apresentem de forma nítida para a correta extração das características da bola, levando ao correto reconhecimento da mesma.

Como abordado anteriormente no Capítulo 1, para vídeos capturados simultaneamente à realização do jogo de tênis a velocidade da bola pode chegar a mais de 200 Km/h. Com isso, a imagem da bola capturada se apresenta distorcida na maior parte do conjunto de imagens do vídeo gravado. A Figura 68 mostra como a bola se apresenta distorcida na imagem durante o jogo quando atinge altas velocidades. Assim, a utilização dos métodos anteriores para a detecção da bola no jogo não consegue obter resultados para grande parte do grupo de dados, uma vez que o treinamento e detecção utilizada no SVM com o grupo de dados que foi treinado se concentra nas características morfológicas da bola bem definidas em uma imagem. Momentos em que a bola se apresenta desta forma na Figura 68 são de difícil detecção para o sistema treinado. Outro fator que dificulta a detecção de bolas em imagens semelhantes à Figura 68 é que o formato da bola distorcido se assemelha também aos riscos no chão da quadra de saibro, podendo gerar muitos falsos positivos para a detecção em lugares do campo onde não existe a bola.

A utilização do SVM no processamento de partidas ao vivo de tênis se faz vantajosa em relação à CNN quando se considera seu tempo de processamento. Para o processamento de 5 segundo de captura do vídeo, o algoritmo leva 70.53 segundos, equivalendo a 0.47 segundos por imagem, que é menor do que o tempo da rede neural convolucional. Porém, para os equipamentos utilizados neste trabalho o método do SVM não foi o ideal, uma vez

Figura 68 – Exemplo de distorção apresentada pela bola em imagens capturadas durante jogo de tênis.



Fonte: próprio autor.

que a taxa com que a câmera utilizada captura as imagens é de 30 fps (Frames Per Second ou Quadros por Segundo). Essa taxa é quem determina a qualidade das imagens da bola quando está estiver em altas velocidades, onde quanto maior o fps da câmera melhor será a integridade do formato da bola na imagem, o que irá gerar boas classificações pelo método do SVM. O problema da adoção deste tipo de câmera é que o preço é proporcional à qualidade do número de quadros por segundo. Desta forma, quanto maior a qualidade da câmera, mais cara ela será. No presente trabalho o objetivo é produzir um sistema capaz de fazer todos os processamentos propostos mas optando sempre por uma abordagem que produza resultados de qualidade porém com baixo custo, então a outra opção para detecção da posição da bola através da utilização de CNN será implementada e estudada.

Para implementações práticas, o usuário deverá escolher entre uma resposta mais rápida do sistema através do investimento em equipamentos mais caros utilizando o SVM para detecção ou a adoção de um sistema de baixo custo através da utilização da CNN, porém com um tempo maior de processamento que pode levar à momentos de interrupção no jogo de pouco mais de 2 minutos.

#### 4.1.2 Detecção Através da Utilização de Redes Neurais Convolucionais

Como alternativa para a implementação em aplicações práticas, foi proposto a utilização da rede neural convolucional TrackNet, que apresentou ótimos resultados de detecção em (HUANG et al., 2019). Para avaliar os resultados de detecção da bola obtidos com o modelo utilizado no presente trabalho, utilizou-se uma ferramenta de representação de desempenho de um classificador, conhecida como matriz de confusão. Através dela é possível fazer comparações entre os resultados das detecções e os valores reais esperados.

Note que as métricas de avaliação são fundamentais para o processo de projeto de uma rede neural artificial. Existem diversos tipos de métricas e cada métrica tem um papel específico. Dentre as diversas métricas existentes (e.g., matriz de confusão, ROC, *sensitivity*, *specificity*, *recall*, *f-score*, acurácia, etc), a matriz de confusão é uma das mais difundidas.

A representação da matriz de confusão é formada como mostrado na Tabela 1. As colunas são referentes às condições reais em que os dados se apresentam. Estes podem ser positivas ou negativas, ou seja, se há a presença da bola na imagem ou não. Já as linhas são referentes à classificação realizada pelo classificador, sendo positivas ou negativas, significando que o modelo detectou a posição da bola ou não, segundo os parâmetros determinados. Para mais informações, (FUENTE, 2018) apresenta as categorias de forma geral que compõem a matriz. Para o presente projeto, a matriz de confusão apresenta quatro categorias possíveis de classificação:

- **Verdadeiros positivos (VP):** a rede neural detectou a posição da bola na imagem em ocorrências onde esta foi detectada corretamente, dentro das métricas estabelecidas
- **Verdadeiros Negativos (VN):** a rede neural não detectou nenhuma posição para bola na imagem em ocorrências onde a posição da bola não se apresentava dentro das métricas estabelecidas
- **Falsos positivos (FP):** a rede neural detectou a posição da bola na imagem em ocorrências onde a posição da bola não se apresentava dentro das métricas estabelecidas
- **Falsos negativos (FN):** a rede neural não detectou a posição da bola na imagem em ocorrências onde a posição da bola se apresentava dentro das métricas estabelecidas

Tabela 1 – Matriz de confusão

	Condição Positiva	Condição Negativa
Predição Positiva	VP	FP
Predição Negativa	FN	VN

Fonte: Adaptado de (FUENTE, 2018)

Como apresentado em (MOLIN, 2019) e (MARGARITIS; KYRIAKIDES, 2019), é possível obter várias métricas que dão sentido aos valores obtidos na construção da matriz de confusão:

- *Precisão*: Representa a proporção de detecções positivas corretas feitas pelo classificador, indicando o grau de variação dos resultados. Através dessa métrica é possível ter informação sobre a quantidade de falsos positivos, onde quanto mais perto de 100% for o valor da precisão, maior é a quantidade de verdadeiros positivos na detecção, indicando o quão precisa é a detecção realizada. Essa métrica é representada pela Equação 4.1.

$$Precisão = \frac{VP}{VP + FP} \quad (4.1)$$

- *Recall*: Indica a proporção de casos positivos que foram preditos corretamente pelo classificador. Através dessa métrica é possível ter informação sobre a quantidade de falsos negativos, verificando quanto o modelo está identificando os casos positivos corretamente. Quanto mais perto de 100% for o valor do *recall*, menor é a quantidade os falsos negativos na detecção. Essa métrica é representada pela Equação 4.2.

$$Recall = \frac{VP}{VP + FN} \quad (4.2)$$

- *F1-Score*: Possibilita a visualização dos parâmetros de *precisão* e *recall*. Uma forma de unir esses valores seria pela média aritmética, mas isso causa um problema em casos onde uma das métricas apresenta valor muito baixos em relação ao valor da outra, gerando um resultado não muito informativo. Desta forma, seu valor é dado pela média harmônica entre *precisão* e *recall*, se aproximando mais da média “comum”. Essa métrica indica o quão preciso e robusto o classificador é, sendo representada pela Equação 4.3.

$$F1 = 2 * \frac{1}{\frac{1}{precisão} + \frac{1}{recall}} = 2 * \frac{precisão * recall}{precisão + recall} \quad (4.3)$$

- *Acurácia*: Indica a proporção de detecções feitas corretamente pelo classificador. É representado pela Equação 4.4.

$$Acurácia = \frac{VP + VN}{VP + FP + VN + FN} \quad (4.4)$$

- *Taxa de erro*: Indica a proporção de detecções incorretas feitas pelo classificador. É representado pela Equação 4.5.

$$taxa\ de\ erro = 1 - acurácia = \frac{FP + FN}{VP + FP + VN + FN} \quad (4.5)$$

Para cada cenário analisado, serão mostrados a seguir a respectiva matriz de confusão e as métricas correspondentes, justificando assim a metodologia adotada. Para a avaliação dos cenários, foram utilizados um conjunto de teste com 5175 imagens com a

posição da bola de tênis previamente armazenadas. Conforme análise feita em (HUANG et al., 2019), a rede proposta foi treinada por 500 épocas com 200 passos por época.

Como mencionado anteriormente, os autores de (HUANG et al., 2019) consideraram como corretas apenas as saídas da rede que possuíam um círculo em sua imagem de saída. Esta abordagem caracteriza o reconhecimento da rede sem levar em conta os reconhecimentos gerados a partir de regiões que não pertencem à posição real da bola. A matriz de confusão e as métricas calculadas desta proposta estão mostrados nas Tabelas 2 e 3.

Tabela 2 – Matriz de confusão da rede considerando como corretas as detecções que apresentam apenas uma bola detectada dentro dos parâmetros especificados por imagem.

	Condição Positiva	Condição Negativa
Predição Positiva	4151	180
Predição Negativa	640	204

Fonte: próprio autor.

Tabela 3 – Métricas da rede considerando como corretas as detecções que apresentam apenas uma bola detectada dentro dos parâmetros especificados por imagem.

Métrica	Valor (%)
Precisão	95.8439
Recall	86.6416
F1-Score	91.0107
Acurácia	84.1546
Taxa de erro	15.8454

Fonte: próprio autor.

O presente trabalho propõe uma alteração nos parâmetros de verificação da validade de uma detecção feita pela rede, incluindo das imagens que apresentarem mais de um círculo no mapa de calor da saída da rede. Todavia, essa inclusão não é feita de forma indiscriminada. Para que uma detecção seja considerada correta, esta deverá atender a exigência de distanciamento entre as localizações detectadas. Essa exigência determina que, para que a detecção de um círculo seja considerada, a localização do centro da bola deverá se apresentar a uma distância menor do que 40 pixels da última bola detectada na imagem anterior. Desta forma, mesmo que exista mais de um círculo presente na imagem, todos serão avaliados. Assim, caso algum dos círculos estejam dentro da distância estabelecida, a imagem será incluída na contagem de detecções corretas da rede. A análise fará então um processo de avaliação em todas as imagens de saída da rede, considerando em cada imagem apenas a localização correta da bola e descartando as incorretas.

Nesse processo de detecção, são levadas em conta as detecções da bola em imagens onde há a detecção correta da bola, porém estas não eram anteriormente consideradas devido às detecções erradas em alguma parte da imagem geradas por características do ambiente que são semelhantes às características da bola extraídas pela rede. Esse aspecto reflete diretamente nas métricas, mostrando resultados melhores para a rede em análise. A matriz de confusão e as métricas correspondentes à esta proposta estão mostrados a seguir nas Tabelas 4 e 5. A redução das detecções classificadas como falso positivo no grupo de teste analisado corresponde à alteração feita no conceito do que é considerado como predição positiva resultante da rede. Essa alteração não prejudica os resultados finais, mas aumenta a quantidade de bolas detectadas corretamente que serão usadas em processamentos.

Tabela 4 – Matriz de confusão da rede considerando como corretas todas as detecções que apresentam pelo menos uma bola dentro dos parâmetros especificados detectada por imagem.

	Condição Positiva	Condição Negativa
Predição Positiva	4302	29
Predição Negativa	640	204

Fonte: próprio autor.

Tabela 5 – Métricas da rede considerando como corretas todas as detecções que apresentam pelo menos uma bola dentro dos parâmetros especificados detectada por imagem.

Métrica	Valor (%)
Precisão	99.3304
Recall	87.0498
F1-Score	92.7855
Acurácia	87.0725
Taxa de erro	12.9275

Fonte: próprio autor.

É possível observar que todas as métricas das redes tiveram seus valores melhorados quando foram adotadas as novas perspectivas para a aceitação de uma imagem como portadora de uma detecção correta da posição da bola de tênis. O classificador teve sua precisão considerada 3.4865% maior, teve 0.4082% mais casos positivos considerados como preditos corretamente, ficou 1.7748% mais robusto, 2.9179% mais acurado e teve sua taxa de erro reduzida em 2.9179%.

A partir da comparação dos resultados apresentados na Tabelas 6 obtidos nos dois cenários apresentados é possível perceber que todos os valores das métricas foram

melhorados, justificando assim a adoção da rede TrackNet com a utilização das modificações propostas.

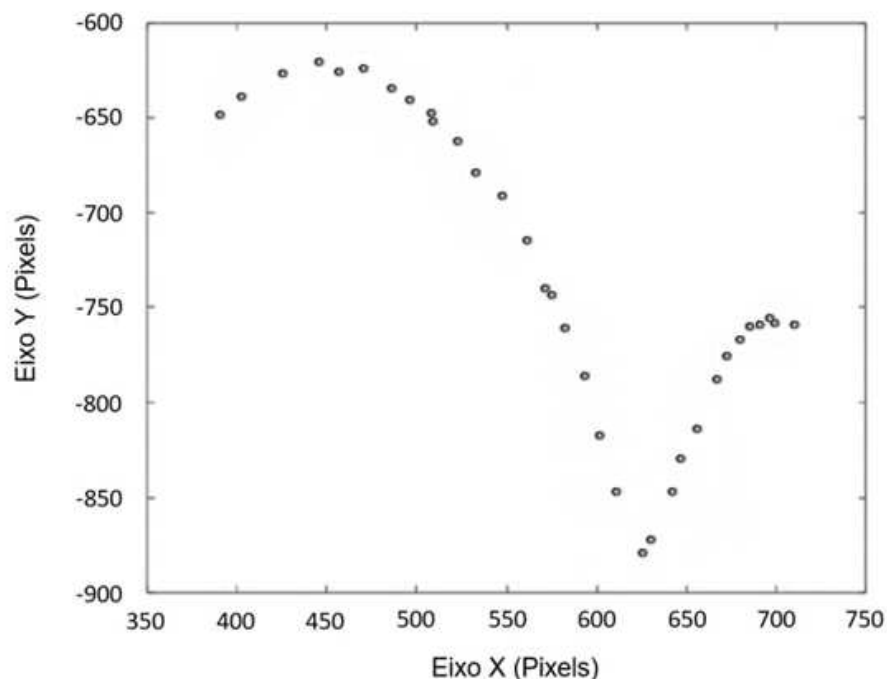
Tabela 6 – Comparação dos valores das métricas da rede TrackNet original e modificada.

MÉTRICA	Valor para TrackNet original (%)	Valor para TrackNet modificada (%)
Precisão	95.8439	99.3304
Recall	86.6416	87.0498
F1-Score	91.0107	92.7855
Acurácia	84.1546	87.0725
Taxa de erro	15.8454	12.9275

Fonte: próprio autor.

Desta forma, foi escolhido o segundo como padrão de detecção durante a execução dos demais passos do processamento proposto neste trabalho devido aos ótimos resultados apresentados. A Figura 69 mostra o gráfico das posições da bola detectada para um momento do jogo.

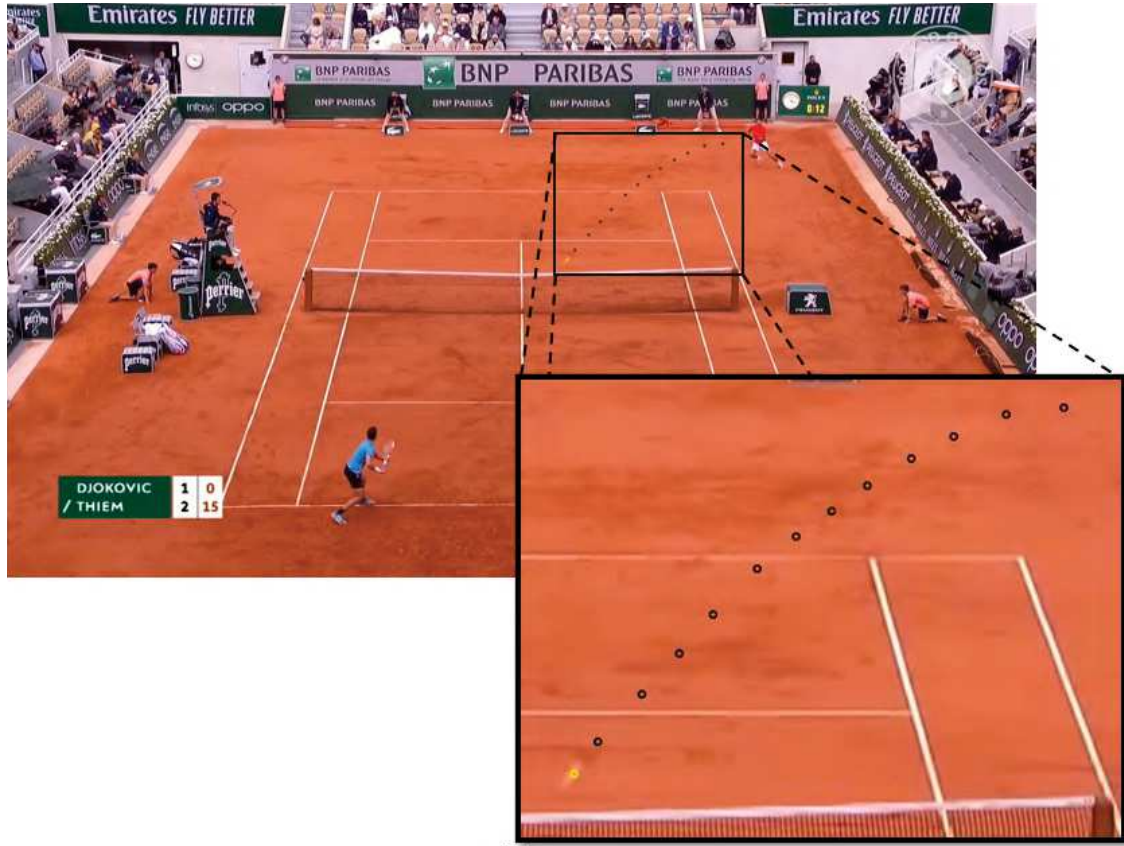
Figura 69 – Gráfico com exemplo de detecções da posição da bola.



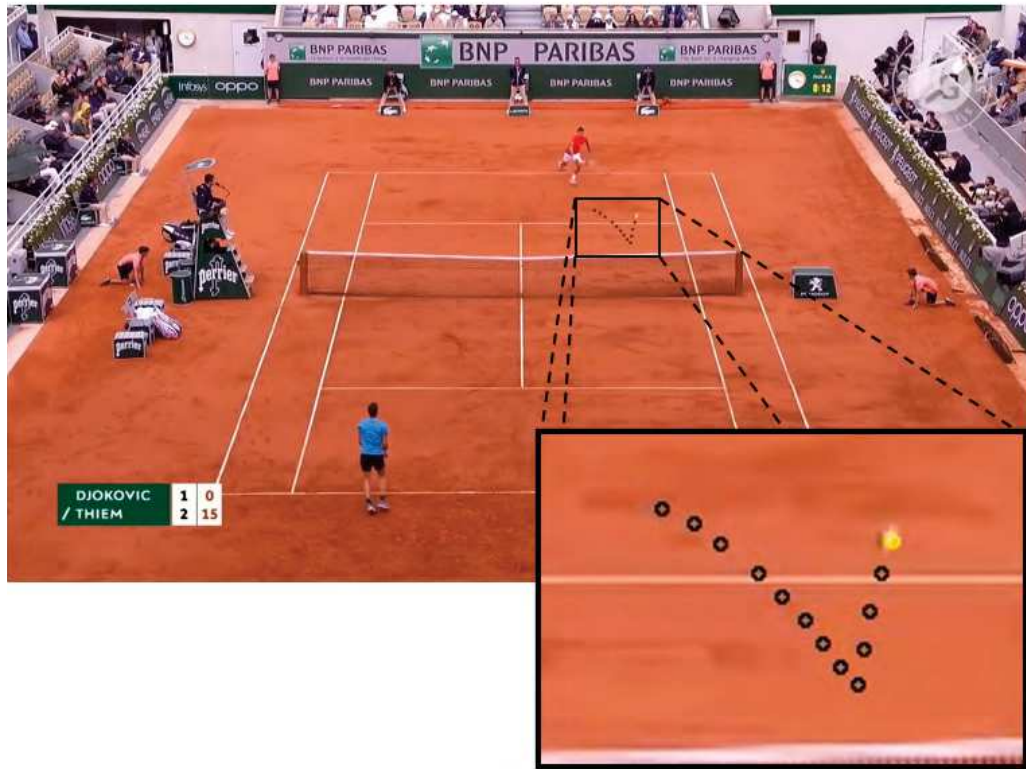
Fonte: próprio autor.

As Figuras 70 (a) e (b) mostram exemplos dos resultados obtidos para a detecção das posições da bola a partir do processamento da rede com as configurações explicadas acima. As posições detectadas são armazenadas em um vetor que se atualiza a cada nova detecção, guardando desta forma as últimas posições da bola. As posições armazenadas

Figura 70 – Exemplos de detecção da posição da bola.



(a)



(b)

Fonte: próprio autor.



no vetor são então exibidas na imagem final, de forma a apresentar o trajeto percorrido pela bola. Para que a imagem final não fique poluída com o trajeto inteiro da bola na jogada, são apresentadas as 13 últimas posições detectadas. Figura 70 (a) apresenta um espaçamento entre as detecções das posições da bola maior do que as detecções da Figura 70 (b) causado pela velocidade da bola. Quanto menor a velocidade da bola, menor será o seu espaçamento entre as detecções.

O processamento do intervalo de 5 segundos da jogada leva 142.84 segundos para ser visualizado pelo jogador. Este valor elevado do tempo de resposta do software desenvolvido é causado pelo processamento da rede neural convolucional que não tem um tempo de detecção tão otimizado para esta aplicação em tempo real, uma vez que seu desenvolvimento foi feito originalmente com a proposta de analisar vídeos previamente gravados.

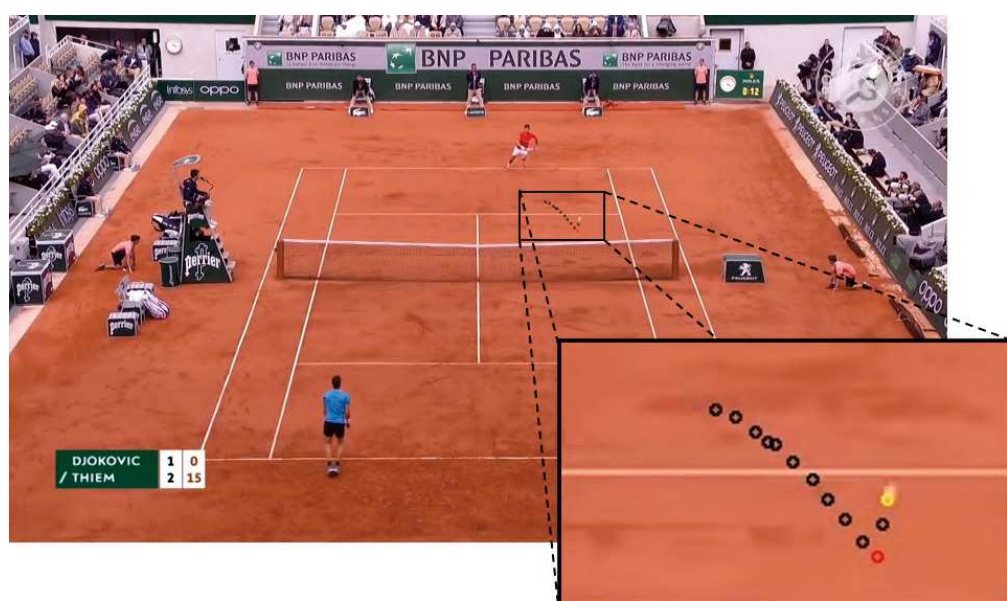
Em aplicações como a implementada neste trabalho, onde a análise é feita durante a realização da partida, a rede apresentará o tempo de processamento como ponto negativo, porém pela qualidade dos resultados obtidos a utilização deste tipo de rede continua se apresentando vantajosa. Este tempo ainda é obtido tendo em vista que a detecção da quadra é realizada apenas uma vez antes do início do jogo, de forma a não ser necessário que esta etapa seja refeita diversas vezes durante o jogo. Esta implementação é possível devido às câmeras se apresentarem estáticas no campo, não se alterando suas perspectiva das posições das linhas da quadra. Desta forma, considera-se que a detecção feita será verdadeira durante todos os momentos do jogo e os pontos da quadra que foram identificados serão utilizados até o fim do processamento. O tempo de processamento computado inclui então apenas os dados que necessitam ser processados nos frames durante o jogo, sendo eles a detecção do jogador, da bola e do momento de quique da bola, juntamente com o tempo que leva para que os dados já processado sejam enviados do computador para o aplicativo. Os métodos utilizados para a detecção do quique e do jogador foram implementados de forma a apresentar um baixo tempo de processamento de forma a entregar resultados rápidos que não aumentassem ainda mais o tempo de processamento gasto pela rede neural convolucional.

## 4.2 Detecção do Quique

Como discutido no capítulo anterior, a detecção do quique da bola de tênis foi realizada através da comparação da altura da bola nas posições detectadas pela rede.

De forma a obter detecções corretas e confiáveis, foi adotada a abordagem de comparação das quatro diferenças de altura provenientes das cinco últimas posições da bola. Nesta lógica, a detecção de duas diferenças de altura positivas seguidas de duas diferenças de altura negativas são interpretadas como um quique. A Figura 71 mostra um quique feito corretamente pela rede.

Figura 71 – Exemplo de detecção correta do quique.

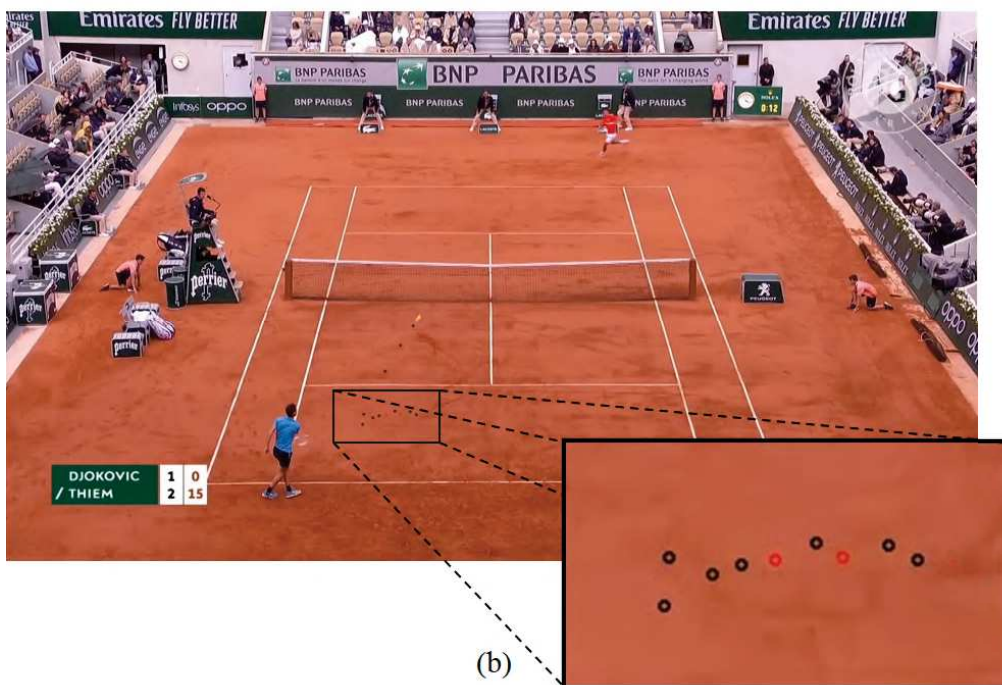
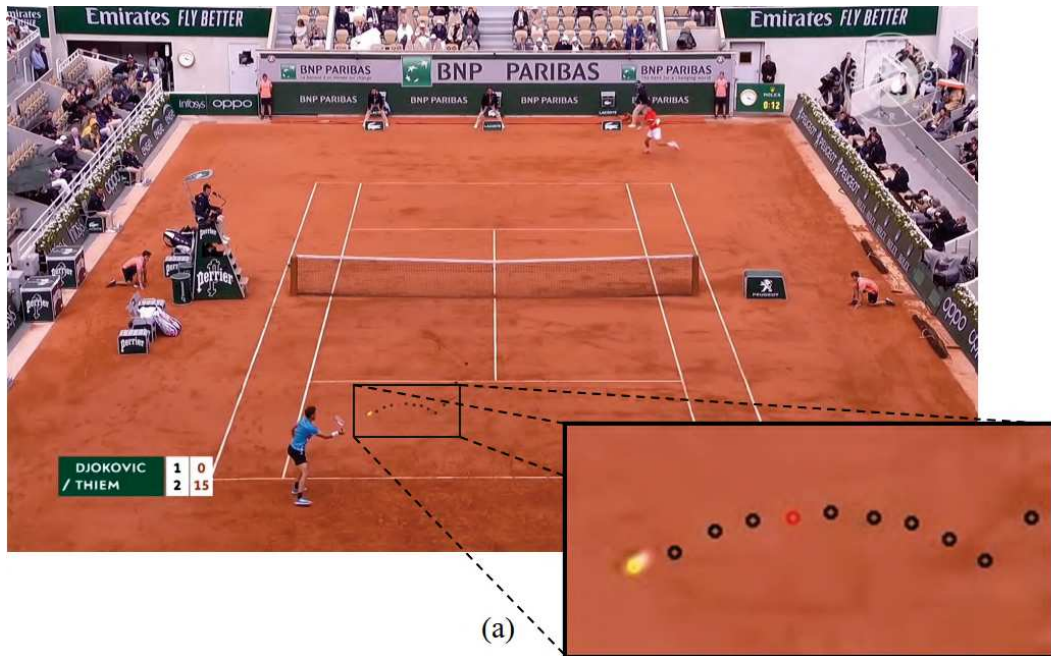


Fonte: próprio autor.

É possível perceber que o desalinhamento de algumas das detecções podem causar problemas no que se refere à detecção quando o quique foi obtido a partir da comparação feita entre as diferenças de altura provenientes de três posições da bola. As Figuras 72 (a) e (b) trazem exemplos dessas ocorrências onde a posição da bola estimada pela rede ficou deslocada em relação à sua real posição, o que gerou uma detecção do quique em situações onde este não havia ocorrido.

Para avaliar o método de detecção proposto, foi analisado o resultado do processamento do vídeo disponível em <<https://youtu.be/bnbM4KZXdlA>> que contém uma compilação de trechos de diversos jogos de tênis realizados na quadra de saibro. O vídeo tem 4 minutos e 42 segundos de duração. A classificação dos quiques foi feita através da observação do vídeo final após o processamento do software de forma visual. Esta classificação não leva em conta onde a localização do quique foi feita, mas sim se houve quique válido ou não.

Figura 72 – Exemplos de erro na detecção do quique.



Fonte: próprio autor.

A Tabela 7 apresenta a quantidade de quiques para cada categoria de classificação listada. O quique da bola pode ser classificado como:

- "Corretos" são aqueles que foram detectados corretamente seguindo a metodologia proposta;
- "Interação com jogador" são aqueles que foram contabilizados como quique mas que na verdade são resultantes do saque ou de uma interação da bola com o jogador;
- "Errados" são aqueles que foram detectados em locais onde não houve quique;
- "Não detectado" são detecções que deveriam ser contabilizadas como ocorrência de quique da bola, porém não foram classificadas segundo a metodologia utilizada.

Tabela 7 – Classificação do quique das bolas no vídeo analisado

Classificação do quique da bola	Número de ocorrências
Corretos	91
Interação com jogador	89
Errados	2
Não detectado	103
<b>TOTAL</b>	<b>285</b>

Fonte: próprio autor.

A partir da análise dos valores presentes na Tabela 7 é possível concluir alguns pontos importantes a respeito da metodologia utilizada.

Os poucos quiques identificados como "Errados" são resultantes da oclusão da bola em momentos consecutivos, não permitindo a correta visualização por parte da câmera. Esta situação é resolvida pelo esquema de ligação das câmeras visto no Capítulo 3, que permite a visualização de um mesmo momento do jogo por dois pontos de vista, fazendo com que seja possível determinar a posição do quique mesmo quando a bola estiver oclusa neste momento em uma das câmeras.

Os valores de quiques classificados como "Interação com jogador" refletem a incapacidade do modelo de diferenciar o quique real da bola e o momento em que o jogador bate na bola com a raquete. Apesar da câmera estar em uma posição que consegue captar a visão geral do jogo, a detecção feita pela comparação das diferenças de altura da bola para este posicionamento das câmeras contabiliza o momento em que o jogador mais próximo a câmera rebate a bola como um quique devido à comparação usar os valores do eixo y, como mostra a Figura 73.

Uma opção para diferenciar o quique real da interação com jogador é a utilização de algumas tecnologias já existentes no mercado. Produtos como Babolat Play e Sony

Smart Tennis Sensor são exemplos de tecnologias capazes de reconhecer os golpes dados durante uma partida e extrair outros dados, como por exemplo a velocidade da batida da bola. A integração das informações captadas por esses equipamentos com o software desenvolvido no presente trabalho permitiriam diferenciar o quique real do momento de interação com o jogador, pois os equipamentos conseguem captar quando a bola bate na raquete. Com esta configuração, o método proposto seria mais assertivo em suas detecções. A detecção da raquete de tênis por meio de visão computacional não foi uma alternativa considerada neste trabalho, pois sua detecção pode trazer problemas para a análise da jogada, como por exemplo momentos em que a raquete esteja muito perto do local onde o quique da bola ocorreu e o quique acabe sendo desconsiderado.

Figura 73 – Exemplo de detecção errada do quique em momento em que jogador rebate a bola.

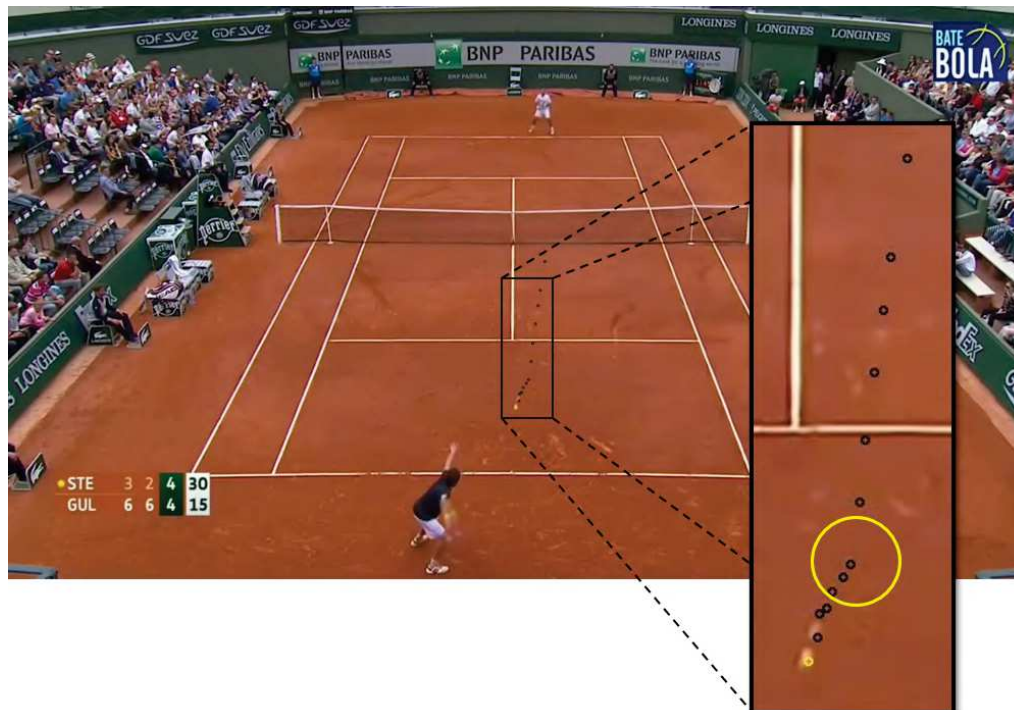


Fonte: próprio autor.

Os quiques classificados como "Não detectado" mostram outra dificuldade relacionada à comparação das alturas causada pelo posicionamento da câmera. Durante a gravação do jogo ocorrem momentos onde apesar de estar ocorrendo um quique a trajetória descrita pela bola não se apresenta da forma mostrada na Figura 43, sendo muito difícil a detecção do quique nesses momentos. Como o apresentado na Figura 74, o quique da bola ocorre na posição destacada pelo círculo amarelo, mas nenhum quique foi detectado pelo algoritmo. A trajetória da bola descrita nessa imagem não permite a comparação de alturas como feito por este método. Um exemplo de oportunidade para solucionar os problemas acima abordados seria através da adoção de um sistema mais robusto de processamento, capaz de determinar a posição 3D da bola. Assim será possível determinar com precisão todos os momentos de quique da bola sem que sua trajetória tenha a exigência de seguir uma determinada forma para ser detectada. Esta nova configuração envolve processos mais

complexos para ser implementado, como a sincronização de todas as câmeras envolvidas. Esta metodologia será abordada no desenvolvimento de futuros trabalhos.

Figura 74 – Exemplo de quique da bola de difícil detecção para posicionamento da câmera.



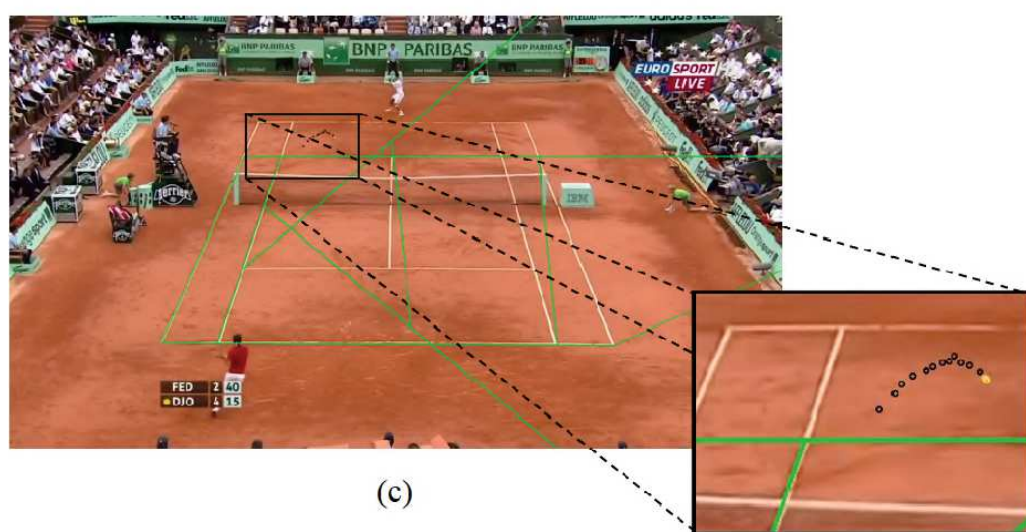
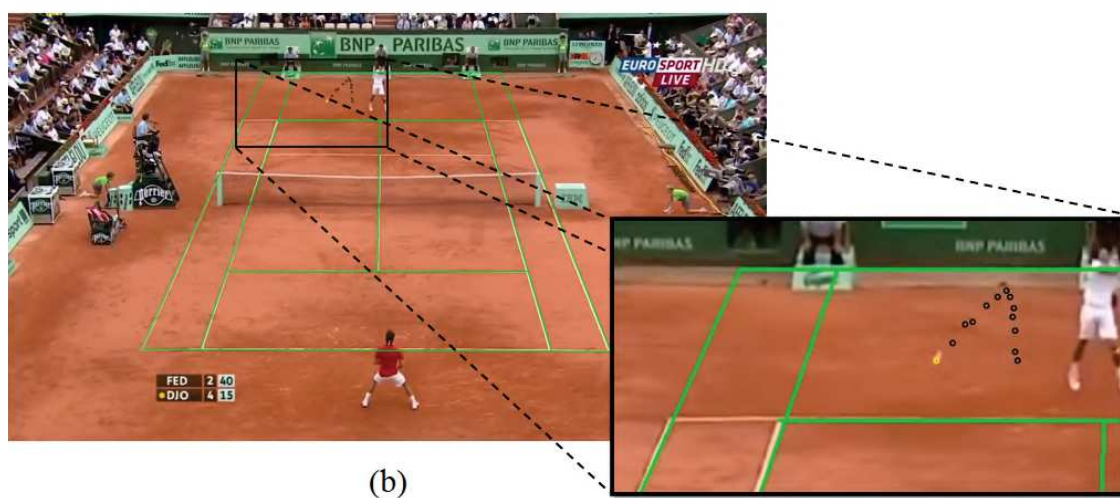
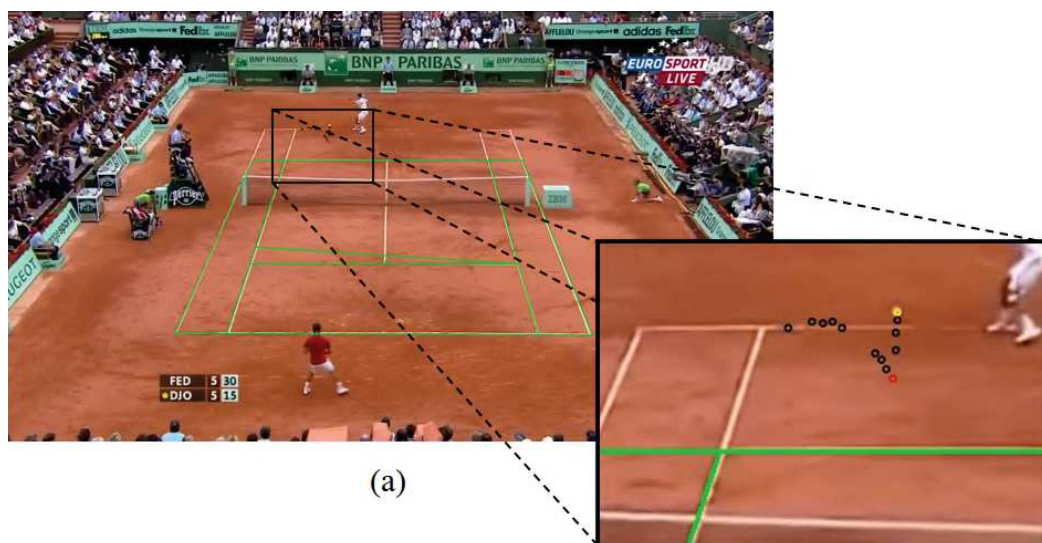
Fonte: próprio autor.

### 4.3 Detecção da Quadra

As Figuras 75 (a), (b) e (c) mostram alguns dos problemas na detecção da quadra quando esta foi feita a partir da tentativa de identificar todos os pontos que formam as arestas da quadra. Esses problemas surgem devido ao fato de que para a lógica de detecção utilizada, a detecção dos vértices externos da quadra são fundamentais para determinação dos demais vértices.

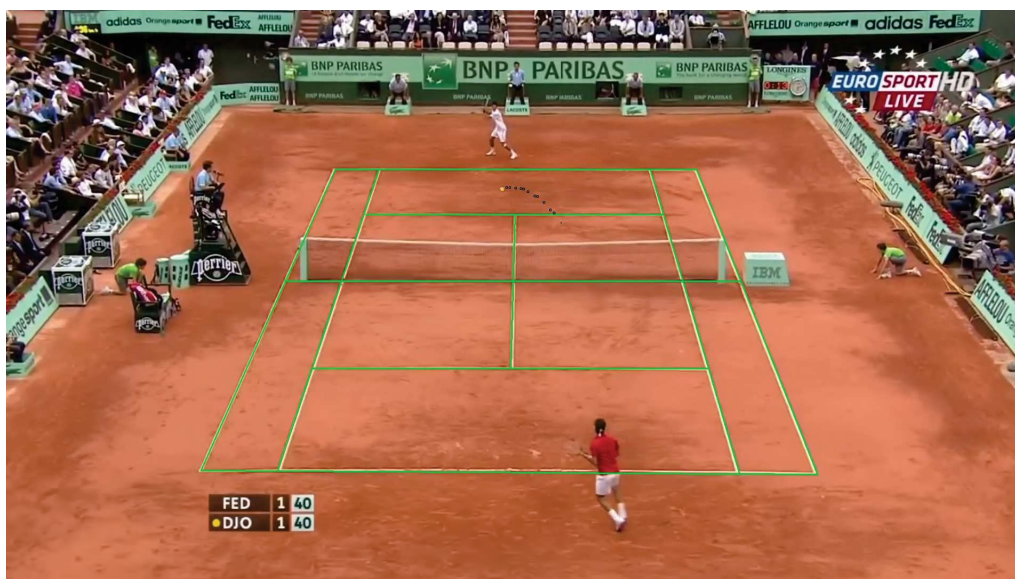
Seguindo a metodologia proposta no Capítulo 3 para a obtenção correta dos vértices da quadra, esses problemas se tornam extintos. Nesse método não é necessário detectar os pontos distantes da câmera, mas sim os pontos referentes à metade da quadra mais próxima à câmera. Como esses pontos são de fácil reconhecimento, os resultados gerados são confiáveis. A partir desses pontos, os vértices do outro campo da quadra serão obtidos projetando os pontos identificados através do ponto do centro da quadra. A precisão dos pontos projetados depende unicamente da precisão dos pontos reconhecidos inicialmente, onde quanto mais precisos forem os pontos reconhecidos, mais precisos serão os pontos projetados. Os resultados para o método adotado para a detecção das linhas do campo são mostrados na Figura 76.

Figura 75 – Exemplos de erro na detecção das linhas da quadra.



Fonte: próprio autor.

Figura 76 – Exemplo de detecção correta das linhas da quadra.



Fonte: próprio autor.

A posição da câmera que captura o jogo faz com que as linhas mais próximas se tornem mais nítidas, enquanto as que estão mais distantes são de difícil detecção devido ao pó do saibro que se solta da quadra durante as jogadas e se deposita sobre a linha do campo, tornando mais difícil sua visibilidade. Uma vez que os vértices externos são detectados incorretamente todos os outros pontos também serão penalizados por este erro.

A Figura 77 mostra a imagem capturada na aplicação prática com suas respectivas coordenadas detectadas pelo algoritmo.

Figura 77 – Exemplo de detecção das linhas da quadra para aplicação prática.



Fonte: próprio autor.



É possível observar que algumas partes das linhas da quadra apresentam algumas distorções na imagem. Esse efeito é devido a distorção causada pela lente da câmera na captura das imagens do jogo. Todas as lentes distorcem as imagens que capturam, umas mais e outras menos, fazendo com que o resultado seja diferente do que é visto na realidade. Como as linhas geradas no método de detecção proposto são retas e não apresentam distorções, elas não conseguem cobrir as linhas originais por completo, deixando as partes distorcidas à mostra. As linhas definidas pelo método de detecção proposto serão consideradas como corretas e sua precisão será avaliada a seguir.

A Tabela 8 traz informações que permitem avaliar a qualidade da detecção dos vértices da quadra.

Tabela 8 – Comparação entre valores reais e detectados para os pontos que compõem as linhas da quadra

Posição Real		Posição Detectada		Erro (Pixels)
x	y	x	y	
2167	1013	2165	1011	2.83
-361	966	-374	967	13.04
1518	86	1517	87	1.41
822	140	818	142	4.47
1777	1006	1778	1004	2.24
-125	971	-137	971	12.00
1428	93	1427	93	1.00
904	135	899	139	6.40
1585	503	1586	505	2.24
402	541	400	540	2.24
1452	156	1452	156	0.00
819	203	819	203	0.00
957	523	953	524	4.12
1124	180	1119	180	5.00
1639	260	1641	264	4.47
552	327	546	331	7.21
1062	299	1062	299	0.00

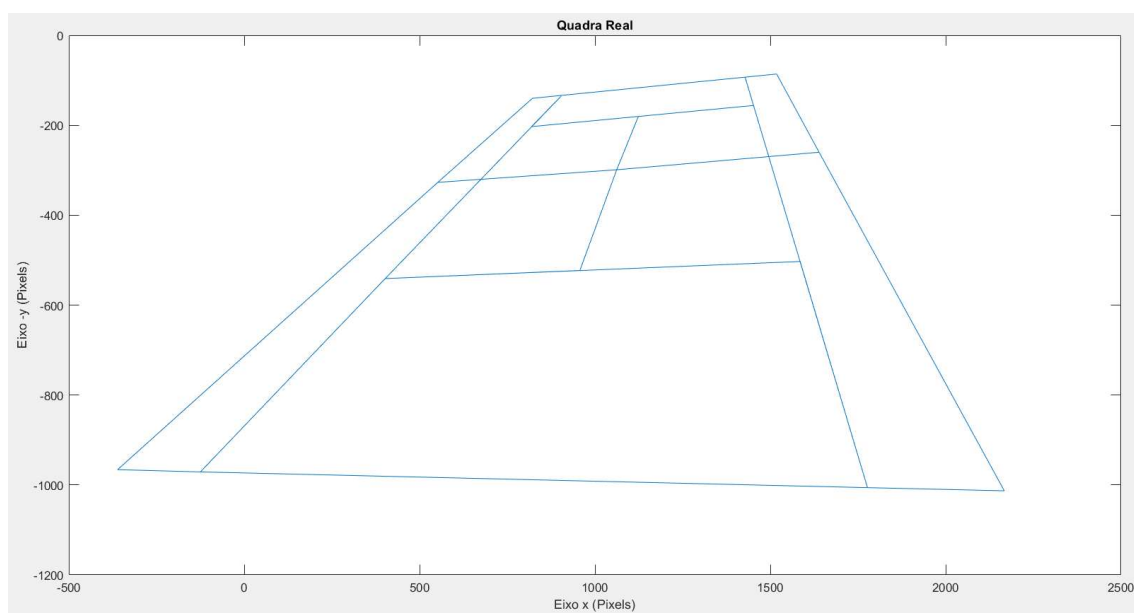
Fonte: próprio autor.

As duas primeiras colunas listam os valores correspondentes às coordenadas verdadeiras x e y dos vértices da quadra na imagem capturada pela câmera na aplicação prática do projeto. Esses valores são os considerados verdadeiros e ideais para correta detecção da quadra na imagem em análise. Importante ressaltar que estes valores foram definidos para a análise da qualidade da detecção pelo método proposto, não sendo em nenhum momento entregues ao algoritmo nem utilizados como realimentação para a correção da detecção, uma vez que o objetivo de aplicação não é apenas para a quadra utilizada, mas sim objetivando detectar as linhas da quadra para qualquer tipo de quadra e qualquer

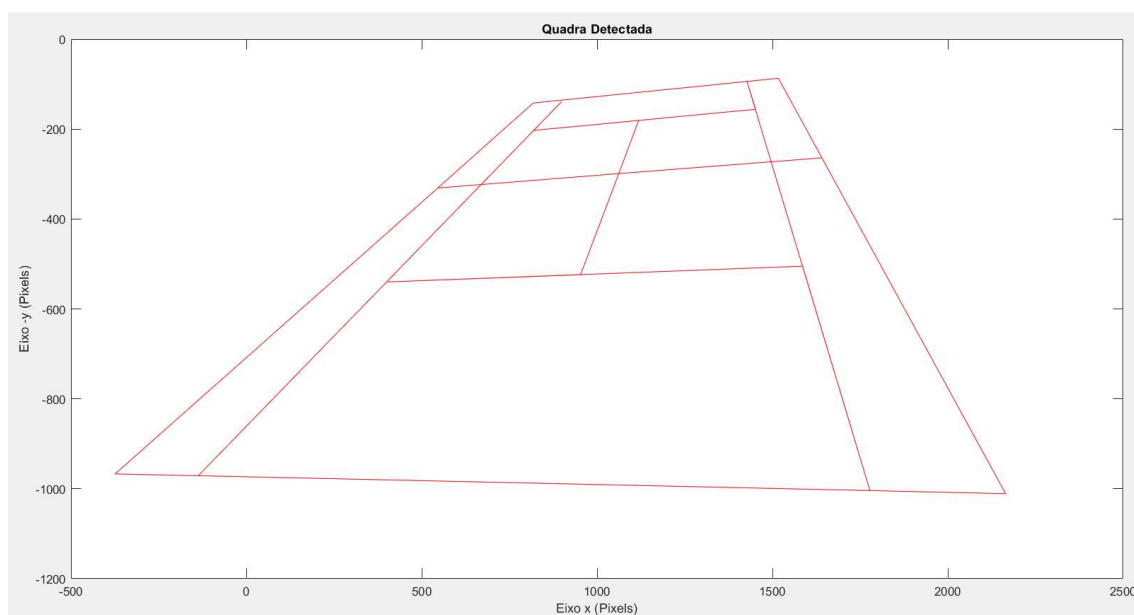
posicionamento da câmera, dentro dos limites já estabelecidos na metodologia. A terceira e quarta coluna mostram as coordenadas x e y detectadas pelo algoritmo de detecção. A quinta coluna traz o resultado do cálculo da distância em pixels entre as coordenadas verdadeiras e as detectadas. A unidade desses valores de erro estão representados em pixels.

A Figura 78 mostra a representação das linhas da quadra detectadas e a posição das linhas reais.

Figura 78 – Representação real e detectada das linhas da quadra de tênis durante experimento. (a) Quadra real. (b) Quadra detectada



(a)



(b)

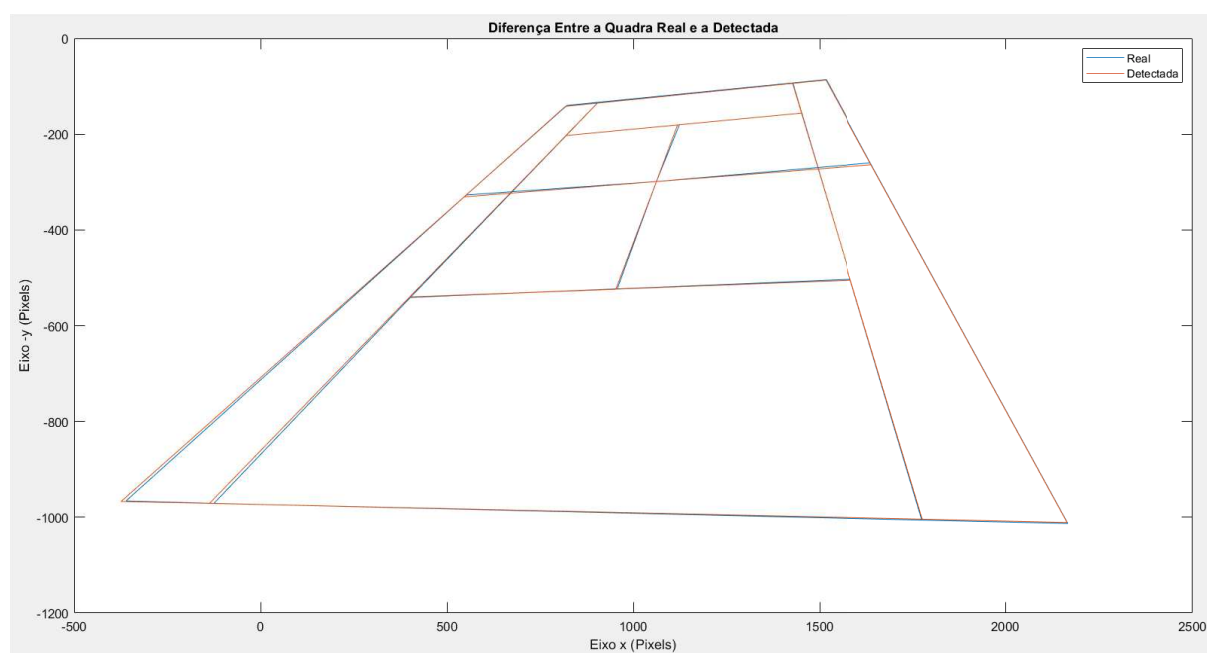
Fonte: próprio autor.

Sabendo que a imagem possui 1920 pixels de comprimento por 1080 pixels de altura, é possível perceber que os valores de erro quando comparados com o tamanho da imagem representam valores muito pequenos.

Com relação aos resultados entregues ao juiz e jogador, o erro entre os pontos que compõem a quadra não representam valores significativos com relação ao tamanho que a bola apresenta próximo da linha. O tamanho da bola de tênis, em pixels, varia na imagem de acordo com sua posição no campo, onde quanto mais perto da câmera, maior será o diâmetro dela em pixels. No campo mais próximo da câmera a bola têm de 10 a 30 pixels de diâmetro, enquanto na quadra mais longe da câmera a bola têm de 5 a 10 pixels de diâmetro. Os quiques que ocorrerem em cima da linha poderão ser difíceis de diferenciar, mas o arbitro sem a utilização de tecnologias também não saberá fazer essa diferenciação. Desta forma, adotar o sistema proposto continua sendo melhor, uma vez que o software é capaz de detectar os quiques duvidosos em geral nas regiões próximas à linha, apresentando incerteza apenas nas regiões que estão a uma distância muito pequena da linha.

A Figura 79 permite a comparação das representações das linhas da quadra detectadas em comparação com a posição das linhas reais. Pela visualização das linhas da quadra é possível perceber que a diferença entre elas é mínima, permitindo concluir que o sistema de detecção da quadra proposto possui boa precisão.

Figura 79 – Gráfico que permite compara representação real e detectada das linhas da quadra de tênis

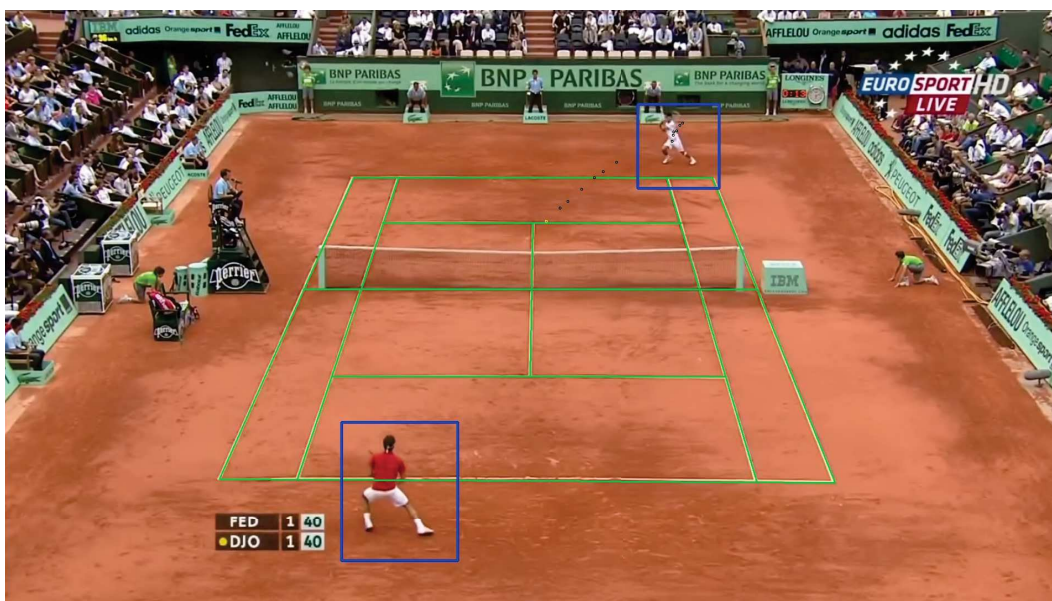


Fonte: próprio autor.

#### 4.4 Detecção dos Jogadores

Inicialmente, o objetivo de detecção dos jogadores era o reconhecimento de ambos os jogadores através das imagens de uma mesma câmera. Tal objetivo foi alcançado nos vídeos utilizados no desenvolvimento do trabalho, como mostra a Figura 80.

Figura 80 – Exemplo de detecção correta dos jogadores.



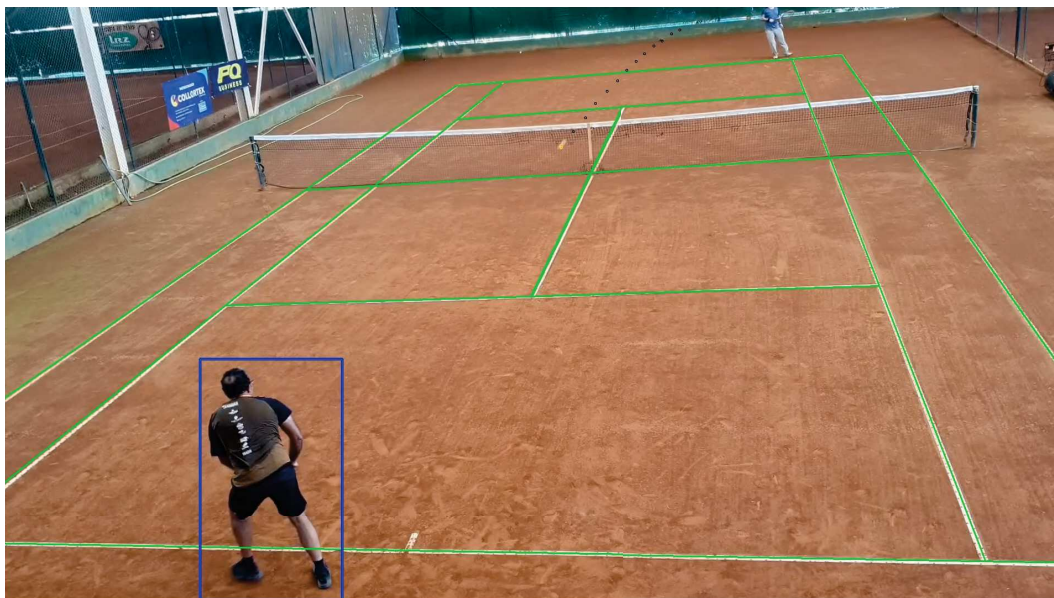
Fonte: próprio autor.

Contudo, na aplicação prática do código desenvolvido no processamento dos vídeos captados pelas câmeras, o algoritmo não foi capaz de identificar o segundo jogador. Essa dificuldade se deve à interferência causada pelo fundo presente na quadra onde a prática fora desenvolvida. Observando novamente a Figura 80, é possível perceber que a detecção dos dois jogadores ocorre em um contexto onde ambos se destacam do restante dos elementos do ambiente. Isso facilita o reconhecimento através do método proposto.

A Figura 81 mostra uma imagem com o resultado final de todo o processamento realizado na prática.

Vale ressaltar que aqui o reconhecimento não é realizado através de processos de deep learning. O reconhecimento é realizado através de operações morfológicas que permitem este processamento. A vantagem dessa abordagem é a obtenção de resultados excelentes de detecção com menor custo computacional e tempo de processamento se comparados ao que seria necessário caso houvesse a adoção de sistemas mais robustos de reconhecimento.

Figura 81 – Imagem capturada pela câmera instalada na quadra com jogador detectado.



Fonte: próprio autor.

#### 4.5 Descrição do Ambiente de Teste

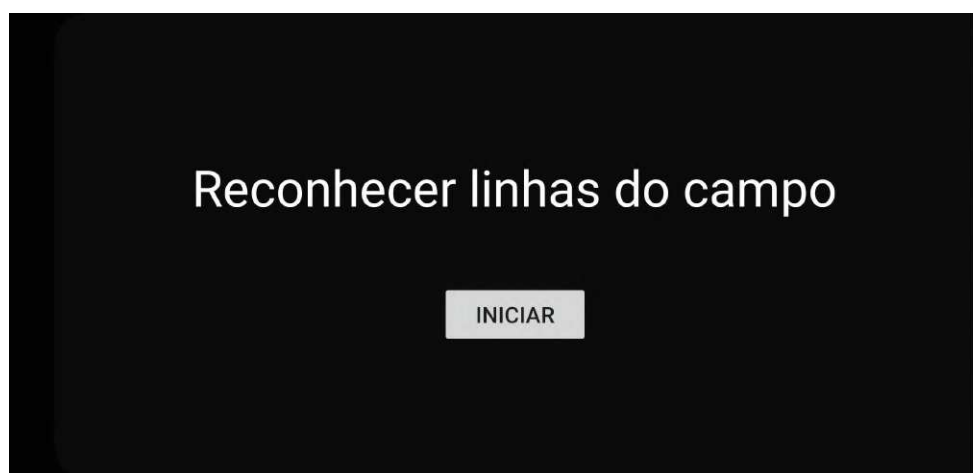
A aplicação desenvolvida é composta por dez telas que definem a interface do usuário. São elas:

- Tela inicial de apresentação da logo do aplicativo
- Tela com botão responsável por iniciar o reconhecimento do campo
- Tela de espera mostrada enquanto o processo de reconhecimento não é finalizado
- Tela que mostra a imagem do campo reconhecido e que solicita ao usuário que verifique se o resultado obtido está correto
- Menu que permite iniciar a gravação do jogo e solicitar revisão da jogada
- Tela que permite ao usuário escolher de qual campo deseja rever o vídeo gravado
- Tela de espera mostrada enquanto o computador envia para o aplicativo o vídeo da câmera selecionada
- Tela que mostra o vídeo completo capturado pela câmera selecionada e que permite a seleção do intervalo que será processado
- Tela de espera mostrada enquanto o computador envia para o aplicativo o vídeo processado
- Tela final que mostra o vídeo processado

A seguir está descrito o funcionamento geral do aplicativo e os processos executados por cada uma das telas abordadas acima.

A primeira tela mostra ao usuário uma animação com a logo do aplicativo. Em seguida, a tela representada na Figura 82, apresenta um botão no qual o usuário deverá clicar quando este desejar iniciar a etapa de reconhecimento das linhas do campo. Nessa etapa, uma mensagem sinalizando que o campo deve ser processado é enviado ao computador através de um socket. O computador recebe essa mensagem e verifica seu conteúdo. Após a mensagem recebida, o código manda uma mensagem para o nó do ROS responsável pelas câmeras 1 e 2 para que uma foto seja tirada de ambos os campos. Essas imagens passarão pelo algoritmo de reconhecimento de campo e os resultados dos dois reconhecimentos nessa etapa são enviados ao aplicativo via socket. Este estágio é realizado antes do início do jogo, uma vez que a presença dos jogadores poderia influenciar negativamente nessa detecção.

Figura 82 – Tela do aplicativo Android que permite o usuário iniciar o reconhecimento do campo.

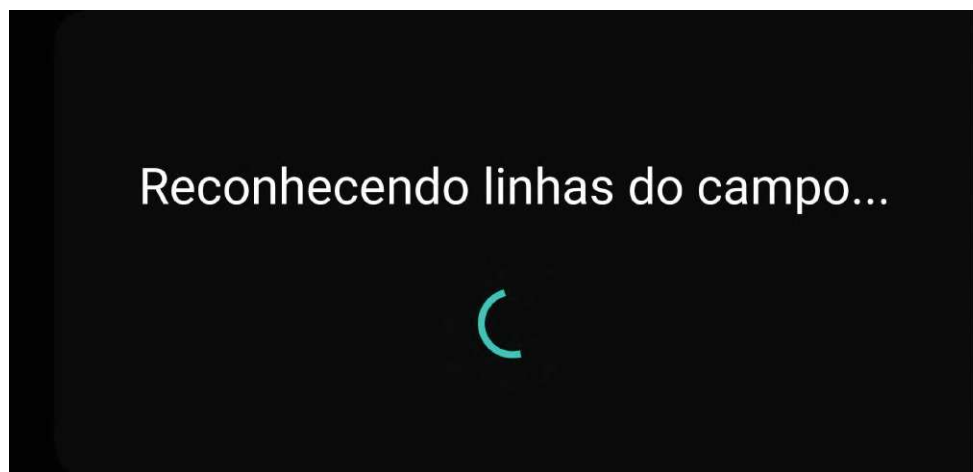


Fonte: próprio autor.

O aplicativo aguardará uma mensagem ao final do processamento para indicar que o reconhecimento da quadra foi concluído. Enquanto essa confirmação não é recebida, a tela da Figura 83 é mostrada ao usuário indicando que o processamento ainda está sendo realizado.

Juntamente com a confirmação de que o campo foi reconhecido, o socket envia para o aplicativo uma mensagem contendo duas imagens, capturadas pelas câmeras instaladas no campo, com a representação do campo detectado em linhas verdes. Essas imagens são mostradas ao usuário através de uma tela como a mostrada na Figura 84. Nessa tela o usuário deverá checar se a detecção foi feita de forma correta. Caso as linhas não tenham sido reconhecidas de forma correta, o usuário poderá selecionar a opção "Sim" e clicar no botão escrito "PRÓXIMO". Essa ação fará com que o aplicativo retorne para a tela da Figura 82 e o reconhecimento poderá ser refeito. Quando o resultado do reconhecimento da

Figura 83 – Tela do aplicativo Android mostrada ao usuário enquanto o aplicativo não recebe confirmação que o campo foi reconhecido.



Fonte: próprio autor.

quadra for satisfatório para o usuário, este deverá escolher a opção "Não", como mostrado na Figura 84, e clicar novamente no botão escrito "PRÓXIMO".

Figura 84 – Tela do aplicativo Android com a imagem da quadra reconhecida com a opção "Não" selecionada.



Fonte: próprio autor.

Como as câmeras estarão em uma posição fixada na quadra, não será necessário refazer esse reconhecimento a cada imagem capturada, pois a posição da quadra reconhecida neste estágio não mudará. A partir desse momento, os contornos mostrados serão tomados como corretos e serão utilizados em todo o processamento. Desta forma, o custo computacional será reduzido pois o reconhecimento será feito apenas uma vez e suas coordenadas serão armazenadas e tomadas como verdadeiras em todo o jogo.

A próxima tela da Figura 85 apresenta o menu que aparecerá para o usuário após o reconhecimento do campo. Através da mesma, o usuário poderá visualizar quanto tempo de jogo foi gravado através do cronômetro presente no meio da tela. Para acionar este cronômetro, o usuário deverá acionar o botão "INICIAR JOGO". Quando este for acionado, um socket será enviado ao computador com uma mensagem que sinalizará para o sistema que o jogo deve começar a ser gravado. Com isso, o código mandará um comando ao nó responsável pelas câmeras para que as imagens capturadas pelas câmeras sejam gravadas. O tempo vai começar a ser contado e o vídeo será captado pelas câmeras. Quando o usuário desejar processar e rever algum momento do jogo, o mesmo deverá apertar o botão "REVER JOGADA". Esse botão, quando acionado, irá zerar o contador, enviar um socket sinalizando que a gravação do jogo deve ser finalizada e fará com que o aplicativo seja direcionado para a tela mostrada na Figura 86.

Figura 85 – Tela do aplicativo Android responsável por iniciar gravação do jogo.



Fonte: próprio autor.

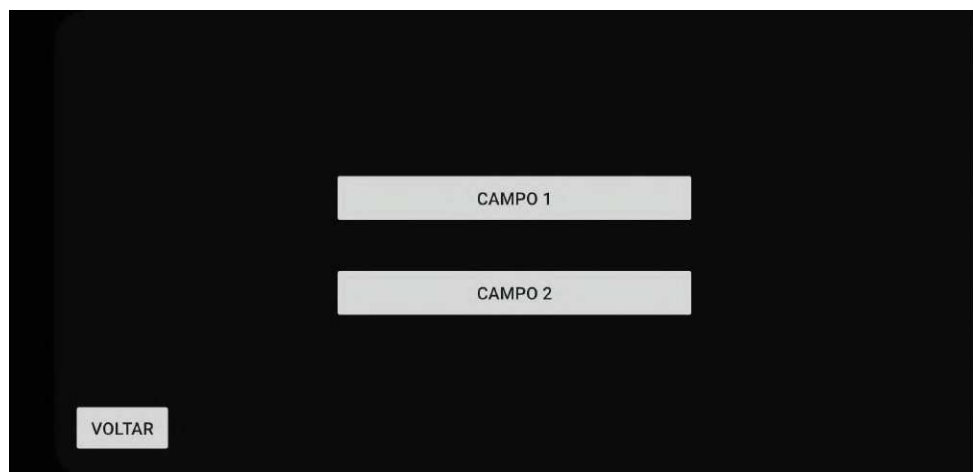
A tela mostrada na Figura 86 apresenta dois botões que servirão para que o usuário selecione de qual câmera deseja visualizar o jogo que fora gravado. Caso o botão "CAMPO 1" seja acionado, o aplicativo envia um socket para o computador sinalizando que o vídeo gravado pela câmera do campo 1 deverá ser convertido. O mesmo ocorre quando o botão "CAMPO 2" é acionado, sinalizando que o vídeo gravado pela câmera do campo 2 deverá ser convertido. O botão "VOLTAR" retorna à tela da Figura 85, zerando o tempo de jogo gravado e ficando pronto para iniciar uma nova gravação.

A seleção de qual vídeo será compilado é necessária pois o vídeo é capturado pelo ROS em um formato .bag, precisando ser convertido para o formato .mp4 para ser reproduzido no *smartphone*. É importante ressaltar que processar todos os vídeos geraria um custo computacional desnecessário, uma vez que apenas um vídeo será mostrado ao usuário por vez e pode ocorrer casos em que este não deseje verificar as duas câmeras.

A tela mostrada na Figura 87 aparecerá em seguida enquanto o vídeo é convertido



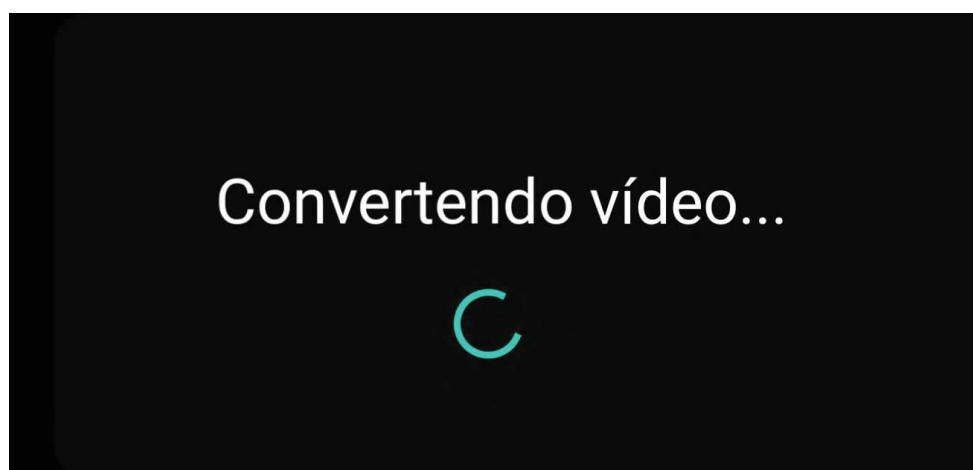
Figura 86 – Tela do aplicativo Android onde é possível selecionar de qual campo deseja-se visualizar o vídeo do jogo.



Fonte: próprio autor.

no computador e será mostrada ao usuário até que a conversão chegue ao fim e o vídeo seja recebido no *smartphone* através de um socket.

Figura 87 – Tela do aplicativo Android mostrada ao usuário enquanto o aplicativo não recebe do computador o vídeo gravado pela câmera selecionada na tela da Figura 86.



Fonte: próprio autor.

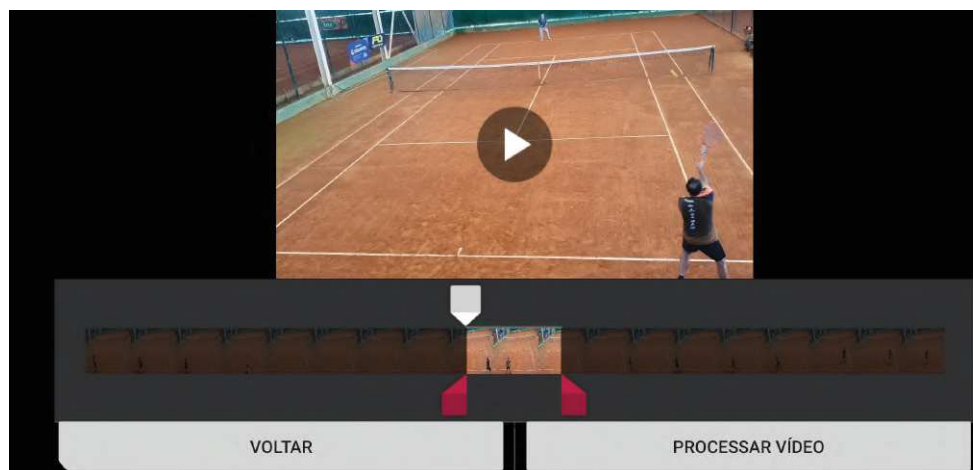
A próxima tela a ser apresentada é a da Figura 88, onde o usuário deverá escolher o trecho do vídeo que será processado pelo algoritmo de visão computacional. O intervalo máximo que poderá ser selecionado é de 5 segundos. A escolha do intervalo é necessária pois o processamento de todos os aspectos relevantes tratados anteriormente geram um custo operacional e tempo de processamento que atrasariam o andamento da partida caso fosse feito o processamento de todo o vídeo.

Desta forma, quando o usuário selecionar o trecho desejado do vídeo gravado da última jogada realizada, seu próximo passo deve ser o acionamento do botão "PROCESSAR

VÍDEO". Essa ação fará com que o intervalo do vídeo selecionado seja enviado ao computador via socket. Quando recebido pelo computador, o intervalo é redirecionado ao processamento do vídeo, que mandará o vídeo resultante ao final do processamento.

O botão "VOLTAR" retorna à tela da Figura 86, possibilitando que a gravação feita seja vista da outra câmera presente na quadra.

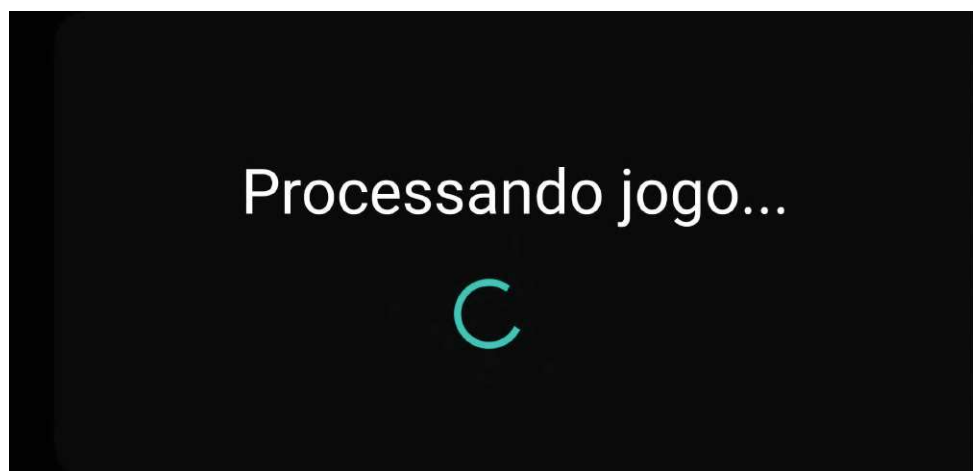
Figura 88 – Tela do aplicativo Android onde é possível selecionar intervalo do vídeo que deverá ser processado.



Fonte: próprio autor.

A tela mostrada na Figura 89 aparecerá em seguida enquanto o vídeo é processado no computador, onde será feita a extração de todas as características de interesse explicadas anteriormente. A tela será mostrada ao usuário até que o processamento chegue ao fim e o vídeo processado seja recebido no smartphone através de um socket.

Figura 89 – Tela do aplicativo Android mostrada ao usuário enquanto o aplicativo não recebe do computador o vídeo processado contendo o trecho selecionado na tela da Figura 88.



Fonte: próprio autor.

Assim que o vídeo for recebido, o mesmo será mostrado ao usuário, tal como visto na Figura 90. A barra de controle do *media player* fica visível quando o usuário toca qualquer lugar na tela, disponibilizando por 1 segundo e desaparecendo em seguida, permitindo que a visualização do vídeo resultante do processamento em modo tela cheia. O botão "VOLTAR" permite ao usuário retornar à tela mostrada na Figura 86, permitindo novamente que o usuário analise a gravação feita pela outra câmera presente na quadra.

Figura 90 – Tela do aplicativo Android que reproduz o vídeo processado.



Fonte: próprio autor.

O vídeo presente no link <<http://bit.ly/Mestrado-TennisApp>> apresenta uma explicação geral do trabalho desenvolvido, mostrando inclusive um exemplo do funcionamento do aplicativo descrito acima.

## 5 Conclusões e Trabalhos Futuros

### 5.1 Conclusões

Este trabalho teve como principal objetivo a aplicação de métodos de *deep learning* e visão computacional no processamento de imagens oriundas de vídeos capturados durante a realização de jogos de tênis. Através da utilização dessas ferramentas, foi possível fornecer dados processados sobre a partida por meio de uma aplicação Android.

Em relação à detecção da bola, o processamento foi feito por meio de duas técnicas diferentes. Os resultados obtidos mostraram que a utilização do SVM em aplicações como a implementada neste trabalho é válida e gera boas detecções em aplicações onde a bola não apresenta distorções na imagem capturada. Todavia, em jogos reais a bola sofre deformações que tornam a detecção uma tarefa difícil, exigindo métodos de aprendizado mais robustos, capazes de realizar detecções confiáveis e assertivas. A alternativa proposta neste trabalho foi a utilização de uma rede neural convolucional de código aberto disponibilizada que tem por objetivo processar as imagens de um jogo de tênis, gerando como resultado imagens contendo à detecção correta da posição da bola durante a partida. Esse método de detecção é amplamente utilizado para classificação de imagens e detecção de objetos devido à sua alta capacidade de extração de características de uma imagem, resultando então em um algoritmo robusto no âmbito do reconhecimento de imagens e objetos em partidas de tênis.

Uma outra contribuição da etapa de processamento das imagens teve como objetivo o reconhecimento das linhas da quadra. A partir do reconhecimento da posição da bola na imagem, foi possível detectar o momento em que a bola toca o chão através da detecção da mudança de direção da bola em cada posição detectada, possibilitando a verificação da posição do quique em relação ao campo, retornando como resultado se este ocorreu dentro ou fora do campo. Por fim, foi feita a detecção dos jogadores durante a partida e apresentada suas posições na tela.

Para entregar esses resultados aos juízes e jogadores durante a realização da partida, desenvolveu-se também neste trabalho um aplicativo Android capaz de fornecer todas as informações de modo prático e acessível. Esse aplicativo permite a visualização de resultados semelhantes aos utilizados em partidas profissionais, podendo ser utilizado tanto em competições quanto em jogos amistosos, permitindo aos jogadores ter resultados profissionais em uma plataforma de baixo custo.

O diferencial deste projeto em relação aos outros presentes na literatura está na metodologia proposta para a obtenção dos resultados. Enquanto a rede neural utilizada originalmente considerava apenas as imagens que apresentavam uma bola detectada por frame, o presente trabalho considerou todas as imagens de saída da rede, tratando os resultados fornecidos pelas imagens onde há mais de uma bola detectada, onde são consideradas apenas as posições que pertencem à trajetória da bola. Esta abordagem

se alinha com a realidade ao reconhecer que o ambiente de aquisição da imagem pode apresentar características semelhantes ao que a rede neural aprendeu como sendo uma característica da bola, ainda mais pela qualidade da rede neural apresentada que é capaz de reconhecer a bola até em situações onde esta se apresenta de forma distorcida. O tratamento dos resultados fornecidos pela rede gerou um aumento de 3,4865% na precisão dos resultados obtidos.

Outro diferencial está nos processos de reconhecimento das linhas da quadra, onde a detecção dos vértices presentes em um campo da quadra permite a dedução de todos os vértices do outro campo através da manipulação matemática dos pontos reconhecidos, gerando uma detecção precisa para todo o campo. Este método supera as dificuldades provenientes do processamento de jogos em quadras de saibro, onde resíduos deste material se depositam sobre as linhas do campo, dificultando o reconhecimento. É importante salientar que essa estratégia é distinta de todas as encontradas na literatura.

Um outro ponto de destaque para o projeto é a detecção dos jogadores no campo. Na grande maioria das pesquisas desenvolvidas são utilizados métodos de reconhecimento de padrões e aprendizado de máquina para a detecção de jogadores. O presente trabalho obteve resultados excelentes de detecção utilizando apenas operações morfológicas na detecção dos jogadores, gerando um custo de processamento e tempo computacional muito abaixo comparado a outras propostas provenientes da literatura, visto a simplicidade de suas etapas de detecção. O trabalho apresentou ótimos resultados, demonstrando assim sua relevância para sua área de pesquisa.

## 5.2 Trabalhos Futuros

Como proposta para outros projetos, sugere-se a sofisticação do aplicativo desenvolvido, adicionando novas funcionalidades relacionadas ao processamento do jogo e às informações que são disponibilizadas ao juiz e aos jogadores.

Quanto ao processamento, o projeto desenvolvido adotou o uso de uma câmera em cada campo para o processamento do jogo, fornecendo dados obtidos de um ponto de referência somente. As ferramentas existentes de visão computacional abrem a possibilidade de se desenvolver uma aplicação mais ampla nesse aspecto, que seja capaz de gerar dados de diversos pontos de vista de um mesmo jogo através da utilização de várias câmeras gravando um mesmo jogo.

Com a adoção de várias câmeras, há também a possibilidade de gerar a projeção de uma nuvem de pontos 3D através da triangulação de duas ou mais câmeras 2D que farão a captação do jogo em posições diferentes, de forma a gerar dados que minimizem áreas oclusas durante o jogo. Assim, pode-se realizar a computação do jogador, da posição da bola, do momento preciso em que está quica no chão e mostrá-lo em uma representação 3D no espaço e analisar outras características que forem alvo de estudo, resultando em

um processamento que permitirá extrair informações diferentes das disponíveis na análise feita em 2D.

Um outro trabalho que pode ser desenvolvido seria um aprimoramento na detecção dos jogadores, utilizando o método já desenvolvido neste trabalho para detectar a primeira posição do jogador. A partir dessa identificação na imagem de onde o jogador está, o processamento faria a extração de mais características dos jogadores, como por exemplo cores e texturas. Com isso o rastreamento seguiria a partir dos dados extraídos e muitas informações poderiam ser utilizadas no rastreamento do jogador.

Outro ponto a ser considerado é a correção das distorções na imagem capturada pela câmera e a avaliação dos efeitos desta correção no método de detecção das linhas da quadra. A distorção da imagem capturada pela câmera ocorre em todas as lentes dependendo de seu nível de curvatura. Em trabalhos futuros propõe-se a análise de algoritmos que sejam capazes de corrigir este efeito.

Uma estudo relevante que pode ser abordado é com relação a necessidade da calibração do sistema no ambiente de teste. Essa etapa consiste em mudar as posições das câmeras e dos odroids, de forma a fazer uma coleta de dados que permita analisar a precisão do sistema. Desta forma, é possível observar se a captura da câmera tem alguma influência na precisão.

Outra abordagem que pode ser feita em pesquisas futuras é quanto a aquisição de dados relativos ao andamento do jogo, como por exemplo a obtenção automática do placar do jogo e da posse de bola, através do uso de visão computacional. Esses dados podem ser fornecidos aos jogadores durante a realização da partida através do smartphone ou smartwatch, permitindo que uma contagem precisa em competições ou até mesmo em partidas amistosas.

Além dessas informações, outros dados de interesse também podem ser processados e entregues ao jogador. O aplicativo pode fornecer por exemplo uma planilha de dados estatísticos do jogo, conhecida como Scout, que é uma ferramenta de avaliação muito utilizada durante os jogos de tênis. Os dados contidos nessa planilha são de grande importância para a verificação da eficiência de cada golpe durante uma partida, mostrando para o atleta análises completas de seu desempenho no jogo, permitindo ao jogador saber quais técnicas ele deve treinar para melhorar suas jogadas e ter mais consistência. O número de primeiros saques feitos de forma correta, a velocidade da bola na partida e o mapeamento dos locais na quadra onde a bola mais acerta são exemplos dos dados que podem ser processados e fornecidos pelo aplicativo.

Finalmente, pretende-se construir um MVP (Minimum Viable Product) com intuito de apresentar o presente projeto em eventos com investidores em projetos de pesquisa de inovação. Para tanto, far-se-á uma pesquisa junto ao INPI para verificar a possibilidade de patenteamento da metodologia aqui apresentada.

## REFERÊNCIAS

- ACADEMY, D. S. *Deep Learning Book*. 2019. <<http://www.deeplearningbook.com.br>>. Acessado em 27/11/2020.
- AGGARWAL, C. C. *Neural Networks and Deep Learning: A Textbook*. [S.l.]: Springer, 2018.
- AHMAD, I. *40 Algorithms Every Programmer Should Know*. [S.l.]: Packt Publishing, 2020.
- ALATAS, B. Sports inspired computational intelligence algorithms for global optimization. *Artificial Intelligence Review*, Springer, v. 52, n. 3, p. 1579–1627, 2019.
- ALBUQUERQUE, O. E. d. Uma interface para o controle de robôs móveis por intermédio de gestos. 2016.
- ANTHONY, S. *Football: A deep dive into the tech and data behind the best players in the world*. 2017. <<https://arstechnica.com/science/2017/05/football-data-tech-best-players-in-the-world/>>. Acessado em 07/01/2021.
- BALSYS, R. *Convolutional Neural Networks (CNN) explained*. 2019. <<https://pylessons.com/CNN-tutorial-introduction/>>. Acessado em 03/12/2020.
- BONACCORSO, G. *Machine Learning Algorithms - Second Edition*. [S.l.]: Packt Publishing, 2018.
- BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer vision in C++ with the OpenCV library*. [S.l.]: O'Reilly Media, Inc, 2012.
- BRASIL, C. O. do. *Tênis - Confederação Brasileira de Tênis*. 2020. <<https://www.cob.org.br/pt/cob/time-brasil/esportes/tenis/>>. Acessado em 14/02/2021.
- BUNKER, R. P.; THABTAH, F. A machine learning framework for sport result prediction. *Applied computing and informatics*, Elsevier, v. 15, n. 1, p. 27–33, 2019.
- BUONOCORE, L.; JÚNIOR, C. N.; NETO, A. de A. Sensor data fusion algorithm for indoor environment mapping using low-cost sensors. *Journal of Control, Automation and Electrical Systems*, v. 24, n. 3, p. 199–211, 2013.
- CANNY, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Image Understanding*, v. 18, 1986.
- CANT, O. et al. Validation of ball spin estimates in tennis from multi-camera tracking data. *Journal of Sports Sciences*, Taylor & Francis, v. 38, n. 3, p. 296–303, 2020.
- CASTRO, R. L.; CANOSA, D. A. Using artificial vision techniques for individual player tracking in sport events. In: *Multidisciplinary Digital Publishing Institute Proceedings*. [S.l.: s.n.], 2019. v. 21, n. 1, p. 21.
- CHEN, C.-C. et al. Unsupervised learning and pattern recognition of biological data structures with density functional theory and machine learning. *Scientific reports*, Nature Publishing Group, v. 8, n. 1, p. 557, 2018.

- CHEN, J.; LITTLE, J. J. Where should cameras look at soccer games: Improving smoothness using the overlapped hidden markov model. *Computer Vision and Image Understanding*, Elsevier, v. 159, p. 59–73, 2017.
- CHITYALA, R.; PUDIPEDDI, S. *Image Processing and Acquisition using Python, 2nd Edition*. [S.l.]: Chapman and Hall/CRC, 2020.
- CIPOLLA, R.; FARINELLA, G. M.; BATTIATO, S. *Machine Learning for Computer Vision*. [S.l.]: Springer, 2012.
- COELHO, F. O. et al. EKF and computer vision for mobile robot localization. In: IEEE. *2018 13th APCA International Conference on Automatic Control and Soft Computing (CONTROLO)*. [S.l.], 2018. p. 148–153.
- COELHO, F. O. et al. Seme signals classification using CNN features extraction as a reliable method. *Anais da Sociedade Brasileira de Automática*, v. 2, n. 1, 2020.
- DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.
- DANIEL, G.; CHEN, M. *Video visualization*. [S.l.]: IEEE, 2003.
- DEY, S. *Hands-On Image Processing with Python*. [S.l.]: Packt Publishing, 2018.
- DEY, S. *Python Image Processing Cookbook*. [S.l.]: Packt Publishing, 2020.
- DUDA, R. O.; HART, P. E. Use of the Hough transformation to detect lines and curves in pictures. In: *Communications of the ACM*. [S.l.: s.n.], 1972.
- ESCRIVA, D. M.; LAGANIERE, R. *OpenCV 4 Computer Vision Application Programming Cookbook - Fourth Edition*. [S.l.]: Packt Publishing, 2019.
- FARIA, E. L. d. *Redes neurais convolucionais e máquinas de aprendizado extremo aplicadas ao mercado financeiro brasileiro*. 2018.
- FRANÇOIS-LAVET, V. et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, Now Publishers, Inc., v. 11, n. 3-4, p. 219–354, 2018.
- FUENTE, A. *Hands-On Predictive Analytics with Python*. [S.l.]: Packt Publishing, 2018.
- FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. 1980.
- GALEONE, P. *Hands-On Neural Networks with TensorFlow 2.0*. [S.l.]: Packt Publishing, 2019.
- GEVORGYAN, M.; BEYELER, M.; MAMIKONYAN, A. *OpenCV 4 with Python Blueprints - Second Edition*. [S.l.]: Packt Publishing, 2020.
- GIRSHICK, R. Fast r-CNN. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1440–1448.



- GOMEZ-GONZALEZ, S. et al. Reliable real-time ball tracking for robot table tennis. *Robotics*, Multidisciplinary Digital Publishing Institute, v. 8, n. 4, p. 90, 2019.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.
- GRILO, F.; FIGUEIREDO, J. Computer vision in industrial automation and mobile robots. In: *Introduction to Mechanical Engineering*. [S.l.]: Springer, 2018. p. 241–266.
- GÉRON, A. *Neural networks and deep learning*. [S.l.]: O’Reilly Media, Inc., 2018.
- HAAREN, J. V. et al. Machine learning and data mining for sports analytics. Citeseer, 2013.
- HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778.
- HOLLAND, J. H. *Adaptation in Natural Artificial Systems – An Introductory Analysis with Application to Biology, Control and Artificial Intelligence*. [S.l.]: MIT Press, 1992.
- HOWSE, J.; MINICHINO, J. *Learning OpenCV 4 Computer Vision with Python 3*. [S.l.]: Packt Publishing, 2020.
- HUANG, J. et al. Video-based sign language recognition without temporal segmentation. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2018.
- HUANG, Y.-C. et al. Tracknet: A deep learning network for tracking high-speed and tiny objects in sports applications. In: IEEE. *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. [S.l.], 2019. p. 1–8.
- HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex. In: *The Journal of physiology*. [S.l.: s.n.], 1962. v. 160.
- HUBEL, D. H.; WIESEL, T. N. Ferrier lecture - functional architecture of macaque monkey visual cortex. In: ROYAL SOCIETY PUBLISHING. *Proceedings of the Royal Society of London. Series B. Biological Sciences*. [S.l.], 1977. v. 198.
- JIANG, X. et al. *Deep Learning in Object Detection and Recognition*. [S.l.]: Springer, 2019.
- KAEHLER, A.; BRADSKI, G. *Learning OpenCV 3*. [S.l.]: O’Reilly Media, Inc, 2016.
- KAMBLE, P.; KESKAR, A.; BHURCHANDI, K. A deep learning ball tracking system in soccer videos. *Opto-Electronics Review*, Elsevier, v. 27, n. 1, p. 58–69, 2019.
- KAPUR, S. *Computer Vision with Python 3*. [S.l.]: Packt Publishing, 2017.
- KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE. *Proceedings of ICNN’95 - International Conference on Neural Networks*. [S.l.], 1995.
- KOS, M.; KRAMBERGER, I. A wearable device and system for movement and biometric data acquisition for sports applications. *IEEE Access*, IEEE, v. 5, p. 6411–6420, 2017.

- KOUBÂA, A. *Ros services*. 2019. <<http://wiki.ros.org/Services>>. Acessado em 06/11/2020.
- LAGANIERE, R. *OpenCV 3 Computer Vision Application Programming Cookbook - Third Edition*. [S.l.]: Packt Publishing, 2017.
- LE, Q. V. et al. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In: IEEE. *CVPR 2011*. [S.l.], 2011. p. 3361–3368.
- LECUN, Y. et al. Gradient-based learning applied to document recognition. 1998.
- LIN, H.-I.; YU, Z.; HUANG, Y.-C. Ball tracking and trajectory prediction for table-tennis robots. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 20, n. 2, p. 333, 2020.
- LIU, J. et al. Automatic player detection, labeling and tracking in broadcast soccer video. *Pattern Recognition Letters*, Elsevier, v. 30, n. 2, p. 103–113, 2009.
- LOY, J. *Neural Network Projects with Python*. [S.l.]: Packt Publishing, 2019.
- MA, J.; ZHAO, L.; HAN, Z. Identification of wiener model using least squares support vector machine optimized by adaptive particle swarm optimization. *Journal of Control, Automation and Electrical Systems*, v. 26, n. 6, p. 609—615, 2015.
- MACIEL, G. M. et al. Methodology for autonomous crossing narrow passages applied on assistive mobile robots. *Journal of Control, Automation and Electrical Systems*, Springer, v. 30, n. 6, p. 943–953, 2019.
- MAHTANI, A. et al. *ROS Programming: Building Powerful Robots*. [S.l.]: Packt Publishing, 2018.
- MANAFIFARD, M.; EBADI, H.; MOGHADDAM, H. A. A survey on player tracking in soccer videos. *Computer Vision and Image Understanding*, Elsevier, v. 159, p. 19–46, 2017.
- MAO, J. Tracking a tennis ball using image processing techniques. 2006.
- MARGARITIS, K. G.; KYRIAKIDES, G. *Hands-On Ensemble Learning with Python*. [S.l.]: Packt Publishing, 2019.
- MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.
- MELO, A. G. et al. 3d correspondence and point projection method for structures deformation analysis. *IEEE Access*, IEEE, v. 8, p. 177823–177836, 2020.
- MESSELODI, S. et al. A low-cost computer vision system for real-time tennis analysis. *International Conference on Image Analysis and Processing*, 2019.
- MOLIN, S. *Hands-On Data Analysis with Pandas*. [S.l.]: Packt Publishing, 2019.
- MORA, S. V. Computer vision and machine learning for in-play tennis analysis: Framework, algorithms and implementation. 2017.
- MOTA, J. *Ros Topics*. 2020. <<http://wiki.ros.org/rostopic>>. Acessado em 17/10/2020.

MYERS, N. et al. The sony smart tennis sensor accurately measures external workload in junior tennis players. *International Journal of Sports Science Coaching*, v. 14, p. 24–31, 2019.

NOH, H.; HONG, S.; HAN, B. Learning deconvolution network for semantic segmentation. In: IEEE. *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.], 2015. p. 1520–1528.

NORVIG, P.; RUSSELL, S. *Artificial Intelligence: A Modern Approach*. [S.l.]: Prentice Hall, 1994.

OTANI, M. et al. Video summarization using textual descriptions for authoring video blogs. *Multimedia Tools and Applications*, Springer, v. 76, n. 9, p. 12097–12115, 2017.

OWENS, N.; HARRIS, C.; STENNETT, C. Hawk-eye tennis system. In: IET. *2003 International Conference on Visual Information Engineering VIE 2003*. [S.l.], 2003. p. 182–185.

PATRÍCIO, D. I.; RIEDER, R. Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review. *Computers and Electronics in Agriculture*, Elsevier, v. 153, p. 69–81, 2018.

PINTO, M. F. et al. A robotic cognitive architecture for slope and dam inspections. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 20, n. 16, p. 4579, 2020.

PINTO, M. F. et al. Case-based reasoning approach applied to surveillance system using an autonomous unmanned aerial vehicle. In: IEEE. *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*. [S.l.], 2017. p. 1324–1329.

PLANCHE, B.; ANDRES, E. *Hands-On Computer Vision with TensorFlow 2*. [S.l.]: Packt Publishing, 2019.

PROTAS Églen da V. Visualização de camadas intermediárias de redes neurais convolucionais de transformação de imagem. 2017.

QUIGLEY, M.; GERKEY, B.; SMART, W. D. *Programming Robots with ROS*. [S.l.]: O'Reilly Media, Inc., 2015.

RAYMOND, C. J.; MADAR, T. J.; MONTTOYE, A. H. Accuracy of the babolat pop sensor for assessment of tennis strokes in structured and match play settings. *Journal of Sport and Human Performance*, v. 7, 2019.

REAL-TIME Motion Template Gradients using Intel CVLib. In: IEEE ICCV Workshop on Framerate Vision. [S.l.: s.n.].

RENÒ, V. et al. Convolutional neural networks based ball detection in tennis games. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, 2018.

ROJAS, R. The backpropagation algorithm. In: *Neural networks*. [S.l.]: Springer, 1996. p. 149–182.

SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. <<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53/>>. Acessado em 03/12/2020.

- SALEH, H. *Machine Learning Fundamentals*. [S.l.]: Packt Publishing, 2018.
- SCHALKOFF, R. J. Pattern recognition. *Wiley Encyclopedia of Computer Science and Engineering*, Wiley Online Library, 2007.
- SHANMUGAMANI, R. *Deep Learning for Computer Vision*. [S.l.]: Packt Publishing, 2018.
- SILVEIRA, G. Direct 3-d tracking for central omnidirectional cameras under general lighting variations. *Journal of Control, Automation and Electrical Systems*, Springer, v. 24, n. 1-2, p. 129–138, 2013.
- SIMONYAN, K.; ZISSERMAN, A. Two-stream convolutional networks for action recognition in videos. *arXiv preprint arXiv:1406.2199*, 2014.
- SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- SOUZA, K. P.; PISTORI, H. Implementação de um extrator de características baseado em momentos da imagem. 2005.
- SPIZHEVOY, A.; RYBNIKOV, A. *OpenCV 3 Computer Vision with Python Cookbook*. [S.l.]: Packt Publishing, 2018.
- STEIN, M. et al. Bring it to the pitch: Combining video and movement data to enhance team sport analysis. *IEEE transactions on visualization and computer graphics*, IEEE, v. 24, n. 1, p. 13–22, 2017.
- STEINBERG, L. Changing the game: The rise of sports analytics. In: *Forbes*. [S.l.: s.n.], April 2015.
- TEACHABARIKITI, K.; CHALIDABHONGSE, T. H.; THAMMANO, A. Players tracking and ball detection for an automatic tennis video annotation. In: IEEE. *2010 11th International Conference on Control Automation Robotics & Vision*. [S.l.], 2010. p. 2461–2494.
- THOMAS, G. Sports tv applications of computer vision. In: *Visual Analysis of Humans*. [S.l.]: Springer, 2011. p. 563–579.
- THOMAS, G. et al. Computer vision for sports: Current applications and research topics. *Computer Vision and Image Understanding*, Elsevier, v. 159, p. 3–18, 2017.
- TRAN, D. et al. Learning spatiotemporal features with 3d convolutional networks. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 4489–4497.
- VALUE, S. *Impactos do coronavírus para a Indústria do Esporte*. 2020. <<https://www.sportsvalue.com.br/estudos/impactos-do-coronavirus-para-a-industria-do-esporte/>>. Acessado em 02/03/2021.
- VAPNIK, V. Pattern recognition using generalized portrait method. In: *Automation and Remote Control*. [S.l.: s.n.], 1963. p. 774–780.
- VIANA, F. X. et al. Aerial image instance segmentation through synthetic data using deep learning. *Learning and NonLinear Models*, 2020.

- VILLAN, A. F. *Mastering OpenCV 4 with Python*. [S.l.]: Packt Publishing, 2019.
- VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. [S.l.], 2004.
- VONDRICK, C. et al. HOGgles: Visualizing Object Detection Features. *ICCV*, 2013.
- WERBOS, P. Beyond regression: "new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.
- WU, Y.; LIM, J.; YANG, M.-H. Online object tracking: A benchmark. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2013. p. 2411–2418.
- YAN, F. et al. Automatic annotation of tennis games: An integration of audio, vision, and learning. *Image and Vision Computing*, Elsevier, v. 32, n. 11, p. 896–903, 2014.
- YANG, C.-S.; YANG, Y.-H. Improved local binary pattern for real scene optical character recognition. *Pattern Recognition Letters*, v. 100, p. 14–21, 2017.
- YANG, H. et al. Recent advances and trends in visual tracking: A review. *Neurocomputing*, Elsevier, v. 74, n. 18, p. 3823–3831, 2011.
- YANG, X. A new metaheuristic bat-inspired algorithm. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, Springer, v. 284, p. 65–74, 2010.
- ZAFAR, I. et al. *Hands-On Convolutional Neural Networks with TensorFlow*. [S.l.]: Packt Publishing, 2018.
- ZHANG, H.; SONG, Y.; SONG, H.-T. Construction of ontology-based user model for web personalization. In: CONATI, C.; MCCOY, K. F.; PALIOURAS, G. (Ed.). *Proceedings of the 11 International Conference on User Modeling*. Corfu, Greece: Springer, 2007. (Lecture Notes in Computer Science, v. 4511), p. 67–76. ISBN 978-3-540-73077-4.