CENTRO UNIVERSITÁRIO FEI

ISAAC JESUS DA SILVA

SISTEMAS COGNITIVOS PARA AGENTES ROBÓTICOS BASEADOS EM APRENDIZADO PROFUNDO

São Bernardo do Campo

2019

ISAAC JESUS DA SILVA

SISTEMAS COGNITIVOS PARA AGENTES ROBÓTICOS BASEADOS EM APRENDIZADO PROFUNDO

Tese de Doutorado, apresentada ao Centro Universitário FEI para obtenção do título de Doutor em Engenharia Elétrica, orientada pelo Prof. Dr. Reinaldo Augusto da Costa Bianchi.

São Bernardo do Campo

2019

da Silva, Isaac Jesus.
Sistemas Cognitivos para Agentes Robóticos Baseados em Aprendizado Profundo / Isaac Jesus da Silva. São Bernardo do Campo, 2019.
156 p. : il.

Tese - Centro Universitário FEI.
Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi.

1. Aprendizado Profundo. 2. Redes Neurais Profundas. 3. Aprendizado por Reforço Profundo. 4. Robôs Móveis. I. Bianchi, Reinaldo Augusto da Costa, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

centro universitário FEI

APRESENTAÇÃO DE TESE
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Doutorado

PGE-10

Aluno: Isaac Jesus da Silva **Matrícula:** 515201-2

Título do Trabalho: Sistemas cognitivos para agentes robóticos baseado em aprendizado profundo.

Área de Concentração: Inteligência Artificial Aplicada à Automação e Robótica

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

Data da realização da defesa: 03/09/2019 **ORIGINAL ASSINADA**

Avaliação da Banca Examinadora

São Bernardo do Campo, / / .

MEMBROS DA BANCA EXAMINADORA	
Prof. Dr. Reinaldo Augusto da Costa Bianchi	Ass.:
Prof. Dr. Flavio Tonidandel	Ass.:
Prof. Dr. Plinio Thomaz Aquino Junior	Ass.:
Prof. Dr. Paulo Lilles Jorge Drews Junior	Ass.:
Prof. Dr. Marcos Ricardo Omena de Albuquerque Máximo	Ass.:

A Banca Examinadora acima-assinada atribuiu ao aluno o seguinte:

APROVADO ☒ REPROVADO ☐

VERSÃO FINAL DA TESE

ENDOSSO DO ORIENTADOR APÓS A INCLUSÃO DAS RECOMENDAÇÕES DA BANCA EXAMINADORA

Aprovação do Coordenador do Programa de Pós-graduação

Prof. Dr. Carlos Eduardo Thomaz

Aos meus pais.

AGRADECIMENTOS

Agradeço aos meus falecidos pais que sempre, incentivaram, financiaram e apoiaram meus estudos.

Agradeço grandemente ao Prof. Dr. Reinaldo Augusto da Costa Bianchi pelas orientações, apoio, motivação, sugestões, pelas aulas, por acreditar no meu trabalho e pelas oportunidades que tem me dado.

Aos professores do Centro Universitário da FEI, pelo incentivo e pelas aulas ministradas: Prof. Dr. Flavio Tonidandel, Prof. Dr. Plinio Thomaz Aquino Junior, Prof. Dr. Paulo Sérgio Silva Rodrigues.

Agradeço aos meus amigos da pós-graduação do Centro Universitário da FEI: Prof. Dr. Danilo H. Perico, Claudio de O. Vilão Jr, Prof. Dr. Thiago P. D. Homem, Vinícius N. Ferreira e Aislan C. Almeida, pelas pesquisas e contribuições a este trabalho.

Agradeço ao Douglas De Rizzo Meneghetti por proporcionar a classe Latex (fei.cls) nos padrões de formatação exigidos pela biblioteca da FEI e ABNT.

Agradeço aos alunos de graduação que trabalharam e contribuíram no projeto RoboFEI durante a realização deste trabalho.

Agradeço à CAPES pela bolsa de estudos de doutorado.

Agradeço a Deus.

“Se o conhecimento pode criar problemas, não é através da ignorância que podemos solucioná-los”

Isaac Asimov

RESUMO

Nos últimos anos o Aprendizado Profundo tem atraído a atenção dos pesquisadores de diversas áreas relacionadas à Aprendizagem de Máquina devido aos resultados que vem apresentando. Estimulado por estes resultados, o objetivo desta pesquisa é analisar, implementar e definir novas formas de aplicações do Aprendizado Profundo em agentes robóticos móveis para a cognição robótica, um tema pouco explorado pela comunidade científica. Desenvolver um sistema de visão adequado para trabalhar com o sistema de tomada de decisões para um robô humanoide jogar futebol de forma autônoma, tem sido um desafio devido a alguns fatores, tais como: limitação do poder computacional do computador embarcado e dinâmica do jogo. Nesta proposta, uma arquitetura com Aprendizado Profundo substitui diversos processos responsáveis por tarefas de cognição em agentes robóticos no intuito de diminuir a complexidade das atuais arquiteturas e elevar sua capacidade de cognição. Foram desenvolvidas duas abordagens, uma com Redes Neurais Profundas e a outra com Aprendizado por Reforço Profundo. Na primeira abordagem foi desenvolvido um sistema de cognição com Redes Neurais Profundas, e também foi apresentado uma nova arquitetura chamada Árvore de Decisão de Redes Neurais Profundas, onde foi possível aumentar a revocação e diminuir o custo computacional. A segunda abordagem apresenta o Aprendizado por Reforço Profundo como uma possível solução para a cognição de agentes robóticos. O sistema de cognição com Aprendizado por Reforço Profundo proporciona ao robô humanoide a capacidade de aprender tarefas apenas pela observação do ambiente através de imagens provenientes da própria câmera do robô, tarefas relacionadas ao papel de um robô jogador de futebol, tais como: goleiro e batedor de pênaltis. Para isso o agente interage com o ambiente recebendo recompensas de acordo com as ações tomada. Os sistemas propostos foram avaliados por experimentos realizados no domínio de futebol de robôs humanoides da RoboCup, em robôs reais e em ambiente simulado, com a implementação de Redes Neurais Profundas e Aprendizado por Reforço Profundo, apresentando resultados notáveis em relação ao objetivo desta pesquisa.

Palavras-chave: Aprendizado Profundo, Redes Neurais Profundas, Aprendizado por Reforço Profundo, Robôs Móveis.

ABSTRACT

In the last years deep learning has attracted the attention of researchers in several areas related to machine learning due to the results it has presented. Encouraged by these results, the aim of this research project is to analyze, implement and define new applications forms of deep learning in mobile robotic agents, an area little explored by the scientific community. Developing a vision system agreed with a decision-making system for a humanoid robot has been a challenge due to some factors, such as: computational power of the embedded computer is limited and the dynamics of the game. In this proposal, an architecture with deep learning replaces several processes responsible for cognition tasks in robotic agents, in order to reduce the complexity of current architectures and increase their cognition capacity. Two approaches were developed, one with Deep Neural Networks and the other with Deep Reinforcement Learning. In the first approach a deep neural network cognition system was developed, and a new architecture called Decision Tree of Deep Neural Networks was presented, where it was possible to increase the recall and decrease the computational cost. The second approach presents Deep Reinforcement Learning as a possible solution for the cognition of robotic agents. The cognition system with deep reinforcement learning provides ability to humanoid robot of learn tasks only by observing the environment through raw images from robot's camera, tasks related to the role of a robot soccer player, such as: goalkeeper and shoot penalty kick. For this the agent interacts with the environment receiving rewards according to the actions taken. The proposed systems were evaluated by experiments performed in the RoboCup humanoid robot soccer field, in real robots and in a simulated environment, with the implementation of deep neural networks and deep reinforcement learning, presenting notable results in relation to the aim of this research.

Keywords: Deep Learning, Deep Neural Network, Deep Reinforcement Learning, Mobile Robots

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS

A3C	Asynchronous Advantage Actor-Critic.
DARwIn-OP	Dynamic Anthropomorphic Robot with Intelligence - Open Platform.
DDQN	Double Deep Q-Learning.
DL	Deep Learning - Aprendizado Profundo.
DNN	Deep Neural Network - Rede Neural Profunda.
DQN	Deep Q-Learning.
DRL	Deep Reinforcement Learning - Aprendizado por Reforço Profundo.
ILSVRC	Imagenet Large Scale Visual Recognition Challenge.
ReLU	Rectified Linear Unit.
RL	Reinforcement Learning - Aprendizado por Reforço.

SUMÁRIO

1 INTRODUÇÃO

Existem problemas que são difíceis de serem resolvidos pelo ser humano e extremamente fáceis de serem resolvidos por computadores, e que podem ser descritos formalmente através de um conjunto de regras matemáticas. Os problemas considerados difíceis para o computador resolver são os problemas que são difíceis de se descrever formalmente por regras matemáticas, sendo normalmente resolvidos de forma intuitiva pelo ser humano, tais como o reconhecimento de palavras através do som da voz ou o reconhecimento de faces em imagens (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para resolver esses problemas combinam-se diversos algoritmos, onde em cada algoritmo descreve formalmente o conhecimento que o computador precisa para executar a tarefa, formando uma hierarquia de conceitos, para que o computador possa executar tarefas complexas. Uma solução é permitir que os computadores construam essa hierarquia de conceitos através de aprendizado. Ao se desenhar um grafo mostrando como esses conceitos são construídos, o grafo será profundo e com muitas camadas. Por esta razão, chama-se esse tipo de abordagem de Aprendizado Profundo (GOODFELLOW; BENGIO; COURVILLE, 2016).

O Aprendizado Profundo é um ramo de aprendizado de máquina (ML do inglês *Machine Learning*), sendo um modelo composto de múltiplas camadas para aprender a representação de dados em múltiplos níveis de abstrações (LECUN; BENGIO; HINTON, G., 2015; DENG, L.; YU et al., 2014). No Aprendizado Profundo foram desenvolvidas técnicas e arquiteturas para aprendizado supervisionado, não supervisionado e aprendizado por reforço.

Existem vários fatores que possibilitaram o recente surgimento do Aprendizado Profundo, um deles é o progresso em paralelização do hardware e do software, possibilitando assim que a execução e o treinamento dos algoritmos de Aprendizado Profundo demandem menos tempo. O uso de novas técnicas tais como ReLUs e *Dropout* também reduziram o tempo de treinamento (LECUN; BENGIO; HINTON, G., 2015). As recentes arquiteturas possuem dezenas e até centenas de camadas de redes convolucionais e centenas de milhões de pesos, para isso o treinamento deve ser realizado com processamento paralelo, utilizando, por exemplo, placas de processamento gráfico (GPU - do inglês *Graphics Process Unit*). Essas mesmas arquiteturas também demandam alguns gigabytes de memória RAM durante o processo de treinamento.

Recentemente o Aprendizado Profundo tem atraído a atenção dos pesquisadores devido ao desempenho que vem apresentando em pesquisas de diversas áreas (KRIZHEVSKY; SUTSKEVER; HINTON, G. E., 2012; MNIH et al., 2015; SILVER et al., 2016; XIONG et al., 2016; HESSEL et al., 2018), surgindo com novos algoritmos e arquiteturas de rede como o atual es-

tado da arte de diversos problemas. Pesquisas em robótica (GIUSTI et al., 2016; SPECK et al., 2016; LEVINE et al., 2016; GU et al., 2017; YAHYA et al., 2017) também tem surgido com o uso de Aprendizado Profundo. No entanto, os robôs possuem diversos problemas relacionados a cognição (ERSEN; OZTOP; SARIEL, 2017).

Cognição Robótica é o processo em que o sistema autônomo percebe seu ambiente, aprende com as experiências, antecipa o resultado de eventos, age para atingir os objetivos e se adapta a circunstâncias mutáveis (VERNON, 2014). A robótica cognitiva é a fusão de duas áreas de pesquisas: agentes robóticos capazes de interagir com o ambiente físico sem restrições e arquiteturas de controle que adquirem e usam a experiência com o ambiente. Para alcançar os objetivos o robô deve perceber seu ambiente, prestar atenção aos eventos que são importantes, planejar o que fazer e aprender com as interações resultantes (IEEE. . . , 2019).

Apesar de estudos recentes apresentarem passos importantes no desenvolvimento de sistemas que possam atuar e aprender autonomamente (PERICO et al., 2014; KIM, S. et al., 2015; LIM et al., 2017; JUMEL et al., 2018), um dos problemas que também estão em abertos na área da robótica é a necessidade de realizar trabalhos com robôs em ambientes não laboratoriais (como por exemplo, carros autônomos que atuam diretamente nas ruas). A realização dessa implantação também exige que os robôs sejam equipados com habilidades avançadas de aprendizado (ERSEN; OZTOP; SARIEL, 2017).

Pesquisas recentes mostram que o Aprendizado Profundo tem a capacidade de aprender tanto as características de baixo nível quanto as de médio e alto nível, além de possuir um grau de generalização (GOODFELLOW; BENGIO; COURVILLE, 2016). Essas pesquisas também mostram que o Aprendizado Profundo tem superado diversas técnicas e algoritmos, apresentando novos algoritmos de Aprendizado Profundo (DL - do inglês *Deep Learning*) como o atual estado da arte de diversos problemas, inclusive na área da robótica. Por isso, uma arquitetura com Aprendizado Profundo poderá substituir diversos processos responsáveis pela cognição de um robô e elevar sua capacidade de cognição.

Neste trabalho serão apresentados experimentos que foram realizados no domínio do futebol de robôs humanoides utilizando Aprendizado Profundo. O futebol de robôs humanoides é uma das categorias da competição RoboCup (KITANO et al., 1997; KITANO; ASADA, 1998), uma competição anual realizada em nível mundial com o intuito de promover pesquisas em Robótica, Inteligência Artificial, Ciência da Computação e Engenharia (VELOSO; STONE, 2012). Nessas competições os robôs são testados em ambientes não laboratoriais, no entanto, um robô inserido em tal domínio tem várias limitações impostas pelas regras da competição:

computador embutido dentro do robô e a dinâmica do ambiente (a restrição em relação a dinâmica do ambiente é que o robô deve agir em um tempo compatível com os objetos e outros robôs a sua volta, ou seja, o robô não deve ser muito mais lento que os robôs adversários ou outros objetos que se movimentam pelo ambiente).

O grande desafio e meta da RoboCup é: "Na metade do século 21, uma equipe de jogadores de futebol de robôs humanoides totalmente autônomos deve ganhar o jogo de futebol, em conformidade com as regras oficiais da Federação Internacional de Futebol (FIFA, do francês: *Fédération Internationale de Football Association*), contra o vencedor mais recente da Copa do Mundo" (KITANO; ASADA, 1998). O jogo de futebol é um grande desafio por ser um ambiente dinâmico e que inclui o desenvolvimento e implementação de várias tecnologias (VELOSO; STONE, 2012). Atualmente a RoboCup é constituída por competições e simpósio.

Na categoria dos robôs humanoides da RoboCup, os robôs devem ser semelhantes aos seres humanos, precisam ser construídos com dispositivos e preceitos análogos a estrutura física do corpo humano. Portanto, não é permitido utilizar alguns sensores, tais como sonar ou laser, sendo que para visão do robô poderá ser utilizada uma ou duas câmeras fixadas em sua cabeça. Também não é permitido utilizar processamento externo, nem enviar mensagens ao robô de dispositivos externos.

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver sistemas cognitivos baseados em Aprendizado Profundo, no intuito de proporcionar a agentes robóticos a capacidade de tomar decisões pela observação do ambiente. Esse sistema cognitivo substitui diversos processos responsáveis por diversas tarefas utilizadas na cognição de agentes robóticos, assim diminuindo a complexidade que as atuais arquiteturas possuem.

1.2 JUSTIFICATIVA

Apesar de estudos recentes (KIM, S. et al., 2015; LIM et al., 2017; JUMEL et al., 2018; PERICO et al., 2014) apresentarem importantes progressos no desenvolvimento de robôs que possam atuar autonomamente, esses robôs possuem uma arquitetura de software composta por diversos processos responsáveis por atividades especificas, tais como: detecção de objetos, sistemas de localização, *path planner*, processo de decisão. Normalmente, esses processos ope-

ram em conjunto e/ou compartilham informações para realizar algumas determinadas tarefas. No entanto, o problema é que todos esses processos devem ser desenvolvidos e configurados pelo próprio pesquisador, e em certos casos não atingem a solução ótima para a tarefa a ser solucionada. Muitas configurações são realizadas manualmente pelo pesquisador e alguns desses processos precisam operar sincronizados com outros processos, assim apresentando uma arquitetura complexa e de difícil configuração para a realização de uma simples tarefa.

Uma solução para substituir ou diminuir a complexidade desse tipo de sistema é a utilização de um sistema cognitivo com Redes Neurais Profundas, ou Aprendizado por Reforço Profundo, para que o próprio robô aprenda a tarefa que lhe é atribuída. No entanto, ainda existe a dificuldade de se aplicar algoritmos de Aprendizado por Reforço Profundo em robôs reais, porque a quantidade de passos necessários para que o algoritmo obtenha um aprendizado satisfatório faz com que seja, em muitos casos, inviável realizar em robôs reais. Por isso, uma alternativa é realizar o aprendizado em simulação e a execução no robô real.

1.3 CONTRIBUIÇÕES

As principais contribuições deste trabalho são: um sistema cognitivo com Redes Neurais Profundas, e um sistema cognitivo com Aprendizado por Reforço Profundo para um robô autônomo móvel. Para observar o ambiente os sistemas usam apenas a imagem da câmera do robô sem qualquer pré-processamento na imagem, e em ambos os sistemas o robô é capaz de realizar tarefas relacionadas ao papel de um robô jogador de futebol.

Outra contribuição que este trabalho apresenta no sistema cognitivo com Redes Neurais Profundas é o desenvolvimento de Árvore de Decisão de Redes Neurais Profundas (DNN do inglês *Deep Neural Network*). Com a árvore de decisão de DNNs foi possível diminuir o tempo de inferência e aumentar a acurácia. Para diminuir o tempo de inferência de uma DNN, os pesquisadores buscam desenvolver arquiteturas que possam diminuir a quantidade de neurônios da rede, no entanto, essa diminuição causa também uma diminuição da taxa de acerto da rede. Esta tese mostra que uma arquitetura de DNN com árvore de decisão, permite que a quantidade de neurônios da rede seja diminuída, e mesmo diminuindo a quantidade de neurônios, essa arquitetura aumenta a acurácia da rede em comparação com as atuais DNNs. Experimentos com várias arquiteturas de redes foram realizados para comparar a taxa de acerto e o tempo de inferência entre as DNNs e também entre a árvore de decisões de DNNs. Foram testadas algumas redes conhecidas pela comunidade cientifica tais como: AlexNet (KRIZHEVSKY;

SUTSKEVER; HINTON, G. E., 2012); GoogLeNet (SZEGEDY et al., 2015); SqueezeNet (HU; SHEN; SUN, 2018); e também variações da arquitetura da AlexNet, bem como uma arquitetura própria desenvolvida para a árvore de decisões de DNNs.

Outra contribuição que este trabalho apresenta no sistema cognitivo com Aprendizado por Reforço Profundo é a possibilidade de realizar um aprendizado no simulador e executá-lo no robô real. Mesmo no simulador o processo de aprendizado gasta dias ou semanas (no simulador a simulação é realizada de forma acelerada) para apresentar um comportamento satisfatório. Porém, no robô real torna-se inviável realizar esse processo de aprendizado, então esse trabalho investiga e demonstra o processo de aprendizado sendo realizado no simulador, e depois de aprendido, a rede é utilizada no robô real.

1.4 ORGANIZAÇÃO DA TESE

Tendo-se esclarecido alguns pontos relacionados à aprendizagem profunda, robótica cognitiva, a contribuição e o objetivo deste trabalho, segue-se a explanação a respeito da estrutura deste trabalho. A seção 2 apresenta uma revisão bibliográfica que aborda: Aprendizado Profundo; Aprendizado por Reforço; e Aprendizado por Reforço Profundo. Contendo as definições dos principais algoritmos que surgiram recentemente e também os algoritmos que foram utilizados como base para o desenvolvimento dos algoritmos de Aprendizado Profundo. A seção 3 apresenta alguns trabalhos com Redes Neurais Profundas realizadas em robôs móveis e Aprendizado por Reforço Profundo em jogos 3D e em robôs. A seção 4 apresenta os Sistemas Cognitivo para Agentes Robóticos Baseados em Aprendizado Profundo: apresenta a definição do problema; o agente robótico utilizado nesta pesquisa; o sistema cognitivo com Redes Neurais Profundas; e o sistema cognitivo com Aprendizado por Reforço Profundo. A seção 5 apresenta os experimentos e resultados com DNN que foram realizados, sendo dividido em quatro seções (seções 5.1, 5.2, 5.3 e 5.4): experimentos realizados no simulador; experimentos realizados no robô real; experimentos com a árvore de decisão de Redes Neurais Profundas; e uma seção de discussão. A seção 6 apresenta os experimentos e resultados que foram realizados com Aprendizado por Reforço Profundo, sendo dividido em três seções: Seção 6.1 com título Robô humanoide aprendendo o papel de um robô goleiro; Seção 6.2 com título Robô humanoide aprendendo a chutar pênaltis; e a transferência dos pesos da rede treinada no simulador e executada no robô real. A seção 7 apresenta a conclusão do trabalho.

2 REVISÃO TEÓRICA

Essa seção está dividida em três seções secundárias: Aprendizado Profundo, aprendizado por reforço e Aprendizado por Reforço Profundo. Na seção 2.1 de Aprendizado Profundo contém as definições de Aprendizado Profundo bem como as principais topologias de Redes Neurais Profundas que surgiram recentemente. Na seção 2.2 de aprendizado por reforço será apresentada uma breve descrição dos algoritmos que foram utilizados como base para o desenvolvimento dos algoritmos de Aprendizado por Reforço Profundo. Na seção 2.3 de Aprendizado por Reforço Profundo será apresentado os principais e mais recentes algoritmos.

2.1 APRENDIZADO PROFUNDO

Essa seção apresentará uma breve descrição de Aprendizado Profundo, no entanto, serão descritos apenas algumas técnicas tais como redes de convolução e redes de neurônios completamente conectados. Isso porque essas redes são empregadas em Aprendizado por Reforço Profundo como também em Redes Neurais ProfundaS.

O Aprendizado Profundo é um conjunto de técnicas de aprendizado de máquina, sendo um modelo composto de múltiplas camadas para aprender a representação de dados em múltiplos níveis de abstrações, onde características de alto-nível são definidas a partir de características de baixo-nível (LECUN; BENGIO; HINTON, G., 2015; DENG, L.; YU et al., 2014).

DL apresentou excelentes avanços no estado da arte em reconhecimento de voz (XIONG et al., 2016), reconhecimento de face (PARKHI; VEDALDI; ZISSERMAN et al., 2015), detecção de objetos e diversas outras áreas de pesquisa (LECUN; BENGIO; HINTON, G., 2015).

Alguns elementos que diferencia o Aprendizado Profundo das outras técnicas de Aprendizado de Máquina são: a quantidade de dados usados nos treinamentos; DL evita a necessidade de operadores humanos especifiquem formalmente todo o conhecimento de que o computador precisa, a aprendizagem profunda escolhe quais características são úteis para a aprendizagem; a maioria das arquiteturas são construídas com um método de camada-por-camada, onde características de alto-nível são definidas a partir de características de baixo nível nessas camadas.

Existem alguns fatores que contribuíram para o surgimento de diversos algoritmos de Aprendizado Profundo nesta última década, tais como: o surgimento de conjunto de dados cada vez maiores, devido a necessidade de analisar com cada vez mais dados, dados capturados de sensores, redes sociais, web, arquivos de *log*, etc (a área de *Big Data* trata da análise em

grandes volumes de dados); o surgimento de hardwares capazes de processar um grande volume de dados, como processadores com vários núcleos que realizam processamento paralelo, e GPUs com cada vez mais núcleos de processamento; o interesse de grandes empresas em desenvolvimento de DL, tais como: Google, Intel, NVIDIA, IBM; o surgimento de linguagens de programação de alto nível, tais como Python, e frameworks específicos para Aprendizado Profundo, tais como: Tensorflow, Keras, Caffe, CNTK, Theano.

No Aprendizado Profundo foram desenvolvidas técnicas e arquiteturas para aprendizado supervisionado, não supervisionado e aprendizado por reforço. Algumas das técnicas de Aprendizado Profundo não supervisionado são: *Deep Belief network* (DBN) (HINTON, G. E.; OSINDERO; TEH, 2006; HINTON, G. E.; SALAKHUTDINOV, 2006; DENG, L.; YU et al., 2014; GOODFELLOW; BENGIO; COURVILLE, 2016); *Deep Boltzmann Machines* (DBM) (DENG, L.; YU et al., 2014; GOODFELLOW; BENGIO; COURVILLE, 2016); *Restricted Boltzmann machine* (RBM) (DENG, L.; YU et al., 2014; GOODFELLOW; BENGIO; COURVILLE, 2016); *Deep Autoencoders* (DENG, L.; YU et al., 2014; GOODFELLOW; BENGIO; COURVILLE, 2016). Essas técnicas também são utilizadas em modelos híbridos.

Em Aprendizado por Reforço Profundo pesquisadores da Google *DeepMind* desenvolveram um algoritmo chamado *Deep Q-Learning* (DQN) (MNIH et al., 2013, 2015). Eles demostraram que o DQN é capaz de aprender a jogar os jogos de Atari. A partir do DQN outros algoritmos foram surgindo, tais como: *Gorila* DQN (NAIR, A. et al., 2015); *Double Deep Q-Learning* (DDQN) (VAN HASSELT; GUEZ; SILVER, 2015); *Deep Q-Learning with Prioritized Experience Replay* (SCHAUL et al., 2015); *Dueling Network Architectures* para Aprendizado por Reforço Profundo (WANG, Z.; FREITAS; LANCTOT, 2016); *Deep Deterministic Policy Gradient* (DDPG) (LILLICRAP et al., 2015); *Continuous DQN* (CDQN) (GU et al., 2016); *Asynchronous Advantage Actor-Critic* (A3C) (MNIH et al., 2016).

Em aprendizado supervisionado existem algumas Redes Neurais Profundas que são conhecidas pelo bom desempenho apresentado, tais como: ALexNet (KRIZHEVSKY; SUTSKEVER; HINTON, G. E., 2012); ZFNet (ZEILER; FERGUS, 2014); VGGNet (SIMONYAN; ZISSERMAN, 2014); GoogLeNet (SZEGEDY et al., 2015); ResNet (HE, K. et al., 2016); SENet (Squeeze-and-Excitation Networks) (HU; SHEN; SUN, 2018).

Desde 2012 surgiram essas Redes Neurais Profundas que vêm sendo aplicadas na competição de classificação e reconhecimento de padrões ILSVRC (ILSVRC - Imagenet Large Scale Visual Recognition Challenge) apresentando resultados superiores a qualquer outro tipo de técnica ou algoritmos (LECUN; BENGIO; HINTON, G., 2015). A competição ILSVCR

(IMAGENET..., 2017) foi criada para fomentar a pesquisa em algoritmos de detecção de objetos em imagens.

Em 2012 a equipe SuperVision da Universidade de Toronto empregou uma rede neural profunda chamada AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, G. E., 2012) (rede neural profunda de 8 camadas) na competição ILSVCR, onde a rede neural profunda (do inglês Deep Neural Network - DNN) foi aplicada em um conjunto de mais de 1 milhão de imagens em 1000 diferentes classes (LECUN; BENGIO; HINTON, G., 2015). A AlexNet ficou conhecida naquele ano por apresentar o melhor desempenho na competição e sendo a equipe vencedora alcançando uma taxa de erro de *top-5* de 0.16, substancialmente menor do que o segundo colocado. O erro de classificação *top-5* é uma medida que considera como acertos, se a correta classe da imagem estiver dentre as 5 primeiras classes classificada pela rede. A rede AlexNet obteve uma taxa de erro *top-5* de 0.15315 enquanto o segundo melhor time o ISI apresentou um erro *top-5* de 0.26172 (RESULTADOS..., 2017). Segundo LeCun, Bengio e Geoffrey Hinton (2015) esse desempenho das DNN tornou-se possível com o uso de novas técnicas tais como: ReLUs, Dropout, e também realizando-se o treinamento em placas de processamento gráfico (GPU - do inglês Graphics Process Unit).

Em 2013 a *startup* Clarifai ganhou a competição de classificação ILSVRC de 2013 apresentando a rede neural profunda ZFNet (ZEILER; FERGUS, 2014). Em 2014 pesquisadores do Google DeepMind apresentaram uma rede neural profunda chamada GoogLeNet vencendo a competição de classificação ILSVRC de 2014. GoogLeNet é uma rede neural de 22 camadas proposta por Szegedy et al. (2015). Essa rede foi o novo estado da arte do ILSVRC de 2014, uma das principais características apresentada foi o aumento da largura e profundidade da rede, melhoria na utilização dos recursos computacionais usados pela rede e desenvolvimento das camadas *inception*.

Pela Tabela 1 pode-se ver que as Redes Neurais Profundas superaram o ser humano, de acordo com o artigo do (RUSSAKOVSKY et al., 2015), o erro top-5 do ser humano foi de 5.1%. A Tabela 1 mostra que o erro de classificação *top-5* na competição ILSVRC diminuiu a cada ano, esta tabela também mostra que todos os times campeões da competição de classificação utilizaram Redes Neurais Profundas. A Figura 1 mostra algumas imagens que foram classificadas pela rede AlexNet usando a classificação *top-5*.

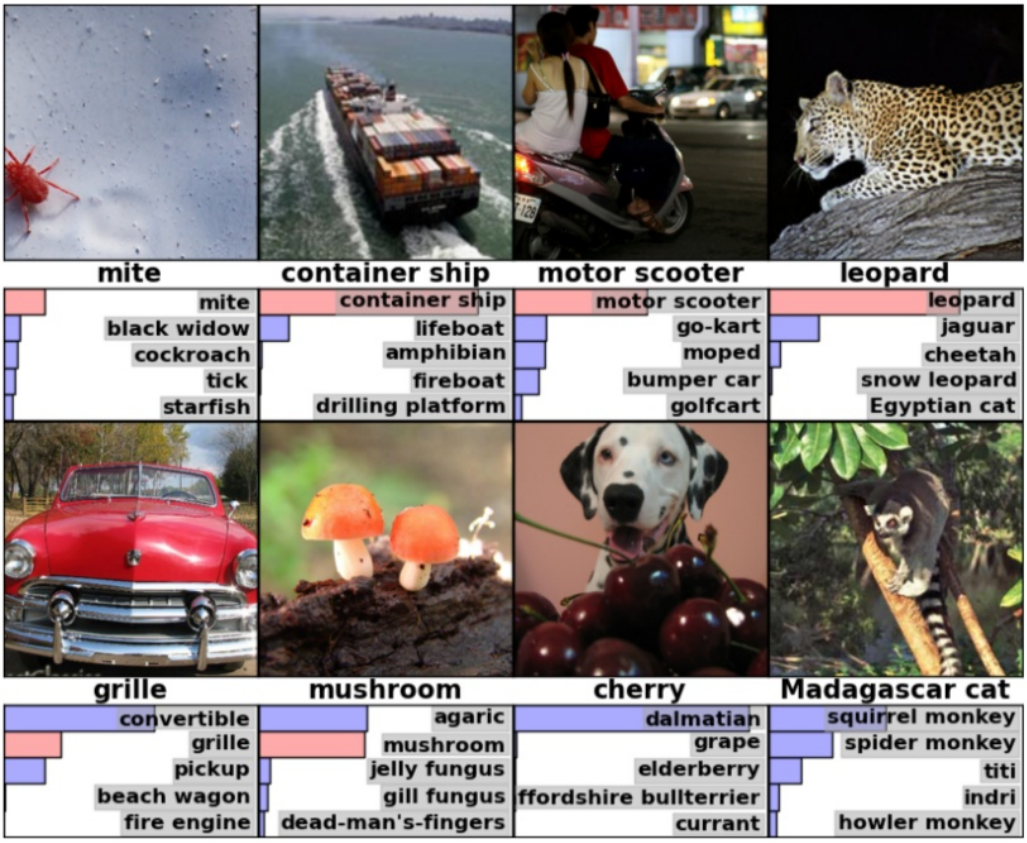
Em 2009, Fei-Fei Li, professora de IA em Stanford na Califórnia, lançou o ImageNet (DENG, J. et al., 2009). A ImageNet 22k é um conjunto de imagens que contém 15 milhões de imagens de 22000 categorias. Para a competição ILSVRC utiliza-se um subconjunto dessas

Tabela 1 – Desempenho das Redes Neurais Profundas na competição ILSVRC.

DNN	Time	Ano	Colocação	Erro (*top-5*)	Uso de dados externos
AlexNet	SuperVision	2012	1	16.4%	não
AlexNet	SuperVision	2012	1	15.3%	Imagenet 22k
ZFNet	Clarifai	2013	1	11.7%	não
ZFNet	Clarifai	2013	1	11.2%	Imagenet 22k
VGGNet	VGG	2014	2	7.32%	não
GoogLeNet	GoogLeNet	2014	1	6.67%	não
ResNet	MSRA	2015	1	3.57%	não
-	Trimps-Soushen	2016	1	2.99%	não
SENet	WMW	2017	1	2.25%	não

Fonte: Autor.

Figura 1 – Imagens classificadas pela rede AlexNet.



Fonte: KRIZHEVSKY; SUTSKEVER; HINTON, G. E., 2012

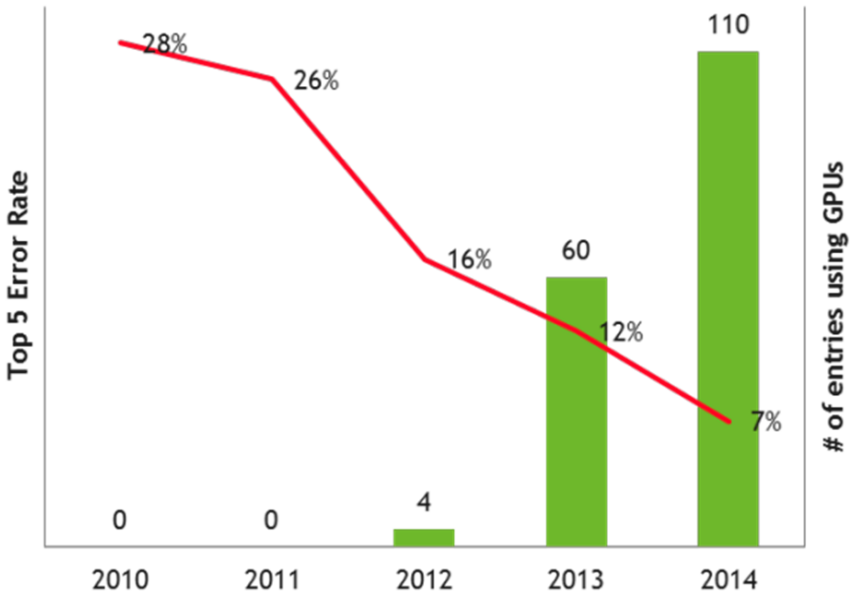
imagens contendo 1.2 milhões de imagens em 1000 categorias. A Figura 2 apresenta duas imagens contidas no subconjunto de 1.2 milhões de imagens da ImageNet referente as duas classes distintas.

Figura 2 – Imagens de classes com características parecidas.

(a) Husky siberiano. (b) Cão Esquimó.

Fonte: SZEGEDY et al., 2015.

Figura 3 – Gráfico do uso de GPU no ILSVRC pelos times participantes.



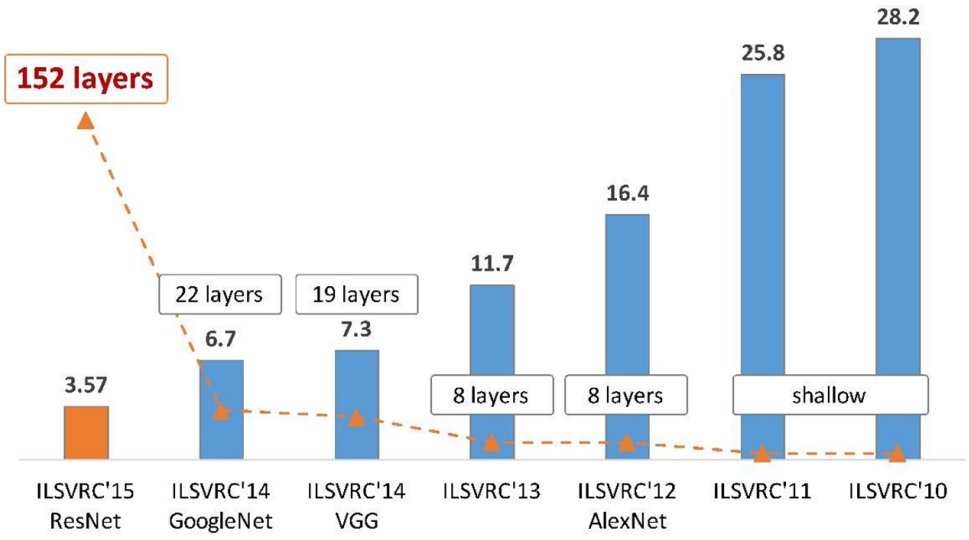
Fonte: , 2017

A Figura 3 apresenta um gráfico disponibilizado pela NVIDIA mostrando que a partir de 2012 os grupos que participaram do desafio ILSVRC passaram a utilizar GPU. Em 2010 a equipe vencedora usou Máquina de Vetores de Suporte (SVM do inglês *Support Vector Machine*), enquanto em 2011 a equipe vencedora empregou uma abordagem comprimida do vetor Fisher (NVIDIA. . . , 2017), sendo que essas equipes não utilizaram GPU. As atuais arquiteturas de DNN possuem dezenas de camadas, centenas de milhões de pesos, com isso a apenas uma década atrás os treinamentos gastariam semanas ou meses para serem realizados, porém com a

paralelização dos hardwares e os softwares os tempos de treinamento reduziram para algumas horas.

A Figura 4 mostra que em 2010 e 2011 as equipes que venceram a competição ILSVRC, não utilizavam técnicas de Aprendizado Profundo. No entanto, em 2012 e 2013 as equipes que venceram a competição utilizaram Redes Neurais Profundas com 8 camadas. Nos anos seguintes as redes passaram a possuir cada vez mais camadas, sendo que em 2015 a equipe vencedora apresentou uma rede neural profunda de 152 camadas chamada ResNet.

Figura 4 – Número de camadas versus o erro top-5.

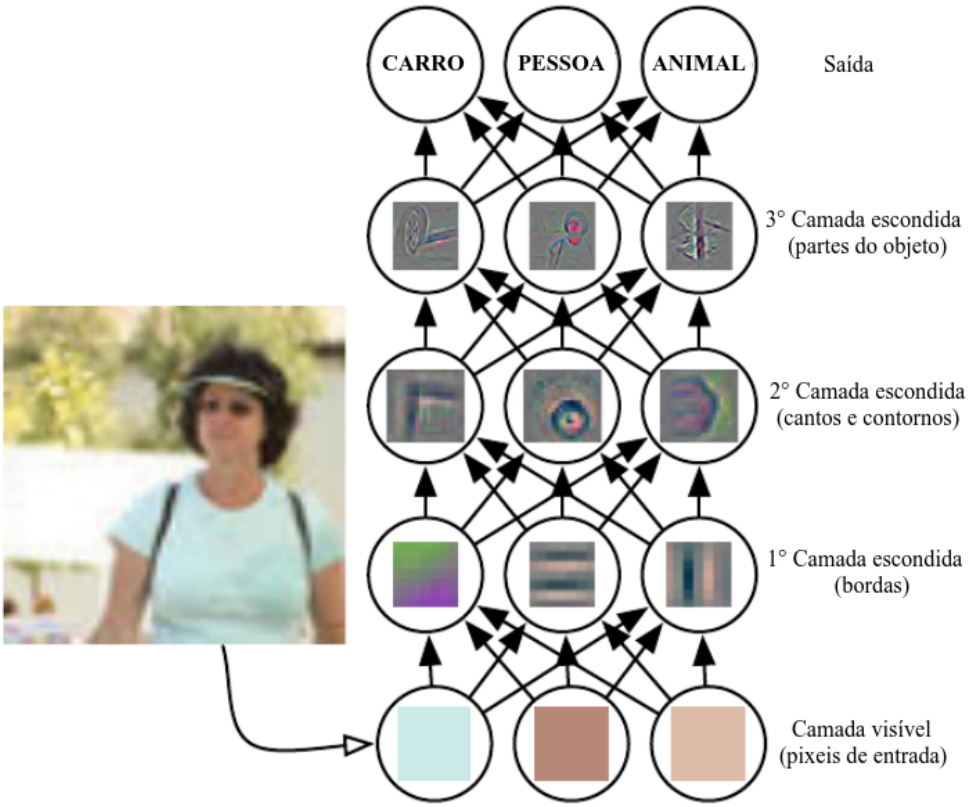


Fonte: HE, K. et al., 2017

Devido ao desempenho que vem sendo apresentado nas pesquisas, o Aprendizado Profundo passou a ser aplicado por grandes empresas como a Google, IBM, Microsoft, Baidu, NVIDIA, Qualcomm, Intel, Twitter, Adobe. Também vem surgindo novas empresas (*start-ups*) para pesquisa e desenvolvimento de projetos em Aprendizado Profundo (LECUN; BENGIO; HINTON, G., 2015). Sendo que em 2014 a rede que apresentou o melhor desempenho na competição ILSVRC14 foi a rede neural profunda da Google (GoogLeNet) (SZEGEDY et al., 2015), e em 2015 a rede que apresentou o melhor desempenho na competição ILSVRC15 foi a rede neural profunda da Microsoft (ResNet) (SZEGEDY et al., 2015).

O conceito de Aprendizado Profundo originou-se de pesquisas em redes neurais com muitas camadas escondidas, no entanto, utilizar redes de múltiplas camadas com somente neurônios *perceptrons*, não funcionam bem quando utiliza-se muitas camadas escondidas (DENG, L.; YU et al., 2014).

Figura 5 – Modelo característico de DL.



Fonte: GOODFELLOW; BENGIO; COURVILLE, 2016

A Figura 5 representa o conceito de detecção de imagens realizado por um método de DL, demonstrando que o DL é capaz de aprender diversas características relacionada as imagens de entrada, onde cada camada é responsável por aprender uma determinada característica das imagens, sendo que cada camada também possui conjuntos em paralelo, assim aprendendo diversas características da imagem (GOODFELLOW; BENGIO; COURVILLE, 2016).

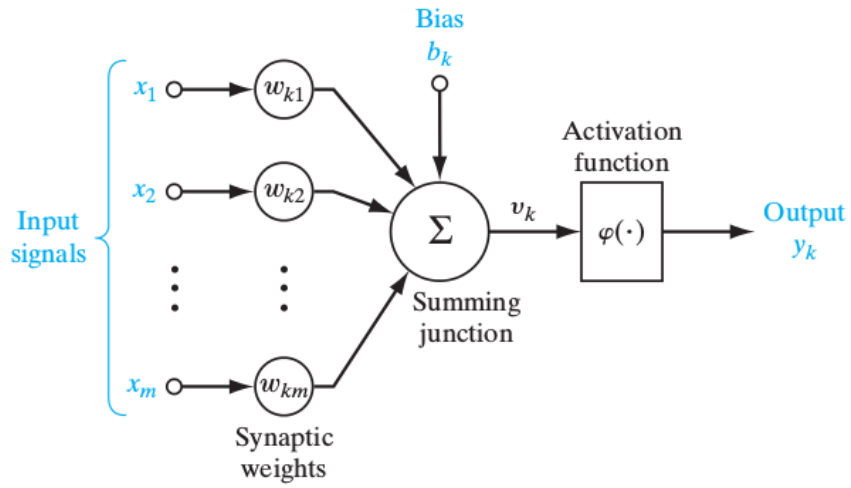
Os algoritmos que serão apresentados neste trabalho possuem redes neurais convolucionais e redes completamente conectadas. Nas camadas convolucionais, aplica-se também a operação *pooling*. Nas redes completamente conectadas aplica-se a técnica do *Dropout* durante o treinamento. O treinamento dessas redes pode ser realizado através do método de retro-propagação (*back-propagation*), no entanto a retro-propagação refere-se apenas a forma como os pesos serão atualizados, durante a retro-propagação pode-se utilizar diversos algoritmos tais como gradiente descendente estocástico. As próximas seções apresentarão as definições teóricas das redes completamente conectadas e redes neurais convolucionais, como também as operações e técnicas aplicadas nessas redes.

2.1.1 Redes Completamente Conectadas

Redes neurais artificiais (RNAs) são inspirados no cérebro humano. As RNA são capazes de realizar o aprendizado para o reconhecimento de diversos padrões, sendo possível armazenar uma base de conhecimento adquirido pela rede através do processo de treinamento. As conexões entre os neurônios são realizadas pelos pesos sinápticos (w_{Kj}) que armazenam o conhecimento adquirido (HAYKIN, 2009). O neurônio artificial que atualmente são usados para construir as redes neurais são modelos simplificados se comparado com os do cérebro humano.

A imagem da Figura 6 apresenta um modelo de neurônio artificial. Pode-se ver que o neurônio recebe as entradas x_j e multiplica-se pelos pelos w_kj. Depois realiza-se o somatório dessas multiplicações resultando no valor u_k conforme Equação (1). Então realiza-se a soma do valor u_k com o viés b_k resultando no valor v_k conforme Equação (2). Por fim, o valor de v_k passa por uma função de ativação $\varphi(.)$ resultando no valor de saída do neurônio y_k conforme Equação (3).

Figura 6 – Modelo de um neurônio artificial.



Fonte: HAYKIN, 2009

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

$$v_k = u_k + b_k \tag{2}$$

$$y_k = \varphi(v_k) \tag{3}$$

Onde: x_j são os valores de entrada ($x_1, x_2, \ldots, x_m$); w_{Kj} são os pesos sinápticos do neurônio k ($w_{k1}, w_{k2}, \ldots, x_{km}$); b_k é o valor do viés (conhecido como *bias* em inglês); $\varphi(.)$ é uma função de ativação; y_k é o valor de saída do neurônio k.

As funções de ativação sigmoide e tangente hiperbólica são as mais convencionais (Equação (5) da sigmoide e Equação (6) da tangente hiperbólica), no entanto, atualmente em DL a função de ativação mais usada é a função ReLU (Equação (4)) (LECUN; BENGIO; HINTON, G., 2015).

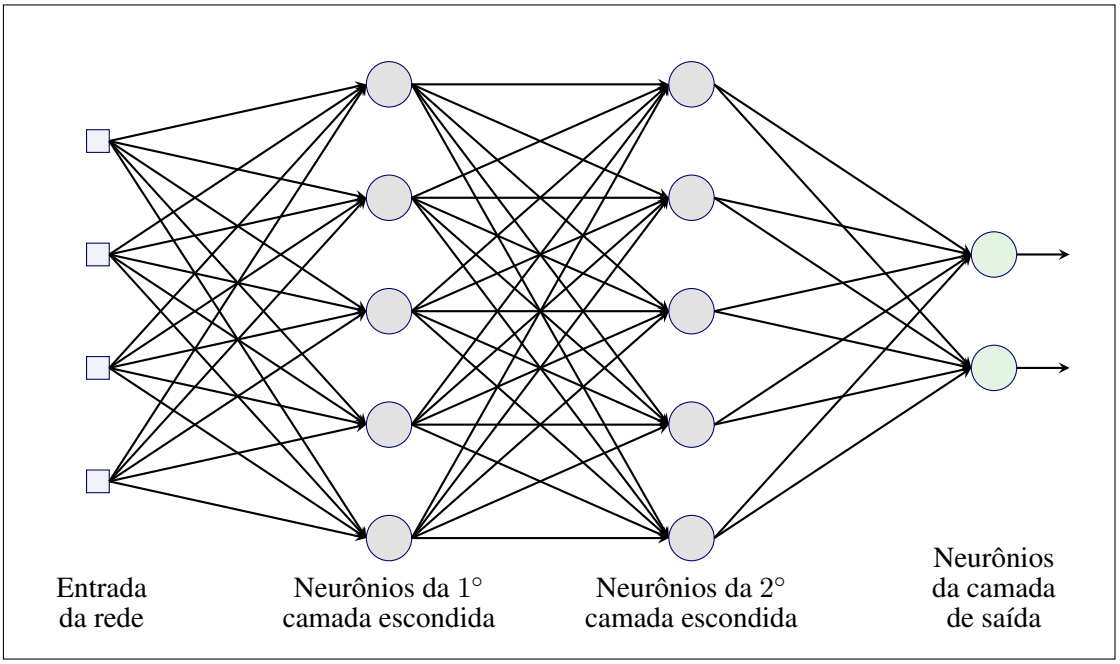
$$\varphi(v) = max(0,v) = \begin{cases} v & \text{se } v \geq 0 \\ 0 & \text{caso contrário} \end{cases} \tag{4}$$

$$\varphi(v) = \frac{1}{1 + \mathrm{e}^{-av}}, \qquad a > 0 \tag{5}$$

$$\varphi(v) = a * tanh(b * v), \qquad (a,b) > 0 \tag{6}$$

Onde o valor de a das Equações (5) e (6) e o valor de b da Equação (6) são parâmetros de ajustes das funções.

Figura 7 – Exemplo de rede neural multicamada.



Fonte: Adaptado de HAYKIN, 2009.

A imagem da Figura 7 apresenta um exemplo de uma rede neural de múltiplas camadas, nesse exemplo há duas camadas escondidas e uma camada de saída. Essa rede neural é

nomeada de completamente conectada, isso porque os neurônios de cada camada são completamente conectados com os neurônios da camada seguinte. Quando alguns neurônios não estão conectados com todos os neurônios da outra camada, nomeia-se essa rede como parcialmente conectada (HAYKIN, 2009). As redes completamente conectadas (também conhecidas como *Multi Layer Perceptron* - MLP) podem resolver problemas não lineares, ou seja, essas redes podem aprender operações lógicas mais complicadas, para isso, deve-se adicionar camadas e aumentar o número de neurônios por camada, assim, terá a capacidade de expressar funções mais complicadas.

2.1.2 Redes Convolucionais

As redes convolucionais foram desenvolvidas por (LECUN et al.)(LECUN et al., 1989) em 1989, já nos anos 90 os pesquisadores dos laboratórios da AT&T desenvolveram um sistema com redes convolucionais para a leitura de cheques bancários. Sendo pela primeira vez lançado comercialmente em 1993, rodando em caixas eletrônicos na Europa e nos EUA para leitura de cheques. Isso também motivou a Microsoft a desenvolver os OCRs (OCR é um acrônimo para o inglês *Optical Character Recognition*) para reconhecimento de escrita (LECUN; KAVUKCUOGLU; FARABET, 2010).

As redes neurais convolucionais profundas se popularizaram a partir de 2012 quando a equipe SuperVision da Universidade de Toronto venceu o desafio da ImageNet de 2012 com a Rede neural Profunda AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, G. E., 2012). A partir desta data, os grupos de pesquisas foram incrementando o número de camadas de redes convolucionais afim de obter uma maior acurácia, no entanto foram desenvolvidas redes mais eficientes em relação a acurácia e tempo de inferência.

A entrada da rede convolucional poderá ser um vetor de 1 dimensão para dados de áudio, poderá ser uma matriz de 2 dimensões para dados de imagens, e 3 dimensões para dados de vídeo (LECUN; KAVUKCUOGLU; FARABET, 2010). Nesta tese serão utilizadas matrizes de duas dimensões para dados de imagem, portanto a entrada da rede será uma matriz de duas dimensões para cada canal de cor da imagem.

No método de convolução aplicam-se vários filtros na imagem para extrair características, esses filtros são chamados de *kernels*. Em processamento de imagens esses filtros também são chamados de máscara, janela ou matriz de convolução (GONZALEZ; WOODS, 2007). Os

filtros são matrizes, onde os valores dessas matrizes são responsáveis por extrair as características da imagem de entrada. Nessa Tese os filtros serão chamados de *kernels*.

Uma típica camada de uma rede convolucional possui três estágios: No primeiro estágio realiza-se várias convoluções em paralelo produzindo um conjunto de ativações. No segundo estágio cada ativação passa por uma função de ativação tais como o ReLU. No terceiro estágio realiza-se a função *pooling*, tornando a camada invariante a pequenas translações (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para realizar o treinamento supervisionado nas redes convolucionais costuma-se utilizar o gradiente descendente estocástico (SGD do inglês *Stochastic gradient descent*) que minimiza a diferença entre a saída desejada e a atual saída da rede, onde todos os *kernels* das camadas são atualizados através do método *back-propagation* (LECUN; KAVUKCUOGLU; FARABET, 2010).

Figura 8 – Exemplo de imagem e *kernel*.

Imagem

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

Kernel

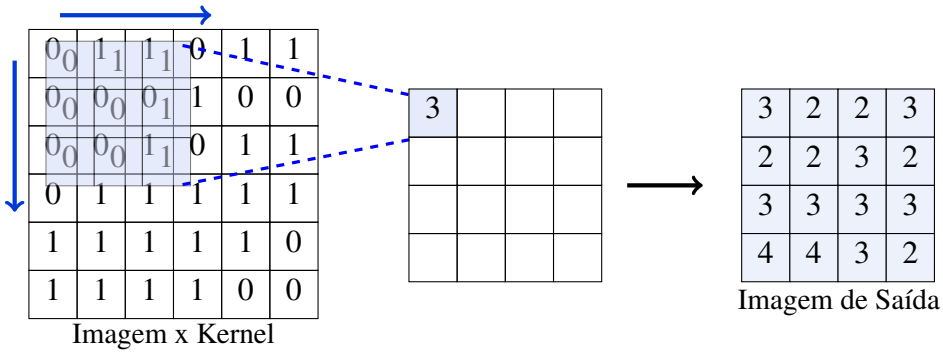
0	1	1
0	0	1
0	0	1

Fonte: Autor.

A imagem da Figura 8 apresenta um exemplo contendo uma matriz de uma imagem binária de 1 canal e também um *kernel*, e a Figura 9 mostra a convolução sendo aplicada na imagem. Para realizar a convolução, multiplicam-se os elementos do *kernel* com os pixels da imagem e somam-se esses valores, esse procedimento é realizado em toda a imagem deslizando o *kernel* pela imagem, assim resultando em uma nova imagem, porém uma imagem com características extraídas pelo filtro. O que determina que tipo de característica será extraído da imagem são: os valores do *kernel*, o tamanho do *kernel*, como também o passo do deslizamento do *kernel* na imagem (*stride*).

A Figura 10 apresenta uma imagem de entrada com 3 canais (imagem X), nesse caso multiplica-se o *kernel* com a imagem de cada canal, assim como realizado no exemplo da Figura 9, e realiza-se a soma entre as imagens geradas de cada canal, gerando apenas uma imagem de

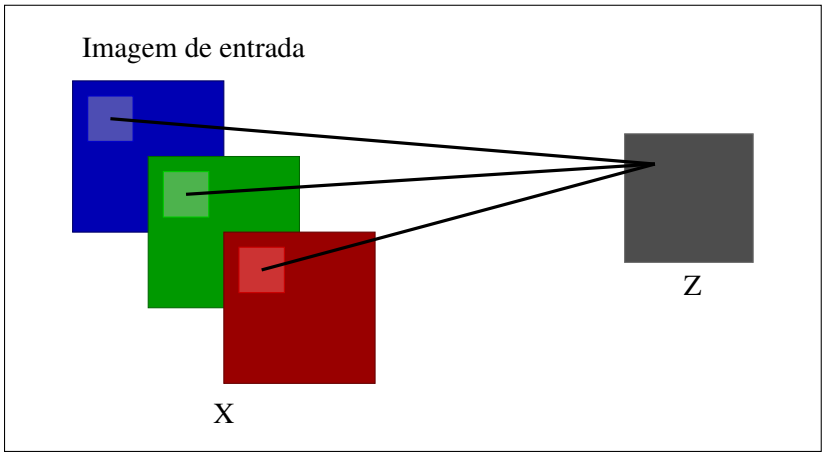
Figura 9 – Exemplo de convolução.



Fonte: Autor.

saída (imagem Z). No entanto, para se extrair diversas características da imagem, utiliza-se vários *kernels* gerando várias imagens de saída como mostra o exemplo da Figura 11, onde há 20 *kernels* para extração de 20 características da imagem, assim gerando 20 imagens de saída. Então, se imagem de entrada possuir 3 canais, haverá uma matriz W (matriz de pesos do *kernel*) para cada canal, ou seja, para 20 *kernels* haverá 60 matrizes de pesos W. Em DL utiliza-se vários *kernels* nas camadas de convolução, para que cada camada possa extrair vários tipos de características das imagens (GOODFELLOW; BENGIO; COURVILLE, 2016).

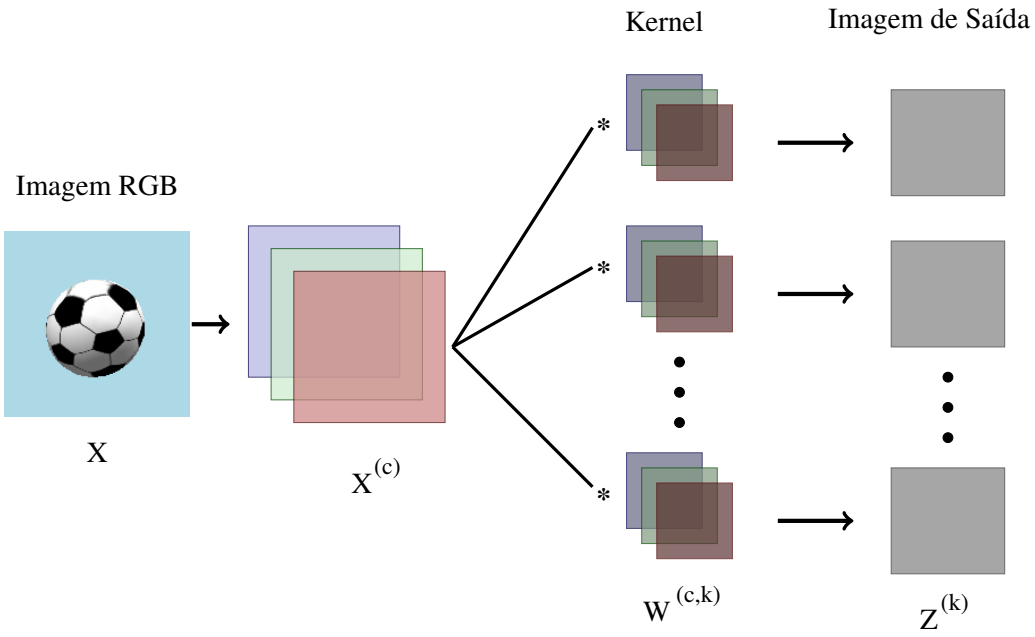
Figura 10 – Exemplo de convolução em uma imagem de 3 canais.



Fonte: Autor.

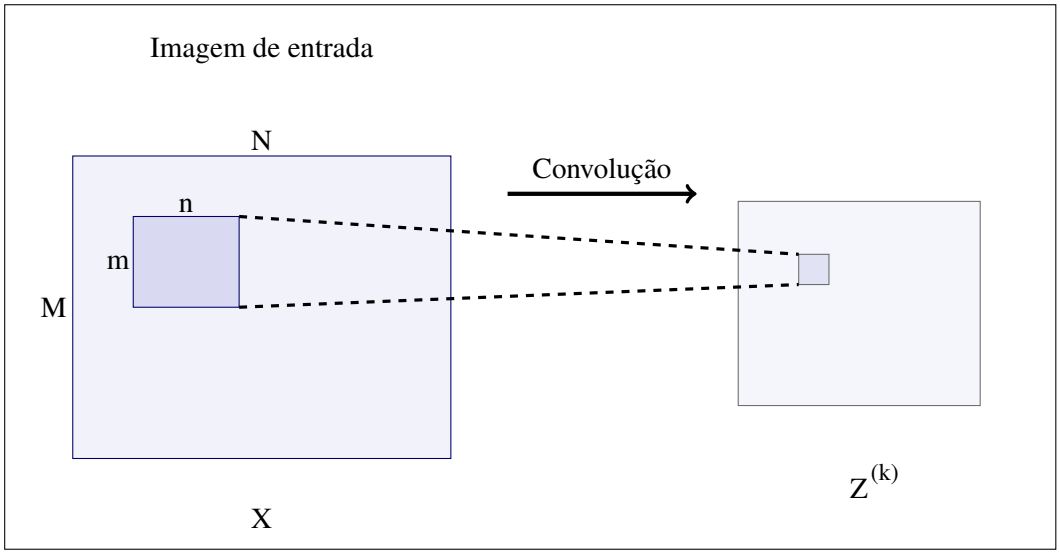
A imagem da Figura 12 também mostra a realização de uma convolução em um quadro de imagem e em apenas um canal, esta imagem mostra as variáveis referente ao tamanho da imagem e do *kernel*. Utilizando-se essas variáveis, a Equação (7) descreve a Equação da convolução.

Figura 11 – Exemplo de convolução em uma imagem de 3 canais e 20 *kernels*.



Fonte: Autor.

Figura 12 – Convolução na imagem.



Fonte: Autor.

$$z_{ij}^{(k)} = \sum_{c} \sum_{s=0}^{m-1} \sum_{t=0}^{n-1} w_{st}^{(k,c)} x_{(i+s)(j+t)}^{(c)} \tag{7}$$

Onde: X é matriz da imagem de entrada, sendo x um elemento da matriz X; Z é matriz da imagem de saída gerada pela convolução, sendo z um elemento da matriz Z; w é o peso do *kernel* ($w \in W$); k é o *kernel*; c é o canal da imagem; i é o índice que percorre a linha (altura da imagem) da matriz da imagem; j é o índice que percorre a coluna (largura da imagem) da matriz

da imagem; s é o índice que percorre em linha a matriz do *kernel*; t é o índice que percorre em coluna a matriz do *kernel*; m é a largura do *kernel*; n é a altura do *kernel*. Considerando uma convolução sem o *zero-padding*, o índice i percorre a imagem até $M - m + 1$ e o índice j percorre a imagem até $N - n + 1$. O passo que o índice i e j percorre na imagem é chamado de *stride* (S).

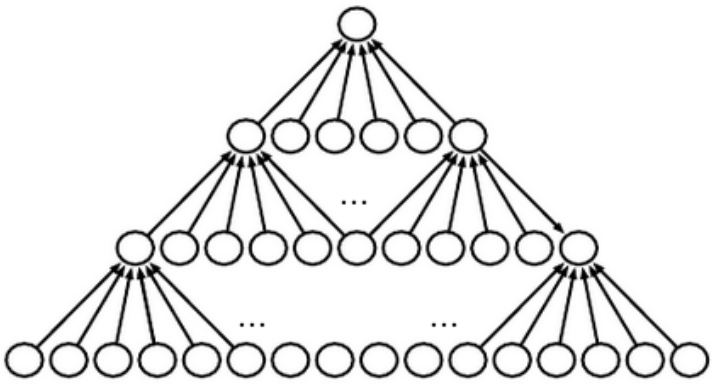
A entrada das redes convolucionais é uma matriz de pixels da imagem contendo o tamanho de $M \cdot N \cdot C$, onde: N é a largura da imagem; M é a altura da imagem; e C é o número de canais que a imagem possui.

$$\begin{aligned} M' &= (M - F + 2P)/S + 1 \\ N' &= (N - F + 2P)/S + 1 \\ C' &= K \end{aligned} \tag{8}$$

O tamanho da imagem na saída de uma camada de convolução ($M' \cdot N' \cdot C'$) é determinada conforme Equação (8), onde: K é a quantidade de (*kernels*); F é o tamanho do *kernel*; S é o passo do deslocamento realizado pelo *kernel* (*stride*); e P é o valor de redimensionamento da imagem (*zero-padding*), sendo utilizado para controlar o tamanho da imagem de saída, com isso é possível manter a saída com o mesmo tamanho da imagem de entrada ($M' = M$ e $N' = N$) (WANG, X. X., 2016).

2.1.2.1 Zero-padding

Figura 13 – Efeito no tamanho da rede sem *zero padding*.



Fonte: GOODFELLOW; BENGIO; COURVILLE, 2016

Figura 14 – Efeito no tamanho da rede com *zero padding*.

Fonte: GOODFELLOW; BENGIO; COURVILLE, 2016

Ao realizar a convolução em uma imagem, a imagem de saída diminui. A solução para este problema é preencher as bordas da imagem com valor 0, sendo essa técnica chamada de *zero-padding*. Se a quantidade de zeros preenchidos nas bordas da imagem possuir uma largura de $n - 1$ e uma altura de $m - 1$, isso permitirá que todos os pixels sejam visitados por todos os elementos do *kernel*, isso é conhecido como convolução completa (*full convolution*) (GONZALEZ; WOODS, 2007).

As Figuras 13 e 14 mostram um exemplo de uma rede convolucional com um *kernel* de largura 6 em todas as camadas. A Figura 13 mostra que ao aplicar o *kernel* sem o *zero-padding* e com *stride* $= 1$ a rede perde 5 pixels por camada. Como neste exemplo possui apenas 16 pixels na camada de entrada, sem o *zero-padding* é possível ter apenas 3 camadas de convolução. Neste caso é possível aumentar o número de camadas diminuindo o tamanho do *kernel*. No entanto diminuir o tamanho do *kernel* ou diminuir o número de camadas da rede, limita o poder expressivo da rede (GOODFELLOW; BENGIO; COURVILLE, 2016). Já a Figura 14 mostra que aplicando o *kernel* com o *zero-padding* a saída de cada camada convolucional permanece do mesmo tamanho da entrada, assim possibilitando a construção de uma rede convolucional com muitas camadas e com tamanhos de *kernels* maiores.

2.1.2.2 Função de ativação

Após a convolução, todos os valores da imagem de saída Z passam por uma função de ativação. Onde essa função de ativação poderá ser uma sigmoide, tangente hiperbólica ou qualquer outra função. No entanto, em DL o mais comum é usar a função de ativação retificadora conforme Equação (9) (ReLU do inglês *Rectified Linear Unit*). Recentemente ReLU (NAIR, V.; HINTON, G. E., 2010) tem sido utilizado com mais frequência por ser mais rápido que

as outras funções de ativações, como sigmoide ou tangente hiperbólica (LECUN; BENGIO; HINTON, G., 2015).

$$a_{ij}^{(k)} = max\Big(0, z_{ij}^{(k)} + b^{(k)}\Big) \tag{9}$$

Onde: b é o viés (*bias*); A é matriz da imagem de saída gerada pela função de ativação, sendo a um elemento da matriz A.

2.1.2.3 Pooling

Pooling é uma operação que quase todas as redes convolucionais empregam em algumas camadas. O *pooling* ajuda a tornar a camada de convolução invariante a pequenas translações (GOODFELLOW; BENGIO; COURVILLE, 2016). Isso significa que modificando-se um pouco a posição do que está sendo apresentado na imagem de entrada, não ocasionarão mudanças nos valores resultantes da saída *pooling*, assim podendo melhorar a eficiência estatística da rede. *Pooling* também é essencial para entradas que variam o tamanho, por exemplo, no caso de uma classificação, deve-se manter igual o tamanho da imagem de entrada, porém a figura que se deseja classificar poderá variar de tamanho.

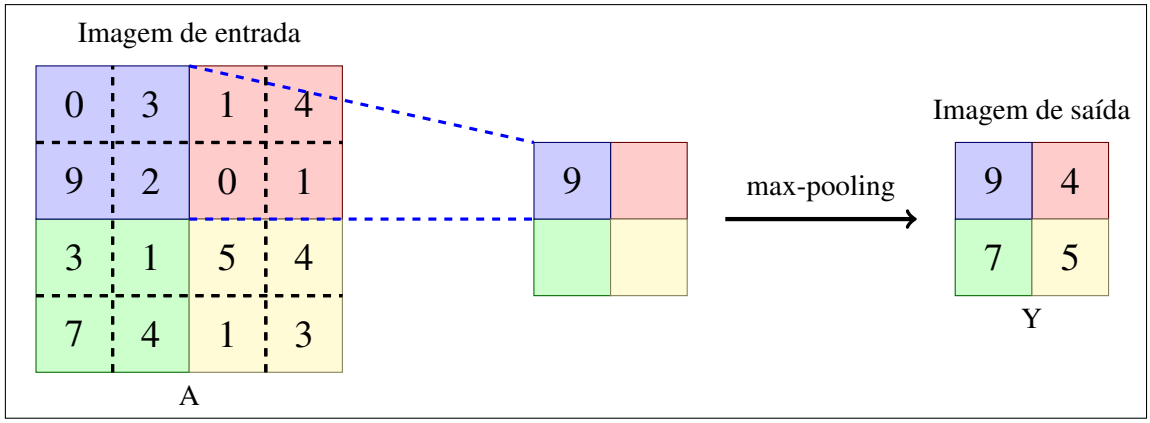
2.1.2.3.1 Max-pooling

O *kernel* do *max-pooling* é utilizado para extrair o valor máximo da região por onde passa. A Figura 15 apresenta um exemplo de um *max-pooling*. Nessa operação, determina-se um *kernel* de tamanho $h * l * K$, sendo esse *kernel* utilizado para extrair o pixel máximo. Da mesma forma que a operação de convolução, desliza-se o *kernel* por toda a imagem, usando um passo determinado pelo valor do *stride*. Nesse exemplo, pode-se observar que a imagem gerada após o *max-pooling* é menor que a imagem da convolução.

A imagem de entrada da função *max-pooling* possui o tamanho de $h \cdot l \cdot K$, onde: l é a largura da imagem; h é a altura da imagem; e K é o número de *kernels*.

$$\begin{aligned} h' &= (h - F)/S + 1 \\ l' &= (l - F)/S + 1 \\ K' &= K \end{aligned} \tag{10}$$

Figura 15 – Exemplo de *max-pooling* com *kernel* 2x2 e *stride* 2.



Fonte: Autor.

O tamanho da imagem de saída é determinado conforme Equação (8), onde: F é o tamanho do *kernel*; S é o passo do deslocamento realizado pelo *kernel* (*stride*).

A Equação (11) descreve a equação da operação *max-pooling*.

$$y_{ij}^{(k)} = max\left(a_{(ui+s)(vj+t)}^{(k)}\right) \tag{11}$$

Onde: A é matriz da imagem de entrada na *max-pooling*, sendo a um elemento da matriz A; Y é matriz da imagem de saída gerada pela operação *max-pooling*, sendo y um elemento da matriz Y; k é o canal da imagem; i é o índice que percorre a linha (altura da imagem) da matriz da imagem; j é o índice que percorre a coluna (largura da imagem) da matriz da imagem; s é o índice que percorre a linha da matriz do *kernel*; t é o índice que percorre a coluna da matriz do *kernel*; u é o passo do deslocamento do *kernel* na imagem (deslocamento na linha da matriz da imagem). v é o passo do deslocamento do *kernel* na imagem (deslocamento na coluna da matriz da imagem). m é a largura do *kernel*; n é a altura do *kernel*. O índice i percorre a imagem até $l - m + 1$ e o índice j percorre a imagem até $N - n + 1$. Normalmente os valores de u e v são iguais, assim define-se apenas o valor do *stride* ($S = u = v$) e costuma-se utilizar um valor entre 2 e 4.

Durante o treinamento, utilizar a operação *max-pooling* faz com que a memória demandada durante o treinamento seja menor, o tempo de *feed-forward* e *feed-backward* também se torna menor. Isso ocorre porque a imagem da saída da camada (Y) é menor quando se utiliza *max-pooling*, assim reduzindo o tempo de treinamento.

2.1.2.3.2 Average-pooling

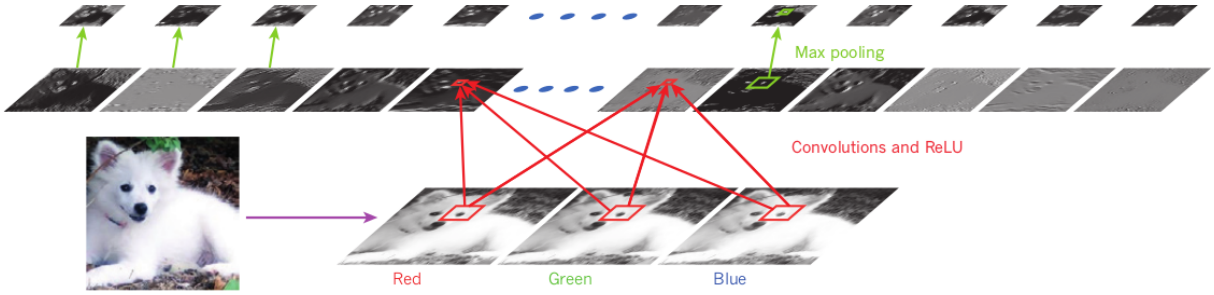
Uma outra operação empregada nas redes convolucionais é o *average-pooling*, para alguns tipos de imagens o *average-pooling* apresenta resultados melhores que o *max-pooling*, por isso sua utilização depende do tipo de dados. Assim como mostrado no exemplo da Figura 15, onde o *kernel* do *max-pooling* é utilizado para extrair o valor máximo da região por onde passa, o *kernel* do *average-pooling* calcula o pixel médio na região por onde passa, resultando na imagem de saída Y, conforme mostra a Equação (12), onde N é o número de termos do *kernel*.

$$y_{ij}^{(k)} = \frac{1}{N} \sum_{i,j} \left(a_{(ui+s)(vj+t)}^{(k)} \right) \tag{12}$$

2.1.2.4 Camada Convolucional

A Figura 16 apresenta um exemplo de uma camada convolucional. Nesse exemplo aplicam-se vários *kernels* na imagem de entrada e também a função de ativação ReLU, depois utiliza-se o *max-pooling*.

Figura 16 – Exemplo de uma camada convolucional.



Fonte: LECUN; BENGIO; HINTON, G., 2015

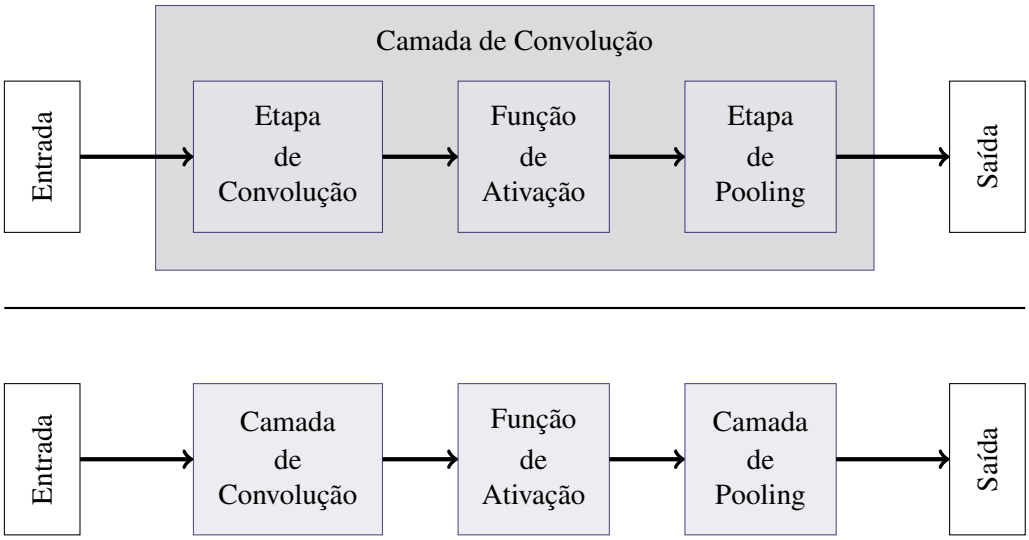
2.1.2.4.1 Calculo da quantidade de parâmetros da rede

Os pesos da rede convolucional são os valores das matrizes do *kernel* (w), os pesos são os valores que serão aprendidos pela rede durante o processo de treinamento. A quantidade de pesos da rede é determinada pela quantidade de canais na imagem de entrada, pelo tamanho e quantidade de *kernels*. Por exemplo, a Figura 9 possui uma imagem de entrada de apenas um

canal e um *kernel* de tamanho 3x3, portanto essa camada de convolução possui 9 pesos (w). No entanto, o valor do viés (b) usado na função de ativação também é aprendido durante o processo de treinamento. Como mostrado na Equação (9), existe um viés para cada *kernel*. Então nesse caso a rede possuí 9 parâmetros referente aos pesos e 1 parâmetro referente ao viés, sendo no total 10 parâmetros.

A Figura 10 apresenta uma imagem de entrada com três canais, e vários *kernels*. Como a imagem de entrada possui três canais o *kernel* também tem três canais. Para calcular a quantidade de pesos dessa camada, aplica-se uma convolução em uma imagem de entrada 6x6x3 em 10 *kernels* de tamanho 4x4x3. Nesse caso teremos 30 matrizes de tamanho 4x4, portanto 480 pesos w. Como essa camada possui 10 *kernels*, haverá 10 parâmetros de viés (b). Então nesse caso a rede possuí 480 parâmetros referente aos pesos e 10 parâmetros referente ao viés, sendo no total 490 parâmetros.

Figura 17 – Calculando a quantidade de camadas da rede.



Fonte: Autor.

A Figura 17 mostra uma típica camada de rede de convolução, no entanto, a Figura mostra que há duas formas de contar a quantidade de camadas de uma rede de convolução. A parte de cima da Figura considera como camada todo conjunto de etapas realizados. Já a parte de baixo da figura considera como camada cada etapa realizada na rede, ou seja, na parte de cima da Figura tem apenas 1 camada, enquanto a parte de baixo da Figura tem 3 camadas. Nesta tese, está sendo considerado como camada todo o conjunto de etapas, como na parte de

cima da Figura 17, isso porque, nessa tese considera-se camadas apenas o conjunto de etapas que possuem parâmetros a serem aprendidos.

2.1.3 Dropout

Dropout (HINTON, G. E. et al., 2012) é uma técnica para reduzir o sobre-ajuste (do inglês: *overfitting*) nas redes neurais. As Redes Neurais Profundas contêm múltiplas camadas ocultas não-lineares de neurônios completamente conectados, assim sendo capaz aprender diversos padrões. No entanto, quando se utiliza um conjunto de dados de treinamento limitado (poucas amostras para o treinamento), acaba resultando em sobre-ajuste. Com o sobre-ajuste o modelo treinado apresentará alta precisão quando testado com um conjunto de dados treinamento (aprenderá até os ruídos da amostra) (SRIVASTAVA et al., 2014), porém apresentará baixa precisão quando testado com dados de validação ou testes (resulta em baixa generalização).

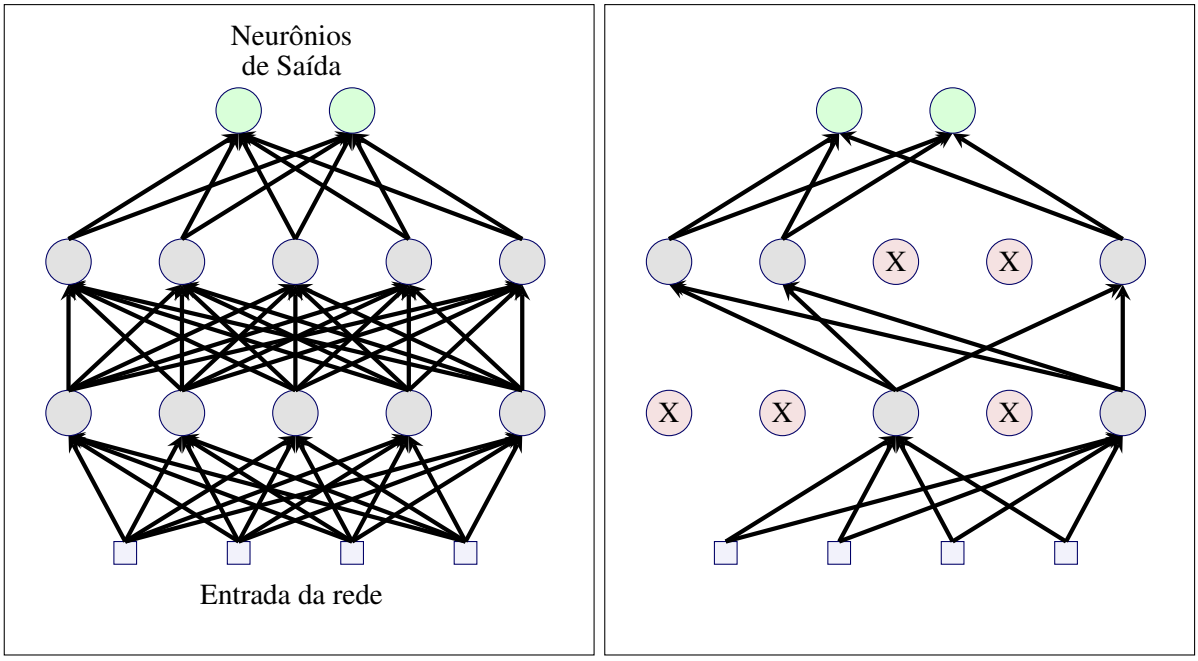
Costuma-se aplicar o *dropout* apenas nas camadas de redes completamente conectadas (ou seja, nas últimas camadas das atuais arquiteturas de DNN, que geralmente possuem redes completamente conectadas), não se aplica *dropout* nas camadas de redes convolucionais porque essas redes são consideravelmente mais resistentes a sobre-ajuste.

Redes com múltiplas camadas demandam bastante tempo com treinamento, tornando difícil lidar com sobre-ajuste. *Dropout* é uma técnica para resolver este problema, onde a ideia é descartar aleatoriamente uma porção de neurônios (junto com suas conexões) da rede neural quando se realiza a retro-propagação (do inglês *backpropagation*), então esses neurônios voltam para a rede e novamente descartar-se aleatoriamente uma porção de neurônios (SRIVASTAVA et al., 2014) para realizar novamente a retro-propagação. A Figura 18 apresenta um exemplo de uma rede com e sem *dropout*.

Na Figura 18, os neurônios nas cores verdes são neurônios de saída e os cinzas são neurônios das camadas escondidas, sendo que essa rede possui duas camadas escondidas. A Figura 18a e 18b pertencem a mesma rede, a diferença é que na Figura 18a a rede está completa, no entanto na Figura 18b foi aplicado o *dropout*, pode-se observar que os neurônios nas cores vermelha estão temporariamente descartados da rede.

A atualização dos pesos da rede durante o treinamento é realizada utilizando-se valores de erro relacionados a um lote de dados (por exemplo, no treinamento com imagens utiliza-

Figura 18 – Exemplo de *Dropout*.



(a) Rede sem *Dropout* (b) Rede com *Dropout*

Fonte: Autor.

se grupos de imagens), onde todas as imagens do lote são passadas através da rede e assim calcula-se um erro médio usado para atualizar os pesos da rede.

2.1.4 Softmax

Softmax é uma função usada frequentemente na saída da rede para representar o valor da classificação apresentado na saída como uma distribuição de probabilidades (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{13}$$

Onde: Z_i é o valor de saída do neurônio i; k é a quantidade total de saídas;

Com a Equação (13) a saída será representada por um valor dentro do intervalo de 0 até 1, e o somatório de todos os valores da saída será sempre 1, assim podendo ser interpretado como uma probabilidade.

Caso deseje aprofundar o conhecimento na área de Aprendizado Profundo, o livro do Goodfellow, Bengio e Courville (2016) apresenta uma boa descrição teórica sobre Aprendizado

Profundo. Tem também os artigos (LECUN; BENGIO; HINTON, G., 2015; DENG, L.; YU et al., 2014), sendo que no artigo (DENG, L.; YU et al., 2014) apresenta também aprendizado não supervisionado. Para saber mais sobre a competição ILSVRC, tem o artigo do Russakovsky et al. (2015), e o artigo da Jia Deng et al. (2009) sobre a ImageNet.

2.2 APRENDIZADO POR REFORÇO

No aprendizado por reforço, o agente aprende através da interação com o ambiente recebendo recompensas negativas ou positivas, de acordo com as ações tomadas. O modelo é descrito por um conjunto de estados $\mathcal{S}$ e um conjunto de ações $\mathcal{A}$, e para cada ação executada a_t em algum estado s_t de acordo com um modelo de transição $P(s_{t+1}|s_t,a_t)$ o agente receberá um reforço (recompensa) $R(s_t,a_t,s_{t+1})$. Neste modelo espera-se que o agente aprenda uma política que maximize a recompensa acumulada ao longo do tempo $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (MITCHELL, 1997).

Durante a interação do agente com o ambiente, o agente executa uma ação e recebe uma observação e uma recompensa r, já o ambiente recebe a ação e emite a observação e a recompensa. Pela equação de Bellman, a recompensa deve ser acumulada com um fator de desconto γ, conforme Equação (14), em que T é o tempo do passo final e R_t é a recompensa acumulada ao longo do tempo obtida através da interação do agente com o ambiente. (SUTTON, Richard S.; BARTO, 1998).

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k} \tag{14}$$

A função valor é a predição da recompensa futura, porque a função valor possui a soma das recompensas recebidas com um fator de desconto durante a interação do agente com o ambiente (Equação (15)). Então, a função valor também pode ser decomposta pela equação de Bellman. Podemos definir uma função valor para cada estado $V(s)$, $s \in \mathcal{S}$, com isso o agente consegue estimar quais estados resultam em maiores recompensas, sendo que as recompensas recebidas em um dado estado depende de quais ações foram tomadas pelo agente (SUTTON, Richard S.; BARTO, 1998).

$$V(s) = r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \dots \tag{15}$$

Da mesma forma pode-se definir uma função valor para o par estado-ação $Q(s,a)$, $s \in \mathcal{S}$ e $a \in \mathcal{A}$ (Equação (16)), com isso o agente consegue estimar quais estados e ações para aquele estado resultam em maiores recompensas.

$$Q(s,a) = r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \dots \tag{16}$$

Uma função valor ótima $Q^*(s,a)$ ou $V^*(s)$ é a que possui o máximo valor alcançado (equações 17 e 18).

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{17}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall s \in \mathcal{S} \quad \text{e} \quad a \in \mathcal{A} \tag{18}$$

Durante a interação do agente com o ambiente para que se maximize a recompensa acumulada ao longo do tempo e assim obter políticas ótimas, deve-se selecionar as ótimas funções valores, satisfazendo as equações de Bellman conforme equações 19 e 20.

$$\begin{aligned} V^*(s_t) &= r(s_t,a_t,s_{t+1}) + \gamma V^*(s_{t+1}) \\ &= \max_a \sum_{s_{t+1}} p(s_{t+1}|s_t,a_t)\Big[r(s_t,a_t,s_{t+1}) + \gamma V^*(s_{t+1})\Big], \quad \forall s \in \mathcal{S} \end{aligned} \tag{19}$$

$$\begin{aligned} Q^*(s_t,a_t) &= r(s_t,a_t,s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1}) \\ &= \sum_{s_{t+1}} p(s_{t+1}|s_t,a_t)\Big[r(s_t,a_t,s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})\Big], \forall s \in \mathcal{S}, \quad a \in \mathcal{A} \end{aligned} \tag{20}$$

Tendo a função valor ótima do par estado-ação Q^*, é possível então escolher a melhor ação a ser tomada conforme Equação (21).

$$\pi^*(s) = arg \max_a Q^*(s,a), \forall a \in \mathcal{A}(s) \tag{21}$$

A seleção de ações pode seguir a regra gulosa $\epsilon - Greedy$, que é uma estratégia bastante utilizada no Q-learning para a escolha das ações (BIANCHI, 2004). Selecionando as ações de forma gulosa, o algoritmo tira proveito da experiência das iterações anteriores. Pela regra gulosa o algoritmo seleciona as melhores ações (explotação), porém com uma parcela de exploração realizada pela seleção das ações de forma aleatória, portanto a seleção de ações segue conforme regra gulosa $\epsilon - Greedy$ dada na Equação (22).

$$\pi(s_t) = \begin{cases} a \sim U(\mathcal{A}(s_t)) & q \leq \epsilon, \\ arg\max_a Q_t(s_t,a), \forall a \in \mathcal{A}(s_t) & \text{caso contrário} \end{cases} \tag{22}$$

Sendo ϵ o parâmetro que define a taxa de exploração/explotação do ambiente podendo possuir um valor que varia de 0 até 1 ($0 \leq \epsilon \leq 1$), e q é um valor aleatório entre 0 e 1 de uma distribuição uniforme $q \sim U(0,1)$.

2.2.1 Q-Learning

O algoritmo Q-learning (WATKINS, 1989; WATKINS; DAYAN, 1992) determina uma função Q (par de estado-ação). Neste algoritmo consideram-se as transições ocorridas a cada iteração pelo par estado-ação, diferente do método de diferenças temporais (TD - *Temporal Difference*) (SUTTON, Richard S, 1988), que considera apenas o valor do estado. A Equação (23), mostra a equação do Q-learning.

$$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha\Big[r(s_t,a_t) + \gamma \max_{a'_t} Q_t(s'_t,a'_t) - Q_t(s_t,a_t)\Big] \tag{23}$$

Onde:

a) s_t : estado atual;
b) a_t : ação que será realizada estando em s;
c) $r(s_t,a_t)$: reforço ao se realizar uma ação a estando no estado s;
d) s'_t : estado futuro observado;
e) a'_t : ação a ser realizada no estado s';
f) γ : fator de desconto $0 \leq \gamma < 1$;
g) α : taxa de aprendizado $0 \leq \alpha < 1$.

2.2.2 Double Q-Learning

O baixo desempenho do algoritmo Q-learning (baixo desempenho significa que o Q-learning precisa de muitos episódios para se aproximar ou encontrar a política ótima) é causado pela superestimação da função valor Q (HASSELT, 2010). Então Hasselt (2010) propôs um

algoritmo chamado Double Q-Learning que possui duas funções valor: Q^A e Q^B. Onde cada função é atualizada com o valor da outra função conforme Algoritmo 1.

Algoritmo 1 – Algoritmo Double Q-learning.

1 Inicialize $Q^A(s,a)$ e $Q^B(s,a)$ com valores aleatórios
2 **enquanto** *o número de episódios desejado não for alcançado* **faça**
3 Inicialize o estado s em uma posição fixa ou aleatório
4 **enquanto** *objetivo não for alcançado* **faça**
5 Escolha uma ação a do estado s utilizando a seleção com $\epsilon - greedy$ baseado em $Q^A(s,\cdot)$ e $Q^B(s,\cdot)$
6 Observe o estado s' e a recompensa r
7 Escolha de forma aleatória entre UPDATE(A) ou UPDATE(B)
8 **se** *UPDATE(A)* **então**
9 Defina $a^* = arg\max_a Q^A(s',a), \forall a \in \mathcal{A}(s')$
10 Atualize o $Q^A(s,a)$ conforme equação:
11 $$Q^A(s,a) \leftarrow Q^A(s,a) + \alpha\Big[r + \gamma Q^B(s',a^*) - Q^A(s,a)\Big]$$
12 **fim**
13 **senão**
14 Defina $b^* = arg\max_a Q^B(s',a), \forall a \in \mathcal{A}(s')$
15 Atualize o $Q^B(s,a)$ conforme equação:
16 $$Q^B(s,a) \leftarrow Q^B(s,a) + \alpha\Big[r + \gamma Q^A(s',b^*) - Q^B(s,a)\Big]$$
17 **fim**
18 Atualize o estado s:
19 $s \leftarrow s'$
20 **fim**
21 **fim**

No Algoritmo 1, a seleção da ação realizada na linha 5 segue a regra gulosa $\epsilon - greedy$, para selecionar a ação utilizam-se ambas funções valores: Q^A e Q^B. Calculando-se a média entre os dois valores de $Q^A(s,a)$ e $Q^B(s,a)$ referente a mesma ação entre as funções valores, então realiza-se a seleção $\epsilon - greedy$ no resultado da média do valor Q. Em alguns experimentos utiliza-se o valor de $\epsilon - greedy$ como $\epsilon = 1/\sqrt{n(s)}$, onde $n(s)$ é o número de estados explorado pelo agente.

Nos experimentos apresentados por Hasselt (2010) ele demonstra que o Double Q-Learning converge para uma política ótima com menos episódios que o algoritmo Q-Learning.

2.2.3 Aprendizado por Reforço com Redes Neurais

A Tese de doutorado do Lin (1993) tendo como orientador o Tom M. Michel combinou o aprendizado por reforço com redes neurais (*back-propagation* com método TD), assim foi

possível aplicar este método em ambientes mais complexos do que os ambientes que estavam sendo aplicados RL naquela época (LIN, 1993).

Um dos problemas do RL daquela época e que se estende até os dias atuais é que o problema para ser solucionado com aprendizado por reforço deverá possuir uma representação em um pequeno espaço de estados (LIN, 1993). No trabalho do Lin (1993) o objetivo é usar uma rede neural para conseguir aumentar esse espaço de estados. Nessa mesma época do trabalho do Lin (1993), várias outras pesquisas buscavam métodos que pudessem generalizar alguns estados pela similaridade, assim abstraindo os estados e aplicando o RL neste espaço de estados com estados abstraídos (LIN, 1993).

O objetivo do trabalho do Lin (1993) foi usar uma rede neural de multicamadas com *back-propagation* para generalização dos estados similares. A generalização também ajuda a reduzir o tempo necessário para o agente aprender a tarefa. Lin (1993) também propôs a técnica chamada *experience replay*, essa técnica armazena as experiências passadas.

2.2.4 Aprendizado por Reforço Acelerado por Heurística

Os algoritmos de aprendizado por reforço necessitam de um número de iterações muito grande para sua convergência, como consequência, à medida que aumenta o espaço de estados e o número de ações, aumenta-se a necessidade de um número maior de iterações para sua aprendizagem, aumentando significativamente o tempo de convergência do algoritmo, tornando em alguns casos o processo de aprendizado extremamente lento. Dessa forma para minimizar esse problema, algumas propostas foram apresentadas, tais como: melhorar o aproveitamento das experiências por meio de generalizações temporais, espaciais ou das ações; o uso de aceleração por abstração temporal ou espacial; a aceleração por distribuição utilizando sistemas multiagentes; e a aceleração baseada em casos (BIANCHI, 2004).

Bianchi (2004) demonstra a aplicação e eficiência da heurística no algoritmo Q-learning. A função heurística é definida a partir de um conhecimento prévio sobre o domínio e é utilizada apenas na seleção das ações a serem executadas pelo agente. Para usar a heurística, a regra de seleção de ações $\epsilon - Greedy$ inclui a função heurística para a escolha da melhor ação, conforme Equação (24).

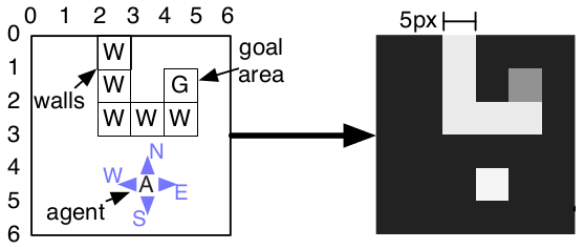
$$\pi(s_t) = \begin{cases} a \sim U(\mathcal{A}(s_t)) & q \leq \epsilon, \\ arg\max_a \Big[Q_t(s_t,a) + \xi H(s_t,a) \Big], \forall a \in \mathcal{A}(s_t) & \text{caso contrário} \end{cases} \tag{24}$$

Onde ξ é um número real que pondera a influência da heurística, cujo valor normalmente é igual a 1 (BIANCHI, 2004), o $H(s,a)$ é a heurística. Deve-se definir um valor de $H(s,a)$ que aplique um conhecimento adicional sobre o referente estado-ação. Essa proposta mantém as principais características dos algoritmos de aprendizado por reforço, porém acaba minimizando o tempo necessário para a convergência.

2.3 APRENDIZADO POR REFORÇO PROFUNDO

Em 2010 Lange e Martin A Riedmiller (2010) apresentaram um algoritmo de Aprendizado por Reforço Profundo usando redes neurais de *autoencoder* que aprendia políticas de ações apenas observando os pixels da imagem. Eles testaram o algoritmo em um mundo de grades de 30x30 pixels conforme imagem da Figura 19, onde o agente pode visitar qualquer posição no espaço de 30x30 (o agente possuí a dimensão de 5x5 pixels), sendo que o agende deve aprender a chegar ao objetivo (*goal*) a partir de qualquer posição do mapa. Este modelo utiliza a rede neural de *autoencoder* para diminuir o espaço de estados da imagem de entrada, onde o algoritmo de aprendizado por reforço (utilizaram o NFQ (RIEDMILLER, M., 2005)) aprende a partir do espaço de estados diminuído pela rede neural de *autoencoder*.

Figura 19 – Mundo de grades.



Fonte: LANGE; RIEDMILLER, M. A., 2010

Em 2013 pesquisadores da Google *DeepMind* disponibilizaram um artigo chamado *Playing Atari with Deep Reinforcement Learning* no Arxiv (MNIH et al., 2013). Neste artigo eles apresentaram um modelo de Aprendizado por Reforço Profundo que aprende a política de ações de um jogo usando como entrada as imagens do jogo. Eles demostraram que o computador foi capaz de aprender a jogar alguns jogos de Atari, apenas observando os pixels da imagem

e recebendo a recompensa positiva pelo incremento da pontuação do jogo. Neste artigo foram testados 7 jogos de Atari.

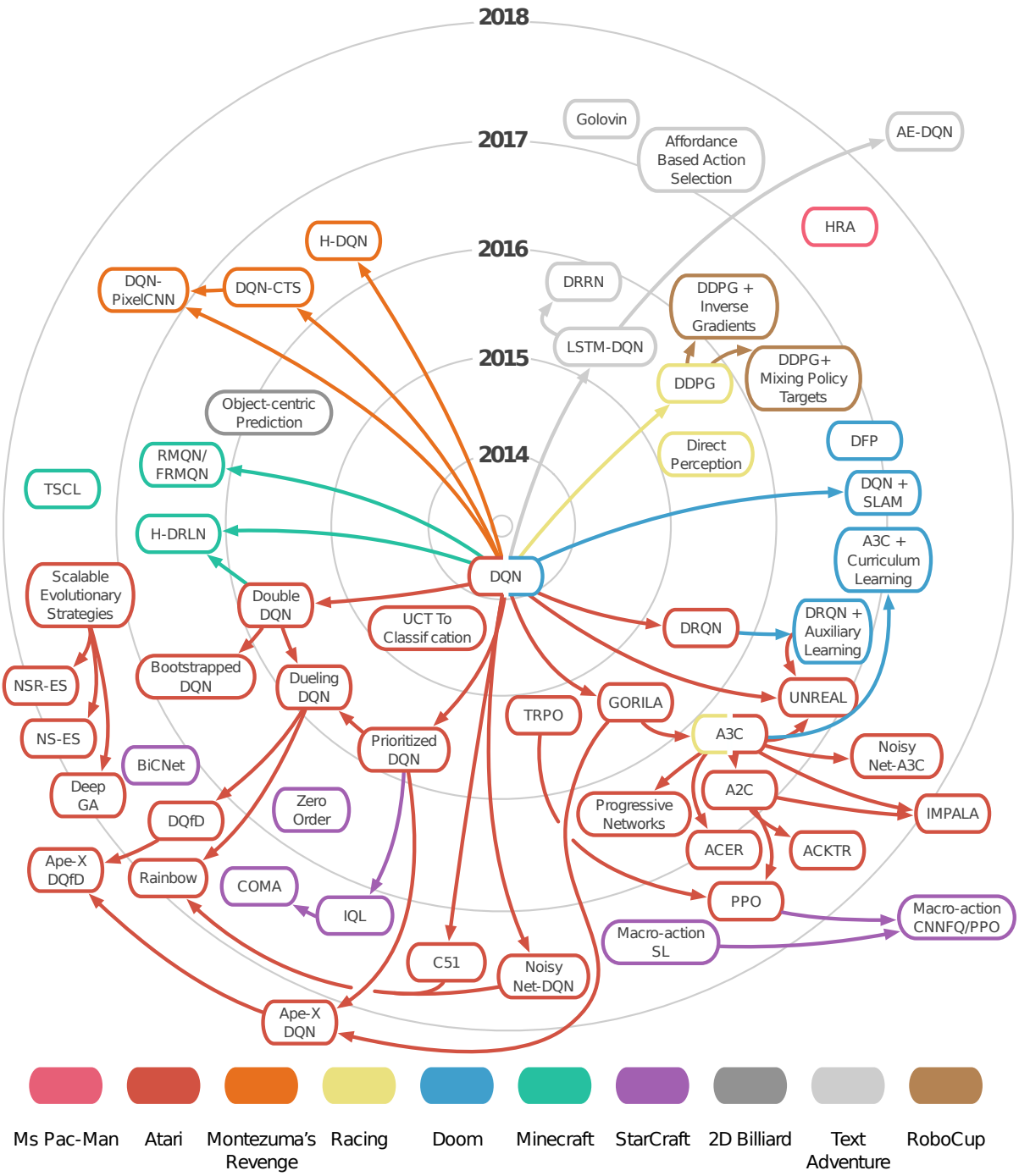
Já em 2015, esses mesmos pesquisadores apresentaram um artigo (MNIH et al., 2015) que demonstra que em jogos de Atari o modelo de Aprendizado por Reforço Profundo chamado de *Deep Q-Network* (DQN) superou o desempenho de todos os algoritmos anteriores, sendo capaz de superar o nível de um jogador humano profissional em alguns jogos. A diferença entre o DQN e a rede neural de *autoencoder* com NFQ, é que o DQN atualiza os pesos da rede aprendendo as características relevantes em relação às ações realizadas e a recompensa recebida, sendo uma abordagem que aplica o RL de ponta a ponta.

A partir dessas pesquisas apresentadas, foram surgindo novas modificações no modelo de DQN objetivando a melhoria, tanto na diminuição do tempo de treinamento (MNIH et al., 2016; SCHAUL et al., 2015), quanto no desempenho em pontuação no jogo (VAN HASSELT; GUEZ; SILVER, 2015; WANG, Z.; FREITAS; LANCTOT, 2016). Também foram surgindo novas aplicações em Aprendizado por Reforço Profundo como em controle contínuo (LILLICRAP et al., 2015), e também sendo aplicado em diferentes jogos como no artigo do Jaderberg et al. (2016) que usou DRL em um jogo de labirinto 3D.

A Figura 20 mostra um diagrama de influência de Algoritmos de DRL, onde cada nó representa um algoritmo de DRL e a cor representa o domínio em que o algoritmo foi desenvolvido e experimentado. A distância do centro representa a data em que o artigo original foi publicado. Setas apontando para um determinado algoritmo mostram quais algoritmos influenciaram no desenvolvimento (JUSTESEN et al., 2019). A Figura 20 mostra que a maioria dos algoritmos de Aprendizado por Reforço Profundo usam como base o algoritmo DQN. Esse histórico do diagrama de influência mostra que o principal ambiente que foi usado para desenvolver e melhorar o DQN foram os jogos de arcade, e que mais tarde foram aplicadas e desenvolvidas variações do algoritmo para jogos mais complexos.

No domínio do futebol de robôs da RoboCup o *Deep Deterministic Policy Gradients* (DDPG) foi aplicado no problema *Half-Field-Offense* (HFO) da RoboCup 2D. HFO (HAUSKNECHT et al., 2016) é um ambiente da RoboCup 2D para a aprendizagem multiagente, onde 2 a 3 jogadores tomam o papel de ataque ou defesa em uma metade de um campo de futebol. Foram criados duas extensões do DDPG, DDPG+*Inverting Gradients* (HAUSKNECHT; STONE, 2015) e DDPG+*Mixing policy targets* (HAUSKNECHT; STONE, 2016). O DDPG+*Inverting Gradients* superou o SARSA e o melhor agente de 2012 da RoboCup 2D (HAUSKNECHT; STONE, 2015).

Figura 20 – Diagrama de influência de Algoritmos de DRL.



Fonte: JUSTESEN et al., 2019

A Figura 20 apresenta diversos algoritmos de Aprendizado por Reforço Profundo, no entanto neste trabalho foram utilizados apenas alguns destes algoritmos, por isso, as próximas seções apresentarão as definições teóricas apenas dos algoritmos que foram utilizados na tese, a não ser o Algoritmo A3C, que apresentou resultados melhores em jogos de Atari, porém não foi utilizado devido ao algoritmo trabalhar de forma paralela.

2.3.1 Deep Q-Network

DQN combina Q-learning com Redes Neurais Profundas. A DQN desenvolvida por Mnih et al. (2013) utiliza como entrada apenas os quadros da imagem do jogo. A saída da rede são as ações possíveis para o referido jogo. A rede é composta por três camadas de redes neurais convolucionais seguida por duas camadas com redes completamente conectadas. Observa-se que nessa rede não há nenhuma camada de *pooling*.

$$y_j^{DQN} = r_j + \gamma \max_a Q(s_{j+1}, a; \theta_i^-) \tag{25}$$

Durante a interação do agente com o ambiente, para que se maximize a recompensa acumulada ao longo do tempo, deve-se selecionar as ótimas funções valores, satisfazendo as equações de Bellman conforme Equação (25).

$$L(\theta_i) = \frac{1}{n} \sum_j^n \left[(y_j^{DQN} - Q(s_j,a_j;\theta_i))^2 \right] \tag{26}$$

Para treinar a rede ajustando os valores dos pesos θ pelo *back-propagation*, é realizado um gradiente descendente (Equação (27)) do erro quadrático médio (MSE do inglês *Mean squared error*) aplicado na função L para cada iteração i conforme Equação (26), onde $\theta_i^- = \theta_{i-1}$ e γ é o fator de desconto.

$$\nabla_{\theta_i} L(\theta_i) = (y^{DQN} - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i) \tag{27}$$

Para melhorar a estabilidade da DQN foi usado um Q diferente para atualizar o y, então a cada C passos realiza-se uma cópia dos valores Q para um $\hat{Q}$. Usando esse valor $\hat{Q}$ o algoritmo torna-se mais estável, portanto, as divergências ou oscilações da política torna-se menos comum (MNIH et al., 2015).

Para armazenar as experiências do agente, é utilizado uma técnica conhecida como *experience replay* (LIN, 1993), portanto durante o jogo todas as experiências $e_t =< s_t,a_t,r_t,s_{t+1} >$ são armazenadas em uma variável chamada *replay memory* $\mathcal{D} = \{e_1, e_2, \ldots, e_n\}$. Durante o treinamento, para evitar que a rede se desloque para um mínimo local, são utilizados pequenas quantidades de transições ($\mathcal{E}$ são pequenas quantidades de transições chamadas de *minibatches transitions*) que são selecionadas de forma aleatória das experiências armazenadas na *replay memory* $\mathcal{D}$, ao invés de utilizar as transições mais recentes.

Algoritmo 2 – Algoritmo Deep Q-Network.

Inicialize a *replay memory* $\mathcal{D}$ vazia mas com capacidade N_r
Inicialize os parâmetros θ da rede
Inicialize a função valor Q com valores aleatórios
enquanto *o número de episódios desejado não for alcançado* **faça**
    Observe o estado s_t do quadro x, ($s_t \leftarrow x$)
    enquanto *O episódio não for finalizado* **faça**
        Selecione a ação a_t do estado s_t utilizando a seleção $\epsilon - greedy$
        Execute a ação a_t no emulador e observe a recompensa r_t e a imagem x
        Observe o estado s_{t+1} do quadro x, ($s_{t+1} \leftarrow x$)
        Armazena a experiência $e_t =< s_t,a_t,r_t,s_{t+1} >$ na memória $\mathcal{D}$
        Selecione de forma aleatória um conjunto de experiências $\mathcal{E}$ da memória $\mathcal{D}$ possuindo uma quantidade N_b de experiências ($\mathcal{E} \sim U(\mathcal{D})$)
        para *cada experiência* $e_j \in \mathcal{E}$ **faça**
            se *episódio termina no passo* $j + 1$ **então**
                $y_j \leftarrow r_j$
            fim
            senão
                $y_j \leftarrow r_j + \gamma \max_a \hat{Q}(s_{j+1}, a; \theta^-)$
            fim
        fim
        Treine a rede realizando a descida de gradiente em $(y_j - Q(s_j, a_j; \theta))^2$ para os parâmetros θ da rede
        $\theta^- \leftarrow \theta$
        $s_t \leftarrow s_{t+1}$
        A cada C passos faça: $\hat{Q} = Q$
    fim
fim

Como é possível ver no Algoritmo 2, a variável x é o quadro da imagem do jogo. Já o estado s_t é considerando como uma sequência de ações e observações (onde nesse domínio cada ação e observação é um passo) $s_t = x_1,a_1,x_2,a_2,x_3,a_3,\ldots,x_t$, considerando t como um tempo finito dessas sequências. Assim teremos um grande e finito MDP em que cada sequência é um estado distinto. N_r é a capacidade máxima de armazenamento de experiências na *replay memory* ($|\mathcal{D}| = N_r$). N_b é a quantidade de experiências que será utilizado da *replay memory* ($|\mathcal{E}| = N_b$) sendo que N_b não pode ser maior que N_r ($1 \leq N_b \leq N_r$). $\mathcal{E}$ é um conjunto de experiências ($\mathcal{E} = \{e_1, e_2, \ldots, e_{N_b}\}$) extraído de forma aleatória da memória $\mathcal{D}$ ($\mathcal{E} \sim U(\mathcal{D})$). A rede executa um número pré-determinado de episódios e cada episódio é finalizado quando o jogo termina. O objetivo do agente é interagir com o jogo através da escolha das ações de forma que maximize a recompensa acumulada. A recompensa futura é descontada por um fator de desconto γ a cada passo.

2.3.2 Double DQN

Hasselt (2010) apresentou um algoritmo chamado Double Q-learning para solucionar o problema do algoritmo Q-learning que superestima a função valor Q em certas condições. Então, os pesquisadores da Google o Van Hasselt, Guez e Silver (2015) observaram que na DQN em certos jogos de Atari, esse efeito de superestimar a função valor Q ocorre de forma mais acentuada. Esses mesmos pesquisadores criaram o algoritmo Double DQN (DDQN) baseado no algoritmo Double Q-learning. Com o DDQN em jogos de Atari foi possível aumentar a pontuação nos jogos comparado com o DQN usando os mesmos valores de parâmetros, portanto DDQN apresenta políticas melhores que o DQN para os jogos de Atari.

O Double Q-learning possui duas funções valores Q, a atualização dos valores de Q é realizado de forma aleatória entre as duas funções valores. Baseado nesta mesma abordagem o DDQN possui dois conjuntos de pesos θ e θ'. Como é possível verificar na Equação (28), um peso é usado para selecionar a ação usando o $argmax$, conforme política gulosa, já o outro peso é usado para determinar o valor Q. A atualização é realizada apenas invertendo a posição dos pesos θ_t e θ'_t na Equação (28), sendo que esta troca é realizada de forma aleatória.

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, arg\max_a Q(S_{t+1}, a; \theta_t); \theta'_t) \tag{28}$$

No entanto, com a utilização da Equação (28) haveria dois pesos para serem atualizados, assim gerando duas redes. Então os pesquisadores realizaram uma simplificação substituindo o θ'_t pelo θ_t^-, onde θ_t^- é uma cópia do θ_t do passo anterior. Sendo que a Equação (29) difere da Equação (28), apenas pelo uso do θ_t^-.

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-) \tag{29}$$

Pode-se ver no Algoritmo 3 e na Equação (29) que os parâmetros θ são utilizados para a seleção de ações, enquanto os parâmetros do passo anterior θ^- são utilizados para seleção do valor do estado ação Q. Com isso foi possível remover o problema de superestimar o valor Q,

Algoritmo 3 – Algoritmo DDQN.

Inicialize a *replay memory* $\mathcal{D}$ vazia mas com capacidade N_r
Inicialize os parâmetros θ da rede e copie θ em θ^-, ($\theta^- \leftarrow \theta$)
Inicialize a função valor Q com valores aleatórios
enquanto *o número de episódios desejado não for alcançado* **faça**
    Observe o estado s_t do quadro x, ($s_t \leftarrow x$)
    enquanto *O episódio não for finalizado* **faça**
        Selecione a ação a_t do estado s_t utilizando a seleção com $\epsilon - greedy$
        Execute a ação a_t no emulador e observe a recompensa r_t e a imagem x
        Observe o estado s_{t+1} do quadro x, ($s_{t+1} \leftarrow x$)
        Armazena a experiência $e_t =< s_t, a_t, r_t, s_{t+1} >$ na memória $\mathcal{D}$
        Selecione de forma aleatória um conjunto de experiências $\mathcal{E}$ da memória $\mathcal{D}$ possuindo uma quantidade N_b de experiências ($\mathcal{E} \sim U(\mathcal{D})$)
        para *cada experiência* $e_j \in \mathcal{E}$ **faça**
            Defina $a^*(s_{j+1}; \theta) = arg\max_a Q(s_{j+1}, a; \theta), \forall a \in \mathcal{A}(s_{j+1})$
            se *episódio termina no passo* $j + 1$ **então**
                $y_j \leftarrow r_j$
            fim
            senão
                $y_j \leftarrow r_j + \gamma Q(s_{j+1}, a^*(s_{j+1}; \theta); \theta^-)$
            fim
        fim
        Treine a rede realizando a descida de gradiente em $(y_j - Q(s_j, a_j; \theta))^2$ para os parâmetros θ da rede
        $\theta_t^- \leftarrow \theta$
        $s_t \leftarrow s_{t+1}$
    fim
fim

2.3.3 Prioritized Experience Replay

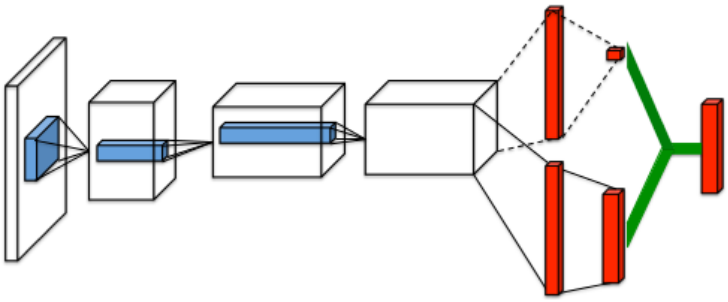
Na DQN o uso do *experience replay* armazenado na *replay memory* $\mathcal{D}$ permite que os agentes lembrem e reutilizem as experiências do passado. Na DQN, a *experience replay* pode reduzir a quantidade de experiência necessária para o agente aprender.

No artigo do Schaul et al. (2015) os pesquisadores desenvolveram um método que pode tornar o aprendizado a partir do *experience replay* mais eficiente, desenvolveram uma forma de priorizar a experiência, assim reproduzindo com mais frequência as transições importantes e eficientes. O objetivo principal de priorizar a experiência é a importância de medir cada transição, atribuindo uma prioridade para a transição. Porque quando se trata as amostras da mesma forma, não está sendo considerado que se pode aprender mais de algumas transições do que de outras. Para medir a importância das transições, utiliza-se o erro, atribuindo uma prioridade para a transição.

2.3.4 Dueling Network Architecture

Os pesquisadores da Google (WANG, Z.; FREITAS; LANCTOT, 2016) desenvolveram uma nova arquitetura de Aprendizado por Reforço Profundo, conforme apresentado na Figura 21. Os pesquisadores combinaram essa nova arquitetura com o algoritmo DDQN e também com o algoritmo *Prioritized Experience Replay*.

Figura 21 – Arquitetura da rede *Dueling*.



Fonte: WANG, Z.; FREITAS; LANCTOT, 2016.

Essa arquitetura possui duas redes em paralelo de neurônios completamente conectados representando a função valor $V(s)$ e a função *advantage* $A(s,a)$, já a saída combina essas duas funções produzindo a função valor estado-ação $Q(s,a)$ ($Q(s,a) = V(s) + A(s,a)$), mas compartilham das mesmas camadas de redes convolucionais. A função *advantage* especifíca como é bom para o agente executar uma ação em comparação com outras ações.

2.3.5 Gorila DQN

A arquitetura Gorila (do inglês General Reinforcement Learning Architecture) foi desenvolvida pelos pesquisadores da Google DeepMind (NAIR, A. et al., 2015) com o intuito de realizar um aprendizado por reforço distribuído, utilizando diversas máquinas que compartilham do aprendizado. Realizar o treinamento da DQN em uma única máquina requer um longo tempo, costuma-se gastar alguns dias de treinamento. O Gorila permite que esse tempo gasto com treinamento seja reduzido (NAIR, A. et al., 2015).

A arquitetura Gorila DQN possui quatro componentes principais: atores em paralelo que geram novos comportamentos; aprendizes em paralelo que são treinados com as experiências armazenadas; uma rede neural distribuída representando a função valor; e uma *experience replay memory* distribuída ($\mathcal{D}$ global) (NAIR, A. et al., 2015).

Este algoritmo é executado com múltiplos agentes em paralelo que interage em múltiplas instancias do mesmo ambiente, cada ator pode armazenar sua própria experiência ($\mathcal{D}$ local) e também compartilhar essa experiência na memória $\mathcal{D}$ global (NAIR, A. et al., 2015).

2.3.6 A3C-LSTM

Mnih et al. (2016) apresentaram um algoritmo de DRL chamado *asynchronous advantage actor-critic* (A3C) que usa gradiente descendente assíncrono. Neste modelo são implementados agentes assíncronos, similar a arquitetura Gorila DQN (NAIR, A. et al., 2015), porém são executadas múltiplas *threads* na CPU e utiliza-se apenas um computador, diferente do Gorila que utiliza os agentes em máquinas separadas. Mantendo o aprendizado em uma única máquina elimina-se o custo de comunicação devido ao compartilhamento de parâmetros entre os agentes (MNIH et al., 2016).

Em vez de utilizar a *experience replay*, executa-se de forma assíncrona vários agentes em paralelo, em várias instâncias do ambiente. Com esse paralelismo os agentes experimentarão uma variedade de estados diferentes.

2.3.7 Deep Q-Network em jogos de Atari

Os pesquisadores da Google (Mnih et al. (2015)) testaram a DQN em um conjunto de 49 jogos de Atari. Nesse domínio a DQN foi capaz de superar todos os algoritmos anteriores e alcançou um nível comparável ao de um jogador humano profissional, usando o mesmo algoritmo e os mesmos parâmetros, recebendo apenas os pixels da imagem do jogo como entrada (MNIH et al., 2015). Nesta subseção será descrito como foi implementado a DQN para os jogos de Atari.

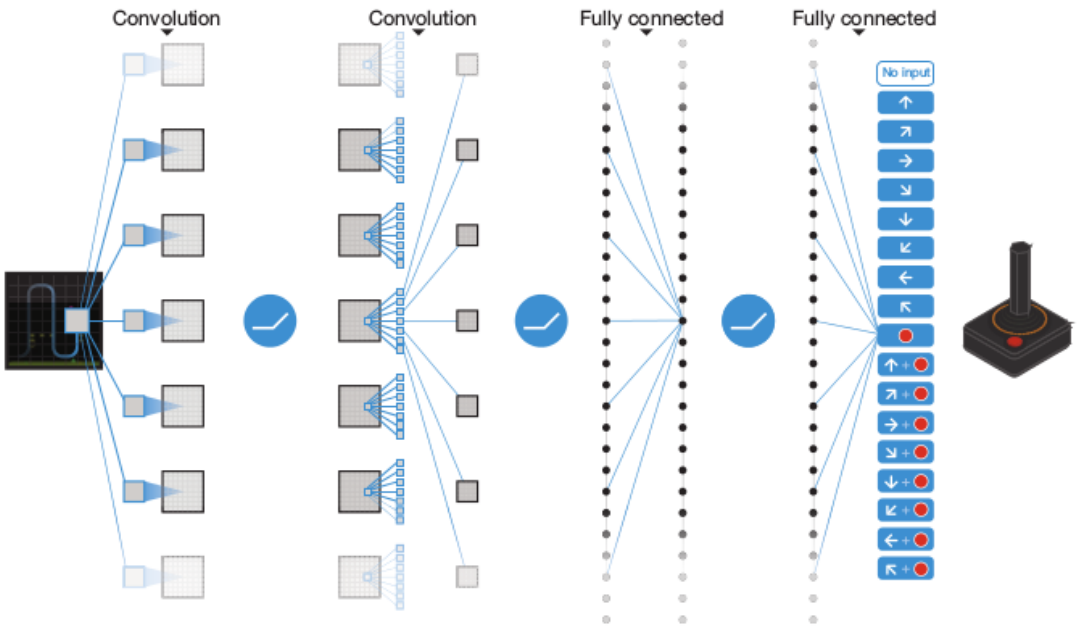
A imagem do jogo de Atari é composta de 210x160 pixels e 128 cores em uma taxa de 60Hz. Para reduzir a dimensionalidade e tratar as particularidades do jogo foi realizado um pré-processamento da imagem, onde primeiramente seleciona-se o valor máximo de cada pixel entre o *frame* atual e o *frame* anterior, para remover o *flickering* usado nos jogos (isso porque alguns objetos só aparecem em quadros pares e outros em quadros ímpares para evitar o *sprites*), em segundo foi extraído o canal Y (luminância), e redimensionado o quadro para 84x84 (MNIH et al., 2015). No entanto na entrada da rede utiliza-se 4 imagens sequenciais, portanto a entrada da rede neural consiste em uma imagem de 84x84x4.

A entrada da rede neural consiste em uma imagem de 84x84x4. A primeira camada escondida e uma rede convolucional de 32 filtros de 8x8 com *stride* 4; a segunda camada escondida é uma rede convolucional de 64 filtros de 4x4 com *stride* 2; a terceira camada escondida e uma rede convolucional de 64 filtros de 3x3 com *stride* 1 seguido por um retificador; a última camada escondida é completamente conectada e consiste em 512 unidades retificadora; e a camada de saída também é completamente conectada com uma saída para cada ação válida. O número de ações válidas varia entre 4 e 18 dependendo do jogo.

As recompensas positivas variam de 0 até 1 e as recompensas negativas variam de 0 até -1. O agente recebe uma recompensa quando ocorre um incremento na pontuação do jogo (MNIH et al., 2015). No jogo *Breakout* por exemplo, cada vez que o agente quebra um bloco, a pontuação aumenta e, portanto, o agente recebe uma recompensa positiva.

O valor do $\epsilon - greedy$ foi decrementado de forma linear entre 1.0 até 0.1 nos primeiros 1 milhão de quadros de imagem, depois permanece em 0.1. Foi treinado um total de 50 milhões de quadros, e o tamanho da memória $\mathcal{D}$ foi de 1 milhão de quadros (a memória sempre contém os quadros mais atuais).

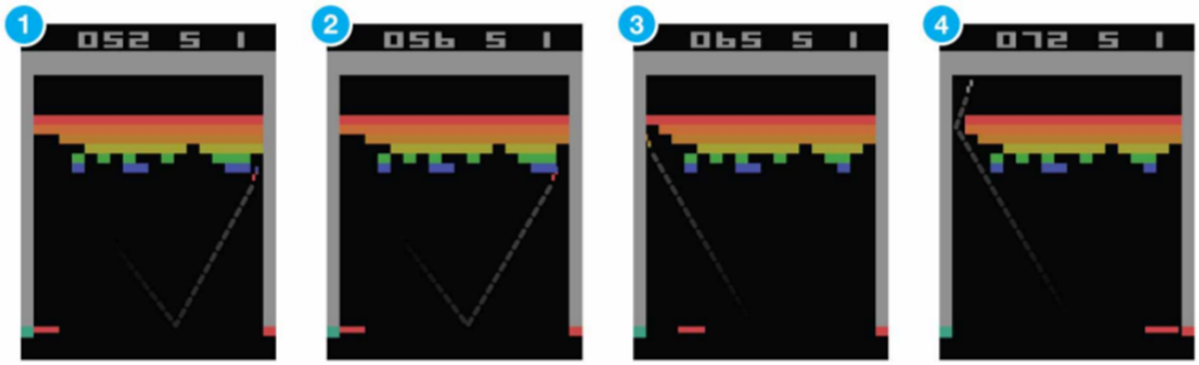
Figura 22 – Arquitetura da DQN em jogos de Atari.



Fonte: MNIH et al., 2015.

O agente observa e seleciona uma ação a cada k quadros ao invés de todos os quadros. Isso reduz a capacidade computacional requerida. Foi utilizado um $k = 4$.

Figura 23 – Imagem do jogo Breakout.



Fonte: MNIH et al., 2015.

Na Figura 23 pode-se ver que o agente com o algoritmo DQN aprendeu a jogar o *breakout* [1] selecionando as ações que fazem com que a bola quebre os blocos, assim recebendo as recompensas positivas cada vez que um bloco é quebrado. No entanto, o que é interessante é que o agente não só aprendeu a quebrar os blocos sem deixar a bola cair, mas também aprendeu que se quebrar os blocos das laterais fazendo com que o bola entre pela lateral, e fique presa na parte superior dos blocos e assim quebrando os blocos superiores, recebe mais recompensa positiva em um menor tempo e ainda sem correr o risco da bola cair.

Tabela 2 – Média do desempenho nos 57 jogos de Atari.

Método	Tempo de Treinamento	Média	Mediana
DQN	8 dias na GPU	121.9%	47.5%
Gorila	4 dias, 100 máquinas	215.2%	71.3%
DDQN	8 dias na GPU	332.9%	110.9%
Dueling DDQN	8 dias na GPU	343.8%	117.1%
Prioritized DQN	8 dias na GPU	463.6%	127.6%
A3C-LSTM	4 dias na CPU	623.0%	112.6%

Fonte: MNIH et al., 2016.

A Tabela 2 mostra o desempenho (pontuação nos jogos) médio dos algoritmos de Aprendizado por Reforço Profundo em 57 jogos de Atari, essa tabela usa como métrica o desempenho do ser humano, portanto o algoritmo DQN apresentou um desempenho de 121.9% superior a um jogador humano. O algoritmo A3C-LSTM foi executado em um computador com 16 núcleos de processamento, enquanto os outros algoritmos foram executados em uma GPU Nvidia K40.

[1] https://www.youtube.com/watch?v=TmPfTpjtdgg

Essa seção 2 apresentou uma breve descrição de Aprendizado Profundo, aprendizado por reforço e Aprendizado por Reforço Profundo. Descrevendo que o Aprendizado Profundo possui diversas técnicas e arquiteturas de aprendizado supervisionado, não supervisionado e aprendizado por reforço. Foi descrito apenas algumas técnicas de aprendizado por reforço, sendo elas as que foram utilizados como base para o desenvolvimento dos algoritmos de Aprendizado por Reforço Profundo, como também o aprendizado por reforço com heurística que poderá ser implementada em Aprendizado por Reforço Profundo. Na seção de Aprendizado por Reforço Profundo foram apresentados os principais e mais recentes algoritmos de Aprendizado por Reforço Profundo.

Na seção 3 será apresentado alguns trabalhos de Redes Neurais Profundas em robótica móvel, e Aprendizado por Reforço Profundo em jogos 3D e em robôs. Esses trabalhos mostram o atual estado da arte do Aprendizado Profundo e estão sendo utilizados como base para as pesquisas realizadas nesta tese.

3 TRABALHOS RELACIONADOS

Nesta seção serão apresentados alguns trabalhos de redes neurais profundas e Aprendizado por Reforço Profundo em jogos 3D e em robôs. Esta seção está dividida em quatro seções secundárias: DNN em robôs móveis; DRL em jogos 3D; DRL para exploração do ambiente com robôs móveis; DRL em robô manipulador.

3.1 DNN EM ROBÔS MÓVEIS

Esta seção apresenta dois trabalhos (GIUSTI et al., 2016; SPECK et al., 2016) com redes neurais profundas em robôs móveis. O trabalho do Giusti et al. (2016) foi utilizado como inspiração para o desenvolvimento dos experimentos que foram realizados e serão apresentados na Seção 5. O trabalho do Speck et al. (2016) mostra como as redes neurais profundas estão sendo utilizadas em robótica móvel.

3.1.1 Controle de um Drone com DNN

No trabalho de Giusti et al. (2016), os autores desenvolveram uma rede neural profunda de 6 camadas com aproximadamente 500 mil neurônios capaz de controlar um drone para se locomover pela trilha de uma floresta, a DNN tem como entrada apenas a imagem em RGB da câmera posicionada na parte da frente do drone.

Os autores também compararam o desempenho da DNN com outras duas técnicas de classificação e também com o ser humano, em relação as outras duas técnicas de classificação a DNN foi superior em precisão, revogação e acurácia, porém apresentou um desempenho comparável ao do ser humano.

Nesse trabalho os pesquisadores realizaram um treinamento supervisionado de uma DNN, uma dificuldade relatada é a grande variação que ocorre entre as imagens devido a diversos fatores, tais como: as diferenças climáticas; diferença na iluminação; os tipos de vegetação; a topografia local; dentre muitos outros fatores. Por isso, houve a necessidade de se incluir uma grande variedade de imagens no conjunto de treinamento (GIUSTI et al., 2016).

Para adquirir o conjunto de imagens, três câmeras foram posicionadas na cabeça de uma pessoa (Figura 24), uma no centro da cabeça posicionada para adquirir imagens do caminho à frente da pessoa, e uma posicionada a 30° a direita e outra a 30° a esquerda adquirindo imagens laterais, as três câmeras juntas cobriam aproximadamente 180° de campo de visão. Essa pessoa

Figura 24 – Câmeras posicionadas na cabeça para gravar vídeos.

Fonte: GIUSTI et al., 2016

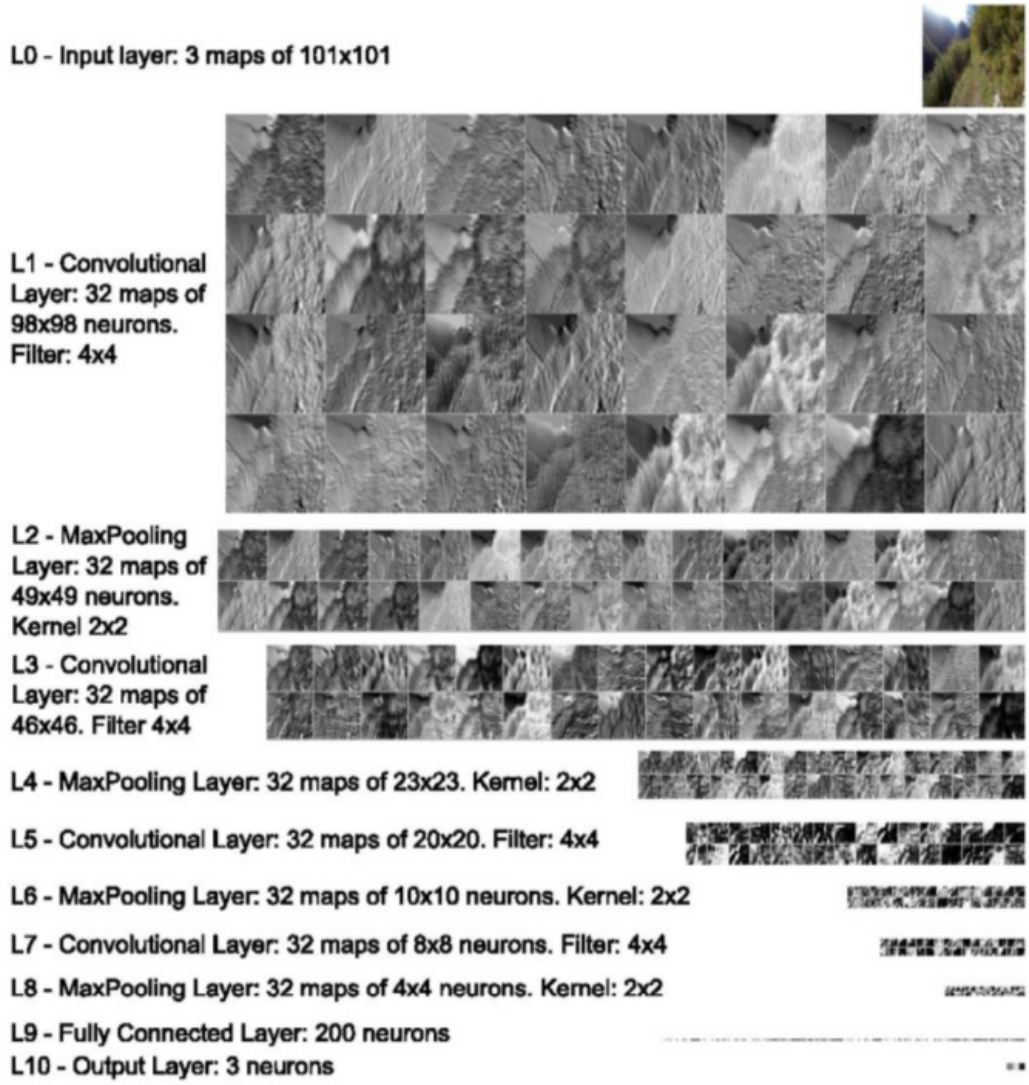
andou pelo caminho da floresta adquirindo imagens para o treinamento. A imagem de cada câmera representou uma classe de imagens do treinamento (GIUSTI et al., 2016).

O conjunto de dados possui 8 horas de gravação em 1920x1080 30fps usando três câmeras GoPro Hero3 Silver, andando aproximadamente 7 quilômetros em diferentes horas do dia e clima, sendo que as imagens foram gravadas somente durante o dia. O conjunto de imagens possui 17119 imagens de treinamento e 7355 imagens de teste (GIUSTI et al., 2016).

A imagem de entrada foi reduzida para 101x101 pixels. A entrada da rede é 3x101x101 (pixels com valores em RGB), seguida por quatro camadas de redes convolucionais com Max-Pooling, uma camada com 200 neurônios completamente conectados e com 3 neurônios na camada de saída, conforme pode ser visto na Figura 25. Os valores dos pixels de entrada foram redimensionados para uma escala de [-1,1]. Foram extraídos dos vídeos 17119 quadros de imagens para ser usado no treinamento, a DNN foi treinada durante aproximadamente 3 dias em um computador com uma GPU GTX 580 conforme descrito pelos autores (GIUSTI et al., 2016).

Com a DNN foi implementado um controle reativo, controlando a rotação *Yaw* do drone proporcional aos valores de saída P(*Turn Right*) e P(*Turn Left*) da DNN, e o valor de P(*Go Straight*) foi usado para controle proporcional da velocidade do drone. Os autores utilizaram um Drone equipado com uma câmera que possui uma resolução de 752x480 pixels, eles também utilizaram *Semi-direct monocular Visual Odometry (SVO)* que é uma técnica de odometria visual (FAESSLER et al., 2015). O SVO e a DNN foram executados em um Odroid-U3, o

Figura 25 – Modelo de DNN.



Fonte: GIUSTI et al., 2016

Odroid processou ambos em uma taxa de 15fps, com isso sendo possível operar o drone de forma autônoma (GIUSTI et al., 2016).

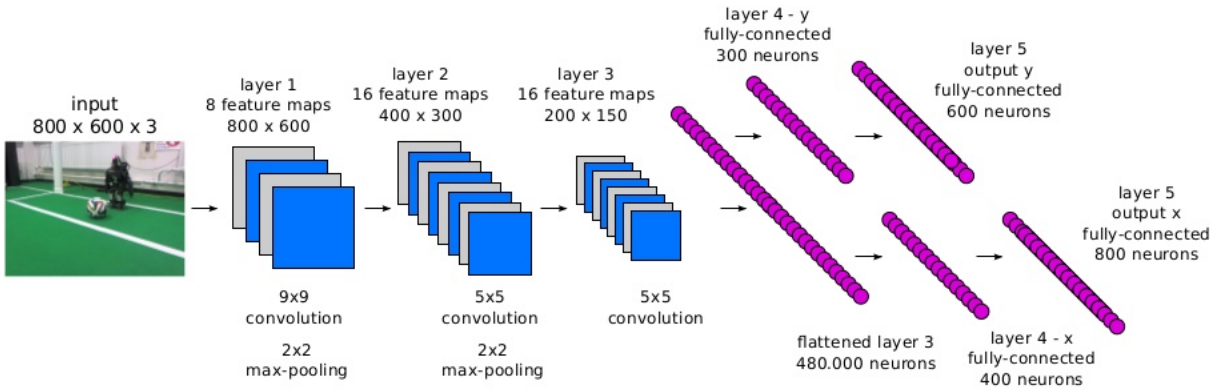
3.1.1.1 Discussão

Neste artigo um dos problemas relacionada a realização dos experimentos, é que o conjunto de imagens foi adquirido com uma câmera GoPro, no entanto foi utilizado a câmera do Drone para executar a rede neural treinada, os próprios autores relatam que houve uma grande diferença de desempenho do classificador entre as câmeras, onde o classificador apresentou um desempenho muito maior com as imagens da GoPro. Para aumentar o desempenho do classificador os autores deveriam realizar o treinamento com as imagens da câmera do drone.

3.1.2 Detecção da localização da bola na imagem usando DNN

O trabalho do Speck et al. (2016) apresenta uma arquitetura de DNN para localizar uma bola no campo da liga humanoide da RoboCup, a localização da bola é informada pela rede neural, que tem como saída a posição x e y da bola na imagem.

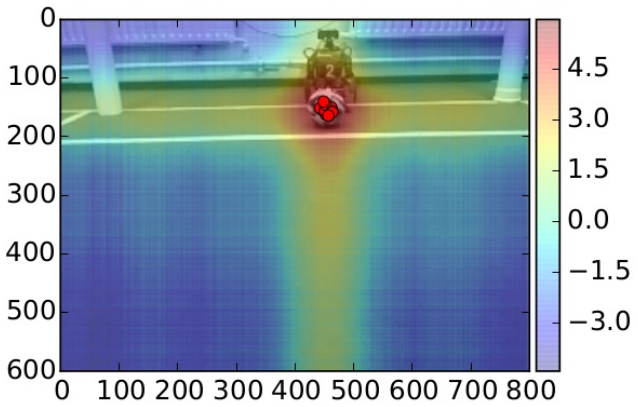
Figura 26 – Arquitetura do modelo com 3 convolucionais.



Fonte: SPECK et al., 2016

Os autores testaram duas arquiteturas de redes, uma com 4 camadas convolucionais e outra com 3, realizando algumas mudanças. A que apresentou um melhor desempenho foi a rede com 3 camadas convolucionais, com *pooling* nas duas primeiras camadas, e três camadas completamente conectada sendo que as duas últimas camadas se separam em duas redes paralelas. Nas últimas camadas foi implementado o *Dropout* com taxa de 0.5. A saída da rede é referente aos eixos x e y da localização da bola na imagem.

Figura 27 – DNN detectando bola na imagem.



Fonte: SPECK et al., 2016

A Figura 27 mostra a detecção da bola realizada pela DNN, as faixas de cores representam os valores da saída da rede.

O conjunto de dados possui 1160 imagens gravadas no campo, onde 80 foram usadas como imagens de testes e 1080 são usadas como imagens de treinamento, as imagens têm uma dimensão de 800x600x3 (largura, altura, canais RGB) sem nenhum pré-processamento.

Os autores informaram que para que a rede possa funcionar no computador do robô humanoide Hambot, a entrada e a saída da rede tiveram que ser redimensionadas, com a entrada 200x150 (largura, altura) e a saída 200x150 referente aos eixos x e y. Com esta configuração a rede classificou cada imagem com uma média de 0.304 segundos. No entanto, com a escala da imagem de entrada em 200x150, a distância entre a câmera e a bola não pode ser maior que 1 metro, sendo que o campo da liga humanoide da RoboCup possui atualmente uma dimensão de 9x6 metros. Então os autores afirmam que há a necessidade de usar computadores com mais poder de processamento.

Pode-se ver que o número de imagens usadas no treinamento são apenas 1160 imagens, isso porque para uma rede identificar e informar a posição do objeto na imagem, é necessário que seja informado em todas as imagens de treinamento, qual a posição do objeto na imagem, sendo que esta operação é realizada de forma manual. Isso faz com que não se utilize muitas imagens devido ao trabalho de ter que selecionar manualmente os objetos em cada imagem.

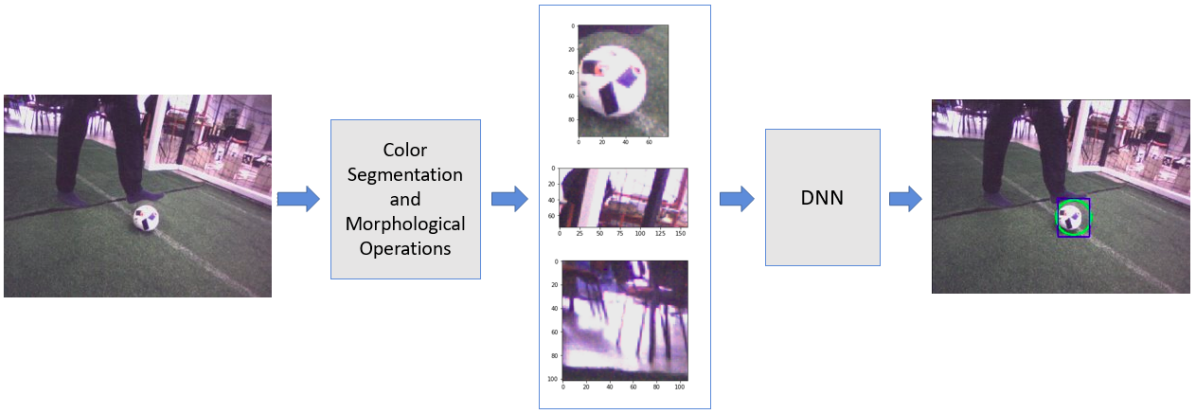
3.1.3 Detecção da bola desenvolvido pelo time RoboFEI

O robô humanoide do time RoboFEI [1] do Centro Universitário FEI utiliza um computador Intel NUC que não tem GPU NVIDIA para executar a DNN, então o sistema de visão desenvolvido em 2017 utilizava a DNN apenas para classificação ao invés de utilizar redes de detecção de objeto, porque uma DNNs de classificação de imagens apresenta um tempo de inferência inferior a uma DNN de detecção de objetos. Portanto, primeiramente o sistema de visão do RoboFEI extrai a Região de Interesse (ROI - do inglês *Region Of Interest*) da imagem, neste processo várias imagens são extraídas (imagens da ROI); estas imagens ROI são encaminhadas para a DNN que infere se alguma das imagens apresentadas é a bola, como mostra a Figura 28.

Neste trabalho, a segmentação de cores e as transformações morfológicas foram utilizadas para extrair a ROI. A segmentação de cores identifica as regiões brancas na imagem gerando uma imagem binarizada dessas regiões. Primeiramente a imagem RGB da câmera do

[1] https://fei.edu.br/robofei/

Figura 28 – Detecção da bola.



Fonte: Autor.

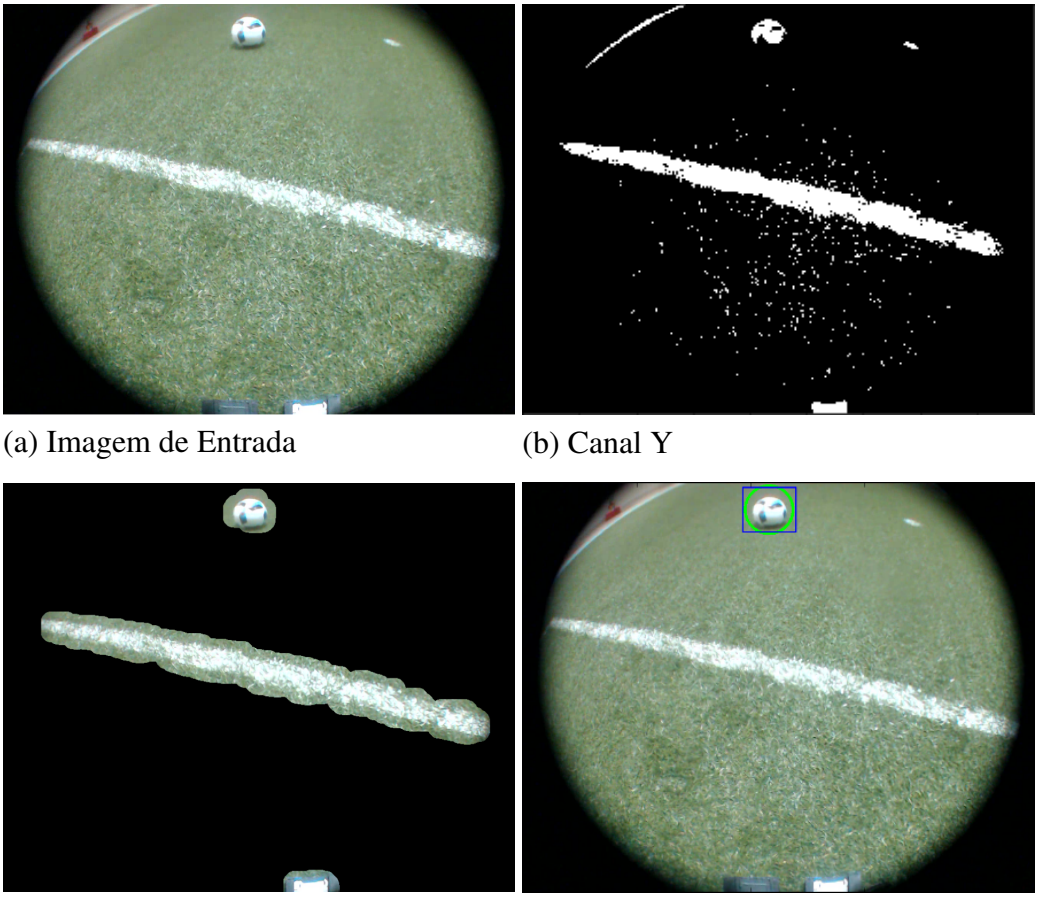
robô é convertida em YUV e o canal Y é extraído, com a imagem do canal Y um *threshold* é utilizado para destacar as regiões brancas, assim binarizando a imagem com as regiões brancas destacadas. Posteriormente uma sequência de dilatação e erosão são realizadas para tratar a imagem excluindo os ruídos e as linhas de campo, como mostra a Figura 29, por fim, os ROIs são enviados para a entrada da DNN que classifica se há alguma imagem de bola. Para realizar as transformações morfológicas o quadro da imagem é dividido em quatro regiões verticais, relacionadas à distância entre a bola e o robô, essas regiões foram criadas para aplicar diferentes *kernels* das transformações morfológicas em cada região, isso porque o tamanho da bola e das linhas de campo diminuem proporcionalmente em relação a distância do robô.

Várias arquiteturas de rede foram testadas, a arquitetura apresentada na Tabela 3 foi a que apresentou uma alta acurácia de 99.95% (uma das soluções para a baixa acurácia foi adicionar mais imagens no conjunto de treinamento). O objetivo foi desenvolver uma arquitetura que consuma pouco tempo de inferência (um tempo inferência suficiente para utilizar a DNN em um intel NUC que atenda à dinâmica do jogo), mas com alta acurácia, uma arquitetura de rede neural profunda simplificada, a fim de gastar menos tempo para treinar e classificar as imagens. A arquitetura desenvolvida possui 128 neurônios na camada totalmente conectada e apenas 2 camadas de redes convolucionais. A função de ativação ReLU foi aplicada na saída das camadas de rede convolucional e nas camadas de redes completamente conectada, já o *dropout* foi implementado apenas nas camadas completamente conectadas.

A rede foi criada e treinada no DIGITS-NIDIA [2]. O treinamento da rede foi realizado em um computador Intel i7-7500U de 2,7 GHz, 16 GB de memória RAM DDR4, NVIDIA GeForce

[2]https://github.com/NVIDIA/DIGITS

Figura 29 – Processo de detecção da bola



(a) Imagem de Entrada (b) Canal Y

(c) Possíveis imagens de bola (d) Classificação realizada pela DNN

Fonte: Autor.

Tabela 3 – Arquitetura.

Layer	**Entrada**	**Kernel**	**Stride**	**N. Kernels**	**Pad**	**Saída**
Conv1	32x32x3	5x5	1	20	0	28x28x20
Pool1	28x28x20	3x3	2	20	0	14x14x20
Conv2	14x14x20	5x5	1	32	2	14x14x32
Pool2	14x14x32	3x3	2	32	0	7x7x32
Fc1	7x7x32			128		128
Fc2	512			2		2

GTX 940MX 4 GB, Linux Ubuntu 16.04. A DNN foi implementada usando a linguagem de programação C++ e Python. Depois de treinado, a DNN foi testada no Intel NUC.

As imagens foram extraídas da câmera do robô, com o robô no campo de futebol. O conjunto de dados tem 40.000 imagens de treinamento e 4.000 imagens de teste (sem *data augmentation*). Existem imagens com vários tamanhos (largura e altura). A fim de verificar o desempenho do DNN treinado, 4000 imagens de teste foram separadas em duas classes, 3000

imagens na classe sem bola e 1000 imagens na classe bola. O conjunto de dados está disponível no google Drive [3].

Tabela 4 – Matriz de Confusão.

	Ball	**noBall**	**acurácia por classe**
Ball	998	2	99.80%
noBall	0	3000	100.0%

A Tabela 4 mostra a matriz de confusão da arquitetura apresentada na Tabela 3. O tempo gasto na DNN para classificar uma imagem em um Intel NUC é de aproximadamente 10 milissegundos. Usando 4000 imagens de teste, a DNN apresentou uma alta acurácia, e usando esta DNN para classificar as imagens de ROI, o sistema de visão detecta a bola na imagem.

A desvantagem deste sistema é que devido ao processo utilizado para extrair as imagens da região de interesse, o sistema de visão detecta a bola a uma distância máxima de um pouco mais de um metro. No entanto, esse trabalho foi realizado em parceria com o time Rhoban [4] do laboratório LaBRI [5] (do francês *Laboratoire Bordelais de Recherche en Informatique*) da Universidade de Boudeaux da França, o time Rhoban utiliza uma outra técnica de extração de imagens da região de interesse, dessa forma o sistema de visão deles detecta a bola um pouco mais distante que o apresentado neste trabalho.

3.1.4 Discussão

No primeiro trabalho apresentado nessa seção (trabalho da seção 3.1.1) a DNN junto com uma odometria visual foi desenvolvida para controlar um Drone para se locomover pela trilha de uma floresta, mostrando que as DNNs podem ser utilizadas para tomada de decisões, os resultados apresentados neste artigo foi utilizado como inspiração para esta tese. Já nos trabalhos das seções 3.1.3 e 3.1.2 a DNN foi utilizada apenas para detecção de bola na imagem, sendo estes dois últimos o uso mais comum das DNNs em sistemas robóticos como pode ser visto em outros trabalhos (TEAM. . . , 2017; JAVADI et al., 2017; ALBANI et al., 2016; SCHNEKENBURGER et al., 2017; DIJK; SCHEUNEMANN, 2019). Portanto, esta tese estende a aplicação das DNNs para a cognição robótica.

[3] https://drive.google.com/drive/folders/1tan78Q0GocKsjcJ57hrPHPaMgMxthCCH?usp=sharing
[4] http://rhoban.com/
[5] https://www.labri.fr/

3.2 DRL EM JOGOS 3D

Alguns trabalhos com Aprendizado por Reforço Profundo mostram a capacidade do DRL de aprender a jogar jogos em ambientes 3D (KEMPKA et al., 2016; LAMPLE; CHAPLOT, 2016; JADERBERG et al., 2016; MNIH et al., 2016; MIROWSKI et al., 2016). Esses trabalhos mostraram que o DRL aprendeu a tomar decisões nos ambientes tridimensionais a partir dos pixels das imagens dos jogos.

No trabalho de Kempka et al. (2016), os pesquisadores desenvolveram uma versão do jogo de videogame Doom chamado VizDoom (Figura 30) para ser usando como plataforma de testes de algoritmos de aprendizado de máquina em um jogo com ambiente tridimensional. Doom é um jogo de tiro de primeira pessoa. Os pesquisadores realizaram dois experimentos com DQN: no primeiro experimento o agente tem que aprender a atirar nos personagens que surgem no ambiente; no segundo experimento os agentes tem que aprender a navegar em um labirinto mais complexo. Assim demonstrando que uma DQN é capaz de aprender essas tarefas em ambientes 3D.

Figura 30 – Imagem do jogo VizDoom.

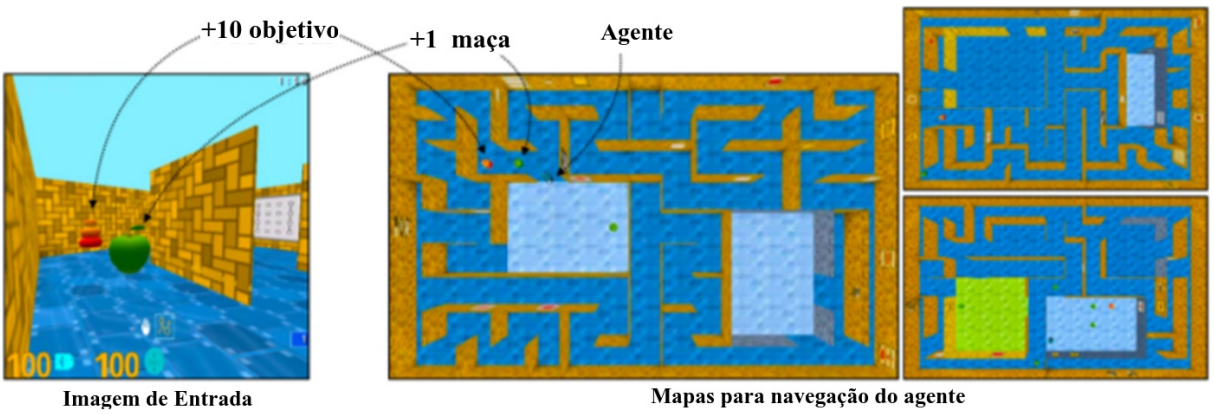
Fonte: KEMPKA et al., 2016

No trabalho de Mirowski et al. (2016), os pesquisadores do *Deep Mind* da Google testaram DRL em um jogo de labirintos 3D [6] (Figura 31), mostrando que o agente é capaz de aprender a navegar no ambiente, mesmo em condições onde a localização do objetivo muda

[6] https://youtu.be/JL8F82qUG-Q

frequentemente. Nesse trabalho os autores fornecem uma análise detalhada do comportamento do agente e a sua capacidade de se localizar, mostrando que o agente aprende as principais habilidades de navegação neste jogo.

Figura 31 – Imagem do jogo de labirintos 3D.



Fonte: JADERBERG et al., 2016

3.3 DRL PARA EXPLORAÇÃO DO AMBIENTE COM ROBÔS MÓVEIS

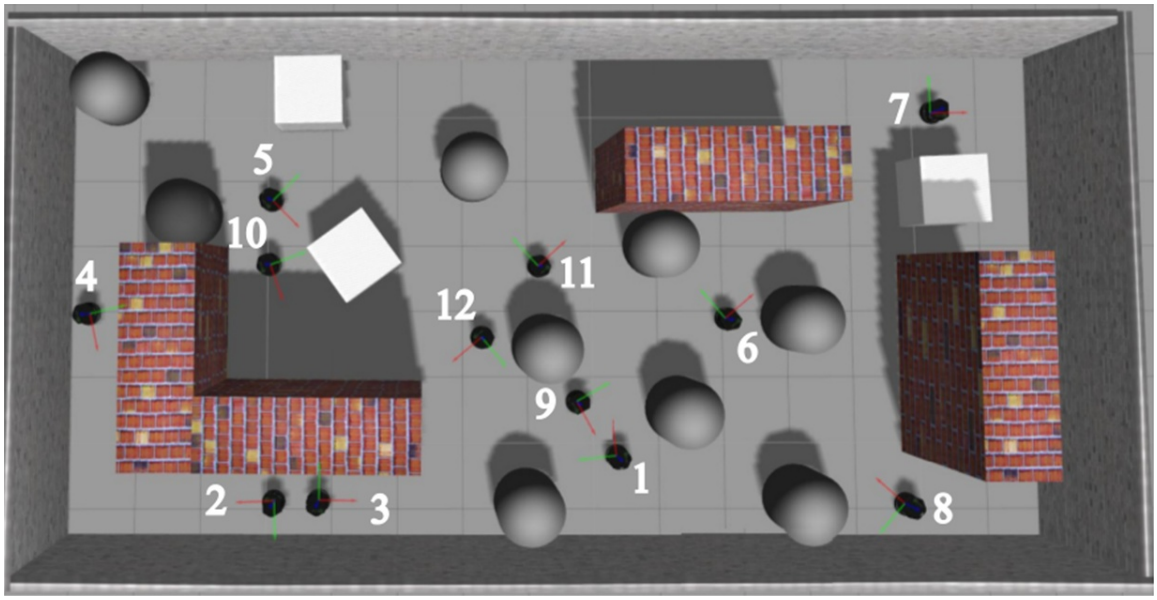
No artigo do Tai e Liu (2016b) os autores utilizaram DQN para que um robô móvel possa aprender a navegar por um ambiente desviando dos obstáculos, esses mesmos autores desenvolveram anteriormente alguns trabalhos com DL em navegação de robôs móveis (TAI; LI; LIU, 2016; TAI; LIU, 2016a,b). Neste artigo (TAI; LIU, 2016b) os autores acreditam que é a primeira vez que um robô móvel desenvolve a capacidade de navegação através do Aprendizado por Reforço Profundo de ponta a ponta usando apenas a informação do sensor em dados bruto (sem qualquer pré-processamento). Neste caso o sensor usado foi um *kinect* que fornece dados da câmera em RGB-D.

Os autores utilizaram o robô *Turtlebot* com um *kinect* em cima do robô. O ambiente para navegação foi implementado no simulador Gazebo[7], contendo vários obstáculos. A Figura 32 mostra que foram posicionados 12 pontos onde o robô poderá inicializar, o ponto escolhido para o robô iniciar a navegação ocorre de forma aleatória (TAI; LIU, 2016b).

No experimento o robô recebe uma recompensa positiva (1) quando age sem esbarrar no obstáculo e recebe uma recompensa negativa (-100) quando esbarra no obstáculo. A rede recebe como entrada apenas a profundidade da imagem fornecida pelo canal D da imagem em

[7] http://gazebosim.org

Figura 32 – Ambiente implementado para realizar a navegação.



Fonte: TAI; LIU, 2016

RGB-D, e a saída da rede corresponde a cinco ações de virar ou andar para frente: esquerda, direita, meio esquerda, meio direita e andar para frente.

A Figura 33 apresenta a trajetória percorrida pela agente. Com apenas 500 interações o agente aprende a girar em torno dele mesmo, isso ocorre porque o agente recebe uma recompensa positiva toda vez que executa uma ação sem esbarrar nos obstáculos, neste caso, o agente deveria receber uma recompensa negativa, e só deve receber uma recompensa positiva quando atinge o objetivo, sendo que neste caso, o objetivo não é simplesmente não esbarrar no obstáculo, o objetivo é desviar dos obstáculos realizando uma navegação pelo ambiente.

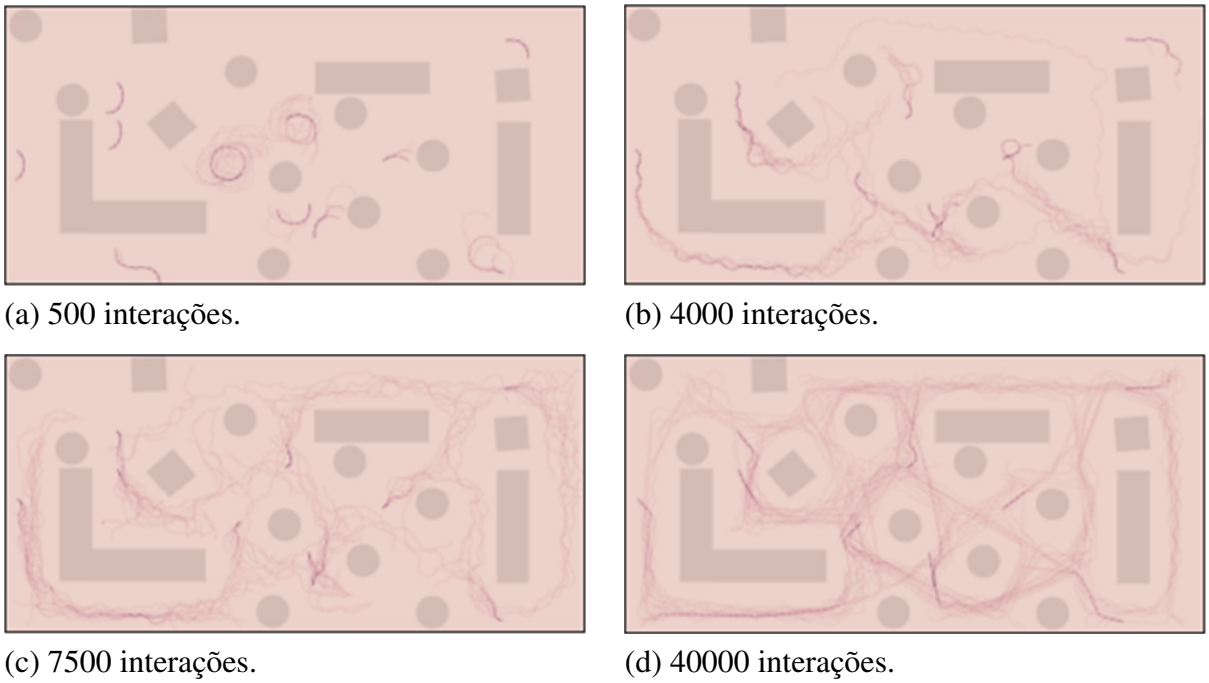
3.4 DRL EM ROBÔ MANIPULADOR

Alguns trabalhos (LEVINE et al., 2016; YAHYA et al., 2017; GU et al., 2017) demonstram que um robô real manipulador com Aprendizado por Reforço Profundo pode aprender algumas tarefas de manipulação de objetos ou até mesmo abrir uma porta.

O artigo do Gu et al. (2017) demonstrou que com Aprendizado por Reforço Profundo o robô pode aprender a realizar tarefas como abrir uma porta. Conforme apresentado na Figura 34.

Os autores demonstram que é possível reduzir o tempo de treinamento executando o treinamento em vários robôs que agrupam suas atualizações de políticas de maneira assíncrona.

Figura 33 – Trajetória percorrida pelo agente.

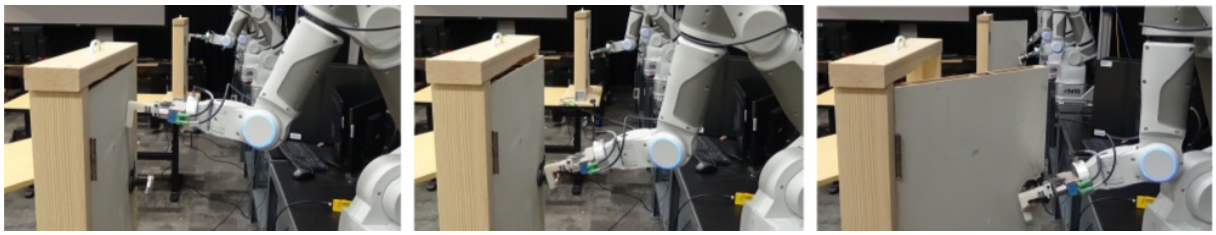


(a) 500 interações. (b) 4000 interações.

(c) 7500 interações. (d) 40000 interações.

Fonte: SZEGEDY et al., 2015.

Figura 34 – Robô manipulador aprendendo a abrir uma porta.



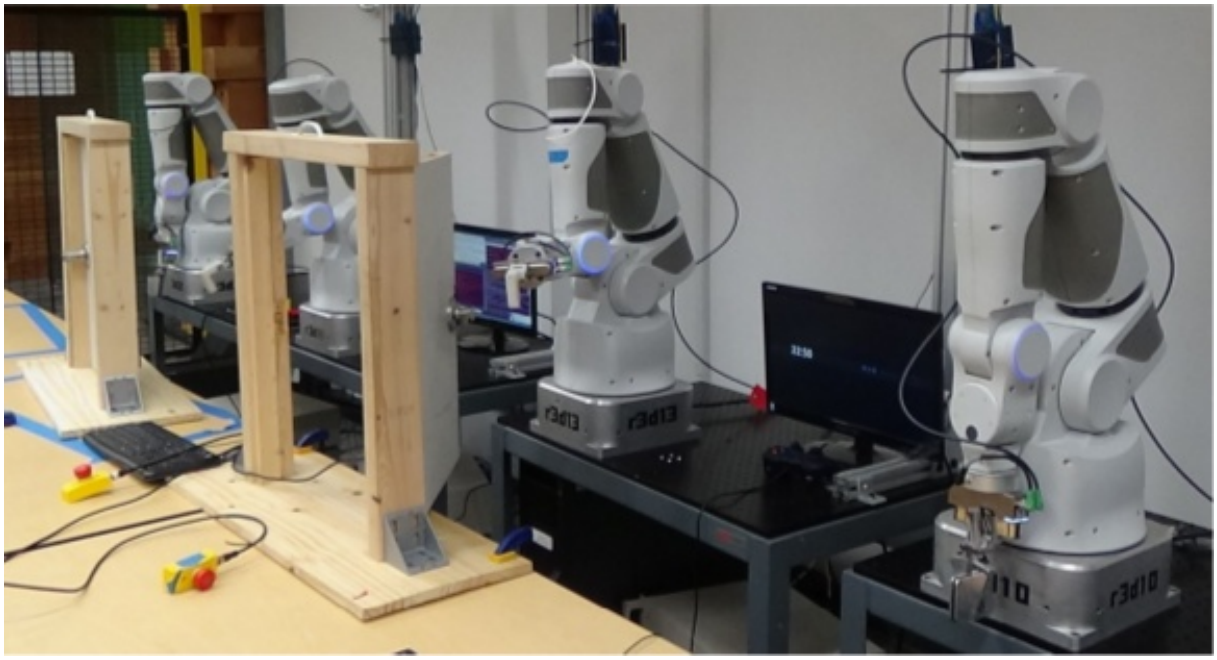
Fonte: GU et al., 2017

O trabalho também mostra que o robô manipulador pode aprender a executar uma tarefa complexa, como abrir uma porta, sem que seja necessário apresentar demonstrações ou representações realizadas de forma manual.

Utilizando apenas dois manipuladores, os robôs aprendem a abrir a porta com 100% de acerto em apenas duas horas e meia. Já usando apenas um robô, demanda 4 horas de treinamento para aprender a abrir a porta com 100% de acerto.

Nessa seção foram apresentados alguns trabalhos com Aprendizado Profundo, esses trabalhos demonstraram o uso de DL em diferentes domínios, como em robótica e em jogos 3D. Apesar dos algoritmos apresentarem algumas diferenças entre esses dois domínios, esses

Figura 35 – Robôs compartilhando a política para aprender a mesma tarefa.



Fonte: GU et al., 2017

trabalhos demonstram os avanços que o Aprendizado Profundo tem apresentado nessas áreas. A próxima seção apresenta o trabalho dessa tese.

4 SISTEMAS COGNITIVOS PARA AGENTES ROBÓTICOS BASEADOS EM APRENDIZADO PROFUNDO

Esta tese apresenta sistemas com aprendizado profundo para substituir alguns dos diversos processos que são utilizados em agentes robóticos, sistemas que possam diminuir a complexidade que as atuais arquiteturas de software possuem. Sistema Cognitivo com Aprendizado Profundo, no intuito de proporcionar a agentes robóticos a capacidade de tomar decisões apenas pela observação do ambiente através da câmera do robô.

Esses sistemas proporciona a agentes robóticos a capacidade de tomar decisões apenas pela observação do ambiente. Política das tarefas que são aprendidas pela interação com ambiente (DRL) ou/e por meio de aprendizado supervisionado (DNN). Assim substituindo arquiteturas que possuem diversos algoritmos responsáveis por diversas tarefas.

4.1 DEFINIÇÃO DO PROBLEMA

Os atuais robôs móveis capazes de atuar autonomamente (PERICO et al., 2014; KIM, S. et al., 2015; TEAM. . . , 2017; LIM et al., 2017; JUMEL et al., 2018) possuem uma arquitetura de software composta por diversos processos responsáveis por atividades especificas, tais como: detecção de objetos, sistemas de localização, *path planner*, processo de decisão. O problema é que todos esses processos são desenvolvidos e configurados pelo próprio pesquisador, e em certos casos não atingindo a solução ótima para a tarefa a ser solucionada, assim apresentando uma arquitetura complexa e de difícil configuração para a realização de uma tarefa simples.

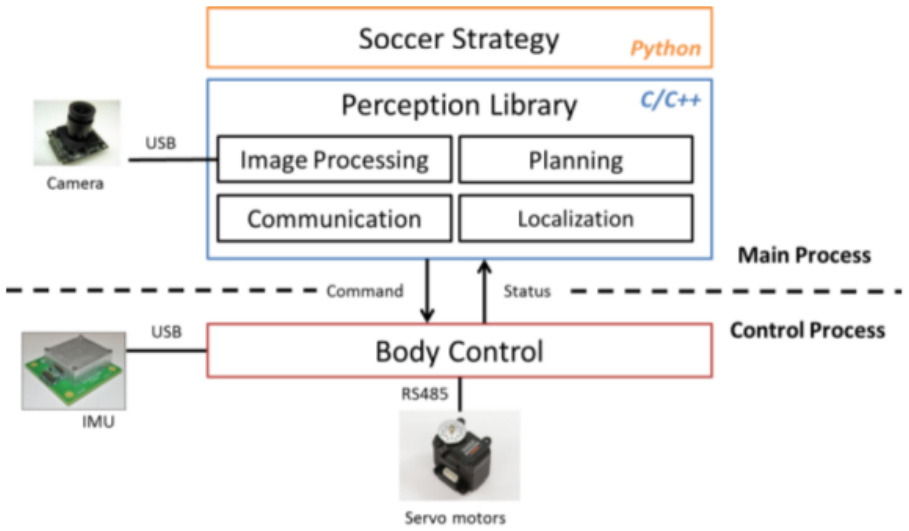
Em algoritmos que não utilizam a abordagem do Aprendizado Profundo os pesquisadores combinam diversos algoritmos, onde em cada algoritmo o operador humano descreve formalmente o conhecimento que o computador precisa para executar uma tarefa, formando uma hierarquia de conceitos, sendo que esta hierarquia permite ao computador aprender ou executar tarefas complexas. Uma outra solução é permitir aos computadores construir essa hierarquia de conceitos através do aprendizado. Ao se desenhar um grafo mostrando como esses conceitos são construídos um sobre o outro, o grafo será profundo e com muitas camadas. Por esta razão, chama-se esse tipo de abordagem de Aprendizado Profundo (GOODFELLOW; BENGIO; COURVILLE, 2016).

Os algoritmos de DL que tem sido apresentados recentemente, tornou-se o estado da arte de diversas áreas como na área de detecção de objetos (KRIZHEVSKY; SUTSKEVER; HINTON, G. E., 2012; SZEGEDY et al., 2015; HU; SHEN; SUN, 2018; HOWARD et al.,

2017), isso faz com que o DL seja uma técnica promissora na área de robótica, com isso muitos pesquisadores tem implementado esses algoritmos em seus robôs. Na competição da RoboCup, algumas equipes (TEAM..., 2017; SPECK et al., 2016) estão implementando Redes Neurais Profundas para detecção de objetos, no entanto, essas mesmas equipes estão usando a DNN apenas como uma técnica de detecção de objeto, substituindo em suas arquiteturas os antigos algoritmos de visão computacional pela DNN.

A Figura 36 mostra a arquitetura de software de um dos robôs do time CITBrains da RoboCup (campeão de 2015 da RoboCup categoria *humanoid kid size*) em que os autores afirmam que implementaram uma DNN para detecção da bola, gols e robôs humanoides para a RoboCup de 2017 (TEAM..., 2017). Pode-se ver que nessa arquitetura há diversos processos, onde cada processo é responsável por algum processamento específico. E esses processos compartilham algumas informações durante a execução de determinadas tarefas, o problema é que algumas configurações são realizadas manualmente pelo pesquisador ou operador, e essas configurações são realizadas separadamente, porém alguns processos precisam operar sincronizados com outros processos, portanto sendo uma arquitetura de difícil configuração.

Figura 36 – Arquitetura de software do robô humanoide do CITBrains.



Fonte: , 2017

As empresas que estão desenvolvendo carros autônomos ainda estão utilizando abordagens tradicionais para algoritmos de tomada de decisões, utilizando a DNN apenas como uma técnica de visão computacional para percepção (HOW..., 2017).

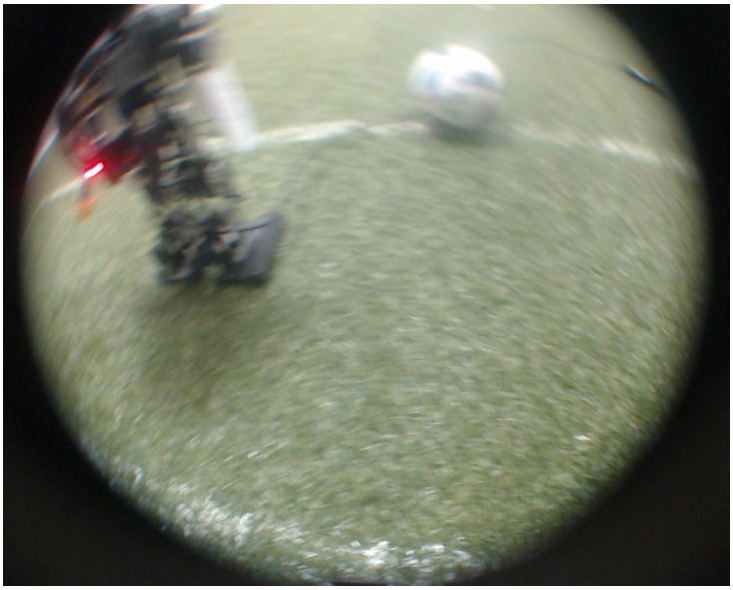
Uma dificuldade que pesquisas em robótica enfrentam quando necessitam que o robô detecte objetos em ambiente reais, é que a técnica de detecção deverá ser capaz de identificar

objetos, independente das variações de iluminação do ambiente que o objeto estiver exposto e dentre outras variações do ambiente. Um bom exemplo disso, são os carros autônomos, que utilizam técnicas de visão computacional para detectar vários objetos em diversos ambientes e situações climáticas. Por exemplo, um carro autônomo deverá detectar o semáforo, independente se está sobre incidência do sol ou na sombra, se o clima local está chuvoso ou ensolarado, se é dia ou noite. Esse mesmo problema pode ser observado no trabalho do Giusti et al. (2016) que foi apresentado na seção 3.1.1 onde os autores relatam a dificuldade de treinar uma rede utilizando imagens proveniente de ambientes reais, especificamente de uma floresta.

Portanto, tratando-se de identificar objetos ou contexto em ambientes reais (robôs em ambientes reais), a técnica utilizada deverá ser capaz de adaptar-se às diversas variações do ambiente. Existem alguns outros problemas que também influenciam na detecção de objetos, como por exemplo o uso de uma câmera *Full-HD* ou uma câmera VGA, câmeras com controle de auto brilho, sendo essas algumas das limitações relacionadas ao hardware utilizado.

Na detecção de objetos em robôs móveis onde a câmera encontra-se posicionada no próprio robô, haverá problemas relacionados a movimentação do robô que também influenciam na detecção de objetos, onde algumas imagens podem apresentar borramento quando o robô realiza determinados movimentos, como mostra a Figura 37.

Figura 37 – Imagem com borramento capturada pela câmera do robô.



Fonte: Autor.

No artigo do Ersen, Oztop e Sariel (2017) os autores relatam que apesar de estudos recentes apresentarem passos importantes no desenvolvimento de sistemas que possam atuar e

aprender autonomamente, um dos problemas que estão em abertos no campo da robótica é a necessidade de realizar-se trabalhos que implantem robôs em ambientes não laboratoriais, como por exemplo carros autônomos que devem atuar diretamente nas ruas, ou competições como a RoboCup e o desafio DARPA que fazem com que o robô atue em um ambiente fora do seu laboratório. Para que os robôs operem nessas situações de forma satisfatória, exige-se que esses robôs sejam equipados com habilidades avançadas de aprendizado que lhes permitam lidar com situações novas e também evite que o robô possa repetir erros já cometidos.

4.2 DEFINIÇÃO DO AGENTE ROBÓTICO UTILIZADO NESTA PESQUISA

Nesta seção será apresentada as principais características de hardware e software do agente robótico, e também o simulador utilizado nesta pesquisa.

Figura 38 – Robôs Humanoides.

Fonte: Autor.

Nesta tese foram utilizados dois modelos de robôs humanoides, um robô real desenvolvido no Centro Universitário FEI que pode ser visto na Figura 38, e um robô humanoide chamado DARwIn-OP (HA et al., 2011) disponível no simulador Webots (SITE. . . , 2019).

Tabela 5 – Características do robô humanoide

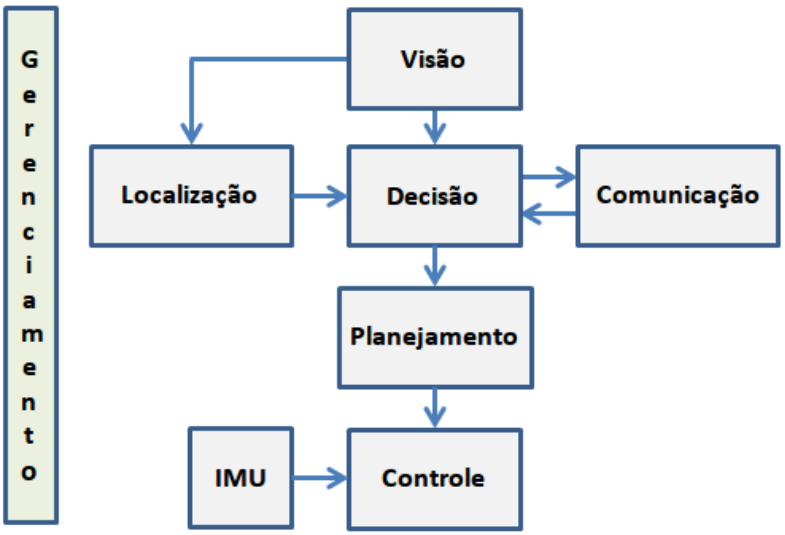
Altura	450 mm
Peso	3.0 Kg
Velocidade do caminhar	10 cm/s
Graus de liberdade	20 no total: 6 em cada perna, 3 em cada braço, 2 na cabeça
Tipo de motor	Dynamixel RX-28
Sensor	UM6 Ultra-Miniature Orientation Sensor
Câmera	Logitech HD Pro Webcam C920 (Full HD)
Computador	Intel NUC Core i5-4250U, 8GB SDRAM, 120GB SDD
Bateria	1 bateria de Li-Po 2Ah, autonomia de 30 minutos

Fonte: Autor.

Os robôs humanoides utilizados nesta tese foram desenvolvidos no Centro Universitário FEI, esses robôs tem a estrutura mecânica e a cinemática baseada no DARwIn-OP (HA et al., 2011). Os robôs possuem 20 graus de liberdade, uma altura de 490mm e um computador Intel NUC, conforme Tabela 5.

Atualmente a arquitetura do software (Figura 39) do robô humanoide desenvolvido no Centro Universitário FEI possui a característica descrita no artigo de Perico et al. (2014), onde os processos de planejamento e decisão recebem informações dos processos de localização, visão e comunicação, e o processo de controle recebe informações do processo de decisão e IMU.

Figura 39 – Arquitetura do robô humanoide desenvolvido no Centro Universitário FEI.



Fonte: PERICO et al., 2014

Os interesses por realizar pesquisas em robôs humanoides, consiste no desejo e necessidade de substituir os humanos em diversas atividades, tais como: atividades perigosas (mineração, energia nuclear, intervenções militares, atividades perigosas em ambientes de desastres), como para o auxílio em atividades do cotidiano (atendentes, auxiliar em atividades doméstica e repetitivas). Uma das vantagens dos robôs humanoides se dá pelo fato de que a locomoção com pernas ser a melhor forma de locomoção em ambientes com descontinuidades no piso, tais como degraus e pedras (WESTERVELT et al., 2007).

Muitos experimentos foram realizados em um simulador de robótica, por ser um ambiente virtual que proporciona testar rapidamente algoritmos usando cenários e robôs próximos da realidade. A vantagem do simulador é poder realizar testes utilizando qualquer sensor ou robôs que forem necessários, sem danificar os robôs e economizar o tempo, devido a possibilidade de acelerar a simulação. A desvantagem do uso do simulador é que atualmente os simuladores ainda não apresentam um fiel modelo físico do robô ou ambiente.

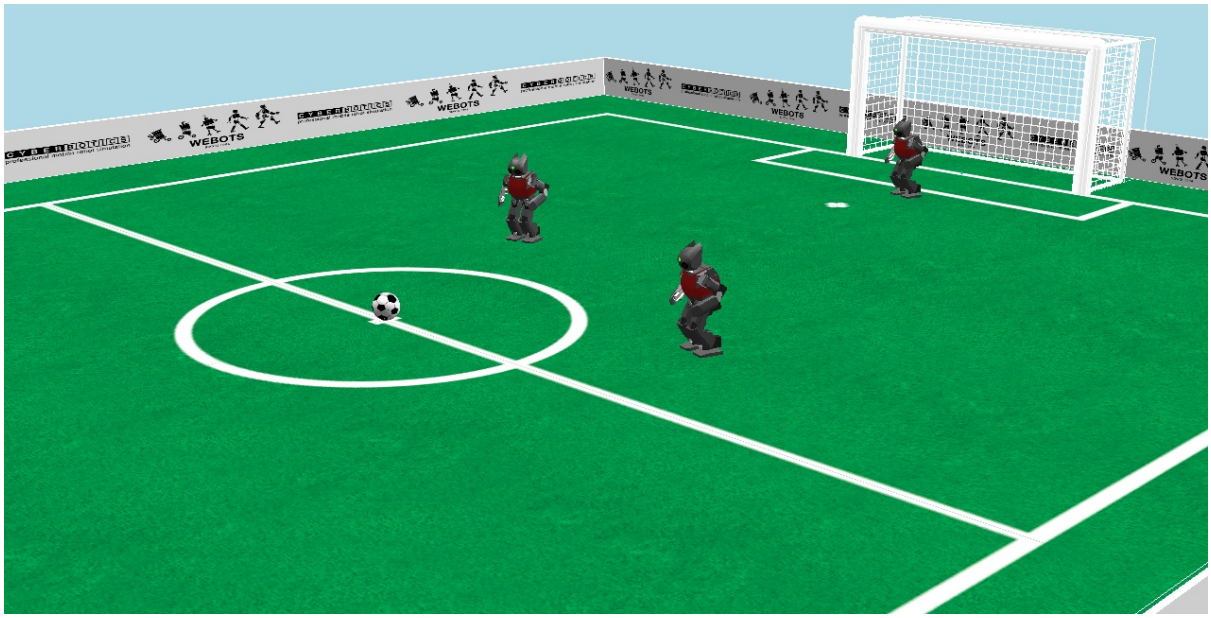
Mesmo que os simuladores de robótica ainda não apresentem um fiel modelo físico do robô e ambiente, a simulação possui diversas vantagens tais como: não há o risco de danificar o robô (caso haja algum problema com o robô durante a simulação, pode-se reiniciar a simulação); Não há necessidade de recarregar a bateria; É possível acelerar a simulação, assim reduzindo o tempo dos experimentos; Os algoritmos podem ser testados nos robôs antes da realização dos testes em robôs reais. Várias morfologias de robôs podem ser testadas antes da construção do robô real.

Neste trabalho foi utilizado o simulador Webots (Figura 40). Webots (SITE..., 2019) é um simulador de robótica da empresa Cyberbotics, um simulador gratuito e de código aberto [1]. O Webots oferece um ambiente virtual tridimensional com propriedades físicas. Esse simulador disponibiliza uma grande variedade de sensores e uma vasta família de robôs. O Webots utiliza a ODE personalizada como motor de física. O Motor de física (do inglês *physics engine*) é o que simula a física em modelos tridimensionais, simulando as condições físicas de um ambiente real. ODE (SITE..., 2018) (do inglês *Open Dynamics Engine*) é uma biblioteca de código aberto de alto desempenho para simular a dinâmica de corpo rígido.

Uma simulação no Webots é composta de três componentes principais: Arquivo Mundo (Arquivo com extensão wbt); Controlador; Plugin físico. Um mundo no Webots é uma descrição das propriedades de cada robô e do seu ambiente. Um controlador é um programa que é responsável pela movimentação do robô através do controle do acionamento dos motores.

[1] https://github.com/omichel/webots

Figura 40 – Simulador Webots.



Fonte: Autor.

No caso do robô humanoide, o controle possui um gerador de caminhadas e também uma programação responsável por fazer o robô levantar, chutar a bola e outras movimentações. Os controladores podem ser escritos nas linguagens de programação suportadas pelo Webots: C, C++, Java, Python ou Matlab. Há também a possibilidade de criar um controlador usando *Robot Operating System* (ROS). O Plugin físico é o ODE (SITE..., 2018). No Webots também há o controlador Supervisor, que é um tipo privilegiado de operações que normalmente só podem ser realizadas por um operador humano, e que também pode ser escrita em qualquer uma das linguagens de programação suportadas pelo Webots.

Neste trabalho o supervisor foi escrito em linguagem Python com a finalidade de auxiliar os processos de aprendizagem. No Sistema com Aprendizado por Reforço Profundo o controlador Supervisor foi utilizado para finalizar o epiśodio caso o número máximo de passos por episódio tenha sido atingido, ou caso a bola tenha saído do campo ou entrado no gol. Depois de finalizado o episódio o controlador Supervisor reposiciona a bola e o robô, e inicia um novo episódio.

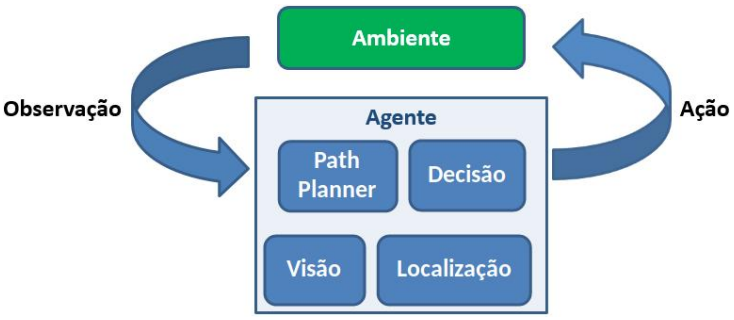
O processo Supervisor também foi utilizado para calcular a recompensa de acordo com a função de recompensa. Nesse caso, o processo Supervisor monitorava a posição da bola no campo e a posição e orientação do robô no campo, isso para realizar o cálculo de distância entre o robô e a bola, verificar se o robô estava de frente ou de costas para o gol, e também monitorar se a bola estava posicionada dentro do campo, fora do campo ou dentro do gol.

4.3 SISTEMAS COGNITIVOS PARA AGENTES ROBÓTICOS COM REDES NEURAIS PROFUNDAS

O objetivo do Sistema com Redes Neurais Profundas é substituir as arquiteturas que possuem diversos processos responsáveis por diversas tarefas utilizadas na cognição de agentes robóticos, assim diminuindo a complexidade da arquitetura do agente robótico.

A Figura 41 mostra o exemplo de uma arquitetura de software de um robô composta por diversos processos responsáveis por atividades especificas, tais como: detecção de objetos, sistemas de localização, *path planner*, processo de decisão, dentre outros (Alguns outros processos não foram apresentados na Figura porque não são substituídos pelo Sistema de Cognição com DNN). E esses processos compartilham informações para realizar algumas determinadas tarefas. No entanto o problema é que todos esses processos devem ser desenvolvidos e configurados pelo próprio pesquisador, e em certos casos não atingindo a solução ótima para a tarefa a ser solucionada, devido à complexidade na configuração, isso porque alguns desses processos precisam operar sincronizados com outros processos, dificultando a sua configuração.

Figura 41 – Agente interagindo com ambiente.



Fonte: Autor.

Esse modelo foi desenvolvido para unificar os processos de visão e decisão através da utilização de uma rede neural profunda. A rede neural profunda recebe uma imagem proveniente da câmera do robô e classifica essa imagem de uma forma que a classificação da imagem seja a ação que o robô deverá executar (Figura 42). Nesse modelo a rede neural profunda é treinada com aprendizado supervisionado.

As vantagens deste modelo são:

a) O robô toma a ação correspondente a imagem observada, sem que haja qualquer pré-processamento na imagem usada na entrada da DNN.

Figura 42 – Modelo proposto com Redes Neurais Profundas.

Fonte: Autor.

b) Durante o treinamento (aprendizado supervisionado) não há a necessidade de informar onde está o objeto na imagem. Quando se usa um algoritmo para identificar um objeto na imagem e também sua posição, durante o treinamento é necessário informar para o algoritmo onde está na imagem o objeto que se deseja identificar, sendo que esta operação é realizada de forma manual.

c) Neste modelo foi usado uma câmera com olho de peixe para aumentar o campo de visão da câmera, com isso foi possível eliminar os servos do *pan* e do *tilt*.

d) Em diversas pesquisas, o DNN tem mostrado ser uma técnica de visão mais eficiente que as demais técnicas de visão que não fazem parte das técnicas de Aprendizado Profundo, sendo atualmente alvo de pesquisa para quem trabalha com visão computacional em robótica, como pode ser visto em alguns trabalhos (TEAM. . . , 2017; JAVADI et al., 2017; ALBANI et al., 2016; SCHNEKENBURGER et al., 2017; DIJK; SCHEUNEMANN, 2019).

Uma desvantagem é que neste modelo há a necessidade de capturar imagens provenientes da câmera do robô e no ambiente em que ocorrerá a navegação, para que possa ser realizado o treinamento da rede. Outra desvantagem é que alguns processos deverão ser integrados com a DNN, por exemplo o processo de comunicação do robô com outros robôs ou também a leitura dos sensores inerciais (IMU).

Nesta Tese foram testadas algumas conhecidas arquiteturas de DNNs, tais como: GoogLeNet; AlexNet e SqueezeNet; e também foram testadas algumas arquiteturas com três e com quatro camadas de redes convolucionais e variações na quantidade de neurônios das camadas completamente conectadas.

4.3.1 Árvore de Decisão de Redes Neurais Profundas

Alguns trabalhos mostram (SPECK et al., 2016; GIUSTI et al., 2016; DIJK; SCHEUNEMANN, 2019) que uma das dificuldades de se utilizar Aprendizado Profundo em robôs móveis pequenos tais como robôs humanoides de até 90 cm de altura ou drones é a limitação do tamanho e peso do computador embarcado. O computador embarcado utilizado nesses robôs são versões de minicomputadores tais como: Intel NUC; fitPC; Raspberry; ODROID; Jetson; dentre outros. O problema destes minicomputadores é que não possuem um massivo poder de processamento paralelo, para que se possa executar Redes Neurais Profundas com um tempo de inferência satisfatório para algumas aplicações, tais como um robô que joga futebol.

Para solucionar esse problema de tempo de inferência da DNN em robôs moveis dotados de computadores que não possuem um massivo poder de processamento paralelo, essa tese apresenta uma arquitetura de árvore de decisões de DNNs, desenvolvida para apresentar um tempo de inferência reduzido se comparado com as típicas DNNs tais como GoogLeNet ou AlexNet, e também apresentou uma acurácia superior a estes modelos.

Este tipo de abordagem já foi apresentado com outros tipos de classificadores (MADZAROV; GJORGJEVIKJ; CHORBEV, 2009) e até mesmo com redes neurais como mostra o trabalho de Drake e Packianather (1998), que apresentou uma árvore de decisão de redes neurais, no entanto, essa tese mostra que este mesmo tipo de abordagem pode ser desenvolvido com DNN, assim apresentando resultados satisfatórios em relação a acurácia e tempo de inferência.

Com a árvore de decisão é possível utilizar arquiteturas de redes neurais menores e mais especializadas, isso porque há menos classes em cada DNN, e essas DNNs apresentam uma acurácia próxima de 100%, resultando em uma acurácia final mais alta do que usando uma única DNN.

Nesse modelo, a acurácia, taxa de acerto ou outras métricas de cada classe podem ser definidas pela Equação (30), onde: $\mathcal{A}$ é a métrica de saída (acurácia, taxa de acerto ou outra métrica) de saída; D é a métrica da DNN (acurácia, taxa de acerto ou outra métrica) referente a camada l ; l é o índice referente as camadas; k é o índice referente as classes da DNN; c é o índice referente as classes da árvore de decisão de DNNs; m é o total de camadas.

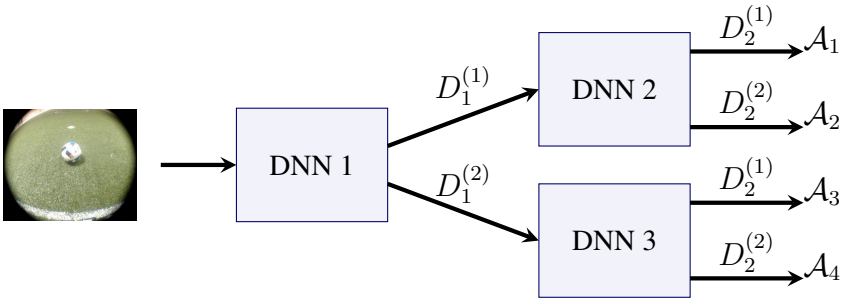
$$\mathcal{A}_c = \prod_{l=1}^{m} D_l^{(k)} \tag{30}$$

Já o valor total da métrica da árvore de decisão de DNNs pode ser definida pela Equação (31), onde: $\mathcal{A}_{total}$ é o valor total da métrica da árvore de decisão de DNNs; $\mathcal{A}$ é a métrica de saída referente a classe c; c é o índice referente as classes; n é o número total de classes.

$$\mathcal{A}_{total} = \frac{1}{n}\sum_{c=1}^{n}\mathcal{A}_c \tag{31}$$

A Figura 43 apresenta um exemplo da árvore de decisão de DNNs, nesse modelo a imagem entra na DNN 1, e a saída da DNN responde com uma probabilidade se a imagem pertence a classe 1 ou 2, então a imagem de entrada é direcionada para uma das DNNs da camada 2. Se na DNN 1 a probabilidade da classe 1 for maior que a classe 2, a imagem será direcionada para a DNN 2 da camada 2; caso contrário a imagem será direcionada para a DNN 3 da camada 2.

Figura 43 – Exemplo de Árvore de Decisão de DNNs.



Fonte: Autor.

O tempo de inferência da arvore de decisão de DNN pode ser calculado somando o tempo de inferência de cada DNN de cada camada conforme Equação (32), onde: $t_{inferência}$ é o tempo de inferência da árvore de decisão de DNN; t é o tempo de inferência da DNN na camada l; l é o índice referente as camadas; m é o total de camadas. Nesse modelo, será utilizado apenas uma DNN de cada camada durante o processo de inferência da imagem, isso porque durante o processo de inferência, seleciona-se apenas uma rede em cada camada, de acordo com a probabilidade das redes selecionadas, ou seja, seleciona-se a rede da próxima camada onde a classificação da imagem apresentou maior probabilidade.

$$t_{inferência} = \sum_{l=1}^{m} t_l \tag{32}$$

Neste trabalho a árvore de decisão de DNNs foi desenvolvida para controlar um robô humanoide, onde a entrada recebe apenas a imagem da câmera do robô e controla o robô a partir dessa imagem.

Os experimentos que serão apresentados na seção 5.3, mostram que com a árvore de decisão de DNNs é possível aumentar a acurácia e reduzir o tempo de inferência, obtendo resultados superiores a modelos com uma única DNN para todas as classes. A árvore de decisão é composta por DNNs menores e mais especializadas, dessa forma trazendo uma solução para o problema da incompatibilidade entre a acurácia e o tempo de inferência, onde uma DNN com maior acurácia possui um maior tempo de inferência que uma DNN com baixa acurácia que possui um baixo tempo de inferência.

4.4 SISTEMA COGNITIVO PARA AGENTES ROBÓTICOS COM APRENDIZADO POR REFORÇO PROFUNDO

Esta seção apresenta um sistema cognitivo que proporciona a agentes robóticos a capacidade de tomar decisões apenas pela observação do ambiente, conforme mostra a Figura 44. A política das tarefas são aprendidas pela interação com ambiente (DRL) durante o processo de aprendizado.

Figura 44 – Modelo proposto com Aprendizado por Reforço Profundo.

Fonte: Autor.

Os algoritmos de Aprendizado por Reforço Profundo que vêm sendo aplicados em jogos 3D (KEMPKA et al., 2016; LAMPLE; CHAPLOT, 2016; JADERBERG et al., 2016; MNIH et al., 2016; MIROWSKI et al., 2016; JUSTESEN et al., 2019), mostraram que o DRL aprendeu a tomar decisões a partir das imagens dos jogos, isso abre a possibilidade de se aplicar DRL em agentes robóticos para realizar tarefas especificas, sendo que alguns recentes trabalhos em robótica (GU et al., 2017) mostraram que DRL aprendeu a realizar determinadas tarefas.

Alguns trabalhos apresentados nas seções 2 e 3 mostram o uso de DL para tomada de decisões a partir da imagem, como no trabalho de Mnih et al. (2013) onde o computador foi capaz de aprender a jogar diversos jogos de Atari recebendo apenas os pixels da imagem do jogo como entrada.

Nesta tese as tarefas estão relacionadas com o papel que o robô deverá desempenhar, por exemplo, foram realizados experimentos que o robô desempenha o papel de um robô goleiro, portanto a sua tarefa é afastar a bola quando ele estiver dentro da área do gol.

Com Aprendizado por Reforço Profundo poderá ser realizado o aprendizado de tarefas referente ao papel que o robô deverá desempenhar, onde poderá haver vários módulos de DRL responsáveis por aprender uma especifica tarefa, tais como: goleiro aprendendo a defender a bola; jogador aprendendo quando deve conduzir a bola ou realizar um passe; jogador atacante aprendendo quando chutar a bola para o gol. Essas tarefas poderão ser gerenciadas por um processo responsável por ativar ou desativar cada módulo de DRL, sendo uma arquitetura onde haverá um processo responsável pela deliberação sobre qual tarefa usar. Sendo que este processo poderá ser também uma DNN ou DRL.

Nesta tese o processo de aprendizado do DRL foi realizado no simulador Webots. A entrada da rede no DRL é a imagem da camerâ do robô, e a saída é a ação que o robô deverá executar. As recompensas são calculadas no controlador Supervisor conforme a funções de recompensas determinadas e apresentadas nas seções 6.1, 6.2 e 6.3.

4.4.1 Transferência dos Pesos Aprendidos no Simulador Para Executar no Robô Real

Essa tese também apresenta que é possível realizar um aprendizado do DRL no simulador para executar no robô real, dessa forma todo o processo de aprendizado é realizado no simulador 3D, e depois de aprendido, os pesos da rede são transferidos para o robô real, nessa proposta não se realiza nenhum processo de aprendizado no robô real.

No conceito de Transferência de Aprendizado, os métodos de aprendizado de máquina são desenvolvidos para usar um conhecimento aprendido em um domínio, para diminuir o tempo de aprendizado em um outro domínio (WEISS; KHOSHGOFTAAR; WANG, D., 2016). No entanto, nessa tese realiza-se apenas a transferência dos pesos da rede, sem a realização do processo de aprendizado, portanto não sendo uma transferência de aprendizado.

Estão disponibilizadas todas as informações necessárias para replicar o estudo (pseudocódigos, código fonte, conjunto de dados) apresentado nesta tese, assim permitindo que os

pesquisadores possam avaliar a validade do estudo, para que os experimentos possam ser replicados, para a análise dos resultados e para dar continuidade a essa pesquisa.

5 EXPERIMENTOS COM REDES NEURAIS PROFUNDA

Nesta seção serão apresentados experimentos que foram realizados para testar e validar o sistema cognitivo com Redes Neurais Profundas apresentados na seção 4. Já o sistema cognitivo com Aprendizado por Reforço Profundo será apresentado na seção 6. Esta seção está dividida em quatro seções secundárias: experimentos realizados no simulador; experimentos realizado no robô real; árvore de decisão de Redes Neurais Profundas; e discussão. Na seção 5.1 serão apresentados experimentos realizados no simulador Webots utilizando o robô humanoide DARwIn-OP. Na seção 5.2 serão apresentados experimentos realizados no robô humanoide real em um campo de gramado sintético com dimensões 9x6 metros. Na seção 5.3 serão apresentados experimentos com a árvore de decisão de DNNs. Na seção 5.4 haverá uma breve discussão sobre os experimentos.

Os experimentos apresentados nesta seção demonstram um modelo que unifica os processos de visão e decisão através da utilização de uma rede neural profunda, assim demonstrando que uma rede DNN é capaz de substituir alguns processos. Neste experimento uma rede neural profunda recebe uma imagem proveniente da câmera do robô e classifica essa imagem de uma forma que a classificação da imagem seja a ação que o robô deverá executar.

Para pesquisa e validação do modelo, a DNN foi aplicada em um robô goleiro. O goleiro deve permanecer posicionado na pequena área e verificar se há alguma bola próxima dele, a partir do momento que o goleiro percebe que a bola está próxima do seu gol, o goleiro deve ir até a bola e chutá-la para longe do gol, após chutar a bola ele deve retornar para a pequena área embaixo das traves do gol. Todas estas ações são controladas pela DNN.

No modelo proposto a rede neural profunda é treinada com aprendizado supervisionado. Todos os treinamentos foram realizados em um computador com processador Intel i7-7700HQ 2.8GHz, memória 32GB DDR4 2133MHZ, SSD de 480GB, GPU NVIDIA GeForce GTX 1060 com 6GB de memória DDR5, Sistema Operacional Ubuntu 16.04. Os experimentos foram realizados em duas etapas; primeiramente em simulação para validar o modelo proposto; depois de comprovado o funcionamento do modelo na simulação, o experimento foi realizado no robô real.

Nos experimentos foram utilizados uma bola branca padrão FIFA (*size* 1), gol branco e o campo contendo as dimensões de 9x6 metros com grama artificial de 30 mm de acordo com as regras de 2016 da liga humanoide da RoboCup (REGRAS..., 2017). Atualmente, na categoria humanoide *KidSize*, os robôs podem ter uma altura de 40 até 90 centímetros, e o número de jogadores é no máximo quatro e no mínimo um (REGRAS..., 2017). A cada ano

a liga humanoide realiza algumas modificações nas regras para que o jogo fique cada vez mais próximo das condições e regras do jogo de futebol dos humanos.

Em todos os experimentos desta seção foram utilizados três grupos de imagens: imagens de treinamento; imagens de validação; e imagens de teste. E todas essas imagens foram extraídas da câmera do robô. As imagens de treinamento e de validação são utilizadas no processo de treinamento. As imagens de teste são utilizadas para aplicar testes estatístico para que se possa interpretar o desempenho da rede e realizar comparações entre os classificadores. Nesta seção as imagens de testes serão utilizadas para verificar a taxa de acerto da rede depois que o treinamento foi finalizado.

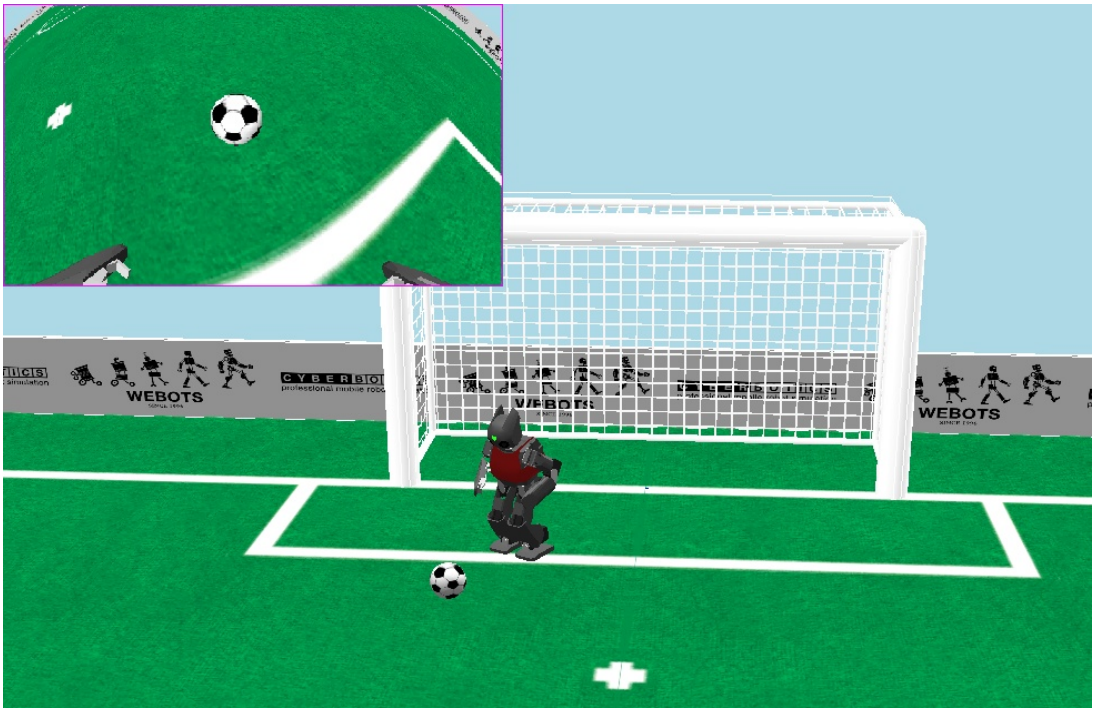
5.1 EXPERIMENTOS REALIZADOS NO SIMULADOR

Nesta seção serão apresentados experimentos do modelo proposto realizados no simulador Webots utilizando o robô humanoide DARwIn-OP em um campo de futebol. Este foi o primeiro experimento realizado para verificar se seria possível utilizar uma DNN para a cognição robótica, onde a DNN recebe apenas a imagem da câmera do robô e a saída sendo ação que o robô deverá executar.

No modelo proposto a rede neural profunda deverá ser treinada com as imagens adquiridas do contexto do ambiente que o robô realiza a navegação, essas imagens devem ser extraídas a partir da câmera do robô, as imagens devem ser separadas em classes referente a ação que se deseja que o robô realize para tais imagens. Por exemplo, na Figura 45 a câmera do robô (imagem do canto superior esquerdo da Figura 45) contém uma imagem em que a bola encontra-se no centro e afastada do robô, portanto essa é uma imagem que deve ser classificada como bola ao centro para que o robô realize a ação de andar para frente.

A taxa de leitura das imagens provenientes da câmera e passada para a entrada da rede neural foi fixada em aproximadamente $8Hz$ (no Desktop utilizado durante o experimento a DNN gastava aproximadamente 4 milissegundos para classificar uma imagem), foi utilizado este valor por ser um tempo adequado para o robô realizar uma ação. Durante o andar, o robô foi configurado para realizar um passo a cada 600 milissegundos, um semiciclo (meio passo) equivale a 300 milissegundos, portanto o sistema de controle não consegue responder a uma decisão que trabalhe abaixo de 300 milissegundos. No robô foi utilizado uma câmera com olho de peixe que permite um campo de visão de 2.3415 radianos (aproximadamente 130 graus).

Figura 45 – Goleiro no simulador.



Fonte: Autor.

Para evitar os erros de classificação que pudessem ocorrer entre as classificações das imagens ($C_{t-n} = C_{t-2} = C_{t-1} \neq C_t$ e $C_{t-1} = C_{t+1} = C_{t+2} = C_{t+n}$, onde C_t é uma classificação errada e os outros C_{t+n} e C_{t-n} são classificações corretas), e assim fazendo com que o robô tomasse uma ação incorreta, a Equação (33) foi utilizada para calcular a soma das probabilidades das últimas classificações com um fator de desconto temporal. Onde $Pr(c)$ é a probabilidade da classe, c é a classe, $gamma$ é o fator de desconto temporal, N é a quantidade total probabilidades passadas que será utilizado.

$$sumPr(c) = \sum_{i=0}^{N} \gamma^i Pr_{t-i}(c) \tag{33}$$

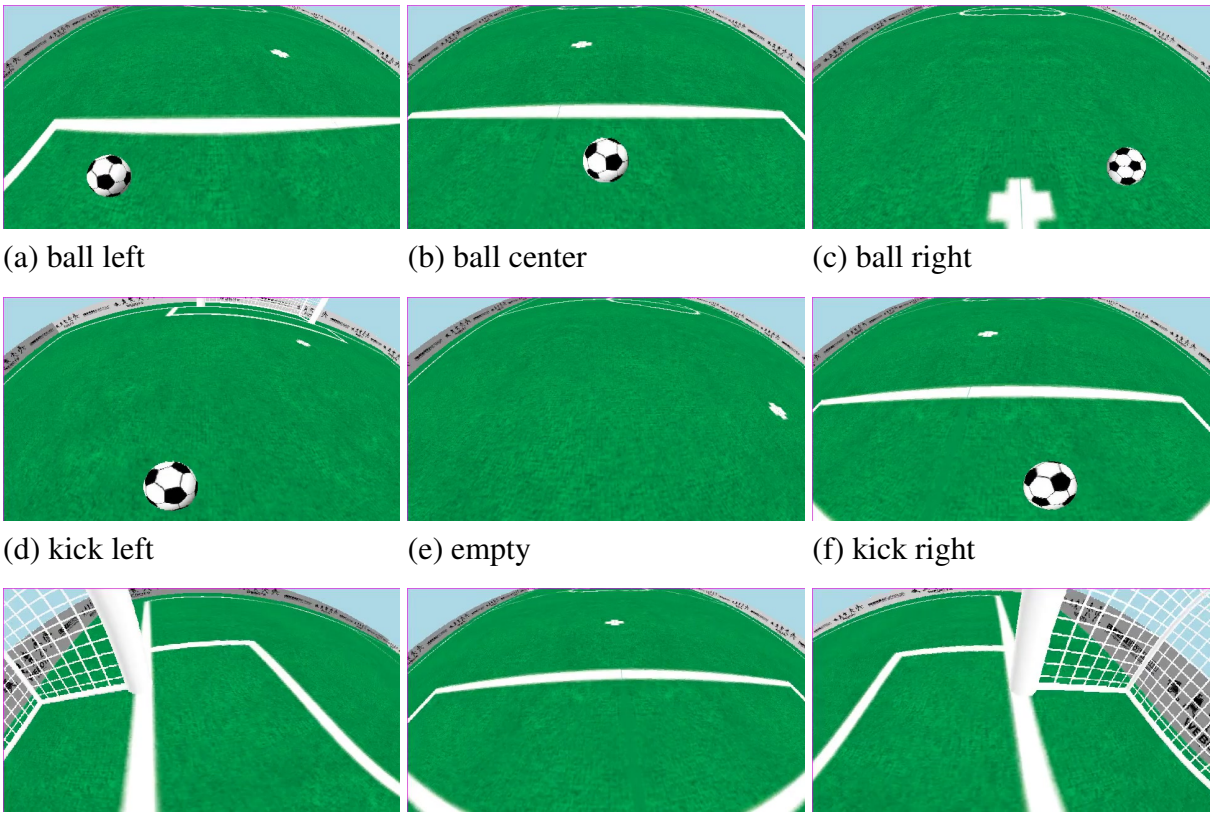
Após realizada esta soma $Pr(c)$, a ação é tomada de acordo com a classe que apresente o maior resultado da soma $sumPr(c)$, conforme Equação (34), onde C é a classe selecionada. Durante os experimentos foram utilizados $\gamma = 0.9$ e $N = 10$.

$$C = arg\max_{c} sumPr(c) \tag{34}$$

Os experimentos foram realizados em um computador com processador Intel i5-4690 3.5 GHz (turbo máximo de 3.9 GHz), placa-mãe ASUS Z97M-PLUS/BR, memória 8GB DDR3 1866MHZ, SSD de 120GB, GPU NVIDIA GeForce GTX 660 2GB memória DDR5, Sistema

Operacional Ubuntu 14.04. Todos os experimentos foram implementados no simulador Webots versão 8.0.5 utilizando a linguagem de programação C++ e Python e a DNN foi implementada em Python utilizando o Caffe[1]. No simulador Webots foi utilizado o robô humanoide DARwIn-OP em um campo de futebol, ambos disponibilizados pelo simulador.

Figura 46 – Classes de imagens.



(a) ball left (b) ball center (c) ball right

(d) kick left (e) empty (f) kick right

(g) at goal (h) at goal (i) at goal

Fonte: Autor.

Foram criadas 7 classes referente as ações que o robô deve tomar: *ball left* ⇒ vire à esquerda; *ball right* ⇒ virar à direita; *ball center* ⇒ andar para frente; *kick right* ⇒ chute direito; *kick left* ⇒ chute esquerdo; *empty* ⇒ andar para trás; *at goal* ⇒ nada a fazer. O treinamento da DNN foi realizado utilizando imagens referente a cada contexto, as imagens foram capturadas com posições aleatórias do robô no campo, sendo imagens da câmera do robô. A Figura 46 mostra as imagens referente as 7 classes, todas as imagens foram capturadas da câmera do robô.

As classes *ball left*, *ball right*, *ball center*, *kick right*, *kick left*, foram criadas para verificar a capacidade e o desempenho da DNN de aprender a localização do objeto na imagem de

[1] http://caffe.berkeleyvision.org/

forma qualitativa (representação qualitativa). Assim a DNN pode informar ao robô qual ação tomar para aquela imagem (andar, chutar, virar). A classe *empty* foi criada para que a DNN pudesse aprender quando que não tem bola na imagem e, portanto, o goleiro deve voltar para o gol. Já a classe *at goal* foi criada para que o robô não necessite de localização para saber que o robô está posicionado no gol, sendo capaz de aprender isso através de imagens do robô posicionado no gol. No entanto, em todos nos experimentos apresentados nessa seção 5 o robô utilizou a orientação proveniente da IMU para manter-se sempre orientado para frente do campo e de costas para o gol.

Para capturar as imagens do conjunto de dados, foram realizadas gravações de vídeos, sendo gravado um vídeo para cada classe, assim podendo extrair os quadros de imagens do vídeo referente a cada classe, sem a necessidade de analisar e separar cada frame nas classes. Para gerar os vídeos, foi utilizado o supervisor do Webots para posicionar o robô em posições aleatórias no campo de forma automática e também para manter a bola em posições aleatórias, porém dentro do limite de cada classe.

Durante o processo de treinamento foi possível notar que quando se usa poucas imagens (300 imagens) a rede apresentava uma baixa taxa de acerto, portanto a rede necessitava de muito mais episódios para aprender o padrão. Nos parâmetros de configurações dos treinamentos da rede foram utilizados os seguintes valores: A taxa de aprendizado foi iniciado com 0.01; *Subtract Mean* = *Pixel*; *Crop Size* = *none*; *Solver type* = *Stochastic Gradient Descent* (SGD), SGD com um *batch size* de 128 exemplos.

Nas subseções 5.1.1, 5.1.2 e 5.1.3 serão apresentados resultados de experimentos realizados com a rede neural profunda: O primeiro experimento apresentado na subseção 5.1.1 mostra como foi realizado o treinamento e o desempenho da DNN após o treinamento; O segundo experimento (subseção 5.1.2) testa a generalização da rede e a capacidade de armazenar diversos padrões de diferentes tipos de bola em uma mesma classe, sendo esta rede aplicada ao mesmo problema do primeiro experimento. E na subseção 5.1.3 será apresentado uma arquitetura de rede com menos neurônios que a Alexnet e com 7 camadas ao invés de 8, buscando uma rede que apresente o mesmo desempenho mostrado nos experimentos das subseções 5.1.1 e 5.1.2, porém com um tempo de inferência menor.

5.1.1 Experimento do robô goleiro com DNN

A DNN utilizada neste experimento foi a AlexNet desenvolvida por Krizhevsky, Sutskever e Geoffrey E Hinton (2012). Após o treinamento também foi realizado o teste com a rede AlexNet sendo executada no mini PC Intel NUC[2]. Executando a rede AlexNet em um Intel NUC Core i5-4250U, 8GB SDRAM, 120GB SSD com as configurações da DNN apresentadas na Tabela 6 em que a rede possuí uma entrada de 110x110x3 (largura, altura, canais RGB), a rede gasta aproximadamente $100ms$ para classificar uma imagem.

Tabela 6 – Arquitetura da DNN AlexNet com entrada 110x110.

Tipo	Camada	Entrada	Kernel	Stride	Pad	Filters	Saída
Convolutional	1	110x110x3	11x11	4	0	96	25x25x96
Max-pooling	1	25x25x96	3x3	2	0	96	12x12x96
Convolutional	2	12x12x96	5x5	1	2	256	12x12x256
Max-pooling	2	12x12x256	3x3	2	0	256	6x6x256
Convolutional	3	6x6x256	3x3	1	1	384	6x6x384
Convolutional	4	6x6x384	3x3	1	1	384	6x6x384
Convolutional	5	6x6x384	3x3	1	1	256	6x6x256
Max-pooling	5	6x6x256	3x3	2	0	256	3x3x256
Fully-connected	6	3x3x256				4096	4096
Fully-connected	7	4096				4096	4096
Fully-connected	8	4096				7	7

Fonte: Autor.

Primeiramente foram geradas 18597 imagens de treinamento, 6199 imagens de validação e 7000 imagens de teste, com resolução de 640x360x3 (largura, altura, canais RGB).

Depois de treinada, foi verificado o desempenho da rede utilizando as 7000 imagens de testes (1000 imagens por classe). O resultado do desempenho da rede pode ser verificado na Tabela 7, nessa tabela pode-se observar que a rede classificou quase todas as classes com aproximadamente 100% de taxa de acerto, e a classe *at goal* apresentou o menor desempenho. Isso porque as duas classes *at goal* e *empty* possuem características muito próximas. O desempenho da DNN durante esse teste foi 98.58% de taxa de acerto.

Após o treinamento da DNN, realizou-se o teste em simulador, com o robô sendo controlado pela DNN (https://youtu.be/b3tebN9dwN8). O robô iniciou posicionado na área do gol e o processo da DNN recebia o *frame* da câmera do robô e respondia com a ação que deveria ser tomada para aquela imagem. Durante esse teste o robô executou as ações corretas, sendo capaz de ir até a bola chutá-la e retornar ao gol. Como havia dois processos sendo executados (robô

[2]http://www.intel.com/content/www/us/en/nuc/overview.html

Tabela 7 – Matriz de confusão.

	ball center	**at goal**	**empty**	**kick left**	**kick right**	**ball left**	**ball right**	**Taxa de acerto**
ball center	1000	0	0	0	0	0	0	100%
at goal	0	931	69	0	0	0	0	93.1%
empty	0	0	1000	0	0	0	0	100.0%
kick left	0	0	4	996	0	0	0	99.6%
kick right	0	0	0	0	1000	0	0	100%
ball left	4	8	3	1	0	984	0	98.4%
ball right	0	5	1	0	4	0	990	99.0%

Fonte: Autor.

no simulador e DNN), as imagens são passadas do simulador para o processo da DNN através da memória compartilhada do sistema.

Figura 47 – Robô aparecendo na imagem.

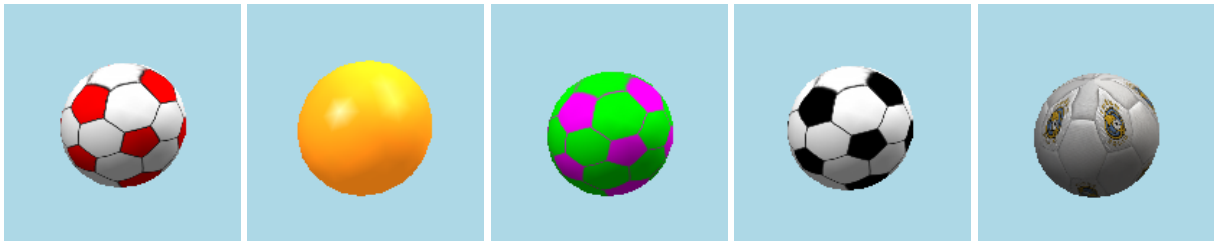
Fonte: Autor.

Visando simular um ambiente de competição, um outro experimento incluiu alguns robôs no campo, no entanto com alguns robôs aparecendo na imagem (Figura 47), o classificador apresentou um desempenho menor (taxa de acerto de 65.16%). Então foram capturadas mais imagens contendo robôs no campo em posições aleatórias, totalizando 36644 imagens de treinamento, 12214 imagens de validação e 14000 imagens de teste. Após o treinamento, a DNN passou a apresentar um desempenho com 99.49% de taxa de acerto.

5.1.2 Verificação de generalização da rede para diversos tipos de bola

Este experimento foi realizado para verificar se a rede seria capaz de aprender com diversos tipos de bolas, e também para testar a generalização da rede. Foram utilizados 15 tipos de bolas, 5 para usar no treinamento e teste, e 10 para usar apenas no teste. Os 5 tipos de bolas usadas nos treinamentos e testes estão apresentados na Figura 48, e as 10 utilizadas apenas no teste estão apresentadas na Figura 49. Neste experimento foram geradas 70583 imagens de treinamento, 23526 imagens de validação e 75000 imagens de teste.

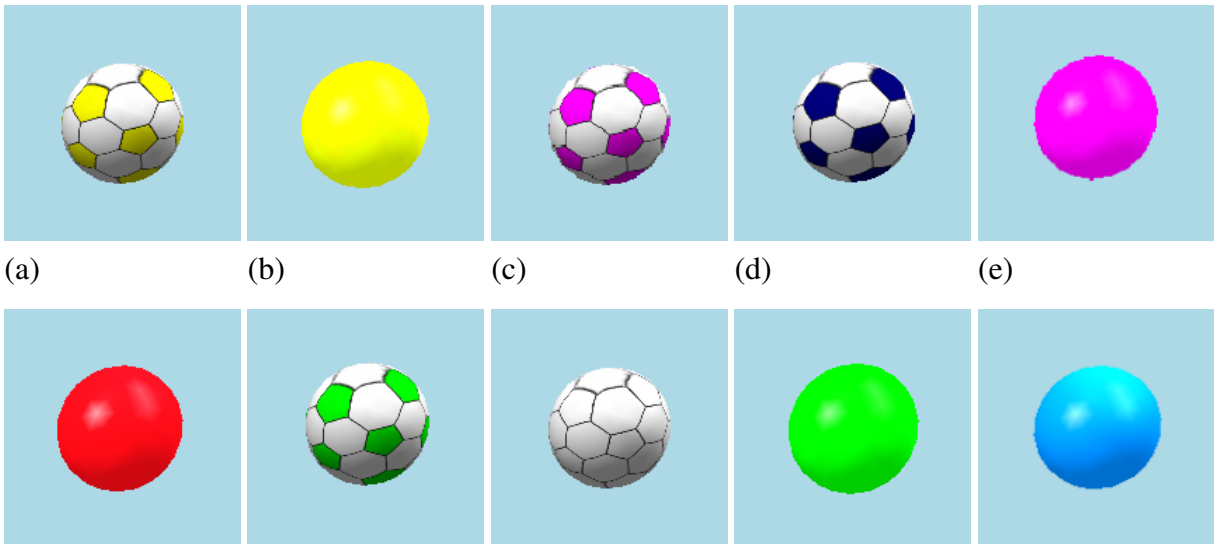
Figura 48 – Bolas usadas no treinamento.



(a) (b) (c) (d) (e)

Fonte: Autor.

Figura 49 – Bolas usadas no teste e não usadas no treinamento.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Fonte: Autor.

A Tabela 8 apresenta a taxa de acerto de cada classe para cada bola usando as imagens de teste. A tabela mostra que para os 5 tipos de bolas usadas no treinamento (Figuras 48a, 48b,

48c, 48d e 48e) a DNN apresentou quase 100% de taxa de acerto. Já nos 10 tipos de bolas que não foram utilizadas no treinamento, a DNN classificou com uma taxa de acerto acima de 95% em 6 tipos de bolas, sendo que 4 bolas não apresentaram uma boa taxa de acerto, portanto a rede possui um grau de generalização.

Uma análise que pode ser realizada em relação aos pixels das bolas é analisar as cores. As 4 bolas que não apresentaram uma boa taxa de acerto, possuem cores azuis, branco e verde, e nas bolas usadas no treinamento não há a cor azul, branco existe em 2 modelos mas que também possuem outras cores (Figuras 48a e 48d), e verde aparece somente na bola da Figura 48c. Em relação as outras 6 bolas que a rede classificou com uma taxa de acerto acima de 95%, uma das possibilidades é que a rede possuía uma bola que tenha RGB laranja (255,165,0) e vermelho RGB($1 < r < 255$,0,0), o que explica por que RGB amarelo RGB($1 < r < 255$,255,0), roxo RGB($1 < r < 255$,0,$1 < b < 255$) e vermelho RGB($1 < r < 255$,0,0), foram classificados corretamente. Parece que a rede ficou tendenciosa, usando cores como descritor e ignorando formas como o círculo externo, um componente-chave nas bolas, a rede também ignorou pentagramas e hexagramas contidos na maioria das bolas usadas para treinamento.

Tabela 8 – Taxa de acerto [%].

	48a	**48b**	**48c**	**48d**	**48e**	**49a**	**49b**	**49c**	**49d**	**49e**	**49f**	**49g**	**49h**	**49i**	**49j**
ball center	100	100	100	100	100	100	100	100	99.6	98.0	98.7	68.4	54.1	20.0	0
kick left	100	100	100	100	100	100	100	100	99.9	92.8	87.8	14.3	0.2	0.4	0
kick right	100	100	100	100	100	100	100	100	100	95.7	90.6	5.2	0	5.8	0
ball left	100	100	100	100	99.8	100	100	99.8	100	99.8	100	99.4	97.9	26.3	1.4
ball right	100	100	100	99.9	100	100	100	100	100	99.2	99.5	58.8	68.3	42.4	1.0
Total	100	100	100	99.99	99.96	100	100	99.96	99.9	97.1	95.3	49.2	43.7	18.98	0.48

Fonte: Autor.

5.1.3 Experimento de redução do tamanho da rede

Neste experimento foi implementado uma arquitetura simplificada de DNN com o objetivo de gastar menos tempo para classificar as imagens mantendo o mesmo desempenho da AlexNet para o problema abordado. Como pode ser visto na Tabela 9, a nova arquitetura possui 512 neurônios nas duas primeiras camadas da *fully-connected* e apenas 4 convolucionais, foi eliminado uma camada convolucional, já todas as outras camadas foram mantidas com a mesma configuração da AlexNet.

A Tabela 10 mostra o desempenho da nova arquitetura e da AlexNet, pode-se ver que as duas DNN apresentaram desempenhos aproximadamente iguais. O conjunto de dados uti-

Tabela 9 – Arquitetura modificada.

Tipo	Camada	Entrada	Kernel	Stride	Pad	Filters	Saída
Convolutional	1	110x110x3	11x11	4	0	96	25x25x96
Max-pooling	1	25x25x96	3x3	2	0	96	12x12x96
Convolutional	2	12x12x96	5x5	1	2	256	12x12x256
Max-pooling	2	12x12x256	3x3	2	0	256	6x6x256
Convolutional	3	6x6x256	3x3	1	1	384	6x6x384
Convolutional	4	6x6x384	3x3	1	1	256	6x6x256
Max-pooling	4	6x6x256	3x3	2	0	256	3x3x256
Fully-connected	5	3x3x256	-	-	-	512	512
Fully-connected	6	512	-	-	-	512	512
Fully-connected	7	512	-	-	-	7	7

Tabela 10 – Taxa de acerto [%] da rede com 4 convolucionais.

	48a	48b	48c	48d	48e	49a	49b	49c	49d	49e	49f	49g	49h	49i	49j
AlexNet	100	100	100	99.99	99.96	100	100	99.96	99.9	97.1	95.3	49.2	43.7	19	0.48
Nova	100	100	99.98	100	99.96	100	100	99.98	99.94	93.2	96.5	47.4	49.1	5.5	0.02

Fonte: Autor.

lizado neste experimento é o mesmo utilizado no experimento da seção anterior (seção 5.1.2). Pode-se observar que mesmo com a redução, a generalização entre os tipos de bola são aproximadamente iguais entre as duas redes. A comparação entre os tempos de inferência que cada arquitetura de DNN gasta para classificar um imagem será apresentado na seção 5.2.

5.2 EXPERIMENTOS REALIZADOS NO ROBÔ REAL

Figura 50 – Robô goleiro.

Fonte: Autor.

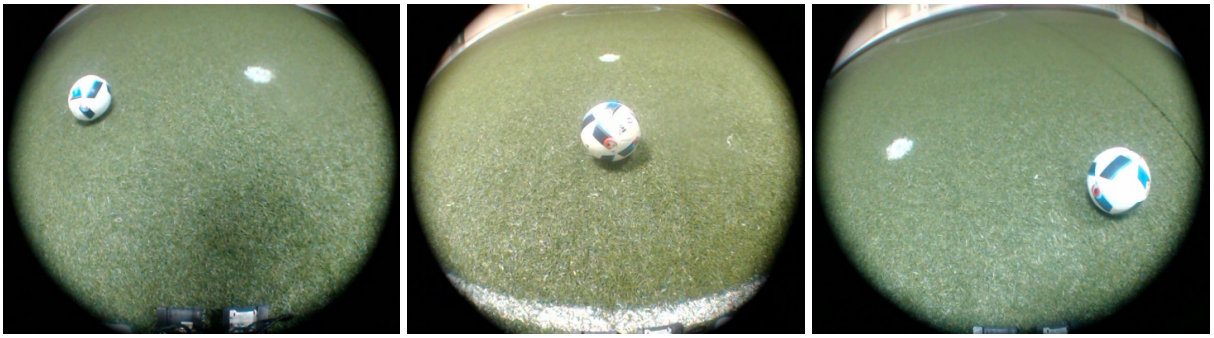
Esta seção apresenta os experimentos que foram realizados no robô humanoide real em um campo de futebol com grama sintética com dimensões de 9x6 metros (Figura 50) conforme as regras da RoboCup.

Todos os experimentos foram implementados no robô real utilizando a linguagem de programação C++ e Python, e a DNN sendo implementada em Python. Os robôs humanoides utilizados nestes experimentos foram desenvolvidos no Centro Universitário FEI, esses robôs têm a estrutura mecânica e a cinemática baseada no DARwIn-OP (HA et al., 2011). Os robôs possuem 20 graus de liberdade, uma altura de 490mm e um computador Intel NUC.

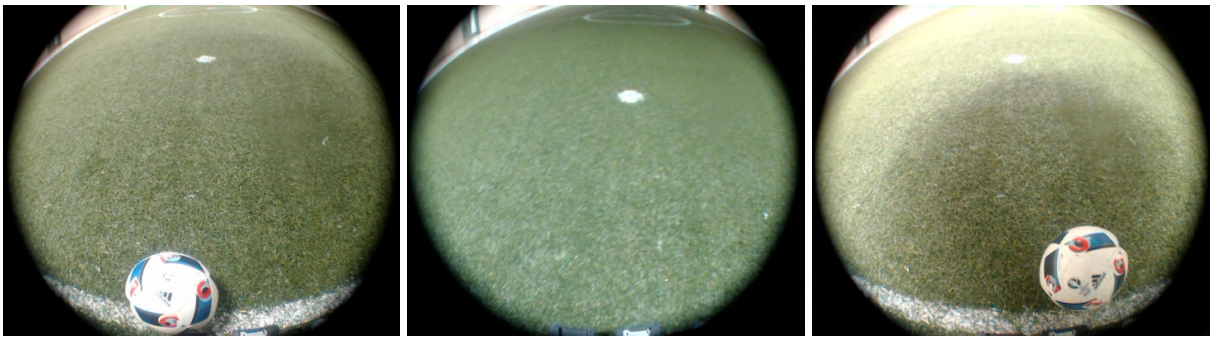
Nestes experimentos também foram criadas 7 classes referentes as ações que o robô deve tomar. A Figura 51 mostra as imagens referentes as 7 classes, todas as imagens foram capturadas da câmera do robô.

Foram treinados e testados 5 tipos de DNN: conv4 com entrada 110x110; AlexNet com entrada 110x110; conv4 com entrada 256x256; AlexNet com entrada 256x256 e a GoogLeNet com entrada 256x256. As imagens de treinamento e de testes foram extraídas da câmera do robô com o robô no campo. O conjunto de dados possuem 16388 imagens de treinamento, 5460 imagens de validação e 700 imagens de teste, com resolução de 1280x720x3 (largura, altura, canais RGB). A conv4 com entrada 110x110 possuí a mesma arquitetura apresentada na

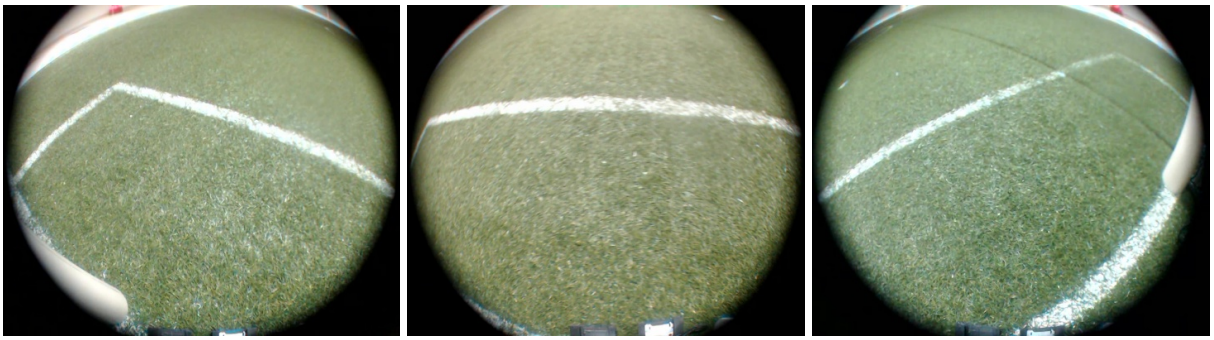
Figura 51 – Exemplos de imagens relacionadas a cada classe.



(a) ball left (b) ball center (c) ball right



(d) kick left (e) empty (f) kick right



(g) at goal (h) at goal (i) at goal

Fonte: Autor.

Tabela 9 da subseção 5.1.3, já a conv4 com entrada 256x256 muda apenas o tamanho da entrada imagem de entrada da rede.

Para verificar o desempenho da rede foram utilizadas 700 imagens de teste, sendo 100 imagens de cada classe. A Tabela 11 mostra o desempenho de cada classe da rede AlexNet com entrada de 256x256 pixels, apresentando uma taxa de acerto de 87.86%. A classe *ball center* apresentou uma taxa de acerto de 98%, e a menor taxa de acerto foi da classe *ball left* com 70%, portanto, há uma variação na taxa de acerto entre as classes.

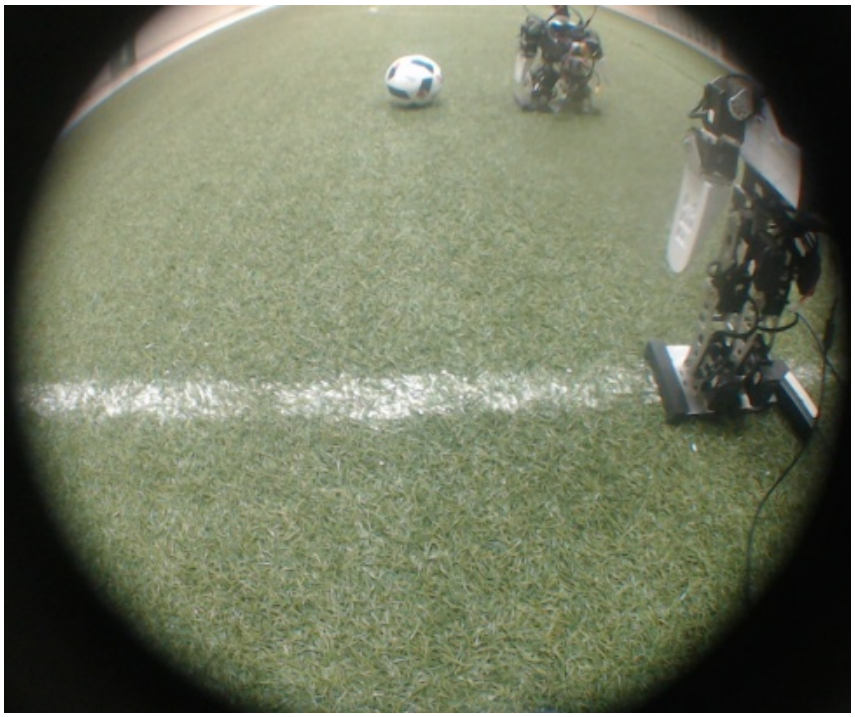
Foram capturadas 700 imagens de testes contendo outros robôs no campo para simular o ambiente de competição, como a imagem da Figura 52. E também foram capturadas mais

Tabela 11 – Matriz de Confusão da AlexNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	98	0	0	0	0	0	2	98.0%
ball left	3	70	0	0	0	21	6	70.0%
ball right	9	7	71	0	0	9	4	71.0%
kick left	0	0	0	98	2	0	0	98.0%
kick right	0	0	0	9	91	0	0	91.0%
at goal	0	0	1	0	0	92	7	92.0%
empty	0	0	1	0	0	4	95	95.0%

Fonte: Autor.

Figura 52 – Imagem da câmera do robô.



Fonte: Autor.

2090 imagens com robôs no campo, totalizando 17956 imagens de treinamento e 5982 imagens de validação.

As Tabelas 12 e 13 apresentam a taxa de acerto das redes treinadas com e sem imagens contendo robôs no campo, sendo testada com imagens contendo robôs no campo. A Tabela 13 mostra que as redes treinadas com imagens contendo robôs no campo apresentaram uma taxa de acerto maior, mesmo sendo apenas 2090 imagens. É possível notar que a rede GoogLeNet apresentou a melhor taxa de acerto apenas incluindo 2090 imagens.

Tabela 12 – Treinamento sem robôs na imagem e teste com robôs na imagem.

Rede	Entrada	Taxa de acerto [%]
Conv4	110x110	61.86
Alexnet	110x110	62.43
Conv4	256x256	58.71
Alexnet	256x256	72.29
GoogLeNet	256x256	59.71

Fonte: Autor.

Tabela 13 – Treinamento com robôs na imagem e teste com robôs na imagem.

Rede	Entrada	Taxa de acerto [%]
Conv4	110x110	74.43
Alexnet	110x110	77.00
Conv4	256x256	83.98
Alexnet	256x256	87.86
GoogLeNet	256x256	96.00

Fonte: Autor.

Tabela 14 – Taxa de acerto por classe.

Rede	Entrada	Taxa de acerto [%]						
		ball center	ball left	ball right	kick left	kick right	at goal	empty
Conv4	110x110	92	2	91	98	42	97	99
Alexnet	110x110	92	17	93	99	41	97	100
Conv4	256x256	88	65	82	86	83	83	100
Alexnet	256x256	98	70	71	98	91	92	95
GoogLeNet	256x256	100	97	92	100	86	98	99

Fonte: Autor.

A Tabela 14 mostra a taxa de acerto das redes para cada classe. Nessa Tabela é possível ver que a rede GoogLeNet (rede de 22 camadas) apresentou um desempenho superior a rede AlexNet (rede de 8 camadas), portanto, como já visto na seção 2.1, utilizar redes com mais camadas melhora o desempenho da classificação. No entanto utilizar redes com muitas camadas exige hardwares com mais poder de processamento, para que se tenha um tempo de classificação compatível com a aplicação.

A Tabela 15 apresenta a taxa de acerto de cada rede aplicada nas imagens de teste, e também o tempo que as redes DNNs gastam para classificar uma imagem em um Intel NUC. As amostras do tempo de inferência foram capturadas durante um período de 10 minutos. Nesse

Tabela 15 – Tempo médio de inferência das redes.

Rede	**Entrada**	**Taxa de acerto [%]**	**Tempo [s]**	
			Média	σ
Conv4	110x110	74.43	0.093	0.00602
Alexnet	110x110	77.00	0.131	0.00811
SqueezeNet	256x256	79.57	0.139	0.01920
Conv4	256x256	83.98	0.458	0.02784
Alexnet	256x256	87.86	0.470	0.01591
Googlenet	256x256	96.00	0.682	0.03386

Fonte: Autor.

experimento também foi testado a DNN SqueezeNet, SqueezeNet é uma DNN com menos parâmetros que a AlexNet, mas que apresenta uma acurácia equivalente. Com isso percebe-se que é possível executar uma rede neural profunda em um computador Intel NUC, e para reduzir o tempo de inferência pode-se reduzir a imagem de entrada, e também é possível reduzir a quantidade de neurônios da rede. No entanto, essas reduções resultam em uma diminuição da taxa de acerto da rede.

Durante o andar, o robô foi configurado para realizar um passo a cada 600ms, um semi-ciclo (meio passe) equivale a 300ms. Portanto, somente a cada 300ms uma ordem do processo de decisão enviada ao processo de controle causa o efeito de mudança da ação executada pelo o andar. Atualmente a dinâmica do jogo na liga humanoide ainda é lenta, isso também faz com que a tomada de decisão possa ser lenta. Por esses dois motivos, a DNN pode demorar 0.3 segundos para tomar uma decisão.

Depois da rede treinada, foi executado um teste da rede DNN no robô real executando o sistema cognitivo para avaliar o comportamento do robô goleiro (https://youtu.be/b3tebN9dwN8). O robô goleiro inicia posicionado no centro do gol, a DNN recebe as imagens da câmera do robô e responde qual ação o robô deverá executar para a imagem observada. Durante esse teste o robô executou as ações corretas em relação ao comportamento de um robô goleiro.

A Figura 53 mostra um dos vários testes que foram realizados e podem ser observados no vídeo disponível https://youtu.be/b3tebN9dwN8. Nesse teste o robô começa posicionado no centro do gol já executando o sistema cognitivo, enquanto não há nenhuma bola perto do gol, o robô permanece parado (Figuras 53.a e 53.b), quando uma bola é posicionada no campo próxima do gol (Figura 53.c) o robô começa a andar em direção a bola (Figuras 53.d, 53.e, 53.f e 53.g), ao se aproximar da bola o robô chuta a bola com o pé direito, isso porque a bola

está próxima do robô no lado direito (Figuras 53.h e 53.i), depois do chute a bola se afasta um pouco do robô (Figuras 53.j e 53.k), no entanto há a necessidade de retirar a bola do campo para verificar se o robô vai voltar para o gol (Figura 53.l), quando não há mais bola no campo próxima do gol, o robô volta para a posição de defesa embaixo das traves do gol conforme Figuras 53.m, 53.n, 53.o, 53.p, 53.q, já na Figura 53.r pode-se observar que o robô fica parado porque não há bola no campo próxima do gol e também já está posicionado embaixo das traves do gol.

5.3 ÁRVORE DE DECISÃO DE REDES NEURAIS PROFUNDAS

Nesta seção será mostrado que para solucionar o problema da relação entre taxa de acerto e o tempo de inferência, pode-se implementar uma Árvore de Decisão de DNNs. O experimento desta seção foi realizado para comparar a árvore de decisão de DNNs com as outras DNNs, comparando-se a taxa de acerto e o tempo de inferência.

O problema é que quando se reduz o tamanho da rede para diminuição do tempo de inferência, perde-se também em taxa de acerto, no entanto, necessitamos de uma rede neural que apresente uma taxa de acerto satisfatória, e que tenha um baixo tempo de inferência para utilizarmos no computador embarcado do robô. Para o sistema de visão dos robôs humanoides da RoboCup, alguns pesquisadores (N'GUYEN et al., 2018) definem que acima de 95% de taxa de acerto como uma taxa de acerto satisfatória.

A árvore de decisão DNNs implementada nos experimentos dessa tese é composta por três DNNs como mostra a Figura 54, onde a primeira DNN (DNN 1) recebe a imagem da câmera do robô e realiza uma classificação dicotômica se há bola na imagem, caso haja bola, a imagem é transmitida para a entrada da segunda DNN (DNN 2) que classifica a imagem da bola e informa na saída da rede a ação que deverá ser tomada devido a posição da bola na imagem, caso a primeira DNN (DNN 1) tenha inferido que não há bola na imagem, a imagem é transmitida para a entrada da terceira DNN (DNN 3) que classifica se o robô está posicionado no gol ou se está fora do gol, para que possa tomar a ação de ficar parado ou retornar para o gol. As três DNNs da árvore de decisão possuem a mesma arquitetura apresentada na Tabela 16.

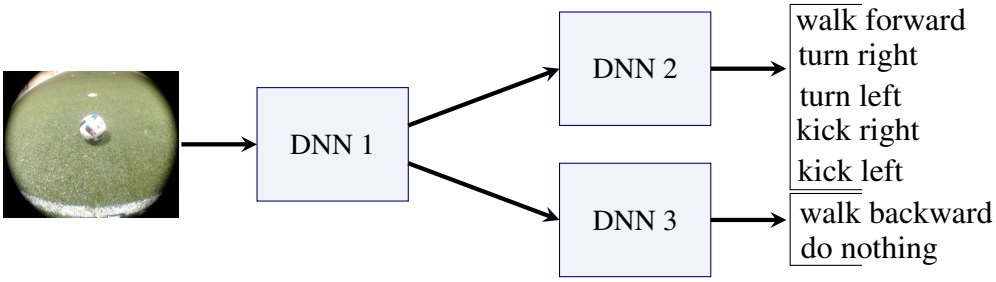
A arquitetura de rede da Tabela 16 foi testada com todas as 7 classes, onde na Tabela 17 mostra que utilizar uma única DNN com 3 convolucionais para todas as classes apresentou baixa taxa de acerto em algumas classes, isso faz com que não seja possível utilizar essa arquitetura com as 7 classes.

Figura 53 – Robô executado o sistema cognitivo com a bola ao centro.

Fonte: Autor.

As Tabelas 18, 19 e 20 mostram a matriz de confusão de cada DNN da árvore de decisão. Essas tabelas mostram que as DNNs com menos classes apresentaram uma maior taxa de acerto que uma única DNN para todas as classes. A Árvore de Decisão pode trabalhar com

Figura 54 – Árvore de Decisão de DNNs.



Fonte: Autor.

Tabela 16 – Arquitetura da DNN.

Tipo	Camada	Kernel	Stride	Pad	Nº de Kernels
Convolucional	1	8x8	4	0	32
Max-pooling	1	3x3	1	0	32
Convolucional	2	4x4	2	0	64
Convolucional	3	3x3	1	1	64
Fully-connected	4	-	-	-	512
Fully-connected	5	-	-	-	7

Fonte: Autor.

Tabela 17 – Matriz de confusão da DNN sem a árvore de decisão.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**
ball center	94	0	0	0	0	3	3
ball left	0	78	1	0	0	10	2
ball right	1	1	95	0	0	3	0
kick left	0	0	0	93	7	0	0
kick right	0	0	0	1	99	0	0
at goal	0	0	10	0	0	53	37
empty	0	0	5	0	0	0	95

Fonte: Autor.

DNNs menores e mais especializadas, dessa forma trazendo uma solução para o problema da incompatibilidade entre a taxa de acerto e o tempo de inferência.

Tabela 18 – Matriz de confusão da DNN 1 com duas classes (*ball* e *no ball*).

	ball	**no ball**	**Taxa de acerto por classe**
ball	489	11	97.8%
no ball	0	200	100%

Fonte: Autor.

Tabela 19 – Matriz de confusão da DNN 3 com duas classes (*at goal* e *empty*).

	at goal	**empty**	**Taxa de acerto por classe**
at goal	96	4	96.0%
empty	0	100	100%

Fonte: Autor.

Tabela 20 – Matriz de confusão da DNN 2 para classificar imagens com bola.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**Taxa de acerto por classe**
ball center	96	2	2	0	0	96.0%
ball left	0	100	0	0	0	100%
ball right	0	2	98	0	0	98.0%
kick left	0	0	0	100	0	100%
kick right	0	0	0	0	100	100%

Fonte: Autor.

A Tabela 20 mostra que a classe *ball center* foi a que apresentou menor taxa de acerto, isso porque nas imagens da bola a direita ou da bola a esquerda muitas vezes a bola estava posicionada no limiar entre duas classes, dessa forma isso não é um problema quando a DNN controla o robô.

Tabela 21 – Taxa de acerto por classe.

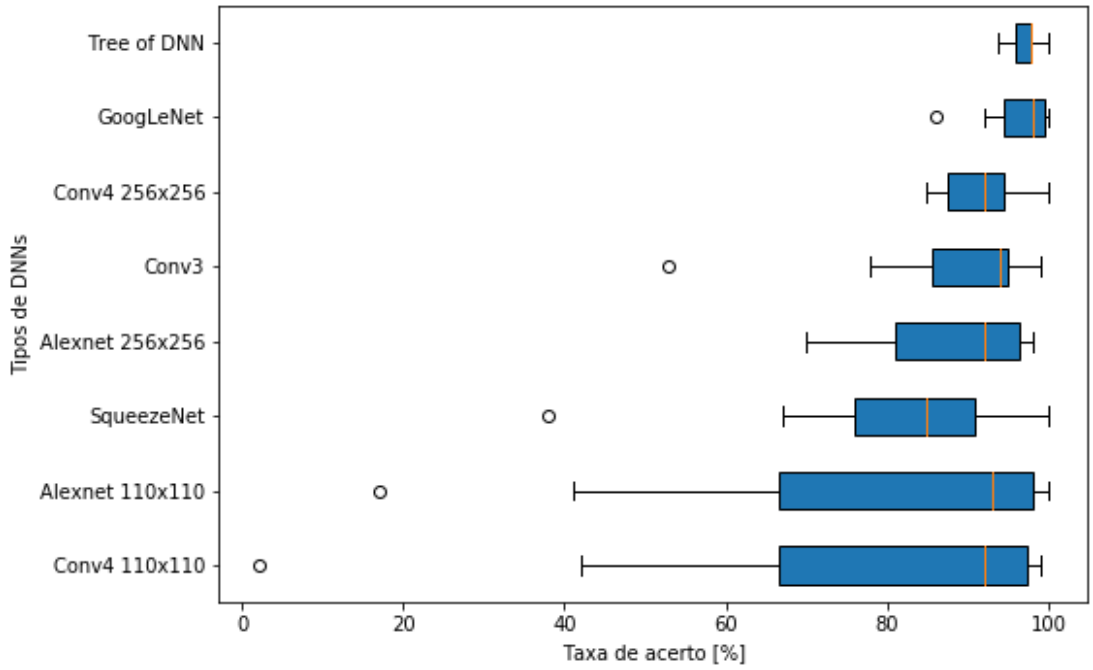
Rede	**Entrada**	**Taxa de acerto [%]**						
		ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**
Conv4	110x110	92	02	91	98	42	97	99
Conv3	110x110	94	78	95	93	99	53	95
Alexnet	110x110	92	17	93	99	41	97	100
SqueezeNet	256x256	95	87	85	67	38	85	100
Conv4	256x256	94	95	89	86	92	85	100
Alexnet	256x256	98	70	71	98	91	92	95
GoogLeNet	256x256	100	97	92	100	86	98	99
Tree of DNN	110x110	93.9	97.8	95.8	97.8	97.8	96.0	100

Fonte: Autor.

A Tabela 21 mostra a taxa de acerto de cada classe das DNNs e da árvore de decisão de DNNs. Na Tabela 21 é possível observar que quase todas as classes da árvore de decisão de DNNs apresentaram uma taxa de acerto acima de 95%, e a classe *ball center* apresentando

a menor taxa de acerto de 93.8%, diferente das outras DNNs que sempre tem uma classe que apresenta uma taxa de acerto inferior a 90%.

Figura 55 – Gráfico que demonstra a dispersão da taxa de acerto entre as classes.

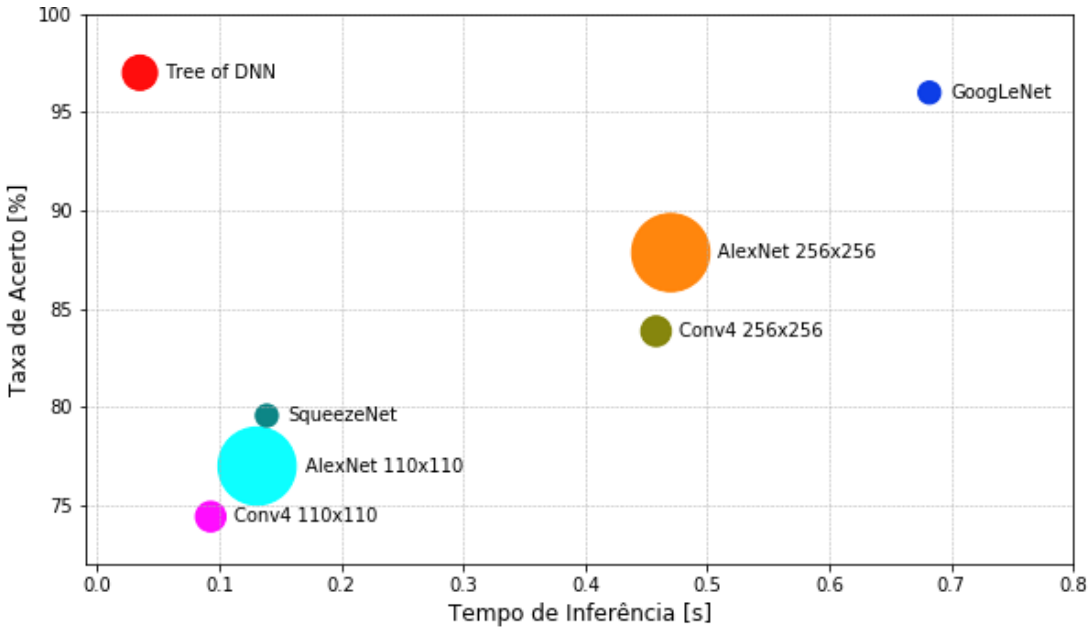


Fonte: Autor.

A Figura 55 apresenta um gráfico de dispersão da taxa de acerto entre as classes para cada DNN, o gráfico apresenta os quartis no retângulo em azul (Q1 e Q3 nas extremidades do retângulo e o Q2 representado por um traço laranja), o valor mínimo e máximo representado por um traço, e também os pontos considerados como *outliers* representados por um circulo no gráfico. A Figura 55 mostra que na árvore de decisão de DNNs a dispersão da taxa de acerto é a menor e não apresenta *outliers*.

A Tabela 22 apresenta a taxa de acerto de cada rede aplicada nas imagens de teste, e também o tempo de inferência para classificar uma imagem no computador Intel NUC. Essas amostras foram capturadas durante 10 minutos. Portanto esses resultados mostram que é possível executar estas DNNs em um computador Intel NUC, dessa forma permitindo que robôs humanoides da RoboCup possam utilizar DNNs. No entanto, ao se diminuir o tamanho da rede, ocorre a diminuição do tempo de inferência, porém diminui também a taxa de acerto da rede. Mas no caso da árvore de decisão de DNNs, apresentou uma taxa de acerto superior a da GoogLeNet, porém com um tempo de inferência 20 vezes menor, portanto usando uma árvore de decisão de DNNs é possível obter uma maior taxa de acerto com um baixo tempo de inferência comparando-se com uma única DNN para todas as classes.

Figura 56 – Gráfico da acurácia pelo tempo de inferência das DNNs.



Fonte: Autor.

A Figura 56 mostra um gráfico que demonstra a acurácia pelo tempo de inferência das DNNs, o tamanho dos círculos é proporcional ao número de parâmetros da rede, sendo: Conv4 com imagem de entrada de 110x110 ou 256x256 possui 9 milhões de parâmetros; AlexNet com imagem de entrada de 110x110 ou 256x256 possui 9 milhões de parâmetros; SqueezeNet possui 5 milhões de parâmetros; GoogLeNet possui 5 milhões de parâmetros; Na *Tree of DNN* cada DNN possui 4 milhões de parâmetros, totalizando 12 milhões de parâmetros, no entanto, durante a inferência apenas 2 DNNs são ativadas;

Esta seção mostrou que com a árvore de decisão de DNNs é possível aumentar a taxa de acerto e reduzir o tempo de inferência. Porque a árvore de decisão é composta por DNNs

Tabela 22 – Taxa de acerto e tempo médio de inferência em um computador Intel NUC.

Rede	**Entrada**	**Taxa de acerto [%]**	**Tempo [s]**	
			Média	σ
Conv4	110x110	74.43	0.093	0.00602
Alexnet	110x110	77.00	0.131	0.00811
SqueezeNet	256x256	79.57	0.139	0.01920
Conv4	256x256	83.86	0.458	0.02784
Alexnet	256x256	87.86	0.470	0.01591
GoogLeNet	256x256	96.00	0.682	0.03386
Tree of DNN	110x110	**97.01**	**0.035**	**0.00216**

Fonte: Autor.

menores e mais especializadas, dessa forma trazendo uma solução para o problema da incompatibilidade entre a taxa de acerto e o tempo de inferência, onde uma DNN com maior taxa de acerto possui um maior tempo de inferência, já uma DNN com baixa taxa de acerto possui um baixo tempo de inferência.

5.3.1 Experimento realizado no simulador

Este experimento mostra os resultados da proposta da árvore de decisão de DNNs sendo aplicada em um simulador de robótica. Para validar a proposta, o experimento foi realizado primeiramente no simulador, depois de validado a proposta, foram realizados nos robôs reais.

A taxa de aquisição de imagens de simulação atinge 30 quadros por segundos quando executada em Computadores equipados com GPU; no entanto, o sistema embarcado do robô possui recursos limitados. Assim, para assemelhar-se à simulação de aquisição de imagem do robô real, realizou-se uma taxa de aquisição de imagens de 8 quadros por segundo $(125ms)$. O argumento por trás da escolha desse valor está associado ao processo de controle, sendo um valor apropriado para o robô classificar uma imagem e executar uma ação.

Para o conjunto de dados foram geradas 36644 imagens de treinamento, 12214 imagens de validação e 7000 imagens de teste, com resolução de 640x360x3 (largura, altura, canais RGB). Na câmera do robô foi definido um campo de visão de 2.3415 radianos, para simular a câmera com uma lente olho de peixe com 130 graus de campo de visão. No simulador foram incluídos alguns robôs no campo de futebol em posições aleatórias para simular um ambiente de competição.

Como podemos ver na Tabela 23 todas as redes apresentaram taxa de acerto de 100% ou próxima de 100% em diversas classes. Essa alta taxa de acerto ocorre porque nas imagens do simulador há poucas variações como iluminação, e também o conjunto de dados no simulador é maior que o conjunto de dados dos experimentos no robô real devido à facilidade de capturar imagens do simulador.

5.4 DISCUSSÃO

Os experimentos apresentados nessa seção 5 provam que é possível utilizar uma rede neural profunda para controlar um robô a partir da imagem bruta da câmera sem qualquer pré-processamento da imagem, nesses experimentos a DNN guia o robô informando ao processo de

Tabela 23 – Taxa de acerto por classe.

Rede	Entrada	Taxa de acerto [%]						
		ball center	ball left	ball right	kick left	kick right	at goal	empty
Conv4	110x110	100	100	99.8	100	100	100	100
Alexnet	110x110	100	99.8	99.8	100	100	100	100
SqueezeNet	256x256	99.9	100	99.6	99.5	99.3	100	100
Conv4	256x256	100	100	99.9	96.3	98.4	100	100
Alexnet	256x256	100	99.9	99.9	99.8	99.6	100	100
GoogLeNet	256x256	100	99.8	100	100	100	100	99.7
Tree of DNN	110x110	99.9	99.9	99.9	99.9	99.9	100	100

Fonte: Autor.

controle qual ação deve ser executada para a imagem observada naquele instante de tempo. No entanto, em sistemas em que o robô precise realizar tarefas relacionadas a diversos processos, como receber informações passadas pelo juiz, interação entre multi-agentes, localização ou até mesmo planejar sua trajetória, a DNN possivelmente deverá ser combinada com os processos responsáveis por essas tarefas específicas.

Esses experimentos mostraram que para a tomada de decisões do robô, não há a necessidade de saber a localização precisa dos objetos na imagem (as informações podem ser abstraídas ou descartadas), a DNN pode apenas informar aos outros processos o que está sendo visto através de uma representação qualitativa daquele contexto do ambiente, por exemplo: a bola está à direita ou a bola está à esquerda.

Um problema encontrado nesse experimento é que quando o robô saía do gol para chutar a bola, na volta se ele se deslocasse muito ele não voltava exatamente no centro do gol. Para solucionar esse problema existem algumas alternativas, tais como: utilizar localização ou odometria. No caso da odometria poderia ser utilizado odometria visual, como a apresentada pelo Forster, Pizzoli e Scaramuzza (2014) e que está disponível no repositório do ROS [3].

A classe *empty* foi criada com dois propósitos: quando o robô não ver a bola volte ao gol e para que a DNN pudesse aprender que a bola é o objeto de maior importância, assim a DNN deverá extrair mais características da bola do que do ambiente (campo), no entanto, assim como as outras classes, a classe *empty* deve possuir o máximo de imagens em diferentes posições do campo e também com robôs do mesmo time ou oponentes no campo.

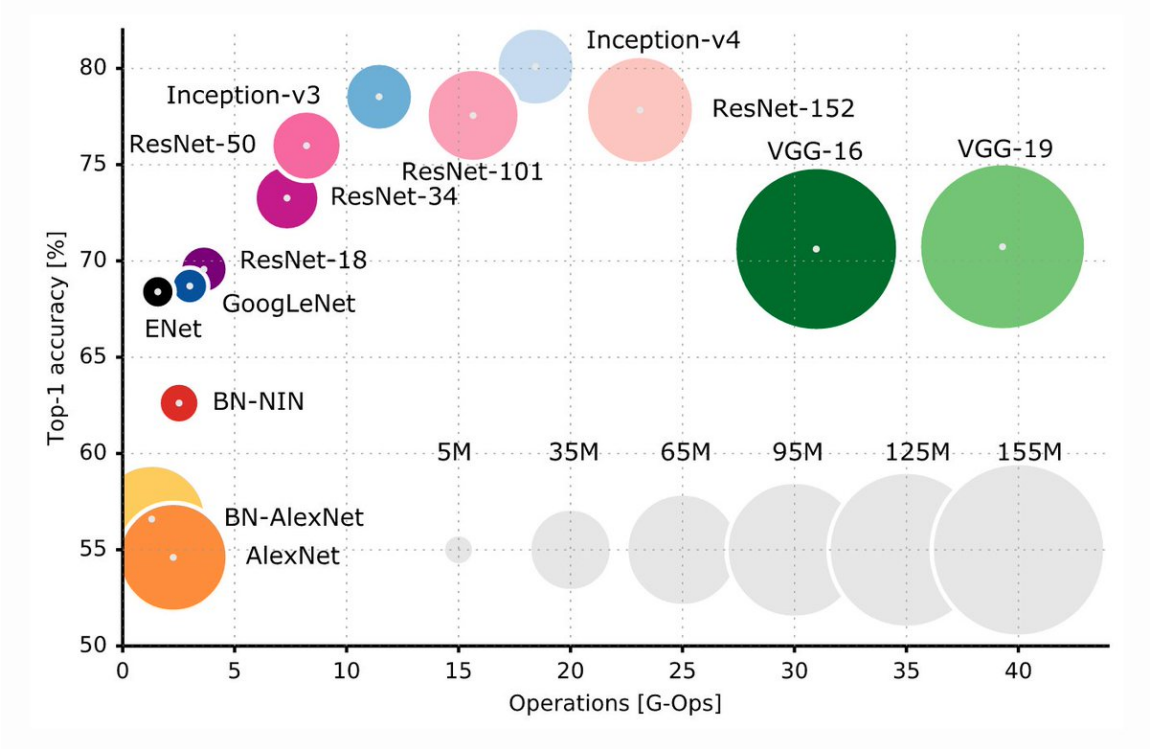
A diferença entre a taxa de acerto do robô real com o robô simulado se dá pelo fato de que durante o processo de captura de imagens de treinamento do robô real, obteve-se menos

[3] http://www.ros.org/news/2014/06/new-package-svo-semi-direct-monocular-visual-odometry.html

imagens de diferentes posições do campo, enquanto no robô simulado o processo de captura foi realizado de forma automatizado, onde o programa posicionou a bola e o robô em posições aleatórias do campo, dentro do limite de cada classe. No ambiente real as imagens da câmera do robô possuem mais variações de iluminação e também borramento ocasionados pelos movimentos do robô.

Para se reduzir o erro de generalização (erro de generalização é uma medida da precisão com que um classificador é capaz de prever resultados para dados nunca vistos, dados que não compõem a base de treinamento), deve-se aumentar o conjunto de dados do treinamento. Uma excelente forma de aumentar esse conjunto de dados é capturar imagens de diferentes posições do objeto em diferentes situações do ambiente, isso ajuda a reduzir o erro de generalização (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 57 – Gráfico que demonstra a acurácia pela quantidade de operações por inferência.



Fonte: CANZIANI; PASZKE; CULURCIELLO, 2016

Nos experimentos apresentados nessa seção foram utilizados as arquiteturas de DNN AlexNet, SqueezeNet, GoogLeNet e algumas arquiteturas com 4 e 3 convolucionais. Foram utilizadas esses tipos de arquiteturas por apresentarem um tempo de inferência de no máximo 600ms (GoogLeNet apresenta um tempo de inferência de aproximadamente 600 milissegundos) em um computador Intel NUC, arquiteturas que possuem um tempo superior a 600 milissegun-

dos torna-se inviável para esse projeto devido a dinâmica do jogo. A Figura 57 mostra um gráfico que demonstra a acurácia pela quantidade de operações (Giga operações por segundo, em inglês *Giga Operations Per Second* - GOPS) para uma inferência de cada DNN, o tamanho dos círculos é proporcional ao número de parâmetros da rede, e a quantidade de parâmetros varia de 5 milhões até 155 milhões de parâmetros, representados pelos círculos cinza na parte inferior direita do gráfico para ser usados como comparação com os círculos das redes.

Comparando-se a rede GoggLeNet com as outras redes, a Figura 57 mostra que não é possível utilizar as arquiteturas ResNets, VGGs ou as Inceptions-v3 e v4, isso porque essas DNNs durante a inferência realizam uma quantidade de operações muito maiores que a GoogLeNet, portanto sendo inviável para o sistema proposto.

Para facilitar o uso deste sistema por outros pesquisadores e encorajar o desenvolvimento de abordagens similares, o código fonte e o conjunto de dados estão disponíveis para download em: https://github.com/Isaac25silva/Goalkeeper-DNN.git

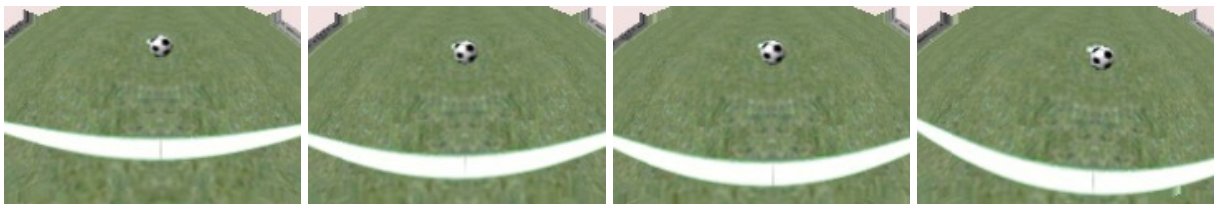
6 EXPERIMENTOS COM APRENDIZADO POR REFORÇO PROFUNDO

Nesta seção serão apresentados experimentos com Aprendizado por Reforço Profundo em um robô humanoide para aprender alguns papeis de um robô jogador de futebol. Os experimentos foram realizados com uma rede *Dueling Double* DQN. Foram realizados experimentos no domínio do futebol de robôs humanoides no simulador Webots e também no robô real.

Esta seção possui três seções secundárias (seções 6.1, 6.2 e 6.3) dos experimentos que foram realizados com DRL. Na seção 6.1 serão apresentados experimentos realizados no simulador Webots onde o robô humanoide aprende a executar o papel de um robô goleiro. Na seção 6.2 serão apresentados experimentos realizados no simulador Webots onde o robô humanoide aprende a chutar pênaltis. Na seção 6.3 serão apresentados experimentos realizados no robô humanoide real em que se realiza a transferência dos pesos aprendidos no simulador para executar no robô real.

A entrada da rede neural consiste em uma matriz de 84x84x4, onde a imagem é transformada no formato YUV e utilizado apenas o canal Y (luminância), e redimensionado o quadro para 84x84. Mas na entrada da rede utiliza-se 4 imagens sequenciais, portanto a entrada da

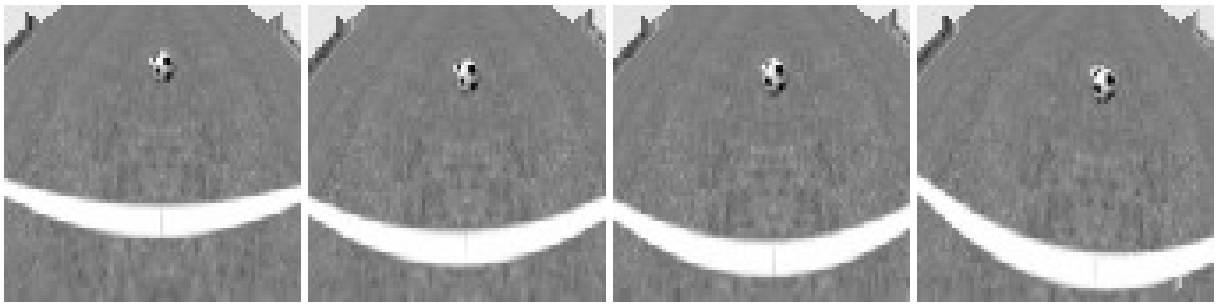
Figura 58 – Imagens da câmera do robô.



(a) (b) (c) (d)

Fonte: Autor.

Figura 59 – Imagens de entrada da rede de tamanho 84x84.



(a) (b) (c) (d)

Fonte: Autor.

Tabela 24 – Arquitetura.

Camada	**Entrada**	***Kernel***	***Stride***	**N. Filtros**	**Saída**
Conv1	84x84x4	8x8	4	32	20x20x32
Conv2	20x20x32	4x4	2	64	9x9x64
Conv3	9x9x64	3x3	1	64	7x7x64
fc1	7x7x64			512	512
fc2	512			7	7

Fonte: Autor.

rede neural consiste em uma imagem de 84x84x4. A primeira camada escondida é uma rede convolucional de 32 filtros de 8x8 com *stride* 4; a segunda camada escondida é uma rede convolucional de 64 filtros de 4x4 com *stride* 2; a terceira camada escondida é uma rede convolucional de 64 filtros de 3x3 com *stride* 1 seguido por um retificador; a última camada escondida é completamente conectada e consiste de 512 unidades; e a camada de saída também é completamente conectada com uma saída para cada ação válida, conforme Tabela 24.

Como mostrado na Figura 59, a entrada da rede consiste em 4 imagens de 84x84, isso faz com que a rede possa tomar decisões relacionadas a dinâmica do jogo. Em todos os experimentos a diferença temporal entre essas imagens foi de 300 milissegundos. Portanto a rede recebe uma imagem referente ao instante atual e 3 imagens passadas. Essas imagens são extraídas da câmera do robô sem realizar qualquer pré-processamento, é realizado apenas a redução da imagem para uma escala de 84x84 e conversão para a escala de cinza. No simulador foi utilizado um tamanho de imagem de 180x120, conforme apresentado na Figura 58.

Os treinamentos foram realizados em um computador com processador Intel i7-7700HQ 2.8GHz, memória 32GB DDR4 2133MHZ, SSD de 480GB, GPU NVIDIA GeForce GTX 1060 com 6GB memória DDR5 e o Sistema Operacional Ubuntu 14.04 e Ubuntu 16.04. Foram utilizadas as linguagens de programação C++ e Python e as DRLs foram implementadas utilizando TensorFlow[1] e como base o código do Plappert (2016) de DRL.

6.1 ROBÔ HUMANOIDE APRENDENDO O PAPEL DE UM ROBÔ GOLEIRO

O objetivo deste experimento é verificar se um algoritmo de Aprendizado por Reforço Profundo é capaz de aprender o papel de um robô goleiro, usando apenas as imagens da câmera

[1] https://www.tensorflow.org

do robô como entrada da rede, assim mostrando ser possível implementar um Sistema Cognitivo com DRL.

O episódio inicia com o robô goleiro parado no centro do gol (Figure 60). O goleiro deve permanecer posicionado na pequena área e verificar se há alguma bola próximo dele, a partir do momento que o goleiro percebe que a bola está próxima do seu gol, o goleiro deve ir até a bola e chutá-la para longe do gol, após chutar a bola ele deve retornar para a pequena área.

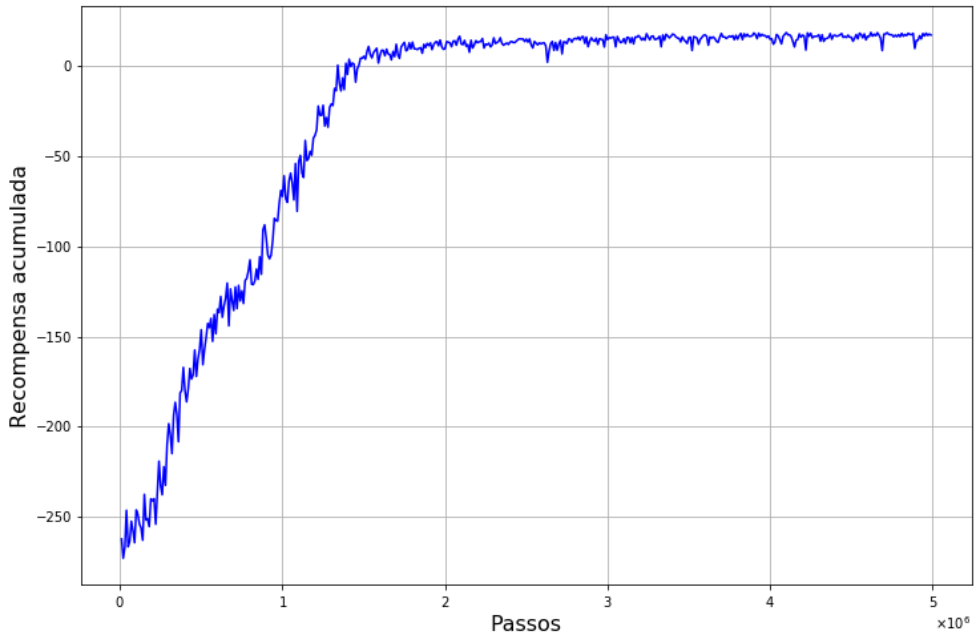
Figura 60 – Robô goleiro.

Fonte: Autor.

As recompensas positivas variam de 0 até 1 e as recompensas negativas variam de 0 até -1. O valor do $\epsilon - greedy$ foi decrementado de forma linear entre 1.0 até 0.1 nos primeiros 1 milhão de passos, depois permanece em 0.1. Foi treinado um total de 3 milhões de passos (a cada passo o DRL lê um quadro da imagem da câmera do robô), e o tamanho da memória $\mathcal{D}$ foi de 500 mil passos (a memória sempre contém os quadros mais atuais, seguindo a regra de FIFO - *First In First Out*). Recompensa = 1.0: Quando o robô chuta a bola para frente ou para o lado; Recompensa = 0.5: Quando a distância euclidiana entre o robô e a bola se torna menor que a distância do estado anterior, e quando a distância euclidiana entre o robô e o gol se torna menor que a distância do estado anterior, isso somente depois que o robô já chutou a bola; Recompensa = -0.1: Quando o robô fica parado no mesmo lugar; Recompensa = -0.5: Quando o robô fica de costas para o campo e de frente para o gol; Recompensa = -1.0: Para qualquer outra situação. Neste experimento o robô pode realizar 7 ações: virar a esquerda; virar a direita; andar para frente; chute direito; chute esquerdo; andar para trás; ficar parado.

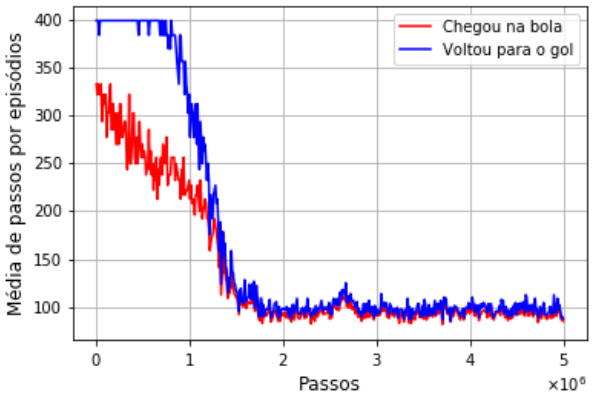
O gráfico da Figura 62.a mostra a progressão do aprendizado, em que o robô aprende a chegar na bola e completar o objetivo cada vez com menos passos por episódio, esse gráfico mostra a média de passos por episódio para cada 10000 passos, com 400 sendo o número máximo de passos por episódio.

Figura 61 – Média da recompensa acumulada por episódio a cada 10000 passos.

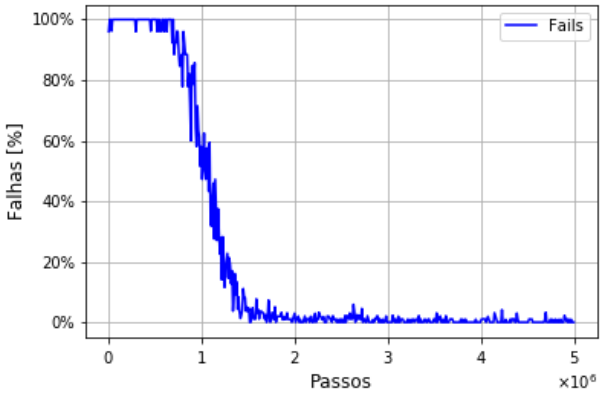


Fonte: Autor.

Figura 62 – Média de passos por episódio e porcentagem de falhas.



(a) Média de passos por episódio.



(b) Porcentagem de falhas.

Fonte: Autor.

O gráfico da Figura 62.b apresenta a porcentagem de falhas que ocorreu para cada 10000 passos, sendo que o número máximo de passos por episódio é 400 passos. Neste experimento a falha significa que o agente não realizou o objetivo, ir até a bola chutar e voltar para o gol.

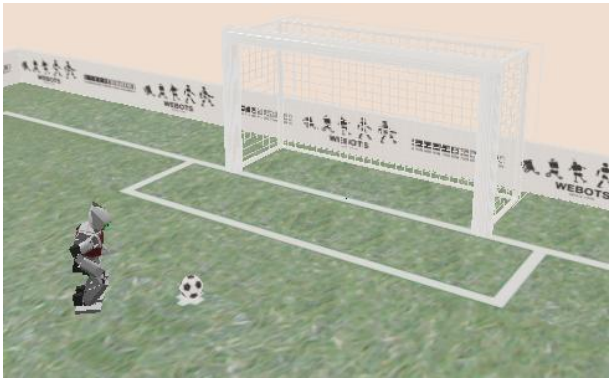
Esse primeiro experimento demonstrou que um algoritmo de Aprendizado por Reforço Profundo é capaz de aprender o papel de um robô goleiro, no entanto, o experimento foi realizado no simulador Webots executando 5 milhões de passos, e a simulação foi executada

em modo acelerado, acelerando entre 10 até 20 vezes. O código fonte está disponível em: https://github.com/Isaac25silva/DRL_goalkeeper.git.

6.2 ROBÔ HUMANOIDE APRENDENDO A CHUTAR PÊNALTIS

Neste experimento o robô deverá aprender a realizar cobranças de pênalti, a bola inicia sempre na marca do pênalti, já o robô inicia distante da bola em posições aleatórias, o robô deverá buscar maximizar o número de gols marcado. O experimento foi realizado no simulador com apenas um robô, não há goleiro ou robôs oponentes neste experimento, o robô deverá apenas aprender a andar até a bola e chutar a bola para dentro do gol. A Figura 63 mostra uma imagem do experimento realizado no simulador.

Figura 63 – Robô cobrador de pênaltis.



Fonte: Autor.

Se o robô aprender somente chegar até a bola, ele encosta na bola e a bola não chega até o gol, então há a necessidade de aprender a chutar a bola. Como o robô inicia em posições aleatórias, se ele simplesmente chegar até a bola e chutar, ele erraria o gol, então há a necessidade de aprender a se posicionar para marcar o gol. Devido a essas condições, esse objetivo de aprender a chutar pênaltis é mais difícil que o objetivo do papel do robô goleiro apresentado na seção 6.1.

Neste experimento há a necessidade do robô se posicionar de frente para o gol antes de chutar a bola, devido a essa condição foi necessário criar mais duas ações: rotacionar em volta da bola à esquerda e rotacionar em volta da bola à direita. Portanto, neste experimento o robô pode realizar 9 ações: virar a esquerda; virar a direita; andar para frente; chute direito; chute esquerdo; andar para trás; ficar parado; rotacionar em volta da bola a esquerda; rotacionar em volta da bola a direita.

Neste experimento a recompensa é diferente do experimento da seção 6.1, isso devido as condições do papel que será exercido pelo agente. Recompensa = 1.0: Quando o robô marca gol; Recompensa = -0.01:Quando o robô chuta a bola. Recompensa = -0.03: Quando a distância euclidiana entre o robô e a bola se torna menor que a distância do estado anterior; Recompensa = -0.05: Quando o robô fica parado no mesmo lugar. Recompensa = -0.1 Para qualquer outra situação.

Também foram testadas algumas outras recompensas, como por exemplo uma recompensa de 1 para fazer gol, -0.5 por se aproximar da bola e -1 para outras situações. No entanto, o agente não aprendeu a fazer gol, aprendeu apenas a chegar na bola.

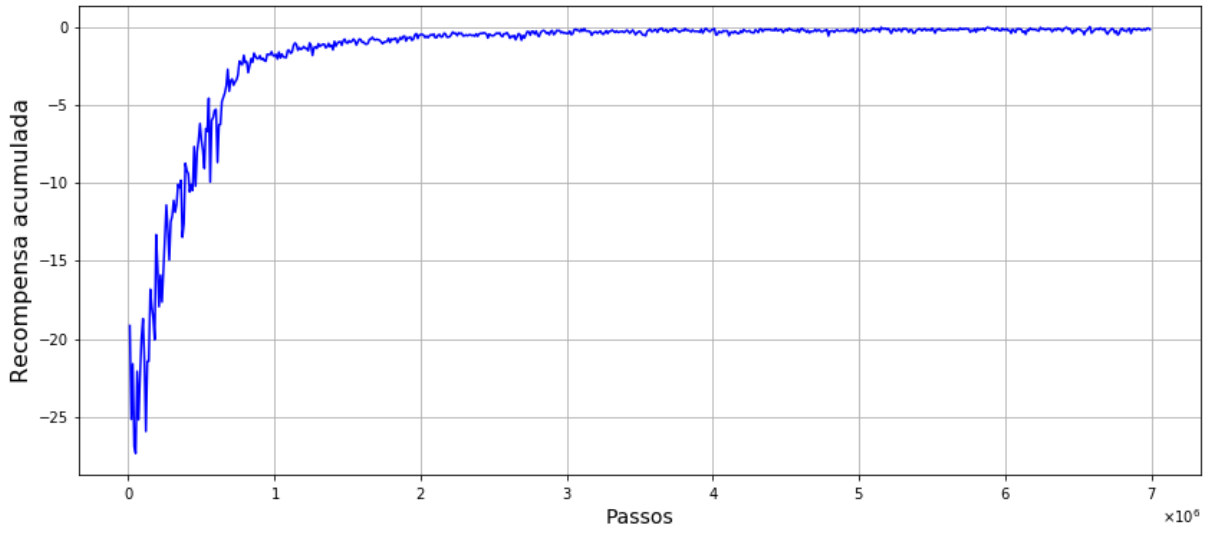
Foi utilizado um $\epsilon - greedy$ com decremento exponencial conforme a Equação (35).

$$\epsilon - greedy(x) = \frac{(1-a)}{e^{(\frac{1}{b}2ex)}} + a \tag{35}$$

Onde: a é o valor mínimo de exploração; b é o número de passos para atingir o valor mínimo de exploração; x é o step que o agente está. Foi usado um $b = 4$ milhões de passos A exploração finalizando em 1 ou 2 milhões de episódios não funcionou, isso porque o decremento exponencial com final em 2 milhões de passos decai mais rápido que um decremento linear de 1 milhão de passos, isso nos primeiros passos. Portanto, foi utilizado o exponencial finalizando em 4 milhões de passos ($b = 4$).

O gráfico da Figura 65.a mostra a progressão do aprendizado em que o robô aprende a alcançar a bola e chutar, esse gráfico mostra a média de passos por episódio para cada 10000

Figura 64 – Média da recompensa acumulada por episódio a cada 10000 passos.



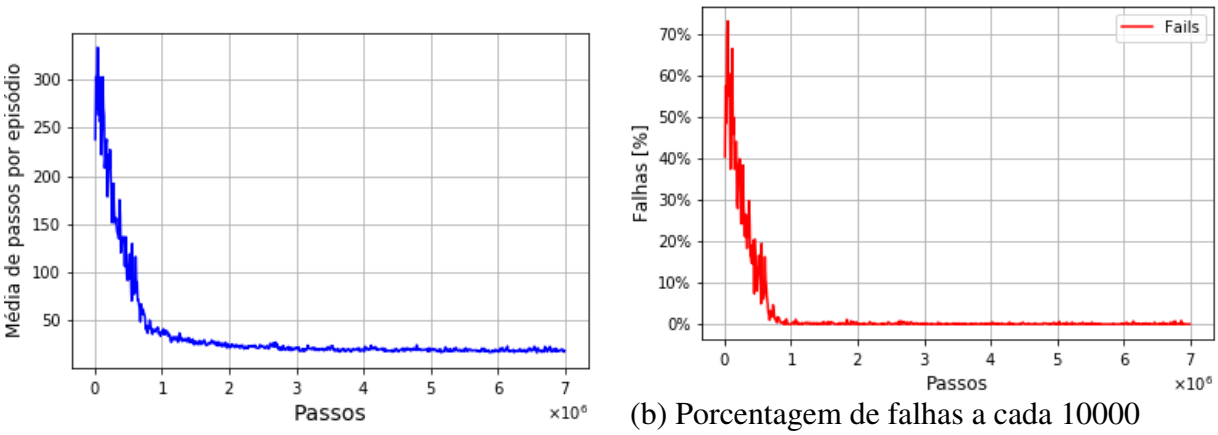
Fonte: Autor.

passos, com 400 sendo o número máximo de passos por episódio. O gráfico da Figura 65.b apresenta a porcentagem de falhas que ocorreu para cada 10000 passos, sendo que o número máximo de passos por episódio é 400 passos. Neste experimento a falha significa que o agente não realizou o objetivo, chutar a bola no gol, e nem sequer chegou até a bola.

O gráfico da Figura 66 apresenta a quantidade de gols marcadas para cada 10000 passos durante o treinamento, essa Figura mostra que durante o processo de aprendizado o agente marca cada vez mais gols, portanto aprendendo a marcar gols, porém o gráfico não apresenta quantos gols foram perdidos durante esses 10000 passos, por isso houve a necessidade de apresentar um gráfico de rendimento apresentado a porcentagem de gols marcados a cada 10000 passos.

O gráfico da Figura 67 apresenta a porcentagem de gols marcados em relação aos pênaltis cobrados para cada 10000 passos durante o treinamento, essa Figura mostra que durante o processo de aprendizado a porcentagem de gols marcados ficou acima de 70% e em alguns momentos ultrapassando os 80%. Após o treinamento, foi realizado teste com a rede aprendida. No teste foi executado 10000 cobranças de pênalti, onde o agente fez 8694 gols de 10000 chutes ao gol (durante estes 10000 episódios o robô foi até a bola, em nenhum episódio foi atingido o número máximo de passos por episódio), portanto uma porcentagem de 86.9% de gols marcados. O código fonte está disponível em: https://github.com/Isaac25silva/DRLpenalty.git.

Figura 65 – Porcentagem de falhas e média de passos por episódio.

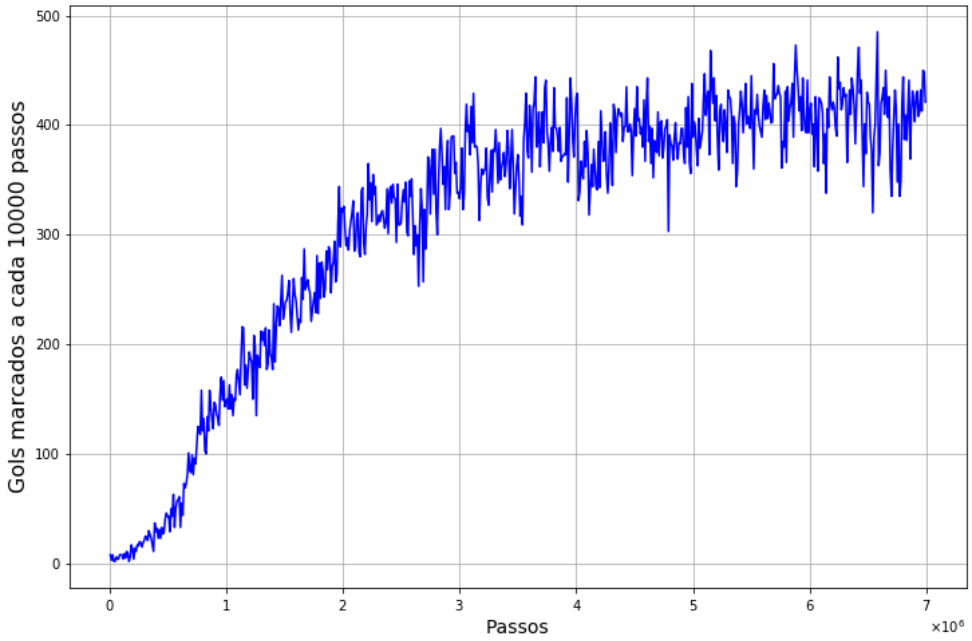


(a) Média de passos por episódio.

(b) Porcentagem de falhas a cada 10000 passos.

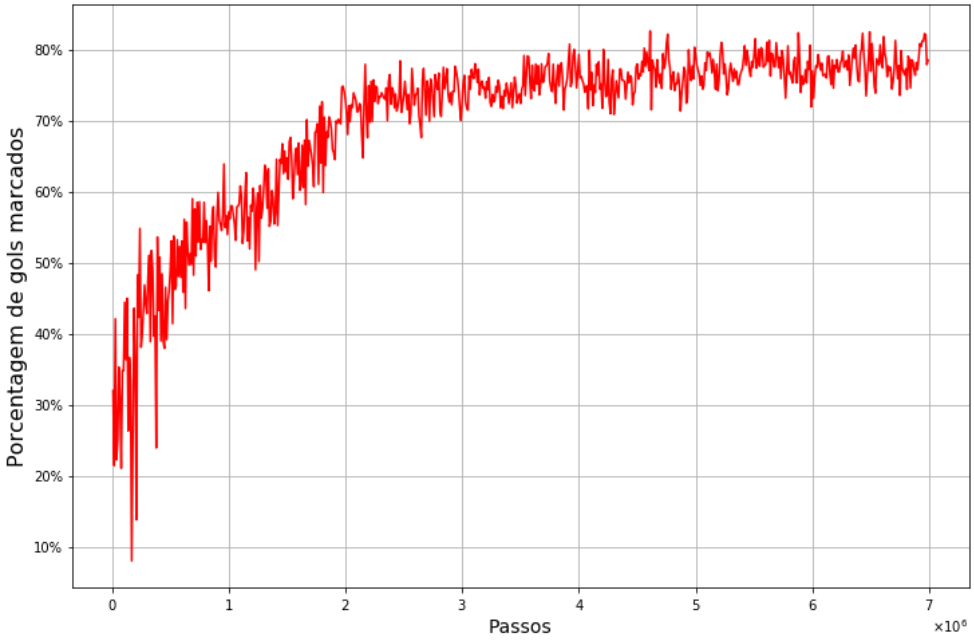
Fonte: Autor.

Figura 66 – Gols marcados para cada 10000 passos.



Fonte: Autor.

Figura 67 – Porcentagem de gols marcados.



Fonte: Autor.

6.2.1 Discussão

Uma diferença que foi observada entre o experimento do robô goleiro e o experimento do pênalti é que o DRL aplicado em um problema como o do goleiro, é mais fácil de ser apren-

dido do que o problema do pênalti, isso porque no problema do robô goleiro para que o agente alcance o objetivo, ele precisa apenas seguir uma sequência de ações. Já no problema do pênalti, se o agente apenas seguir uma sequência de ações, ele encostará na bola, mas possivelmente não fará o gol, então o problema que ocorre no aprendizado desse papel é que o objetivo é apresentado apenas no último passo, onde o robô no último passo recebe uma recompensa por fazer ou não o gol, e essa recompensa deve ser espalhada para os outros estados. Nos primeiros episódios o robô aprende rapidamente a chegar na bola, no entanto, para aprender a se posicionar para chutar a bola em direção do gol, o aprendizado ocorre de forma mais lenta. Foi possível perceber que neste experimento a função de recompensa é importante para determinar o que o agente deve aprender e também para acelerar o aprendizado.

Outro problema encontrado neste experimento foi a estabilização da recompensa acumulada após milhares de passos executados (Figura 64), porém essa estabilização não significava que o agente aprendeu a bater pênaltis. Um outro gráfico que também não demonstrava o aprendizado do agente foi o gráfico do número de passos por episódios, onde após milhares de passos executados pelo agente, o gráfico apresentava poucos passos por episódio, porém isso representava apenas que o agente aprendeu a chegar na bola e chutar, porém chutava a bola para qualquer direção.

Então para monitorar de forma eficaz o aprendizado do agente neste experimento, houve a necessidade de criar um gráfico de rendimento do agente que mostra a porcentagem de gols marcados para cada 10000 passos (Figura 67). Com isso, após vários testes foi possível desenvolver uma função de recompensa adequada a esse problema.

6.3 TRANSFERIR OS PESOS APRENDIDOS NO SIMULADOR PARA EXECUTAR NO ROBÔ REAL

Neste experimento o objetivo é realizar uma transferência dos pesos aprendido no simulador para executar no robô real, isso porque realizar o aprendizado no robô humanoide real é algo inviável devido a quantidade de passos necessários. Todo o processo de aprendizado foi realizado no simulador 3D, depois de realizado o processo de aprendizado no simulador, os pesos da rede foram transferidos para o robô real, no robô real não foi realizado nenhum processo de aprendizado, apenas foi executado a rede. Nesta seção será descrito todo o processo que resultou em uma transferência sem necessidade de aprendizado no robô real. O código fonte desse experimento está disponível no github em .

Em Transferência de Aprendizado, os métodos de aprendizado de máquina são desenvolvidos para usar um conhecimento aprendido em um domínio, para diminuir o tempo de aprendizado em um outro domínio (WEISS; KHOSHGOFTAAR; WANG, D., 2016). Nessa pesquisa o objetivo é realizar a transferência dos pesos da rede aprendido em um simulador para ser executada no robô real sem a necessidade de realizar nenhum treinamento no robô real, não sendo uma transferência de aprendizado, porque não foi realizado nenhum treinamento no robô real.

Este experimento foi realizado usando a *Dueling Double DQN* em um robô humanoide no simulador Webots. No simulador foi utilizado o robô humanoide DARwIn-OP em um campo de futebol, disponível no simulador versão 8.0.5 usando linguagens de programação C++ e Python. A arquitetura do *Dueling Double DQN* é a mesma descrita na seção 6.

Para obter sucesso na transferência dos pesos da rede treinada no simulador e poderem ser executados no robô real com um similar desempenho, algumas premissas foram consideradas, dessa forma buscando manter a imagem no simulador com as características similares as da imagem do ambiente real. No entanto o mundo virtual do simulador não possui as mesmas variações do ambiente que o mundo real (variação de iluminação, textura e cor), então para que a DQN se tornasse menos sensível a essas variações, a primeira premissa foi realizar no simulador diversas variações de iluminação, textura e cor do campo e da bola.

A segunda premissa considerada nesse experimento foi em relação ao tamanho dos objetos observado pela rede, para que a DQN se tornasse invariante ao tamanho da bola, foi realizado variações no tamanho da bola durante o treinamento e também realizado distorções na lente olho de peixe.

A terceira premissa foi manter a mesma relação de tamanho dos quadros da câmera do robô tanto do simulador quanto do robô real, isso porque modificar a relação da câmera faz com que a distorção dos objetos sejam diferentes na entrada da rede.

Antes de iniciar o experimento foi realizado um teste de quanto tempo a rede gasta durante um *feedforward* no robô real. A Tabela 25 apresenta o tempo que uma rede DQN gasta de inferência em um Intel NUC i5-4250U, 8GB SDRAM, 120GB SDD [2]. As amostram foram capturadas durante um período de aproximadamente 10 minutos. O tempo de inferência de aproximadamente 15 milissegundos é considerado satisfatório, porque o tempo mínimo que o robô gasta para a tomada de ação é de aproximadamente 300 milissegundos.

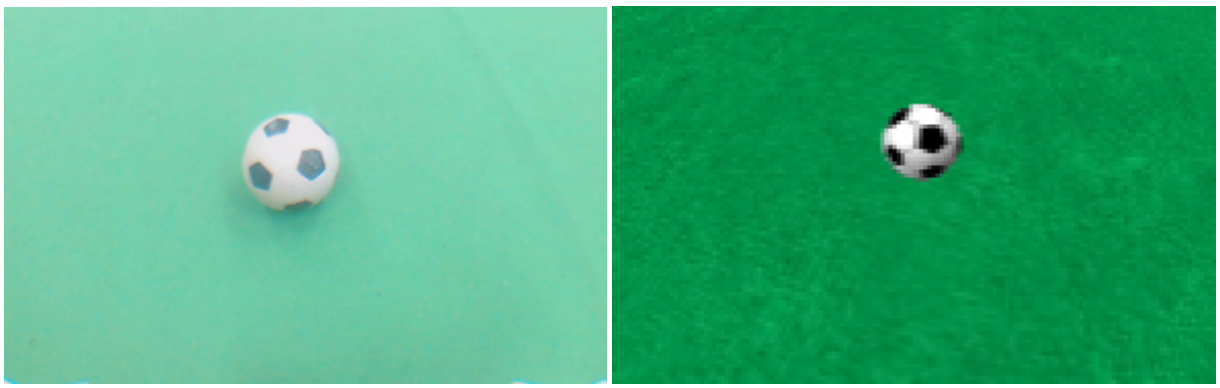
[2]http://www.intel.com/content/www/us/en/nuc/overview.html

Tabela 25 – Tempo de inferência da DQN em um computador Intel NUC.

Rede	**Entrada**	**Tempo [s]**	
		Média	σ
Duel DQN	84x84	0.0148	0.00341

Fonte: Autor.

Figura 68 – Imagem da câmera do robô.



(a) Robô real. (b) Simulação.

Fonte: Autor.

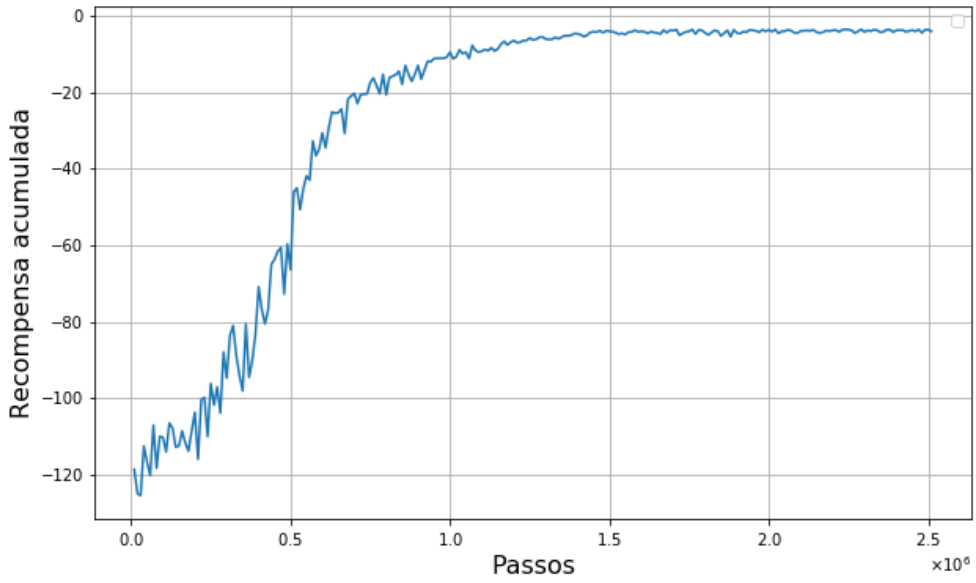
Figura 69 – Imagem da câmera do robô.

(a) Esquerda. (b) Centro. (c) Direita.

Fonte: Autor.

A Figura 68 mostra que as imagens da câmera do robô no simulador são parecidas com as imagens do robô real, porém possuem pequenas diferenças na cor do campo, da bola e também diferenças no formato da bola. No entanto variações relacionadas as essas diferenças foram realizadas no simulador durante o treinamento. O tempo para executar uma ação foi fixado em 300 milissegundos para dar tempo de o robô durante o andar realizar pelo menos 1 passo a cada ação. Na câmera do robô só aparecia o campo e a bola conforme apresentado na Figura 68. Vale ressaltar que o sucesso deste experimento só foi possível respeitando-se as premissas apresentadas nessa seção.

Figura 70 – Média da recompensa acumulada por episódio a cada 10000 passos.



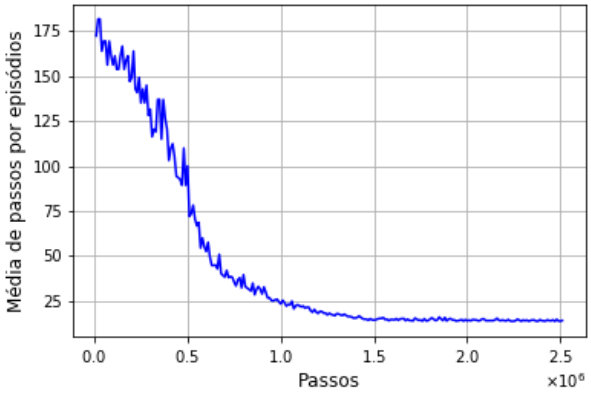
Fonte: Autor.

Neste experimento a bola foi posicionada próxima do robô, e o objetivo é que o robô fosse até a bola. Foram realizados três experimentos conforme mostrado na Figura 69: um com a bola a frente do robô, bola na frente direita do robô; e bola na frente esquerda do robô, conforme Figura 69. O episódio inicia quando a bola está posicionada dentro do campo de visão do robô, e o episódio termina quando o robô encosta na bola.

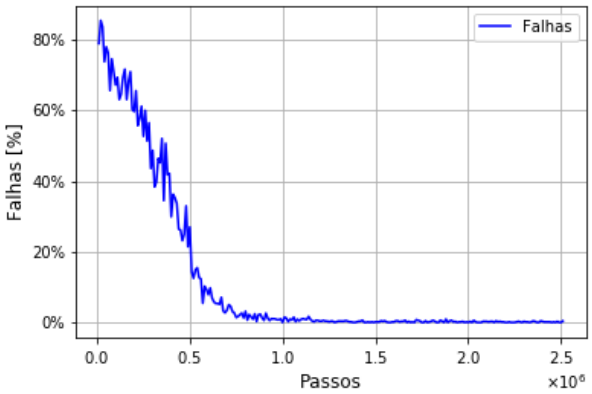
Durante a execução do aprendizado na simulação a Figura 70 mostra que o algoritmo de DRL foi capaz de maximizar a recompensa. Os gráficos da Figura 71 mostra que a quantidade de passos por episódio foi diminuindo ao longo do tempo. E a porcentagem de falha por episódio foi reduzido a zero falhas (é considerado falha no episódio quando o robô atinge o limite máximo de 200 passos no episódio).

Recompensa = 1.0: quando o robô chuta a bola para frente ou para a lateral, e quando o robô retornou para o gol após chutar a bola; Recompensa = -0.3: quando a distância euclidiana entre o robô e a bola se torna menor que a do estado anterior; Recompensa = -0.5: quando o robô fica parado; Recompensa = -1.0 para qualquer outra situação. O valor do ϵ-greedy foi decrementando de 1.0 até 0.1 no primeiro 1 milhão de passos, e depois permanece 0.1. O treinamento foi realizado em 2.5 milhões de passos e o tamanho da memória *replay* foi de 500000 passos. Neste experimento o robô pode realizar 5 ações: virar a esquerda; virar a direita; andar para frente; andar para trás; e ficar parado.

Figura 71 – Passos por episódios e porcentagem de falhas.



(a) Média de passos por episódio.



(b) Porcentagem de falhas.

Fonte: Autor.

Tabela 26 – Média de passos por episódio no robô real e simulado.

Posição da bola	**Robô real**			**Simulado**		
	Média	σ	**Falhas**	**Média**	σ	**Falhas**
Centro	15.7	7.74	0	9.90	0.88	0
Direita	16.6	6.06	2	21.4	2.75	0
Esquerda	19.2	6.65	2	18.1	2.01	0

Fonte: Autor.

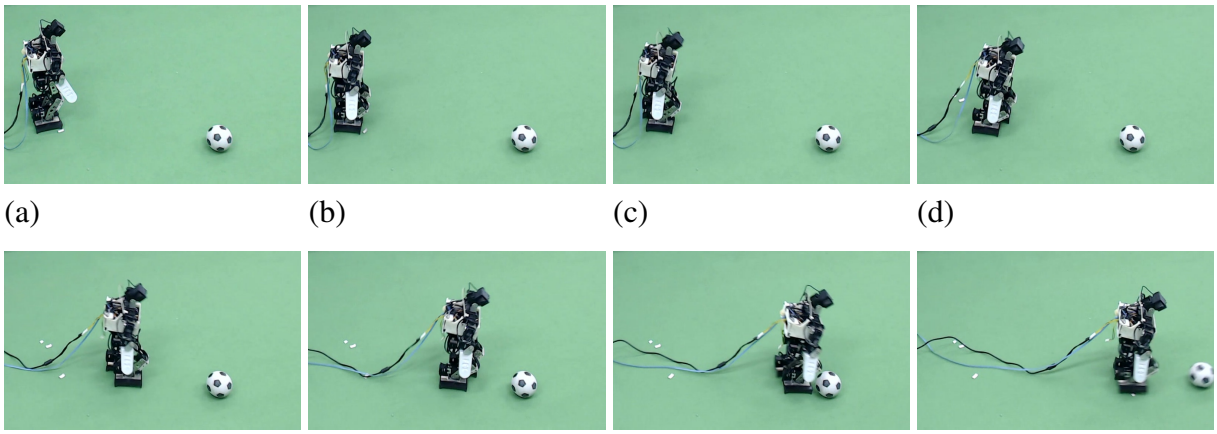
O gráfico da Figura 71.a mostra a progressão do aprendizado em que o robô aprende a alcançar a bola, o gráfico mostra a média de passos por episódio para cada 10000 passos, com 200 sendo o número máximo de passos por episódio. O gráfico da Figura 71.b apresenta a porcentagem de falhas que ocorreu para cada 10.000 passos. Neste experimento a falha significa que o agente não realizou o objetivo de chegar até a bola.

O experimento apresentado na Tabela 26 foi realizado no robô real e também no simulador, o robô inicia de uma posição fixa no campo, e a bola também inicia em uma posição fixa determinada no experimento, o objetivo é que o robô ande em direção a bola e o episódio é finalizado quando o robô encosta na bola ou quando atinge o limite máximo de 200 passos por episódio. Foram realizados três experimentos: um com a bola iniciando na frente a direita do robô, na frente a esquerda do robô, e na frente no centro (foram escolhidas posições onde a bola pudesse estar dentro do campo de visão do robô). Os resultados podem ser observados na Tabela 26 e foram realizadas 30 amostras para cada posição da bola.

A Tabela 26 mostra a média e o desvio padrão de passos por episódio realizados no robô real. Podemos observar que quando a bola iniciou a direita ou a esquerda do robô houveram

duas falhas entre as 30 amostras, essas falhas ocorreram porque quando a bola inicia a direita ou a esquerda, a bola aparece nas extremidades do campo de visão do robô, e no inicio o robô tomava uma ação que fazia com que a bola saísse do campo de visão, com isso os 4 quadros de imagem usados no DRL não foram suficientes para guardar a última posição da bola.

Figura 72 – Bola à direita.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fonte: Autor.

A imagem da Figura 72 apresenta a movimentação do robô em uma das amostras do experimento, onde a bola inicia da posição frente direita. A Figura 72.a apresenta o início do episódio, a Figura 72.b mostra que o robô executou a ação virar a direita, as ações das Figuras 72.b até Figura 72.g o robô executou as ações andar para frente, e a Figura 72.h mostra que o robô encostou na bola empurrando a bola para frente.

Esse experimento mostrou que é possível realizar o treinamento no simulador e usar os pesos da rede no robô real, assim possibilitando que todo o treinamento seja realizado apenas em simulação. No entanto para isso, deve-se respeitar algumas premissas que foram apresentadas nessa seção. Esse treinamento foi realizado em uma rede de Aprendizado por Reforço Profundo chamada Dueling Double DQN. O código fonte está disponível em: https://github.com/Isaac25silva/DRLtransfer.git.

6.3.1 Discussão

Esse experimento demonstrou que é possível realizar um aprendizado no simulador e executá-lo no robô real. No entanto, o experimento mostra que o ambiente do simulador deverá ser parecido com o ambiente do robô real, inclusive apresentando variações de iluminação, cor

e distorções que ocorrem no mundo real, mas mesmo que o ambiente no simulador não seja exatamente como o ambiente real, o DRL aprendido no simulador apresentou um comportamento similar no robô real.

Inicialmente, foi realizado um experimento de transferência dos pesos da rede aprendida no papel do robô goleiro no simulador para o robô real, porém não houve êxito devido a diversos problemas identificados em diversos testes que foram realizados. Esses problemas são: variações de iluminação, cor e distorções que ocorrem no mundo real, relação entre altura e largura da imagem, objetos no plano de fundo da imagem, textura e cor do campo, tamanho da bola. Então foi realizado um experimento obedecendo algumas premissas que foram consideradas nesta tese, dessa forma buscando manter a imagem no simulador com as características similares as da imagem do ambiente real.

No experimento a bola foi posicionada próxima do robô, e o objetivo e que o robô vá até a bola, dessa forma sendo um experimento mais simples que o do robô goleiro, foi necessário também simplificar o ambiente removendo objetos que pudessem aparecer na imagem de fundo, assim tornando a imagem do robô muito parecida com a imagem do simulador.

7 CONCLUSÃO

Essa tese apresentou dois Sistemas Cognitivos para agentes robóticos com Aprendizado Profundo, sendo um Sistema Cognitivo com Redes Neurais Profundas e um Sistema Cognitivo com Aprendizado por Reforço Profundo, e para observar o ambiente os sistemas usam apenas a imagem da câmera do robô sem qualquer pré-processamento na imagem. Os experimentos e resultados mostram que em ambos os sistemas o robô foi capaz de realizar tarefas relacionadas ao papel de um robô jogador de futebol.

O Sistema Cognitivo com Redes Neurais Profundas mostra que uma DNN pode controlar um robô em substituição aos processos de visão e decisão, usando apenas imagens brutas da câmera do robô sem qualquer pré-processamento de imagem. Uma vantagem deste modelo é que, durante o treinamento, não há necessidade de informar a localização do objeto na imagem, diferente de outras técnicas, onde é necessário informar manualmente a posição do objeto na imagem. Também foi apresentado uma nova arquitetura sendo uma Árvore de Decisão de Redes Neurais Profundas, assim sendo possível aumentar a acurácia diminuindo o custo computacional (diminuindo o tempo de inferência da DNN).

Os Sistemas de Cognição com Redes Neurais Profundas e com a Árvore de Decisão de Redes Neurais Profundas foram testados primeiramente em simulação e posteriormente no robô real, essa tese apresentou parte dos resultados obtidos dos experimentos, e todo o código fonte e o conjunto de dados está disponível para download em: https://github.com/Isaac25silva/Goalkeeper-DNN.git

O Sistema Cognitivo com Aprendizado por Reforço Profundo mostra que o Aprendizado por Reforço Profundo pode proporcionar aos robôs a capacidade de aprender tarefas apenas pela observação do ambiente através de imagens proveniente da própria câmera do robô, tarefas relacionadas ao papel de um robô jogador de futebol, tais como: goleiro e batedor de pênalti. No entanto, o aprendizado ocorreu no simulador, isso porque aprender tais tarefa exigem que o agente execute milhares de vezes determinada tarefa durante o processo de aprendizado. Com isso, mesmo no simulador o processo de aprendizado gasta dias ou semanas (no simulador a simulação é realizada de forma acelerada). Porém, no robô real torna-se inviável realizar esse processo de aprendizado, então esse artigo investiga e demonstra o processo de aprendizado sendo realizado no simulador, e depois de aprendido, o modelo é usado no robô real.

Esse trabalho também demonstrou que é possível realizar um aprendizado no simulador e executá-lo no robô real. No entanto, o experimento mostra que o ambiente do simulador deverá ser parecido com o ambiente do robô real, inclusive apresentando variações de ilumi-

nação, cor e distorções que ocorrem no mundo real, mas mesmo o ambiente no simulador não sendo exatamente como o ambiente real, o DRL aprendido no simulador apresentou um comportamento similar no robô real. No entanto, para obter êxito no experimento, foi necessário simplificar o ambiente removendo objetos que pudessem aparecer na imagem de fundo. Então um dos trabalhos futuro é investigar a realização do aprendizado com uma imagem de fundo mais poluída (com vários objetos aparecendo no fundo da imagem).

Comparando-se o Sistema Cognitivo com Aprendizado por Reforço Profundo com o Sistema Cognitivo com Redes Neurais Profundas, pode-se concluir que uma das desvantagens do Sistema Cognitivo com DNN é não aprender a dinâmica do ambiente, isso porque a DNN recebe apena uma imagem na entrada, já o Sistema Cognitivo com DRL recebe 4 imagens na entrada, assim aprendendo também a dinâmica do ambiente. No entanto a atual desvantagem do Sistema Cognitivo com DRL é a inviabilidade de se realizar o aprendizado nos robôs reais.

A partir do trabalho realizado nesta tese foram publicados e submetidos alguns artigos. O artigo intitulado: *"Towards Robotic Cognition Using Deep Neural Network Applied in a Goalkeeper Robot"* foi publicado no *Latin American Robotics Symposium* (LARS) em 2017, este artigo apresenta o sistema de cognição com DNN. O artigo intitulado: *"Toward Robotic Cognition by Means of Decision Tree of Deep Neural Networks applied in a Humanoid Robot"* foi submetido para a revista *Journal of Robotics and Autonomous Systems*, este artigo apresenta o sistema de cognição com árvore de decisão de DNNs. O artigo intitulado: *"Deep Reinforcement Learning in a Humanoid Robot to Learn Play Soccer"* será submetido para a revista *Journal of Control, Automation and Electrical Systems*, este artigo apresenta o sistema de cognição com Aprendizado por Reforço Profundo. Mais artigos publicados durante o Doutorado são apresentados no Anexo A. Esta Tese também foi premiada como melhor Tese de Doutorado em Robótica de 2019.

Como um trabalho futuro, poderá ser aplicado o modelo aos demais tipos de jogadores no futebol de robôs, podendo também integrar o modelo com os demais processos informando o que está sendo visto através da representação qualitativa das observações do ambiente.

Um outro trabalho futuro é realizar o Aprendizado por Reforço Profundo em outras tarefas de um robô jogador de futebol tais como: goleiro aprendendo a defender a bola; jogador aprendendo quando deve conduzir a bola ou realizar um passe; jogador atacante aprendendo quando chutar a bola para o gol. Tendo todas essas tarefas aprendidas, elas poderão ser gerenciadas por um processo responsável por ativar ou desativar cada DRL de cada tarefa, onde esse processo será responsável pela deliberação sobre qual tarefa usar.

REFERÊNCIAS

ALBANI, Dario et al. A deep learning approach for object recognition with nao soccer robots. In: SPRINGER. ROBOT World Cup. [S.l.: s.n.], 2016. p. 392–403.

BIANCHI, Reinaldo Augusto da Costa. **Uso de heurísticas para a aceleração do aprendizado por reforço.** 2004. Tese (Doutorado) – Universidade de São Paulo.

CANZIANI, Alfredo; PASZKE, Adam; CULURCIELLO, Eugenio. An analysis of deep neural network models for practical applications. **arXiv preprint arXiv:1605.07678**, 2016.

TEAM Description Paper: CITBrains (Kid Size League). CITBrains. 2017. Disponível em: <https://www.robocuphumanoid.org/qualification/2017/2afcdec2438a38a31a888ceb8e1a0f0302809d7a/CIT_Brains_Kid_Humanoid_KidSize_regularanddrop_in_2017_TDP.pdf>.

DENG, Jia et al. Imagenet: A large-scale hierarchical image database. In: IEEE. COMPUTER Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. [S.l.: s.n.], 2009. p. 248–255.

DENG, Li; YU, Dong et al. Deep learning: methods and applications. **Foundations and Trends® in Signal Processing**, Now Publishers, Inc., v. 7, n. 3–4, 2014.

DIJK, Sander G van; SCHEUNEMANN, Marcus M. Deep Learning for Semantic Segmentation on Minimal Hardware. **RoboCup 2018**, Springer Verlag, 2019.

DRAKE, PR; PACKIANATHER, MS. A decision tree of neural networks for classifying images of wood veneer. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 14, n. 4, p. 280–285, 1998.

ERSEN, Mustafa; OZTOP, Erhan; SARIEL, Sanem. Cognition-Enabled Robot Manipulation in Human Environments: Requirements, Recent Work, and Open Problems. **IEEE Robotics & Automation Magazine**, IEEE, 2017.

FAESSLER, Matthias et al. Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle. **Journal of Field Robotics**, John Wiley & Sons, v. 1, 2015.

FORSTER, Christian; PIZZOLI, Matia; SCARAMUZZA, Davide. SVO: Fast semi-direct monocular visual odometry. In: IEEE. ROBOTICS and Automation (ICRA), 2014 IEEE International Conference on. [S.l.: s.n.], 2014. p. 15–22.

GIUSTI, Alessandro et al. A machine learning approach to visual perception of forest trails for mobile robots. **IEEE Robotics and Automation Letters**, IEEE, v. 1, n. 2, p. 661–667, 2016.

GONZALEZ, Rafael C; WOODS, Richard E. Image processing. **Digital image processing**, v. 2, 2007.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org.

GU, Shixiang et al. Continuous deep q-learning with model-based acceleration. In: INTERNATIONAL Conference on Machine Learning. [S.l.: s.n.], 2016. p. 2829–2838.

GU, Shixiang et al. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In: IEEE. 2017 IEEE international conference on robotics and automation (ICRA). [S.l.: s.n.], 2017. p. 3389–3396.

HA, I. et al. Development of Open Humanoid Platform DARwIn-OP. IEEE in Proceedings of SICE Annual Conference (SICE), p. 2178–2181, 2011.

HASSELT, Hado V. Double Q-learning. In: ADVANCES in Neural Information Processing Systems. [S.l.: s.n.], 2010. p. 2613–2621.

HAUSKNECHT, Matthew; STONE, Peter. Deep reinforcement learning in parameterized action space. **arXiv preprint arXiv:1511.04143**, 2015.

_____. On-policy vs. off-policy updates for deep reinforcement learning. In: DEEP Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop. [S.l.: s.n.], 2016.

HAUSKNECHT, Matthew et al. Half field offense: An environment for multiagent learning and ad hoc teamwork. In: AAMAS Adaptive Learning Agents (ALA) Workshop. [S.l.: s.n.], 2016.

HAYKIN, Simon S. **Neural networks and learning machines**. [S.l.]: Pearson Upper Saddle River, NJ, USA, 2009. v. 3.

HE, Kaiming et al. **Deep Residual Learning**. 2017. Disponível em: <http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf>.

HE, Kaiming 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.

HESSEL, Matteo et al. Rainbow: Combining improvements in deep reinforcement learning. In: THIRTY-SECOND AAAI Conference on Artificial Intelligence. [S.l.: s.n.], 2018.

HINTON, Geoffrey E; OSINDERO, Simon; TEH, Yee-Whye. A fast learning algorithm for deep belief nets. **Neural computation**, MIT Press, v. 18, n. 7, p. 1527–1554, 2006.

HINTON, Geoffrey E; SALAKHUTDINOV, Ruslan R. Reducing the dimensionality of data with neural networks. **science**, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006.

HINTON, Geoffrey E et al. Improving neural networks by preventing co-adaptation of feature detectors. **arXiv preprint arXiv:1207.0580**, 2012.

HOWARD, Andrew G et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

HU, Jie; SHEN, Li; SUN, Gang. Squeeze-and-Excitation Networks. In: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [S.l.: s.n.], 2018. p. 7132–7141.

HOW Drive.ai Is Mastering Autonomous Driving With Deep Learning. IEEE Spectrum. 2017. Disponível em: <http://spectrum.ieee.org/cars-that-think/transportation/self-driving/how-driveai-is-mastering-autonomous-driving-with-deep-learning?utm_source=Tech+Alert&utm_medium=Email&utm_campaign=TechAlert_03-16-17&bt_ee=5cuijb9FOLVF4Y4DDF3azCR76Knhu2YXv0E7wD93/GVsmK3nxymqWkiLsxxWGh0n&bt_ts=1489674242217>.

IEEE Robotics and Automation Society - Cognitive Robotics. IEEE-RAS. 2019. Disponível em: <http://www.ieee-ras.org/cognitive-robotics>.

RESULTADOS da Imagenet Large Scale Visual Recognition Challenge 2012. ILSVCR2012. 2017. Disponível em: <http://image-net.org/challenges/LSVRC/2012/results.html>.

IMAGENET Large Scale Visual Recognition Challenge. ImageNet. 2017. Disponível em: <http://www.image-net.org/challenges/LSVRC/>.

JADERBERG, Max et al. Reinforcement learning with unsupervised auxiliary tasks. **arXiv preprint arXiv:1611.05397**, 2016.

JAVADI, Mohammad et al. Humanoid Robot Detection using Deep Learning: A Speed-Accuracy Tradeoff. In: SPRINGER. ROBOT World Cup. [S.l.: s.n.], 2017. p. 338–349.

JUMEL, Fabrice et al. Context Aware Robot Architecture, Application to the RoboCup@ Home Challenge. In: ROBOCUP symposium. [S.l.: s.n.], 2018. p. 1–12.

JUSTESEN, Niels et al. Deep learning for video game playing. **IEEE Transactions on Games**, IEEE, 2019.

KEMPKA, Michał et al. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In: IEEE. 2016 IEEE Conference on Computational Intelligence and Games (CIG). [S.l.: s.n.], 2016. p. 1–8.

KIM, Sanghyun et al. Approach of team snu to the darpa robotics challenge finals. In: IEEE. HUMANOID Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on. [S.l.: s.n.], 2015. p. 777–784.

KITANO, Hiroaki; ASADA, Minoru. The RoboCup humanoid challenge as the millennium challenge for advanced robotics. **Advanced Robotics**, Taylor & Francis, v. 13, n. 8, p. 723–736, 1998.

KITANO, Hiroaki et al. RoboCup: The Robot World Cup Initiative. In: PROCEEDINGS of the First International Conference on Autonomous Agents. Marina del Rey, California, USA: ACM, 1997. (AGENTS '97), p. 340–347. Disponível em: <http://doi.acm.org/10.1145/267658.267738>.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2012. p. 1097–1105.

LAMPLE, Guillaume; CHAPLOT, Devendra Singh. Playing FPS games with deep reinforcement learning. **arXiv preprint arXiv:1609.05521**, 2016.

LANGE, Sascha; RIEDMILLER, Martin A. Deep learning of visual control policies. In: CITESEER. ESANN. [S.l.: s.n.], 2010.

LECUN, Yann et al. Generalization and network design strategies. **Connectionism in perspective**, Zurich, Switzerland: Elsevier, p. 143–155, 1989.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LECUN, Yann; KAVUKCUOGLU, Koray; FARABET, Clément. Convolutional networks and applications in vision. In: IEEE. CIRCUITS and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. [S.l.: s.n.], 2010. p. 253–256.

LEVINE, Sergey et al. End-to-end training of deep visuomotor policies. **The Journal of Machine Learning Research**, JMLR. org, v. 17, n. 1, p. 1334–1373, 2016.

LILLICRAP, Timothy P et al. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015.

LIM, Jeongsoo et al. Robot system of DRC-HUBO+ and control strategy of team KAIST in DARPA robotics challenge finals. **Journal of Field Robotics**, Wiley Online Library, v. 34, n. 4, p. 802–829, 2017.

LIN, Long-Ji. **Reinforcement learning for robots using neural networks**. [S.l.], 1993.

MADZAROV, Gjorgji; GJORGJEVIKJ, Dejan; CHORBEV, Ivan. A multi-class SVM classifier utilizing binary decision tree. **Informatica**, v. 33, n. 2, 2009.

MIROWSKI, Piotr et al. Learning to navigate in complex environments. **arXiv preprint arXiv:1611.03673**, 2016.

MITCHELL, Thomas M. **Machine Learning**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

MNIH, Volodymyr et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

MNIH, Volodymyr et al. Asynchronous methods for deep reinforcement learning. In: INTERNATIONAL conference on machine learning. [S.l.: s.n.], 2016. p. 1928–1937.

MNIH, Volodymyr et al. Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013.

N'GUYEN, Steve et al. Rhoban Football Club: RoboCup Humanoid Kid-Size 2017 Champion Team Paper. **RoboCup 2017: Robot World Cup XXI**, Springer, v. 11175, p. 423, 2018.

NAIR, Arun et al. Massively parallel methods for deep reinforcement learning. **arXiv preprint arXiv:1507.04296**, 2015.

NAIR, Vinod; HINTON, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In: PROCEEDINGS of the 27th international conference on machine learning (ICML-10). [S.l.: s.n.], 2010. p. 807–814.

NVIDIA and IBM Cloud Support ImageNet Large Scale Visual Recognition Challenge. NVIDIA. 2017. Disponível em: <https://devblogs.nvidia.com/parallelforall/nvidia-ibm-cloud-support-imagenet-large-scale-visual-recognition-challenge/>.

SITE da Open Dynamics Engine. ODE. 2018. Disponível em: <http://www.ode.org/>.

PARKHI, Omkar M; VEDALDI, Andrea; ZISSERMAN, Andrew et al. Deep face recognition. In: 3. BMVC. [S.l.: s.n.], 2015. v. 1, p. 6.

PERICO, Danilo H et al. Newton: a high level control humanoid robot for the RoboCup Soccer KidSize League. In: ROBOTICS. [S.l.]: Springer, 2014. p. 53–73.

PLAPPERT, Matthias. **keras-rl**. [S.l.]: GitHub, 2016. https://github.com/matthiasplappert/keras-rl.

RIEDMILLER, Martin. Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method. In: SPRINGER. ECML. [S.l.: s.n.], 2005. v. 3720, p. 317–328.

REGRAS da Liga Humanoide. RoboCup. 2017. Disponível em: <https://www.robocuphumanoid.org/materials/rules/>.

RUSSAKOVSKY, Olga et al. Imagenet large scale visual recognition challenge. **International Journal of Computer Vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

SCHAUL, Tom et al. Prioritized experience replay. **arXiv preprint arXiv:1511.05952**, 2015.

SCHNEKENBURGER, Fabian et al. Detection and localization of features on a soccer field with feedforward fully convolutional neural networks (fcnn) for the Adultsize humanoid robot Sweaty. In: PROCEEDINGS of the 12th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Birmingham. [S.l.: s.n.], 2017.

SILVER, David et al. Mastering the game of Go with deep neural networks and tree search. **nature**, Nature Publishing Group, v. 529, n. 7587, p. 484, 2016.

SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SPECK, Daniel et al. Ball Localization for Robocup Soccer using Convolutional Neural Networks, 2016.

SRIVASTAVA, Nitish et al. Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 1929–1958, 2014.

SUTTON, Richard S. Learning to predict by the methods of temporal differences. **Machine learning**, Springer, v. 3, n. 1, p. 9–44, 1988.

SUTTON, Richard S.; BARTO, Andrew G. **Introduction to Reinforcement Learning**. 1st. Cambridge, MA, USA: MIT Press, 1998.

SZEGEDY, Christian et al. Going deeper with convolutions. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2015. p. 1–9.

TAI, Lei; LI, Shaohua; LIU, Ming. A deep-network solution towards model-less obstacle avoidance. In: IEEE. INTELLIGENT Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on. [S.l.: s.n.], 2016. p. 2759–2764.

TAI, Lei; LIU, Ming. Mobile robots exploration through cnn-based reinforcement learning. **Robotics and biomimetics**, Springer Berlin Heidelberg, v. 3, n. 1, p. 24, 2016.

_____. Towards cognitive exploration through deep reinforcement learning for mobile robots. **arXiv preprint arXiv:1610.01733**, 2016.

VAN HASSELT, Hado; GUEZ, Arthur; SILVER, David. Deep reinforcement learning with double Q-learning. **CoRR, abs/1509.06461**, 2015.

VELOSO, Manuela; STONE, Peter. Video: Robocup robot soccer history 1997–2011. In: IEEE. INTELLIGENT Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on. [S.l.: s.n.], 2012. p. 5452–5453.

VERNON, David. **Artificial cognitive systems: A primer**. [S.l.]: MIT Press, 2014.

WANG, X Xu. Deep reinforcement learning: case study with standard RL testing domains. Technische Universiteit Eindhoven, 2016.

WANG, Ziyu; FREITAS, Nando de; LANCTOT, Marc. Dueling network architectures for deep reinforcement learning. **arXiv preprint arXiv:1511.06581**, 2016.

WATKINS, C. J. C. H. **Learning from Delayed Rewards.** 1989. Tese (Doutorado) – Cambridge University.

WATKINS, C. J. C. H.; DAYAN, Peter. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992.

SITE do Webots. Webots. 2019. Disponível em: <http://www.cyberbotics.com>.

WEISS, Karl; KHOSHGOFTAAR, Taghi M; WANG, DingDing. A survey of transfer learning. **Journal of Big Data**, Nature Publishing Group, v. 3, n. 1, p. 9, 2016.

WESTERVELT, Eric R. et al. **Feedback control of dynamic bipedal robot locomotion**. Boca Raton: CRC Press, 2007. (Control and automation). Disponível em: <http://opac.inria.fr/record=b1134075>.

XIONG, Wayne et al. Achieving human parity in conversational speech recognition. **arXiv preprint arXiv:1610.05256**, 2016.

YAHYA, Ali et al. Collective Robot Reinforcement Learning with Distributed Asynchronous Guided Policy Search. In: IEEE. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [S.l.: s.n.], 2017. p. 79–86.

ZEILER, Matthew D; FERGUS, Rob. Visualizing and understanding convolutional networks. In: SPRINGER. EUROPEAN conference on computer vision. [S.l.: s.n.], 2014. p. 818–833.

ÍNDICE

APÊNDICE A – APRESENTAÇÃO DAS MÉTRICAS DAS REDES NEURAIS PROFUNDAS IMPLEMENTADAS NESTA TESE

Está seção apresenta uma análise detalhada dos dados extraídos dos experimentos com a Redes Neurais Profundas apresentado as métricas mais utilizadas para modelos de classificação, para que seja possível observar o comportamento das DNNs no modelo apresentado nesta Tese. Nesta seção as análises são realizadas nas imagens de teste.

As métricas utilizadas nesta seção são:

a) Acurácia (*accuracy* (ACC));
b) Precisão (*precision* ou *positive predictive value* (PPV));
c) Revocação ou Sensibilidade (*sensitivity*, *recall*, *hit rate*, ou *true positive rate* (TPR));
d) Especificidade (*specificity*, *selectivity* ou *true negative rate* (TNR));
e) F-Score ou F-measure.

Deve-se lembrar que as métricas utilizadas nesta seção não são determinísticas, devem ser interpretadas como probabilidades. Isso porque as imagens de testes representam uma pequena amostra do domínio que o classificador será inserido.

Nesta análise serão apresentadas as matrizes de confusão de cada DNN, a partir da matriz se extrai os dados que serão utilizados nas equações de cada métrica. Essas informações são: Verdadeiros Positivos (*true positive* (TP)); Verdadeiros Negativos (*true negative* (TN)); Falsos Positivos (*false positive* (FP)); Falsos Negativos (*false negative* (FN)).

O valor da acurácia pode ser calculado pela Equação (36). A acurácia é uma métrica que demonstra a proporção de quanto que o classificador consegue classificar corretamente os dados. No entanto está métrica é sensível a desbalanceamentos do conjunto de dados, assim podendo induzir a uma conclusão errada sobre o desempenho do classificador em alguma classe específica.

$$Acurácia = \frac{TP + TN}{TP + TN + FP + FN} \tag{36}$$

O valor da precisão pode ser calculado pela Equação (37). A precisão é uma métrica que representa que de todos os dados que foram classificados como dados da classe (TP + FP), qual a proporção de quantos realmente são dados da classe. Quantos elementos selecionados são relevantes.

$$Precisão = \frac{TP}{TP + FP} \tag{37}$$

O valor da revocação pode ser calculado pela Equação (38). A revocação é uma métrica que representa que de todos os dados que são de uma determinada classe (VP + FN), qual a

proporção que classifica como sendo dessa classe. Quantos dos elementos relevantes foram selecionados.

$$Revocação = \frac{TP}{TP + FN} \tag{38}$$

O valor da especificidade pode ser calculado pela Equação (38). A especificidade é uma métrica que representa que dos dados corretamente classificados como negativos (TN), qual a proporção que esse classificador avalia corretamente como sendo negativos.

$$Especificidade = \frac{TN}{TN + FP} \tag{39}$$

O valor do F-score pode ser calculado pela Equação (40). O F-score é uma média harmônica da precisão e revocação, sendo uma métrica que combina precisão e revocação de modo a trazer um valor que indica a qualidade da classificação. E está métrica é robusta a desbalanceamentos do conjunto de dados.

$$F - Score = 2 * \frac{PPV * TPR}{PPV + TPR} \tag{40}$$

A.1 MATRIZ DE CONFUSÃO E MÉTRICAS DOS EXPERIMENTOS REALIZADOS NO ROBÔ REAL

Nesta seção são apresentadas as matrizes de confusão dos experimentos nas Tabelas 27, 29, 31, 33, 35, 37. 39, e as métricas de cada DNN por classe nas Tabelas 28, 30, 32, 34, 36, 38, 40. Também é apresentado uma média das métricas de cada arquitetura na Tabela 41.

Tabela 27 – Matriz de confusão da Conv4 com entrada 110x110.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	92	3	0	0	0	2	3	8
ball left	0	2	70	0	0	18	10	98
ball right	0	4	91	0	0	5	0	9
kick left	0	0	0	98	2	0	0	2
kick right	0	0	0	58	42	0	0	58
at goal	0	0	0	0	0	97	3	3
empty	0	0	0	0	0	1	99	1
FP	0	7	70	58	2	26	26	-

Fonte: Autor.

Tabela 28 – Métricas da Conv4 com entrada 110x110.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	1.0000	0.9200	1.0000	0.9583	0.9849
ball left	0.2222	0.0200	0.9867	0.0367	0.8323
ball right	0.5652	0.9100	0.8600	0.6973	0.8683
kick left	0.6282	0.9800	0.8794	0.7656	0.8967
kick right	0.9545	0.4200	0.9958	0.5833	0.8967
at goal	0.7886	0.9700	0.9422	0.8700	0.9473
empty	0.8609	0.9900	0.9635	0.9209	0.9684
Média	0.7171	0.7443	0.9468	0.6903	0.9135

Fonte: Autor.

Tabela 29 – Matriz de confusão da Conv3 com entrada 110x110.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	94	0	0	0	0	3	3	6
ball left	0	78	1	0	0	19	2	22
ball right	1	1	95	0	0	3	0	5
kick left	0	0	0	93	7	0	0	7
kick right	0	0	0	1	99	0	0	1
at goal	0	0	10	0	0	53	37	47
empty	0	0	5	0	0	0	95	5
FP	1	1	16	1	7	25	42	-

Fonte: Autor.

Tabela 30 – Métricas da Conv3 com entrada 110x110.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	0.9895	0.9400	0.9981	0.9641	0.9886
ball left	0.9873	0.7800	0.9981	0.8715	0.9635
ball right	0.8559	0.9500	0.9697	0.9005	0.9666
kick left	0.9894	0.9300	0.9981	0.9588	0.9870
kick right	0.9340	0.9900	0.9864	0.9612	0.9870
at goal	0.6795	0.5300	0.9568	0.5955	0.8940
empty	0.6934	0.9500	0.9242	0.8017	0.9281
Média	0.8756	0.8671	0.9759	0.8647	0.9592

Fonte: Autor.

Tabela 31 – Matriz de confusão da AlexNet com entrada 110x110.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	92	1	0	0	0	1	6	8
ball left	0	17	40	1	3	22	17	83
ball right	0	3	93	0	0	4	0	7
kick left	0	0	0	99	1	0	0	1
kick right	0	0	0	59	41	0	0	59
at goal	0	0	0	0	0	97	3	3
empty	0	0	0	0	0	0	100	0
FP	0	4	40	60	4	27	26	-

Fonte: Autor.

Tabela 32 – Métricas da AlexNet com entrada 110x110.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	1.0000	0.9200	1.0000	0.9583	0.9854
ball left	0.8095	0.1700	0.9924	0.2810	0.8610
ball right	0.6992	0.9300	0.9177	0.7983	0.9198
kick left	0.6226	0.9900	0.8800	0.7645	0.8983
kick right	0.9111	0.4100	0.9920	0.5655	0.8953
at goal	0.7823	0.9700	0.9424	0.8661	0.9473
empty	0.7937	1.0000	0.9441	0.8850	0.9540
Média	0.8026	0.7700	0.9527	0.7312	0.9230

Fonte: Autor.

Tabela 33 – Matriz de confusão da Conv4 com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	88	1	5	0	0	1	5	12
ball left	1	65	2	0	0	21	11	35
ball right	0	11	82	0	0	5	1	18
kick left	0	0	0	86	14	0	0	14
kick right	0	0	0	17	83	0	0	17
at goal	0	0	0	0	0	83	17	17
empty	0	0	0	0	0	0	100	0
FP	1	12	7	17	14	26	34	-

Fonte: Autor.

Tabela 34 – Métricas da Conv4 com entrada 256x256.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	0.9888	0.8800	0.9980	0.9312	0.9783
ball left	0.8442	0.6500	0.9775	0.7345	0.9259
ball right	0.9213	0.8283	0.9863	0.8723	0.9607
kick left	0.8350	0.8600	0.9672	0.8473	0.9498
kick right	0.8557	0.8300	0.9730	0.8426	0.9498
at goal	0.7545	0.8300	0.9492	0.7905	0.9303
empty	0.7463	1.0000	0.9347	0.8547	0.9452
Média	0.8494	0.8398	0.9694	0.8390	0.9486

Fonte: Autor.

Tabela 35 – Matriz de confusão da SqueezeNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	95	4	0	0	0	1	0	5
ball left	0	87	12	0	0	1	0	13
ball right	9	6	85	0	0	0	0	15
kick left	1	0	0	67	32	0	0	33
kick right	0	0	0	62	38	0	0	62
at goal	2	3	0	0	0	85	10	15
empty	0	0	0	0	0	0	100	0
FP	12	13	12	62	32	2	10	-

Fonte: Autor.

Tabela 36 – Métricas da SqueezeNet com entrada 256x256.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	0.8879	0.9500	0.9747	0.9179	0.9704
ball left	0.8700	0.8700	0.9731	0.8700	0.9554
ball right	0.8763	0.8500	0.9752	0.8629	0.9538
kick left	0.5194	0.6700	0.8877	0.5852	0.8543
kick right	0.5429	0.3800	0.9419	0.4471	0.8556
at goal	0.9770	0.8500	0.9958	0.9091	0.9704
empty	0.9091	1.0000	0.9786	0.9524	0.9824
Média	0.7975	0.7957	0.9610	0.7921	0.9346

Fonte: Autor.

Tabela 37 – Matriz de confusão da AlexNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	98	0	0	0	0	0	2	2
ball left	3	70	0	0	0	21	6	30
ball right	9	7	71	0	0	9	4	29
kick left	0	0	0	98	2	0	0	2
kick right	0	0	0	9	91	0	0	9
at goal	0	0	1	0	0	92	7	8
empty	0	0	1	0	0	4	95	5
FP	12	7	2	9	2	34	19	-

Fonte: Autor.

Tabela 38 – Métricas da AlexNet com entrada 256x256.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	0.8909	0.9800	0.9773	0.9333	0.9777
ball left	0.9091	0.7000	0.9873	0.7910	0.9433
ball right	0.9726	0.7100	0.9963	0.8208	0.9520
kick left	0.9159	0.9800	0.9829	0.9469	0.9824
kick right	0.9785	0.9100	0.9962	0.9430	0.9824
at goal	0.7302	0.9200	0.9390	0.8142	0.9361
empty	0.8333	0.9500	0.9647	0.8879	0.9624
Média	0.8901	0.8786	0.9777	0.8767	0.9623

Fonte: Autor.

Tabela 39 – Matriz de confusão da GoogLeNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**FN**
ball center	100	0	0	0	0	0	0	0
ball left	1	97	2	0	0	0	0	3
ball right	4	4	92	0	0	0	0	8
kick left	0	0	0	100	0	0	0	0
kick right	0	0	0	14	86	0	0	14
at goal	1	1	0	0	0	98	0	2
empty	1	0	0	0	0	0	99	1
FP	7	5	2	14	0	0	0	-

Fonte: Autor.

Tabela 40 – Métricas da GoogLeNet com entrada 256x256.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	0.9346	1.0000	0.9879	0.9662	0.9897
ball left	0.9510	0.9700	0.9914	0.9604	0.9882
ball right	0.9787	0.9200	0.9966	0.9485	0.9853
kick left	0.8772	1.0000	0.9761	0.9346	0.9796
kick right	1.0000	0.8600	1.0000	0.9247	0.9796
at goal	1.0000	0.9800	1.0000	0.9899	0.9970
empty	1.0000	0.9900	1.0000	0.9950	0.9985
Média	0.9631	0.9600	0.9931	0.9599	0.9883

Fonte: Autor.

Tabela 41 – Métricas das DNNs.

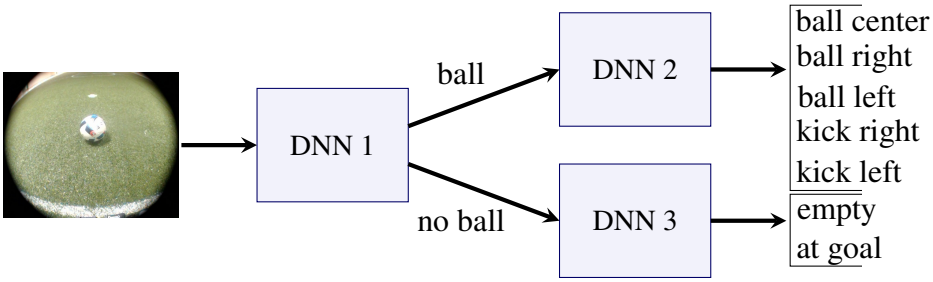
	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
Conv4 110x110	0.7171	0.7443	0.9468	0.6903	0.9135
AlexNet 110x110	0.8026	0.7700	0.9527	0.7312	0.9230
SqueezeNet 256x256	0.7975	0.7957	0.9610	0.7921	0.9346
Conv4 256x256	0.8494	0.8398	0.9694	0.8390	0.9486
Conv3 110x110	0.8756	0.8671	0.9759	0.8647	0.9592
AlexNet 256x256	0.8901	0.8786	0.9777	0.8767	0.9623
GoogLeNet 256x256	0.9631	0.9600	0.9931	0.9599	0.9883

Fonte: Autor.

A.1.1 Matriz de Confusão e Métricas das DNNs da Árvore de Decisão

A Árvore de Decisão de DNNs aplicada nos experimentos dessa tese é composta por três DNNs como mostra a Figura 73, onde a primeira DNN (DNN 1) recebe a imagem da câmera do robô e realiza uma classificação se há bola na imagem, caso haja bola, a imagem é transmitida para a entrada da segunda DNN (DNN 2), caso contrário a imagem é transmitida para a entrada da terceira DNN (DNN 3). As três DNNs da árvore de decisão possuem a mesma arquitetura. As Tabelas 42, 44, 46 apresentam a matriz de confusão das DNNs, e as métricas de cada DNN por classe são apresentadas na Tabela 43, 45, 47. A Tabela 48 apresenta as métricas da Árvore de Decisão de DNNs para cada classe, e a Tabela 49 apresenta uma média das métricas da Árvore de Decisão de DNNs e também das outras arquitetura de DNNs.

Figura 73 – Árvore de Decisão de DNNs.



Fonte: Autor.

Tabela 42 – Matriz de confusão da DNN 1.

	ball	**no ball**	**FN**
ball	489	11	11
no ball	0	200	0
FP	0	11	-

Fonte: Autor.

Tabela 43 – Métricas da DNN 1.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball	1.0000	0.9780	1.0000	0.9889	0.9843
no ball	0.9479	1.0000	0.9780	0.9732	0.9843
Média	0.9739	0.9890	0.9890	0.9811	0.9843

Fonte: Autor.

Tabela 44 – Matriz de confusão da DNN 2.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**FN**
ball center	96	2	2	0	0	4
ball left	0	100	0	0	0	0
ball right	0	2	98	0	0	2
kick left	0	0	0	100	0	0
kick right	0	0	0	0	100	0
FP	0	4	2	0	0	-

Fonte: Autor.

Tabela 45 – Métricas da DNN 2.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	1.0000	0.9600	1.0000	0.9796	0.9920
ball left	0.9615	1.0000	0.9899	0.9804	0.9920
ball right	0.9800	0.9800	0.9950	0.9800	0.9920
kick left	1.0000	1.0000	1.0000	1.0000	1.0000
kick right	1.0000	1.0000	1.0000	1.0000	1.0000
Média	0.9883	0.9880	0.9970	0.9880	0.9952

Fonte: Autor.

Tabela 46 – Matriz de confusão da DNN 3.

	at goal	**empty**	**FN**
at goal	96	4	4
empty	0	100	0
FP	0	4	-

Fonte: Autor.

Tabela 47 – Métricas da DNN 3.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
at goal	1.0000	0.9600	1.0000	0.9796	0.9800
empty	0.9615	1.0000	0.9600	0.9804	0.9800
Média	0.9808	0.9800	0.9800	0.9800	0.9800

Fonte: Autor.

Tabela 48 – Métricas da Árvore de Decisão de DNNs.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
ball center	1.0000	0.9389	1.0000	0.9687	0.9764
ball left	0.9615	0.9780	0.9899	0.9695	0.9764
ball right	0.9800	0.9584	0.9950	0.9691	0.9764
kick left	1.0000	0.9780	1.0000	0.9889	0.9843
kick right	1.0000	0.9780	1.0000	0.9889	0.9843
at goal	0.9479	0.9600	0.9780	0.9533	0.9646
empty	0.9114	1.0000	0.9389	0.9541	0.9646
Média	0.9715	0.9701	0.9860	0.9704	0.9753

Fonte: Autor.

Tabela 49 – Métricas das DNNs e da Arvore de Decisão de DNNs.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
Conv4 110x110	0.7171	0.7443	0.9468	0.6903	0.9135
AlexNet 110x110	0.8026	0.7700	0.9527	0.7312	0.9230
SqueezeNet 256x256	0.7975	0.7957	0.9610	0.7921	0.9346
Conv4 256x256	0.8494	0.8398	0.9694	0.8390	0.9486
Conv3 110x110	0.8756	0.8671	0.9759	0.8647	0.9592
AlexNet 256x256	0.8901	0.8786	0.9777	0.8767	0.9623
GoogLeNet 256x256	0.9631	0.9600	0.9931	0.9599	0.9883
Tree of DNNs	0.9715	0.9701	0.9860	0.9704	0.9753

Fonte: Autor.

A.2 MATRIZ DE CONFUSÃO E MÉTRICAS DOS EXPERIMENTOS REALIZADOS NO SIMULADOR

Está seção apresenta a matriz de confusão dos experimentos realizados no simulador. Essas matrizes são apresentadas nas Tabelas 50, 51, 52, 53, 54, 55. A Tabela 56 apresenta uma média das métricas de cada arquitetura.

Tabela 50 – Matriz de confusão da Conv4 com entrada 110x110.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	1000	0	0	0	0	0	0	100%
ball left	0	1000	1	0	0	0	0	100%
ball right	0	0	998	0	0	1	1	99.8%
kick left	0	0	0	1000	0	0	0	100%
kick right	0	0	0	0	1000	0	0	100%
at goal	0	0	0	0	0	1000	0	100%
empty	0	0	0	0	0	0	1000	100%

Fonte: Autor.

Tabela 51 – Matriz de confusão da AlexNet com entrada 110x110.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	1000	0	0	0	0	0	0	100%
ball left	0	998	1	0	0	0	1	99.8%
ball right	0	0	998	0	0	1	1	99.8%
kick left	0	0	0	1000	0	0	0	100%
kick right	0	0	0	0	1000	0	0	100%
at goal	0	0	0	0	0	1000	0	100%
empty	0	0	0	0	0	0	1000	100%

Fonte: Autor.

Tabela 52 – Matriz de confusão da SqueezeNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	999	1	0	0	0	0	0	99.9%
ball left	0	1000	0	0	0	0	0	100%
ball right	0	4	996	0	0	0	0	99.6%
kick left	1	4	0	995	0	0	0	99.5%
kick right	4	0	1	2	993	0	0	99.3%
at goal	0	0	0	0	0	1000	0	100%
empty	0	0	0	0	0	0	1000	100%

Fonte: Autor.

Tabela 53 – Matriz de confusão da Conv4 com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	1000	1	0	0	0	0	0	100%
ball left	0	1000	0	0	0	0	0	100%
ball right	0	4	999	0	0	0	0	99.9%
kick left	1	4	0	963	0	0	0	96.3%
kick right	4	0	1	2	984	0	0	98.4%
at goal	0	0	0	0	0	1000	0	100%
empty	0	0	0	0	0	0	1000	100%

Fonte: Autor.

Tabela 54 – Matriz de confusão da AlexNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	1000	0	0	0	0	0	0	100%
ball left	0	999	0	0	0	0	1	99.9%
ball right	0	0	999	0	1	0	0	99.6%
kick left	0	12	0	998	0	0	0	99.5%
kick right	4	0	0	0	996	0	0	99.3%
at goal	0	0	0	0	0	1000	0	100%
empty	0	0	0	0	0	0	1000	100%

Fonte: Autor.

Tabela 55 – Matriz de confusão da GoogLeNet com entrada 256x256.

	ball center	**ball left**	**ball right**	**kick left**	**kick right**	**at goal**	**empty**	**Taxa de acerto**
ball center	1000	0	0	0	0	0	0	100%
ball left	2	998	0	0	0	0	0	99.8%
ball right	0	0	1000	0	0	0	0	100%
kick left	0	0	0	1000	0	0	0	100%
kick right	0	0	0	0	1000	0	0	100%
at goal	0	0	0	0	0	1000	0	100%
empty	0	3	0	0	0	0	997	99.7%

Fonte: Autor.

Tabela 56 – Métricas das DNNs.

	Precisão	**Revocação**	**Especificidade**	**F-score**	**Acurácia**
Conv4 110x110	0.9996	0.9996	0.9999	0.9996	0.9999
AlexNet 110x110	0.9994	0.9994	0.9999	0.9994	0.9998
SqueezeNet 256x256	0.9976	0.9976	0.9996	0.9976	0.9993
Conv4 256x256	0.9976	0.9975	0.9996	0.9976	0.9993
AlexNet 256x256	0.9974	0.9974	0.9996	0.9974	0.9993
GoogLeNet 256x256	0.9993	0.9993	0.9999	0.9993	0.9998

Fonte: Autor.

ANEXO A – TRABALHOS PUBLICADOS

Nesta seção encontra-se todos os trabalhos que foram publicados durante o doutorado, em ordem cronológica, respeitando a seguinte sequência: periódicos, capítulos de livros, artigos publicados em congressos internacionais, trabalhos apresentados em congressos internacionais e artigos publicados em congressos nacionais.

A.1 PERIÓDICOS

Humanoid Robot Framework for Research on Cognitive Robotics. PERICO, D. H.; HOMEM, THIAGO P. D.; ALMEIDA, A. C.; SILVA, ISAAC J.; VILAO, C. O.; NICASSIO, V.; BIANCHI, R. A. C.. Journal of Control, Automation and Electrical Systems. 2018.

Heuristically Accelerated Reinforcement Learning by Means of Case-Based Reasoning and Transfer Learning. BIANCHI, REINALDO A.C.; SANTOS, PAULO E.; SILVA, ISAAC J.; CELIBERTO, LUIZ A.; LOPEZ DE MANTARAS, RAMON. Journal of Intelligent & Robotic Systems. 2017.

A.2 CAPÍTULOS DE LIVROS

Humanoid Robot Gait on Sloping Floors Using Reinforcement Learning. Communications in Computer and Information Science. SILVA, ISAAC J.; PERICO, DANILO H.; HOMEM, THIAGO P. D.; VILÃO, CLAUDIO O.; TONIDANDEL, FLAVIO; BIANCHI, REINALDO A.C.. Communications in Computer and Information Science. Springer. 2016.

A.3 ARTIGOS PUBLICADOS EM CONGRESSOS INTERNACIONAIS

Towards Robotic Cognition Using Deep Neural Network Applied in a Goalkeeper Robot. SILVA, ISAAC J.; VILÃO, CLAUDIO O.; BIANCHI, REINALDO A. C.. Latin American Robotics Symposium (LARS), Curitiba - Paraná - Brazil. 2017.

A Robot Simulator Based on the Cross Architecture for the Development of Cognitive Robotics. PERICO, DANILO H.; HOMEM, THIAGO P.D.; ALMEIDA, AISLAN C.; SILVA, ISAAC J.; VILÃO, CLAUDIO O.; FERREIRA, VINICIUS N.; BIANCHI,

REINALDO A.C.. Latin American Robotics Symposium (LARS), Recife-PE - Brazil. 2016.

A.4 APRESENTAÇÃO DE TRABALHO EM CONGRESSOS INTERNACIONAIS

Development a vision system with Deep Neural Networks to object detection in RoboCup Humanoid League. DA SILVA, ISAAC JESUS; BIANCHI, REINALDO A.C. ; LY, OLIVIER. Apresentação de poster na RoboCup Symposium - Montreal - Canada. 2018.

A.5 ARTIGOS PUBLICADOS EM CONGRESSOS NACIONAIS

Development a vision system with Deep Neural Networks to ball detection in RoboCup Humanoid League. DA SILVA, ISAAC JESUS; BIANCHI, REINALDO A.C. ; LY, OLIVIER. II Workshop de Robôs Humanoides do Brasil (BRAHUR) e III Workshop do Robôs de Serviço (BRASERO). 2019.

Estudo e implementação de um sistema de visão computacional baseado em Redes Neurais Profundas para detecção de objetos no domínio do Futebol de Robôs Humanoide. OLIVEIRA, JONAS H. R.; DA SILVA, ISAAC JESUS; BIANCHI, REINALDO A.C.. II Workshop de Robôs Humanoides do Brasil (BRAHUR) e III Workshop do Robôs de Serviço (BRASERO). 2019.

Desenvolvimento de uma Placa Eletrônica para um Robô humanoide da Liga RoboCup Humanoid Teensize. OLIVEIRA, MARIANA A.; DA SILVA, ISAAC JESUS; BIANCHI, REINALDO A.C.. II Workshop de Robôs Humanoides do Brasil (BRAHUR) e III Workshop do Robôs de Serviço (BRASERO). 2019.

Using Reinforcement Learning to Optimize Gait Genaration Parameters of a Humanoid Robot. SILVA, ISAAC J.; PERICO, D. H. ; COSTA, A. H. R. ; BIANCHI, REINALDO A.C. . Simpósio Brasileiro de Automação Inteligente (SBAI), 2017, Porto Alegre - RS. 2017.