Pontifícia Universidade Católica
do Rio de Janeiro

Evelyn Conceição Santos Batista

Um estudo de transfer learning em deep reinforcement learning em ambientes robóticos simulados

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica da PUC-Rio.

Orientador: Dr. Wouter Caarls

Rio de Janeiro
Abril de 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Evelyn Conceição Santos Batista

Um estudo de transfer learning em deep reinforcement learning em ambientes robóticos simulados

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Dr. Wouter Caarls
Orientador
Departamento de Engenharia Elétrica – PUC-Rio

Dr. Leonardo Alfredo Forero Mendoza
Universidade do Estado do Rio de Janeiro – UERJ

Dra. Karla Figueiredo
Universidade do Estado do Rio de Janeiro – UERJ

Dra. Bianca Zadrozny
International Business Machines – IBM

Rio de Janeiro, 30 de Abril de 2019

Evelyn Conceição Santos Batista

Graduou-se em Engenharia Elétrica pela Universidade do Estado do Rio de Janeiro.

Ficha Catalográfica

Santos Batista, Evelyn Conceição

Um estudo de transfer learning em deep reinforcement learning em ambientes robóticos simulados / Evelyn Conceição Santos Batista; orientador: Wouter Caarls. - 2019.

v., 88 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2019.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Aprendizado com reforço profundo;. 3. Rede Neural Convolucional;. 4. Transferência de aprendizado;. 5. Robôs autônomos;. 6. Ambientes Complexos;. I. Caarls, Wouter. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 621.3

Agradecimentos

Primeiramente a Deus, por me dar forças e iluminar a minha caminhada.

Aos meus orientadores Professor Wouter Caarls e Professor Leonardo Mendoza pelos ensinamentos e estímulo para a realização deste trabalho.

Ao CNPq, pelos auxílios concedidos.

Aos laboratórios ICA e LCI, pelo suporte disponibilizado.

Aos meus pais, principalmente à minha mãe, que sempre me ajudou e me incentivou em todo tempo deste curso.

Aos meus amigos, principalmente Cristian Muñoz, Smith Arauco e Guilherme Fadul, pelo apoio, paciência e pelas muitas vezes que compartilhamos conhecimentos.

E a todos que, diretamente ou indiretamente, me ajudaram, deixo registrada a minha gratidão.

Resumo

Santos Batista, Evelyn Conceição; Caarls, Wouter. **Um estudo de transfer learning em deep reinforcement learning em ambientes robóticos simulados**. Rio de Janeiro, 2019. 88p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Esta dissertação de mestrado consiste em um estudo avançado sobre aprendizado profundo por reforço visual para robôs autônomos através de técnicas de transferência de aprendizado. Os ambientes de simulação testados neste estudo são ambientes realistas complexos onde o robô tinha como desafio aprender e transferir conhecimento em diferentes contextos para aproveitar a experiência de ambientes anteriores em ambientes futuros. Este tipo de abordagem, além de agregar conhecimento ao robô autônomo, diminui o número de épocas de treinamento do algoritmo, mesmo em ambientes complexos, justificando o uso das técnicas de transferência de aprendizado.

Palavras-chave

Aprendizado com reforço profundo; Rede Neural Convolucional; Transferência de aprendizado; Robôs autônomos; Ambientes Complexos;

Abstract

Santos Batista, Evelyn Conceição; Caarls, Wouter (Advisor). **A simulation study of transfer learning in deep reinforcement learning for robotics**. Rio de Janeiro, 2019. 88p. Dissertação de mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

This master's thesis consists of an advanced study on deep learning by visual reinforcement for autonomous robots through transfer learning techniques. The simulation environments tested in this study are highly realistic environments where the challenge of the robot was to learn and transfer knowledge in different contexts to take advantage of the experience of previous environments in future environments. This type of approach besides adding knowledge to the autonomous robot reduces the number of training epochs the algorithm, even in complex environments, justifying the use of transfer learning techniques.

Keywords

Deep Reinforcement Learning; Convolution Neural Network; Transfer Learning; Robot Autonomous; Complex Environments;

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

RL – *Reinforcement Leaning*
AR – Aprendizado por reforço
PDM – Processo de Decisão de Markov
MDP – *Markov decision Process*
POMDP – *Partially observable Markov decision process*
DP – *Dynamic Programming* ou Desvio Padrão
DL – *Deep Learning*
CNN – *Convolutional Neural Network*
FC – *Fully Connected*
TL – *Transfer Learning*
DQN – *Deep Q-Network*
GPU – *Graphics Processing Unit*
CPU – *Central Processing Unit*

Lista de Símbolos

S – Conjunto de estados
s_t – Estado no instante t
A – Conjunto de ações
a_t – Ação no instante t
f – Função de transição ou função
R – Função de recompensa
r_t – Recompensa no instante t
π – Política
Q – Função valor
$Q*$ – Função Q ótima
$\pi*$ – Política ótima
γ – Taxa de desconto
α – Taxa de aprendizagem
x_t – Imagem não processada no instante t
ϕ – Sequência de imagens que caracteriza um estado no DQN
$\mathcal{D}$ – *Replay Memory*

1
Introdução

O aprendizado de agentes autônomos é um campo de pesquisa atualmente bastante explorado e desenvolvido, pelos avanços em algoritmos sofisticados e hardware especializado (1)(2)(3). Estes avanços contribuíram para que diferentes tipos de aplicações se desenvolveram na área de simulação de autônomos, por exemplo carros autônomos (4)(5), veículos aéreos e marinhos autônomos (6)(7) e robôs autônomos para diferentes áreas (8)(9). Estas áreas estudam técnicas que permitem que um robô adquira habilidades inovadoras ou se adapte a um ambiente através de algoritmos de aprendizado, permitindo aprender e desenvolver habilidades para diferentes aplicações (10)(11).

1.1
Motivação

Um robô autônomo deve interagir com o ambiente para atingir seus objetivos: deve ser capaz de reunir informações sobre o ambiente, tomar decisões baseadas nestas informações e iniciar uma ação específica baseada nessas decisões.

A aprendizagem pode acontecer através de auto-exploração autônoma, como na aprendizagem por reforço, que é uma área de aprendizado de máquina preocupada com a forma como os agentes de software devem tomar ações em um ambiente para maximizar alguma noção de recompensa cumulativa. A Aprendizagem por Reforço (AR) é definida como um problema de aprendizagem, onde um sistema inteligente sabe através de recompensas ou reforços se as ações que está realizando no ambiente são boas ou ruins. Neste tipo de aprendizagem o sistema age independente de exemplos externos. Essa definição pode ser exemplificada nos jogos de xadrez, onde o reforço pode ser recebido apenas no final do jogo, ou em outras situações em que o reforço é fornecido com mais frequência, como no jogo de ping e pong, onde a recompensa pode ser fornecida a cada ponto ganho ou perdido (12).

O processo de aprendizagem por reforço pode ser caracterizado como um processo de decisão sequencial, onde as sequências de transições entre os estados e o recebimento de recompensas associadas a cada transição determinam o resultado final. As transições de estados em um ambiente de aprendizagem por

reforço seguem a regra do Processo de Decisão de Markov (PDM), ou seja, a probabilidade de transição de um estado no ambiente, depende exclusivamente da ação realizada e do estado atual que o agente se encontra, e não da sequência dos estados previamente visitados, criando assim uma total independência em relação a qualquer ação e estados executados anteriormente. Nos problemas modelados com o PDM, existe o conceito de política comportamental, ou seja, o conjunto de ações escolhidas e realizadas ao longo da execução do experimento. Assim, a aprendizagem por reforço possui como objetivo, usar as recompensas observadas para aprender e definir a política ótima, no ambiente (12). Estes algoritmos foram muito usados e pesquisados para simulação de robôs autônomos e sistemas de múltiplos agentes (13), baseados unicamente em algoritmos de aprendizado por reforço. Os algoritmos de AR exploram os ambientes, mas não aproveitam os conhecimentos aprendidos em ambientes anteriores, desperdiçando experiência adquirida, deixando o treinamento assim mais demorado. Uma das opções para encontrar soluções a estes problemas são algoritmos da área de *Deep Learning* entre eles especificamente Redes Convolutivas e *Transfer Learning*. Excelentes resultados recentes na aplicação de técnicas de *Deep Learning*, em especial redes convolutivas em algoritmos de Aprendizado de Reforço levaram a uma onda de avanços revolucionários na teoria de agentes e estabeleceram o campo de estudo *Deep Reinforcement Lerning* (DRL)(14)(15). As redes convolutivas permitem que os modelos de RL tenham imagens de vídeo como entrada do modelo. Isso é um grande avanço na área da robótica e abre espaço para novas aplicações nessa área. Outra técnica muito usada em *Deep Learning* é *Transfer Learning* onde os algoritmos aproveitam o aprendizado de outros cenários para adquirir habilidades que servem em cenários diferentes. Esta técnica permite que o robô consiga levar a expertise aprendida em ambientes anteriores a ambientes presentes e diminuir o número de épocas de treinamento da rede convolutiva que sempre são bastante altas. A união de redes convolutivas e *Transfer Learning* já se apresenta em trabalhos recentes na simulação de robôs para ambientes simples e pouco complexos como o jogo Doom (16) de baixa resolução. Os ambientes realistas são muito mais complexos mas abrem as portas a novas aplicações da robótica tornando-se alvo de novas pesquisas. Este trabalho apresenta um novo estudo mostrando *Deep Reinforcement Learning* aplicando a técnica de *Transfer Learning* para simular o comportamento de um robô em diferentes tipos de ambientes desde simples até altamente complexos e realistas, mostrando excelentes resultados em todos os ambientes e com a técnica *Transfer learning* diminuindo o tempo de épocas de treinamento em ambientes realistas.

Atualmente, em várias aplicações, há a necessidade de inserir maior

autonomia nos sistemas robóticos, isto é fazer com que os robôs incorporem a capacidade de decidir, sem intervenção humana, o que deve ser executado para satisfazer determinados objetivos previamente especificados pelo operador. Cada vez mais as pesquisas na área se focam no desenvolvimento de aplicações que visam atender a essa necessidade, pois os agentes percebem o ambiente através de sensores e agem nesse ambiente de forma independente, através de atuadores. Esta autonomia permite que os robôs tomem decisões rápidas, levando em conta várias variáveis de um ambiente. Havendo assim várias aplicações como carros autônomos, drones inteligentes, robôs para aplicações agrárias e de segurança, por exemplo. A motivação principal deste trabalho é simular robôs autônomos em ambiente reais e complexos para avançar cada vez mais na autonomia e complexidade de tarefas. Para isso será usado aprendizado por reforço visual conhecido como *deep reinforcement learning* e como há a dificuldade de treinar um autômato no mundo real, este trabalho visa fazer simulações para que o agente não tenha que aprender tudo no ambiente real, assim é utilizada a técnica de *transfer learning*, para que seja viável a transferência de aprendizado da simulação para o mundo real.

1.2 Objetivos e Contribuições

O objetivo dessa dissertação é desenvolver e aplicar algoritmos de aprendizado por reforço visual em um simulador, para um robô aprender a alcançar um determinado objetivo a partir de imagens provenientes de uma câmera no autômato em diversos ambientes, e após esse aprendizado será possível o autônomo reutilizar o que já fora aprendido para a navegação em outros ambientes. Além de um estudo aprofundado sobre aprendizado por reforço visual e técnicas de transfer learning em redes convolutivas, simulando robôs autônomos em diferentes tipos de ambientes mostrando o comportamento dos algoritmos e a viabilidade do uso de transfer learning.

1.3 Estrutura da dissertação

Esta dissertação será dividida em 4 capítulos da seguinte forma:

Capítulo 1: Fundamentos teóricos – apresenta conceitos sobre *Reinforcement Learning*, *Deep learning* e *Deep Reinforcement Learning*.

Capítulo 2: Modelo - apresenta os ambientes usados nos testes, a arquitetura da rede neural e a síntese de treinamento que será utilizada no capítulo 3.

Capítulo 3: Experimentos - explica cada experimento exibindo seus resultados, os comparando quando há uso de *transfer learning*.

Capítulo 4: Conclusões e trabalhos futuros - apresenta conclusões tiradas após observar os resultados dos experimentos e mostra onde as soluções deste trabalho serão aplicadas em trabalhos futuros.

2
Fundamentos Teóricos

2.1
Reinforcement Learning

A aprendizagem por reforço (*reinforcement learning* (RL)) é uma área dentro da aprendizagem de máquina dedicada ao desenvolvimento de algoritmos que permitem a um agente (robô, sistema, personagem de videogame, etc.) aprender uma tarefa onde se tem que tomar decisões sequenciais para alcançar um objetivo, maximizando um valor acumulado de recompensa. Em RL, um agente pode ser uma instância de diferentes tipos, por exemplo um robô ou simplesmente um sistema ou algoritmo encarregado de controlar uma planta industrial. Existe um grande número de aplicações práticas para esses tipos de problemas, onde a principal vantagem é que não requer um especialista para encontrar a solução para o problema, mas simplesmente o problema deve ser formulado de maneira apropriada, especificando as recompensas ou penalidades para que o agente possa aprendê-lo.

Há mais algumas coisas para definir no aprendizado por reforço que são centrais para o assunto. Primeiro, há um conceito de recompensa. Isso é o que diferencia os algoritmos de aprendizado por reforço de outros tipos de algoritmos de aprendizado de máquina. Um agente tentará maximizar não apenas sua recompensa imediata mas também futuras recompensas. Frequentemente, os algoritmos de aprendizado de reforço encontraram novas maneiras de realizar isso.

O propósito de um agente durante a aprendizagem por reforço é encontrar uma estratégia que leve a eleger a melhor ação e obter a maior recompensa acumulada esperada em qualquer estado. Considera-se que o agente aprendeu uma estratégia ótima, usualmente chamada política ótima, quando é capaz de acumular a maior recompensa possível para a tarefa especificada. Muitas vezes para resolver a tarefa adequadamente é suficiente encontrar uma política próxima a ótima.

2.1.1 Processo de decisão de Markov(MDPs)

Os Processos de Decisão de Markov (Markov Decision Processes, ou MDPs) são os modelos mais utilizados para se determinar os problemas de aprendizagem por reforço, os MDPs podem ser considerados como um tipo de processo de decisão seriada, onde se cumprem as propriedades de Markov (17).

O MDP é definido por meio da tupla $< S, A, f, R >$, seguindo uma notação similar a apresentada por Sutton e Barto (12), na qual S é um conjunto de estados, sendo $s_t \in S$ o estado no qual se encontra o agente no momento t, A é o conjunto de ações que o agente pode executar quando está no estado s_t, sendo $a_t \in A(s_t)$ a ação que o agente processa no instante t quando estava no estado s_t, f é uma função de transição e for fim, R é uma função de recompensa.

O agente tem a tarefa de encontrar a política $\pi : S \times A \rightarrow [0, 1]$, na qual $\pi_t(s, a)$ indica a probabilidade de que o agente selecione a ação a no estado s, para que assim se maximize o parâmetro de reforço.

2.1.1.1 Propriedade de Markov

A propriedade de Markov diz que para um processo estocástico ter a propriedade de Markov se a probabilidade condicional do próximo estado do processo, s_{t+1}, depende apenas do estado no qual o agente estava posicionado, s_t, e da ação executada neste estado em a_t.

Pode-se definir um MDP finito pelo seu conjunto de ações, estados e funções de transição de estados e reforços, assim de acordo com a propriedade de Markov, pode-se obter a função de transição dos estados, como mostrada na equação 2-1:

$$f(s, a, s') = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2-1}$$

na qual Pr é a probabilidade que fora comentada. Verifica-se que $\forall s \in S, \forall a \in A, \sum_{s_i \in S} f(s, a, s_i) = 1$, uma vez que a função f determina uma distribuição de probabilidades.

Na equação 2-2 é exibida a função de recompensa, na qual E define uma esperança sobre o valor a ser recebido.

$$R(s, a) = E\{r_{t+1} | s_t = s, a_t = a\} \tag{2-2}$$

2.1.2 Políticas

A política π é utilizada para que o agente escolha a ação para realizar em cada estado. Portanto, dado um MDP $\{S, A, f, R\}$, tem-se uma política

determinística com uma função $\pi : S \to A$, que mapeia cada estado a ação. Quando um agente segue uma política π, quer dizer que, em qualquer instante k e estado s_k, a ação executada será dada por $a_k = \pi(s_k)$.

Caso as políticas executem um mapeamento entre estados e distribuições no espaço de ações, $\pi : S \times A \to [0, 1]$, elas serão estocásticas, respeitando para cada estado $s \in S$ as seguintes condições (equações 2-3 e 2-4)

$$\pi(s, a) \geq 0 \tag{2-3}$$

$$\sum_{a \in A} \pi(s, a) = 1 \tag{2-4}$$

A política é parte do agente, e em geral, o objetivo dos algoritmos de RL é encontrar uma política ótima (18). Um exemplo de como uma política funciona pode ser dado da seguinte maneira: supondo um MDP determinístico onde uma distribuição de estado inicial, $I : S \to [0, 1]$, define a probabilidade de que o sistema se inicie no estado s_0, é então empregada a política π, sendo assim a ação realizada pelo agente será $a_0 = \pi(s_0)$ e como consequência dessa ação, de acordo com a função de transição f, será gerada uma transição até o estado $s_1 = f(s_0, a_0)$ e então o agente receberá uma recompensa $r_0 = R(s, a, s')$.

O processo descrito acima se repetirá gerando a trajetória $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, ...$ No caso de um problema episódico, a trajetória se encerrará em um estado terminal $s_{term} \in S_{term}$, onde $S_{term} \subseteq S$, e então uma nova sequência começará com um estado determinado por I. Se o problema for contínuo, a trajetória pode se prolongar indefinidamente.

2.1.3 Função Valor e equações de Bellman

Os algoritmos de aprendizado por reforço em sua maioria têm como objetivo tentar obter a política de ação π por aproximação das funções valor. Tais funções avaliam a vantagem para o agente de se estar em um estado ou a vantagem do mesmo executar uma ação a partir de um estado. Portanto há dois tipos de função valor, a *função valor estado-ação* Q e a *função valor estado* V

Tem-se que a função Q de uma política π , $Q^\pi : S \times A \to \mathbb{R}$ é igual ao retorno obtido quando, a partir de um estado s e a ação a, se segue a política π (equação 2-5)

$$Q^\pi(s, a) = \sum_{k=0}^{\infty} \gamma^k R(s_k, a_k, s'_k) \tag{2-5}$$

Desta fórmula se pode tirar outra expressão da função Q que, apesar de ser equivalente, é mais útil para inserir conceitos posteriores. Se $(s_0, a_0) =$

(s, a), $s_{k+1} = f(s_k, a_k, s'_k)$ para $k \geq 1$, então se pode desassociar primeiro o término do somatório, obtendo 2-6

$$\begin{aligned} Q^\pi(s,a) &= R(s,a) + \sum_{k=1}^{\infty} \gamma^k R(s_k, a_k) \\ &= R(s,a) + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} R(s_k, \pi(s_k)) \\ &= R(s,a) + \gamma R^\pi(f(s,a)) \end{aligned} \tag{2-6}$$

As funções valor tem uma propriedade fundamental, que é poderem ser caracterizadas de forma recursiva mediante a *equação de Bellman* (17). A equação de Bellman estabelece que, para uma função Q^π, o valor de realizar a ação a no estado s seguindo a política π é igual a soma da recompensa imediata e o valor com desconto alcançado por π no próximo estado, isto é (equação 2-7)

$$Q^\pi(s,a) = R(s,a) + \gamma Q^\pi(f(s,a), \pi(f(s,a))) = R(s,a) + \gamma Q^\pi(s', a') \tag{2-7}$$

A equação de Bellman para a função Q pode ser obtida a partir de $Q^\pi(s,a) = R(s,a) + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} R(s_k, \pi(s_k))$ (equação 2-8):

$$= R(s,a) + \gamma \left[R(f(s,a), \pi(f(s,a))) + \gamma \sum_{k=2}^{\infty} \gamma^{k-2} R(s_k, \pi(s_k)) \right]$$

$$= R(s,a) + \gamma Q^\pi(f(s,a), \pi(f(s,a))) \tag{2-8}$$

sendo $(s_0, a_0) = (s, a), s_{k+1} = f(s_k, a_k)$ para $k \geq 0$ e $a_k = \pi(s_k)$ para $k \geq 1$. Q^π é a única solução para o sistema de equações definido por $Q^\pi(s,a) = R(s,a) + \gamma Q^\pi(f(s,a), \pi(f(s,a))) = R(s,a) + \gamma Q^\pi(s', a')$. Importante dizer que há várias políticas que podem ter a mesma função Q, mas, para uma mesma dada política π, Q^π é única.

Normalmente, dado um MDP, o objetivo é encontrar a melhor política possível, aquela que obtenha o maior retorno. Então uma função Q ótima, apresentada como Q^*, se define como a melhor função Q que pode ser obtida com qualquer política (equação 2-9)

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{2-9}$$

Pode-se dizer que uma política ótima é aquela que em cada estado seleciona a ação com o maior valor da função Q ótima, maximizando o retorno obtido (esquação 2-10)

$$\pi^*(s) = argmax_a Q^*(s,a) \tag{2-10}$$

na qual $argmax_a g(a)$ para uma função qualquer $g(.)$ devolve o valor de a que proporciona o máximo da função $g(a)$.

Para qualquer dada função Q, uma política π que satisfaça a equação 2-11

$$\pi(s) = argmax_a Q(s,a) \tag{2-11}$$

é considerada gulosa em relação a Q. Logo, uma possível forma de se encontrar a política ótima é calcular primeiro Q^* e depois aplicar $\pi^*(s) = argmáx_a Q^*(s,a)$ para obter uma política gulosa em relação a Q^*.

Através da equação de otimalidade de Bellman, as funções valor ótimas Q^* também podem ser caracterizadas de forma recursiva. A equação mencionada determina que o valor ótimo de executar uma ação a em um estado s é igual a soma da recompensa imediata obtida mais o valor de Q^*, que foi obtido pela ação ótima no estado subsequente (equação 2-12)

$$Q^*(s,a) = R(s,a) + \gamma \max_{a'} Q^*(f(s,a), a') \tag{2-12}$$

2.1.4 Q-Learning

É possível fazer com que Q se aproxime diretamente de Q^*, independentemente da política de seleção adotada. Esse algoritmo é conhecido como *Q-Learning*(20). *Q-Learning* aprende as funções valor para as ações-estados no lugar da função valor dos estados. Em particular, se deve estimar $Q^\pi(s,a)$ para a política π, para todos os estados s, e para as ações a. Este algoritimo se define principalmente pela atualização da função mostrada na equação 2-13

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \tag{2-13}$$

Um agente em tempo t localizado no estado $s_t \in S$, elege uma das possíveis ações, $a_t \in A_{st}$, no estado de acordo com a política de seleção (ex.: ϵ-*greedy*). A equação 2-13 , atualiza a função de avaliação para esse par estado-ação considerando o valor da recompensa obtido, r_t, e a função de avaliação máxima para todas as possíveis ações no estado futuro s'. Para isso, é necessário utilizar a taxa de desconto γ, a qual determina o peso temporal relativo dos reforços e a taxa de aprendizagem α, que determina em que medida as informações recém-adquiridas substituirão as informações antigas.

Q-Learning é um dos métodos empregados com maior frequência para a solução de problemas de aprendizagem por reforço, isto se deve provavelmente a

sua simplicidade e robustez que permite aplicá-los de forma rápida e eficiente com requisitos mínimos de computação. Por um lado, o *Q-Learning* é um algoritmo que possui uma base teórica, o que é importante porque seu uso pode ser estendido para a solução de problemas de otimização. Por outro lado, o *Q-Learning* estima a função valor de um par estado-ação usando as estimativas das funções valor de outros pares estado-ação, mas sem necessitar de um modelo do ambiente. Essas duas propriedades, associadas ao fato de que o Q-learning pode ser aplicado de forma incremental e aprender diretamente das iterações do agente com o ambiente, o tornam um algoritmo de aprendizado de reforço extremamente atrativo.

Este trabalho tem como base o algorítmo de aprendizagem por reforço proposto por Watkins (20) (*Q-Learning*), porém um pouco modificado, como será explicado nos próximos capítulos.

Basicamente, o objetivo do algoritmo *Q-Learning* (Algoritmo 2.1) é encontrar todos os estados possíveis para certas ações e manter um registro deles. Para cada ação realizada (a), uma recompensa (r) será concedida, seja favorável ou negativa, de modo que em visitas subsequentes ao mesmo estado (s), possa ser determinada qual ação é a mais apropriada.

Q-learning: Função de aprendizado $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$

Require:
Estados $\mathcal{S} = \{1, \ldots, n_x\}$
Ações $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Taxa de aprendizado $\alpha \in [0, 1]$, tipicamente $\alpha = 0.1$
Fator de desconto $\gamma \in [0, 1]$
procedure QLearning($\mathcal{X}$, A, R, α, γ)
Inicializa $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ arbitrariamente
while Q não converge **do**
Inicia o estado em $s \in \mathcal{S}$
while s não é terminal **do**
Calcula π de acordo com Q e estratégia de exploração
(ex. $\pi(x) \leftarrow \arg\max_a Q(x, a)$)
$a \leftarrow \pi(s)$
Executa ação a e Recebe recompensa r e próximo estado s' do simulador
$Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
$s \leftarrow s'$
return Q

Figura 2.1: Algoritmo Q-Learning

2.2 Deep Learning

Redes neurais são modelos inspirados em cérebros biológicos. Seus componentes simulam as interações entre axônios e dendritos, que emitem e recebem impulsos nervosos respectivamente. Redes neurais tem demonstrado bom resultado quando utilizadas em problemas muito complexos, como: segmentação de objetos em imagens ou vídeos, a descrição de cenas ou a extração semântica de palavras. Neste trabalho modelos de rede neurais e *deep-learning* são utilizadas em todas as tarefas de aprendizado, nas seções seguintes serão introduzidos os modelos e técnicas baseadas em redes neurais utilizados neste projeto.

2.2.1 Redes convolucionais

As redes neurais convolucionais, em inglês *Convolutional Neural Network* (CNN), podem ser consideradas a parte da inteligência artificial (IA) mais inspirada na biologia, em especial na neurociência, já que seu trabalho é análogo ao que o córtex visual primário faria. As CNNs representam um modelo profundo bem sucedido. Hoje, elas são utilizadas em diversas aplicações, como tarefas de visão computacional, com bons resultados. Por esta razão, estas redes são de grande interesse para este trabalho, uma vez que CNNs serão utilizadas para retirar as características principais das imagens geradas pelos simuladores. Nota-se que, para a visão computacional, são as redes mais robustas e bem sucedidas da atualidade, como foi demonstrado em 2012 quando uma CNN ganhou o desafio de reconhecimento de imagens do ImageNet (21).

Importante ressaltar que as entradas da CNN podem variar de tamanho, fato que representa uma grande vantagem no trabalho com imagens.

2.2.1.1 Camadas convolucionais

Uma convolução é uma operação matemática entre duas funções f e h, que produz uma terceira função vista como uma versão modificada de f como uma função de h. Neste trabalho, f é delimitado como uma sequência de vetores e h é um filtro linear que faz com que cada novo elemento da saída seja uma soma ponderada dos elementos no contexto de cada elemento processado na sequência. Assim, na sequência de entrada (ou nas saídas de convolução anteriores) é aplicado um conjunto de filtros lineares com os mesmos pesos para toda a sequência, no caso de redes convolucionais, aprendidas por *back-*

propagation. Isso alcança certas propriedades na sequência de saída, sendo uma das mais importantes a invariância à tradução.

Quando há uma entrada f e um filtro h, cada elemento (i, j) da saída g de uma convolução se define como $g(i, j) = \sum_{k,l} f(i-k, j-l)h(k, l)$, onde k e l são a altura e largura do filtro respectivamente. Na figura 2.2 mostra-se um exemplo desta operação.

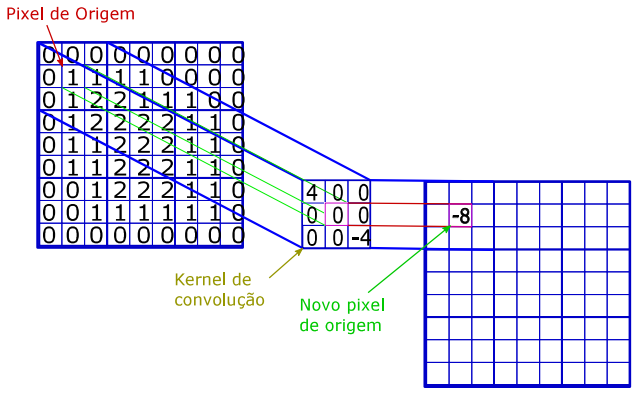

Figura 2.2: Convolução 2D

Em seu estudo, Min Lin e Qiang Chen (22) apontam três fatores que contribuem para a eficiência dos algoritmos de Deep Learning. Estes são: (1) a conectividade ou interações dispersas, (2) o compartilhamento de parâmetros e (3) a invariância para a localização do objeto.

Em relação à primeira vantagem, os parâmetros de interação entre cada entrada e saída são compartilhados, fazendo com que existam mais interações que parâmetros. Por consequência, os requisitos de memória diminuem e as correlações entre os pixels vizinhos são aprendidas.

A vantagem do segundo fator está ligado ao conceito anterior, como várias funções do modelo usam os mesmos parâmetros, a memória é salva.

A contribuição do terceiro fator deriva de sua equivalência, já que, se a entrada muda, a saída também o faz na mesma medida.

Devido a essas vantagens, as camadas convolucionais por vezes substituem as camadas *fully connected* para acelerar o processo de aprendizado. Isto acontece, por exemplo, com as técnicas NIN (Network to Network) (22).

2.2.1.2 Camadas de agrupamento ou Pooling

As operações de *pooling* tem como objetivo principal reduzir o tamanho das matrizes de saída em problemas onde o tamanho da entrada é muito grande. Ou seja, após a aplicação de convoluções, a saída permanecerá grande,

com a utilização do *pooling*, o tamanho da saída é reduzido, assim há uma também uma redução no custo de tempo de cálculo das convoluções subsequentes, já que o tamanho de sua entrada fora reduzido previamente pelo *pooling*.

Os operadores pooling oferecem uma maneira de realizar *down-sampling*, geralmente não-linear, sobre a entrada. O que os permite lidar com várias escalas focando em um componente do resultado obtido após a aplicação desta operação em uma entrada.

Há vários tipos de operações de *pooling*, as mais utilizadas são *max pooling* e *average pooling*, que substituem regiões de $k \cdot l$ da entrada por um único componente calculado como o máximo ou a média da região, nesta ordem. Um exemplo dessas operações é mostrado na Figura 2.3

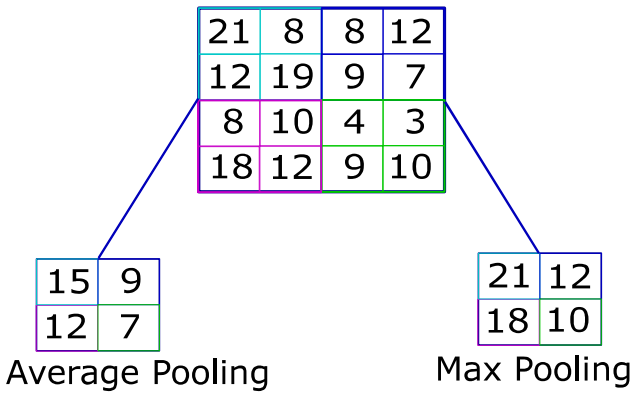

Figura 2.3: Exemplo de *max pooling* e *average pooling*

2.2.1.3 Camadas totalmente conectadas (*Fully Connected*- FC)

As camadas FC funcionam como uma rede neural tradicional e contêm aproximadamente 90% dos parâmetros de rede (23). O vetor de saída 1D da rede é geralmente de comprimento predefinido. Por exemplo, para uma tarefa de classificação de imagem será de comprimento igual ao número de categorias (classes) que são tomadas. Outra opção comum na análise de imagens é tomá-lo como um vetor de características 1D para processamento posterior (o que mais tarde será chamado de extração de características).

Como esse tipo de rede manipula uma grande quantidade de parâmetros, é muito custoso treiná-las, uma vez que necessitam de uma grande carga computacional para treiná-las. Por esta razão muitos autores defendem diminuir as conexões entre os neurônios dessas camadas utilizando algum tipo de método, como no caso do *GoogleLeNet* (24), reduzi-los em número, ou até mesmo eliminá-los (25).

A parte mais complicada quando se trabalha com esse tipo de arquitetura é o treinamento. Se for decidido usar uma estratégia supervisionada, tanto no *feedforward* e *backpropagation*, será necessário computadores com alta capacidade computacional.

2.2.2 *Rectified Linear Unit* - ReLU

Tipicamente, funções como a *sigmoid* ou *hyperbolic tangent* têm sido utilizadas funções de ativação, porém a suas derivadas tem a peculiaridade de assumir valores muito baixos em determinadas situações, o que causa a paralisia da rede.

Foram propostas várias soluções para solucionar este problema, entre elas está o uso da função de ativação ReLU ($f(x) = \max(0, x)$), que resolve o problema dito anteriormente. A sua derivada é $f'(x) = 1$ se $x > 0$ e zero caso contrário, o que faz com que o gradiente se propague o erro mensurado até as camadas iniciais da rede. Porém, apesar de vantagens como essa e outras, como a simplicidade da computação e a invariância à escala, a ReLU sofre de um problema conhecido como *dying ReLU problem*, que ocorre quando as pré-ativações são em sua maioria negativas. Para tal, foram propostas funções como *Leaky ReLU* (LReLU) (26).

2.2.3 Generalização

Como há um grande número de parâmetros para os modelos de *deep learning* utilizarem e, geralmente, um pequeno número de amostras disponíveis para ajustá-los, ocorrem problemas de generalização, dentre eles, pode-se destacar o *underfitting* e *overfitting* (27), sendo este último mais frequente durante a experimentação de redes neurais.

2.2.4 Segmentação

A segmentação tem como objetivo simplificar e/ou alterar a representação de uma imagem em algo que é mais significativo e mais fácil de analisar, sendo assim, a segmentação de imagem normalmente é usada para localizar objetos e/ou limites nas imagens. Pode-se dizer que a segmentação de imagens atribui um rótulo a cada pixel em uma imagem, de modo que os pixels com o mesmo rótulo tenham as mesmas características, isso foi feito adaptando as arquiteturas existentes para gerar resultados de classificação em nível de pixel e, portanto, ser útil em tarefas como a segmentação semântica, assim as CNNs

são usadas, pois são capazes de fazer essas previsões em conjuntos de dados muito grandes.

Segmentação semântica tem uma ampla gama de aplicações que vão desde a compreensão da cena até a condução autônoma, observada muito atualmente. A SegNet (28) (Figura 2.4), por exemplo, é uma rede neural convolucional projetada para ser uma arquitetura eficiente para segmentação semântica em pixels. É motivada principalmente por aplicações de compreensão de cenas de estrada que exigem a habilidade de modelar a aparência, forma e entender a relação espacial entre diferentes classes.

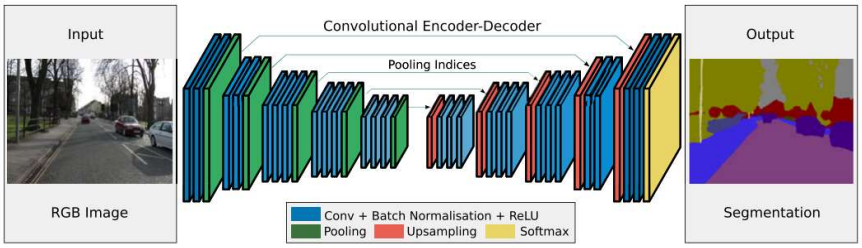

Figura 2.4: Figura retirada de *Segnet: A deep convolutional encoder-decoder architecture for image segmentation* (28), exibindo a arquitetura da rede SegNet

Como falado anteriormente, os maiores esforços para alcançar esta adaptação foram baseados nas *Fully Convolutional Networks* (FCN) (29). Através delas, converteu-se as CNNs em redes completamente convolucionais e realizou-se um conjunto de operações de *upsampling* nos mapas de características obtidas até atingir o tamanho de entrada das imagens originais, a fim de obter a classificação no nível de pixel desejado. Essa primeira abordagem se mostra insuficiente devido aos processos de *dowsampling* de sinais incluídos nas redes originais. Estes buscam aumentar as informações contextuais utilizadas na análise da imagem, no entanto, as saídas obtidas são de baixa precisão (pixels espessos). Para obter precisão no nível de pixel procurado, um processo de refinamento adicional é exigido nos resultados. Para tanto, uma solução foi proposta: combinar os mapas de características obtidos nas primeiras camadas da rede - que conservam praticamente todos os resolução da imagem - com as saídas finais obtidas através de conexões entre ambas as camadas.

2.2.4.1 PSPNet

A PSPNet (30) foi a rede que apresentou os melhores resultados ao usar um decodificador para estudar as informações fornecidas pelo codificador combinando blocos de tamanhos diferentes para analisar a imagem em diferentes

resoluções e assim beneficiar a informação contextual. Os resultados obtidos por esta rede foram superiores aos apresentados por redes como LinkNet (31), ENet (32) e SegNet (28), mas o tempo de processamento requerido foi extremamente alto, ruim para imagens muito grandes.

A PSPNet foi escolhida para ser utilizada neste trabalho pelos bons resultados apresentados, e por ser treinada com o dataset ADE20K (33) , uma vez que as imagens são parecidas com as produzidas pelo simulador *Realistic Rendering*, que será abordado no próximo capítulo. Os resultados da segmentação da rede PSPNet, treinada com ADE20k, são exibidos na figura 2.5.

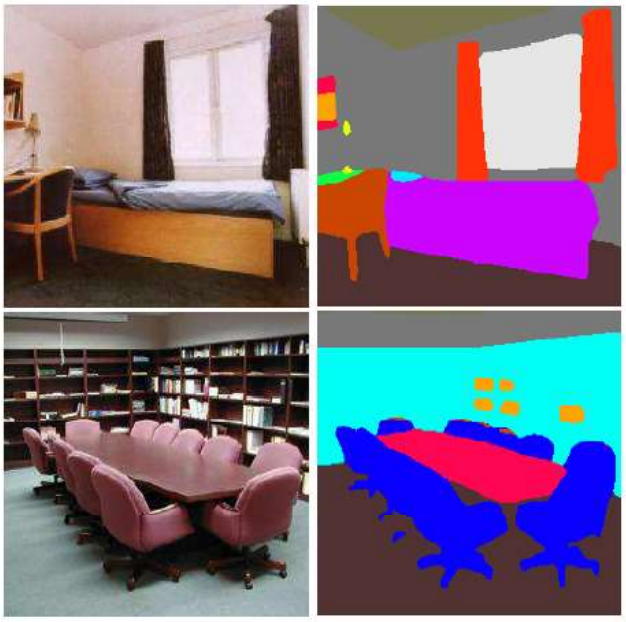

Figura 2.5: Figura retirada de (30), exibindo imagem resultante da segmentação feita pela rede PSPNet

2.2.5 Transfer Learning

Algoritmos de aprendizado automático progrediram nos últimos anos, em geral, eles assumem que os exemplos de treino e teste têm os mesmos atributos e vêm da mesma distribuição. Se a distribuição mudar, geralmente necessita-se reconstruir os modelos, em algumas aplicações os dados são escassos ou estão desatualizados ou não se pode coletar os exemplos de treinamento para reconstruir um modelo. Para situações como essas, que a técnica de *Transfer Learning* é utilizada, uma vez que visa utilizar o conhecimento previamente adquirido para novos problemas, considerando que os domínios, tarefas e distribuições utilizados em treinamentos e testes podem ser diferentes (34).

Basicamente neste trabalho, para reduzir o tempo de treinamento sem diminuir a precisão, treinou-se rede neural usando *Transfer Learning*, mantendo as primeiras camadas e apenas treinando as camadas recém-adicionadas, pôde-se aproveitar o conhecimento adquirido pelo algoritmo pré-treinado e usá-lo para o próximo treinamento, como visto na Figura 2.6.

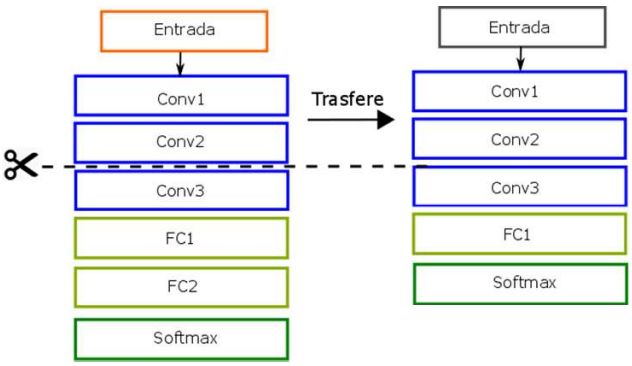

Figura 2.6: Exemplo de *transfer learning* de duas camadas convolucionais

2.3 Deep Reinforcement Learning

Como visto nas seções anteriores, o *Q-Learning* utiliza a tabela Q como um método de representação para a função Q. Este método permite representar, de maneira muito precisa, o valor das recompensas de longo prazo que podem ser obtidas em cada estado para cada uma das ações disponíveis. No entanto, essa representação também implica em um problema muito importante: aumentar a complexidade do estado também aumenta o tamanho da tabela Q, porém de maneira exponencial.

Desse modo, o objetivo é criar um modelo capaz de resolver um problema complexo, viabilizando que o tamanho da tabela Q seja grande, porém possível de armazenar na memória. Então a ideia foi utilizar uma aproximação da função Q através do uso de uma rede neural, a qual foi empregada, em 2013, pela equipe da DeepMind no seu projeto Playing Atari with Deep Reinforcement Learning (35).

2.3.1 *Deep Q-Network*

No projeto mencionado acima, foi desenvolvido um agente capaz de aprender a jogar sete jogos do console Atari utilizando os pixels da tela como variáveis de estado, após um pequeno pré-processamento no qual foram convertidos em uma escala de cinza e o tamanho foi redimensionado para 110x84 pixels, em vez dos 210x160 originais. Os resultados obtidos foram

impressionantes, conseguindo superar um jogador humano experiente em três desses jogos. Tal agente foi nomeado como *Deep Q-Network* (ou DQN), sendo essa a primeira implementação de um agente *Deep Reinforcement Learning*.

Como dito anteriormente, sua principal diferença em relação a um agente usual de *Q-Learning* foi a substituição da tabela Q por uma rede neural que aproximou os valores da função Q para qualquer par estado-ação, ou seja, a rede neural tem como saída a uma tabela Q aproximada.

Essa rede precisava ser treinada para que essa abordagem fosse a mais precisa possível, de modo que o objetivo era minimizar a diferença entre o valor aproximado da rede e seu valor esperado.

Por fim, as *Deep Q-Networks* não deixam de ser um MDP, então as ações tomadas dependem apenas do estado atual. Assim em alguns jogos (como o *Breakout* (36)) em que algo tão importante, quanto a velocidade da bola, era impossível de se estimar considerando apenas o último *frame*. Assim, eles decidiram que o estado foi formado pelos últimos quatro *frames* observados, de modo que as mudanças que ocorreram no ambiente ao longo do tempo pudessem ser consideradas.

O Algoritmo que pode ser observado na Figura 2.7 pode ser explicado da seguinte forma: primeiramente inicializa-se a *replay memory*, que é uma espécie de memória que guarda as transições passadas (será melhor explicada na seção 2.3.3), é inicializada também a função Q com pesos aleatórios. Após esses procedimentos, para cada época é inicializado um estado, que consiste em uma sequência de imagens que são retiradas do simulador e pré-processadas. E para cada passo que o agente faz, são realizadas as seguintes etapas: uma ação é escolhida de acordo com uma política gulosa, a ação é então executada no simulador e se obtém do mesmo uma recompensa e a próxima imagem, esta imagem é processada, e então chamada de estado. As transições são guardadas na *replay memory*, e só então se escolhe um *minibatch* de transições a partir desta memória. Utilizando este *minibatch*, é possível calcular a recompensa futura e assim estimar Q a partir de uma rede neural.

Deep Q-Learning

Inicializa replay memory $\mathcal{D}$ de tamanho N
Inicializa função Q com pesos aleatórios
for época = 1, M **do**
  Inicializa sequência $s_1 = \{x_1\}$ e preprocessa sequência $\phi_1 = \phi(s_1)$
  for $t = 1$, T **do**
    Seleciona com probabilidade ϵ uma ação aleatória a_t,
    caso contrário seleciona $a_t = max_a Q^*(\phi(s_t), a; \phi)$
    Executa ação a_t no simulador e observa recompensa r_t e imagem x_{t+1}
    Preprocessa a imagem x_{t+1} e Seta $s_{t+1} = s + 1, a_t, x_{t+1}$
    Guarda a transição $(\phi_t, a_t, r_t, \phi_{t+1})$ em $\mathcal{D}$
    Escolhe aleatoriamente minibatch de transições $(\phi_j, a_j, r_j, \phi_{j+1})$ de $\mathcal{D}$
    Seta $y_j = r_j$ se o estado for terminal
    Seta $y_j = r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta)$ se o estado não for terminal
    Performa o gradiente descendente em $(y_j - Q(\phi_j, a_j; \theta))^2$

Figura 2.7: Algoritmo Deep Q-Learning

2.3.2 *Target Q-Network*

Um dos problemas que podem ser observados é que o uso da rede para inferir o valor da função Q para o estado atual e para o próximo estado tornou o processo de aprendizagem muito instável, devido a dependência dos valores aprendidos pela rede, que mudavam constantemente com cada uma das atualizações.

Para reduzir o nível de dependência com os valores atuais da rede neural, foi introduzida uma segunda rede, denominada "rede de destino"ou *target network*, que é uma cópia da rede original. A rede original foi utilizada para inferir o valor da função Q para o estado atual, enquanto a segunda foi utilizada para inferir o valor do próximo estado. Ao contrário da rede original, *target network* não era constantemente atualizada, mas sincronizada com o valor dos parâmetros da rede original depois de ter sido treinada com vários conjuntos de treinamento.

A presença dessa rede proporcionou estabilidade aos valores previstos para os estados seguintes, o que, por sua vez, também proporcionou estabilidade para o aprendizado.

2.3.3 *Experience Replay*

Para aprender com amostras de dados consecutivas não se mostra muito eficiente, uma vez que elas tendem a ter um alto grau de correlação entre

si. Isso é uma grande desvantagem, uma vez que o objetivo dos conjuntos de treinamento é ser o mais representativo possível.

Então para que o treinamento não fosse realizado com amostras de dados consecutivos, foi utilizado um método chamado *experience replay*, que possibilita reduzir a correlação entre os dados e minimizar a possibilidade de que o aprendizado ficasse restrito a um mínimo local. Esse método consiste em armazenar em cada passo tuplas da forma $< s, a, r, s' >$ na *replay memory*, onde s é o estado atual, a ação tomada, r a recompensa obtida e s' próximo estado; e durante o treinamento da rede, um conjunto de tuplas selecionadas aleatoriamente dentre todas as armazenadas. Dessa forma, a rede poderia treinar usando amostras em uma ordem diferente na qual elas ocorreram.

O número de amostras armazenadas foi limitado, de modo que, quando o tamanho máximo fosse atingido, os mais antigos seriam substituídos pelos novos, para que o conjunto fosse sempre atualizado. Além disso, a possibilidade de utilizar a mesma amostra em diversos conjuntos de treinamento aumentou consideravelmente a eficiência dos dados obtidos.

2.3.4 *Transfer learning* em *Deep Q-Networks*

Como algoritmos de *deep Q-learning* necessitam de um grande poder computacional para serem treinados, empregar técnicas de *Transfer Learning* faz com que se diminua o tempo e processamento no treinamento dos agentes. Por isso neste trabalho foi empregado esta técnica, afim de baixar o tempo gasto do treinamento sem que os resultados finais sejam comprometidos. Alguns exemplos de *Transfer Learning* em *Deep reinforcement learning* podem ser vistos em (37) (38) (39)

3
Modelo

Esta seção descreve os modelos criados para avaliar a utilização de *transfer learning* em ambientes treinados por *deep Q-learning*.

3.1
Ambientes

Há dois tipos de ambientes, os de coleta de objetos, nos quais o agente deve achar e pegar um objeto, e os ambientes de navegação, cujo objetivo é o agente se auto-localizar e encontrar uma porta no cenário. Há três ações disponíveis para o agente realizar: *Move Forward*, *Turn Left*, e *Turn Right*. Os valores das recompensas para os de coleta de objetos, são de 100 ao se chegar ao objetivo e -1 para cada passo dado até se chegar ao destino. Recompensas para os ambientes de navegação o valor das recompensas foram um pouco diferentes, em caso do agente encontrar a porta, a recompensa é de 200 e em caso de colisão com a parede, a recompensa é de -10. E para o *Health Gathering* como objetivo é sobreviver o maior tempo possível em uma sala com um piso que causa danos periodicamente, "medikits"são gerados aleatoriamente nessa sala, ajudando o agente a sobreviver, pois curam os danos causados. Assim o agente é recompensado por 1 a cada passo em que está ativo e -100 por morrer.

Outros cenários foram criados utilizando o *Unreal*, cujo modelo se assemelha aos outros ambientes, apresentando o mesmo número de ações e de recompensas.

A capacidade de transferir conhecimento de experiências anteriores é fundamental para um agente se adaptar rapidamente a diferentes ambientes e aprender efetivamente novas tarefas. Como dito anteriormente, este trabalho faz um estudo simulado de DQN, onde o agente é avaliado em ambientes que o mesmo ainda não conhece, mostra que se pode treinar uma rede robusta para navegação e através de sua eficácia em generalizar para mapas desconhecidos com texturas de fundo desconhecidas e com diferentes tarefas.

Assim foi investigada a eficácia do pré-treinamento para transferir conhecimento de vários cenários. Em particular, mostra-se que os recursos aprendidos pela rede de navegação podem ser efetivamente utilizados para transferir co-

nhecimento entre um conjunto diversificado de tarefas, como coleta de objetos e localização de portas.

3.2 Configuração experimental

Foram criados vários cenários (Figura 3.1) para investigar transfer learning em DQNs. Estes cenários foram desenvolvidos no ambiente de jogo Doom usando a API ViZdoom (40) e o editor Doom de código aberto, Slade 3 (41). A API do ViZDoom dá acesso direto ao motor do jogo ZDoom (42) e permite enviar comandos ao agente do jogo de forma síncrona e receber entradas (imagens) do estado atual do jogo. Além disso foi feita a interação com o mecanismo de jogo Doom usando scripts ACS dentro do editor Doom para calcular recompensas para todos os cenários.

Para os primeiros testes foram utilizados os ambientes do Doom, foram criados ambientes com diferentes níveis de dificuldade de navegação diferentes, que serão discutidos mais a frente. O código utilizado foi inspirado no que foi empregado no ViZDoom, porém com algumas diferenças, como por exemplo tendo como entrada da rede 4 imagens, como explicado em (35).

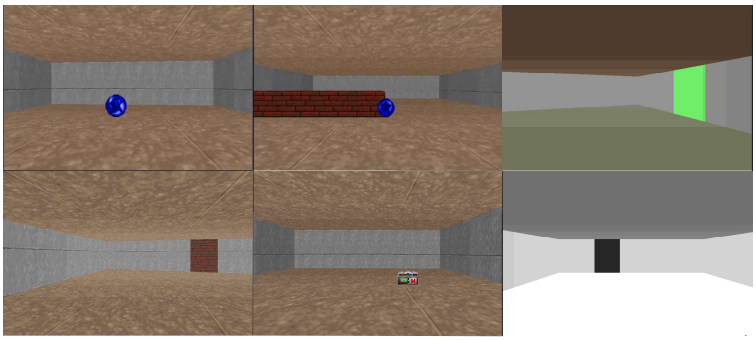

Figura 3.1: Cenários criados para os testes utilizando Doom como ambiente

Este simulador foi escolhido para os primeiros testes por apresentar menos complexidade para o algoritmo, assim os resultados seriam obtidos mais rapidamente.

Além do ZDoom, foi utilizado o Unreal (43) , uma plataforma de criação de jogos muito conhecida, de onde foi retirado como template o ambiente *Realistic Rendering* (44), o código aplicado a esse ambiente foi inspirado no encontrado em (43), porém houve modificações, como a rede utilizada, tendo como entrada 4 imagens, com menos camadas e sem a utilização de pooling como explicado em (35), o pré-processamento havendo segmentação no caso da utilização de *transfer learning*. Este simulador foi escolhido por apresentar

imagens com grande semelhança com a realidade (Figura 3.2) , ponto de grande importância que será explicado nas próximas seções.

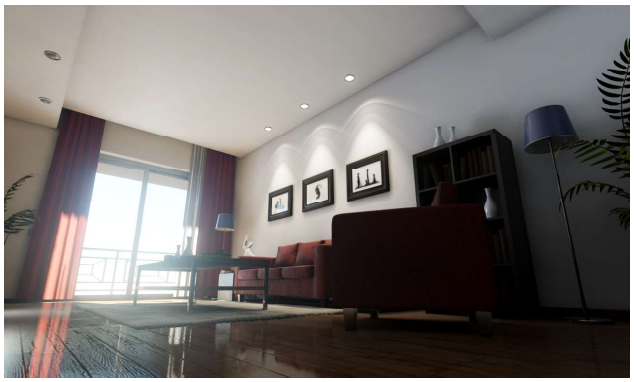

Figura 3.2: Ambiente *Realistic Rendering*

Importante ressaltar que em todos os treinamentos foi utilizado um computador com uma GPU (Graphics Processing Unit) 1080ti da Nvidia e CPU(Central Processing Unit) Intel i5-3450. Como a maior parte do processamento era seriado, já que os passos dados pelo agente dependiam exclusivamente do último passo executado, apenas aproximadamente 23% da GPU foi utilizada para o processamento das imagens (estados).

3.3 Arquitetura de rede utilizada

Para os ambientes mais simples como o Doom, foi utilizado um modelo de rede neural criado utilizando Tensorflow (45) como base. Este modelo foi um pouco modificado, como dito na seção de Configuração experimental, ele é um modelo simples, apresentado apenas 2 camadas convolucionais e 2 *full connected*, como visto na Figura 3.3.

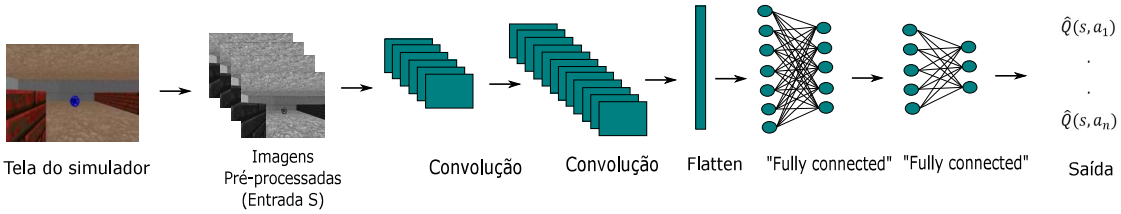

Figura 3.3: Modelo de rede utilizado para os ambientes simples como os do Doom

A tabela 3.1 exibe os parâmetros da rede apresentada.

Tabela 3.1: Arquitetura da rede utilizada pelos ambientes Doom

Camada	**Saída**	**Tamanho do Kernel**	***Stride***	**Ativação**
Conv2d	8	(6,6)	(3,3)	ReLU
Conv2d	16	(3,3)	(2,2)	ReLU
Fully Connected	128			ReLU
Fully Connected	8			

Já para o ambiente *Realistic Rendering*, foi utilizada uma rede 3.4 um pouco mais profunda, por o ambiente ser mais complicado que os já citados, uma vez que apresenta uma textura com mais detalhes e mais objetos espalhados pelo ambiente, dificultando a navegação do agente até o objetivo.

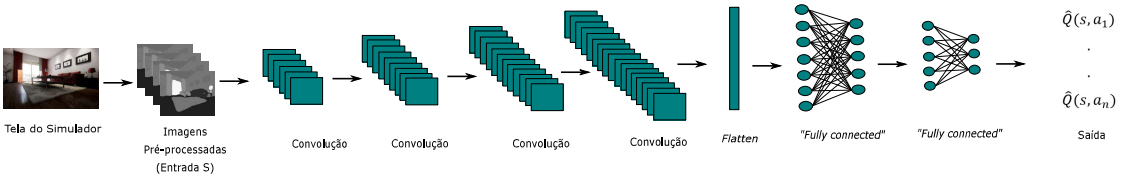

Figura 3.4: Modelo de rede utilizado para o ambiente *Realistic Rendering*

Tabela 3.2: Arquitetura da rede utilizada pelo *Realistic Rendering*

Camada	**Saída**	**Tamanho do Kernel**	***Stride***	**Ativação**
Conv2d	8	(6,6)	(3,3)	ReLU
Conv2d	16	(3,3)	(2,2)	ReLU
Conv2d	32	(6,6)	(2,2)	ReLU
Conv2d	64	(6,6)	(2,2)	ReLU
Fully Connected	128			ReLU
Fully Connected	8			

3.4 Síntese do treinamento

O objetivo deste trabalho é fazer com que um autômato aprenda a navegar em um ambiente desconhecido e para isso uma rede neural é treinada, com a entrada da rede descrita acima requer apenas 4 frames (35) , e para cada frame é feito um pré-processamento antes das mesmas serem dadas como entrada na rede neural.

O pré-processamento dos ambientes criados com Zdoom consiste em redimensionar cada imagem para (30, 45), colocá-la em tons de cinza e em seguida normalizá-la.

Já para os criados com Unreal, antes do redimensionamento, é utilizada segmentação semântica, feita pela PSPNet, rede já treinada utilizando o dataset ADE20K, explicada no capítulo 3. Essa segmentação é necessária pelo fato de a rede treinada anteriormente ter aprendido com imagens de um ambiente segmentado. Sendo assim as imagens do ambiente *Realistic Rendering* deveriam se parecer com o que fora treinado anteriormente, e afim de solucionar essa questão foi utilizada uma rede para segmentar as imagens retiradas do *Realistic Rendering*. Em alguns casos é utilizado o *transfer learning* para evitar retrabalho e retreinamento dos pesos já treinados. Após pré-processar as imagens elas são dadas como entrada na rede que foi descrita na seção anterior, treinando assim o agente.

Após as redes serem treinadas foram criados gráficos para ser possível exibir os resultados da melhor forma. Os gráficos utilizando os dados dos testes foram gerados da seguinte forma:

Dados como número de épocas, passos (*steps*), recompensas acumuladas e tempo de treinamento foram coletados ao final de cada época. Importante mencionar que cada época pode consistir em vários episódios, ou seja, uma época tem no máximo 2000 *steps* e o agente deve chegar até o seu objetivo neste limite de *steps*, e cada vez que ele chega ao objetivo um episódio é concluído, ou seja, um época pode ser igual a um episódio caso o agente não chegue ao objetivo durante os 2000 *steps*, como mostrado na figura 3.5.

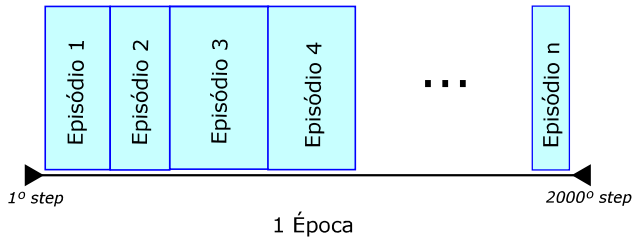

Figura 3.5: Época e episódios

Levando em conta a estocasticidade do algoritmo, um mesmo experimento foi executado 10 vezes, e então a média das recompensas acumuladas em cada época foi feita. Porém para se obter a média de cada época foi feita a média das recompensas acumuladas de cada episódio dentro de uma época.

Algoritmos que utilizam o "Doom"como ambiente de simulação apresentam os seguintes hiperparâmetros exibidos na tabela 3.3

Tabela 3.3: Hiperparâmetros Doom

Variável	**Valor**
Épocas	100
Passos por época	2000
Tamanho da replay memory	10000
Tamanho do batch	128
Taxa de aprendizado	0.0002
Taxa de desconto	0.99
Episódios de validação por época	20

Já os que utilizaram o *Realistic Rendering*, os hiperparâmetros foram um pouco modificados, devido ao ambiente ser mais complexo, exibidos na tabela 3.4

Tabela 3.4: Hiperparâmetros *Realistic rendering*

Variável	**Valor**
Épocas	8000
Passos por época	2000
Tamanho da replay memory	50000
Tamanho do batch	128
Taxa de aprendizado	0.0001
Taxa de desconto	0.95
Episódios de validação por época	20

Para calcular o intervalo de confiança foi utilizado desvio padrão para assim calcular o erro padrão com 95% de confiança (Eq. 3-1), logo foi possível plotar os gráficos de comparação de resultados utilizando as médias e o intervalo de confiança (Eq. 3-2).

$$s = \frac{\sigma}{\sqrt{n}} \tag{3-1}$$

$$IC = 1.96 * \sigma \tag{3-2}$$

Outro parâmetro utilizado para comparar a eficiência do treinamento é o *rise time* (tempo de subida), que é o tempo que se leva para mudar de um valor baixo para um valor alto, ou seja, o tempo que o agente levou até ser treinado. Ele foi calculado observando a diferença de recompensa de épocas, isto é, quando a diferença entre as médias das recompensas das 5 últimas épocas já treinadas é de aproximadamente 10%.

4
Experimentos

Os experimentos que serão apresentados nesse trabalho utilizaram algoritmos de *deep Q-learning*, técnicas de *transfer learning* e ambientes criados para os testes.

O primeiro ambiente que foi utilizado foi o *Health Gathering*, ele é um ambiente padrão criado para testes de *deep reinforcement learning*, cujo objetivo é o agente capturar a maior quantidade de objetos (*'medikit'*) possível, ele foi utilizado para testar a eficiência da rede neural que seria utilizada nos testes. Após a rede ser validada, foram realizados testes para se verificar a qual era a melhor inicialização do agente e do objetivo, para isso foi utilizado um ambiente contendo uma esfera (*'soul sphere'*) como objetivo, ambiente foi utilizado por apresentar poucas dificuldades e ser fácil de ser treinado. Em seguida, foi empregada a técnica de *transfer learning* para averiguar a transferência entre tarefas, para isso foi criado um ambiente semelhante ao dos testes de inicialização, porém tendo como objetivo alcançar uma caixa (*'medikit'*).

Para verificar a robustez do algoritmo, foi criado um ambiente com barreiras, tendo como objetivo o agente alcançar uma esfera. O objetivo da criação desse ambiente foi testar o a transferência de aprendizado entre ambientes diferentes. Como os resultados foram bons, foram criados mais ambientes afim de se testar *transfer learning* para cenários diferentes, para isso foi criado um ambiente com apenas uma porta, no qual o agente tem como objetivo chegar a mesma e um outro ambiente semelhante a este, porém segmentado manualmente.

Como o objetivo era se ter um ambiente cada vez mais complexo para os testes, foi utilizado o *Realistic Rendering*, cujo objetivo é o agente chegar a apenas uma porta das três existentes no cenário. Foi utilizado por apresentar maior complexidade, semelhança ao mundo real e obstáculos como mesa e plantas. Uma vez que se gostaria de utilizar *transfer learning* com este cenário, foi criado, utilizando o ViZDoom, um ambiente segmentado com 3 portas que apresenta a mesmo mesmo objetivo que o *Realistic Rendering*, porém o este ambiente não apresenta obstáculos e é segmentado.

4.1 Teste da rede neural

Como uma forma de testar a rede neural, foi treinando o ambiente *Health Gathering* (Figura 4.1). Este é um cenário Doom padrão muito utilizado para treinar algoritmos de *reinforcement learning* (40).

Figura 4.1: Ambiente Health Gathering

Na Tabela 4.1 estão os resultados e os gráficos nas Figuras 4.2 e 4.3 demonstrando que utilizando aproximadamente 68 épocas é possível treinar um ambiente como o *Health Gathering*. Isso mostra que o agente a chegar ao seu objetivo é capaz de aprender a partir da rede utilizada (Figura 3.3). Assim a mesma será usada nos próximos experimentos.

Importante ressaltar que como visto no gráfico, Figura 4.3, primeiro o agente aprende a pegar algumas caixas, mas acaba morrendo, pois ainda não tem muita noção do que tem que fazer, depois de umas 40 épocas de treinamento, o agente começa a andar em forma circular, pois há muitas caixas espalhadas pelo chão, então ele andando dessa forma acaba por pegar algumas delas e por fim, depois de aproximadamente 68 épocas o agente aprende a distinguir o que é realmente uma caixa e que ele precisa pegar a mais próxima para que assim tenha a maior recompensa.

Tabela 4.1: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do *Health Gathering*

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Health Gathering	21.76	83.63	17.84	48.26	13.91	49.95	15.57

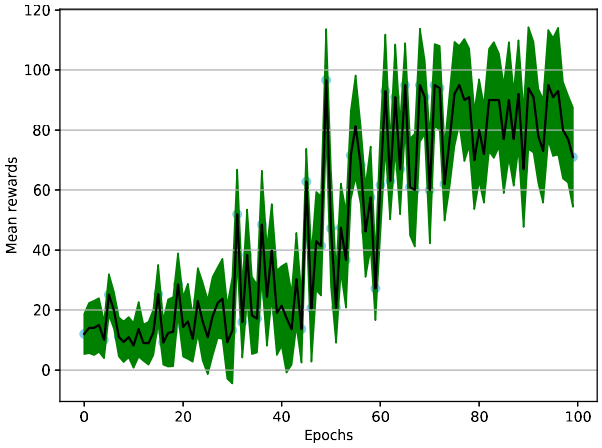

Figura 4.2: Média e desvio padrão das recompensas do treinamento do *Health Gathering*

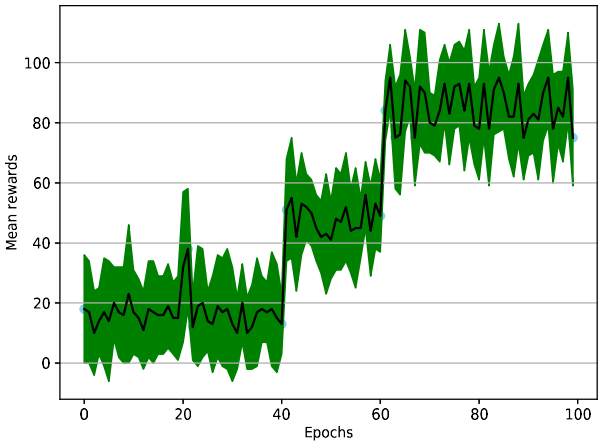

Figura 4.3: Média e desvio padrão das recompensas da validação do *Health Gathering*

4.2 DQN em um ambiente simples

Os primeiros testes utilizaram ambientes bem simples, como o que será apresentado agora, foi criado um cenário similar ao *Health Gathering*, porém utilizando apenas um objeto de formato esférico - no Doom conhecido como "soul sphere" - como objetivo a ser alcançado, Figura 4.4.

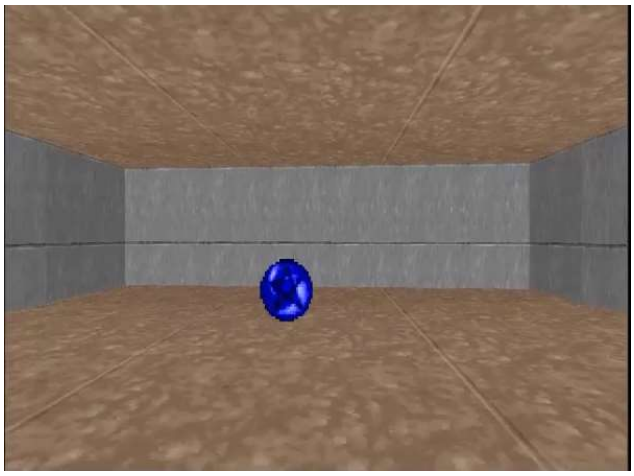

Figura 4.4: Ambiente simples com 'esfera'

Neste mesmo ambiente foram feitas algumas alterações relativas a inicialização do agente e da esfera no cenário, como por exemplo começar as simulações com o agente em um lugar fixo e esfera em um lugar aleatório, ou agente e esfera fixos. Foi notado que esses diferentes tipos de inicialização influenciam no resultado do treinamento do agente.

4.2.1 Simulação inicializada com agente e esfera na mesma posição

Nesse ambiente o agente e o objeto foram criados em todos os episódios ocupando sempre a mesma posição (Figura 4.5).

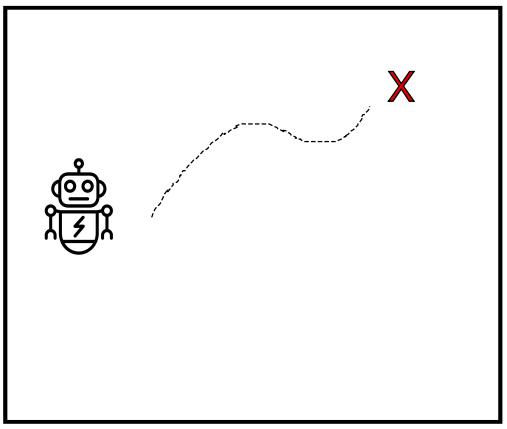

Figura 4.5: Inicialização do ambiente com alvo e agente fixos

Concluíu-se que com aproximadamente 30 épocas o agente aprendeu a chegar até o objetivo, como pode ser visto nos gráficos (Figuras 4.6 e 4.7). Na tabela 4.2 pode-se verificar os resultados.

Importante ressaltar que a diferença de recompensa entre os gráficos 4.6 e 4.7 acontece porque o agente quando feita a validação tem até 2000 steps para executar, se ele não acertar o alvo, os 2000 passos serão dados, caso contrário, se ele chegar ao objetivo, menos passos serão dados, e a recompensa será maior. Neste caso, não faria sentido transferir o aprendizado de modelo treinado numa posição fixa para posições aleatórias, já que isso só retardaria a aprendizagem.

A validação do agente parte do mesmo ponto até onde foi treinado o agente na determinada época.

Tabela 4.2: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente com alvo e agente fixos

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Agente e ambiente fixos	13.26	95.37	2.60	78.45	21.19	-176.27	71.7

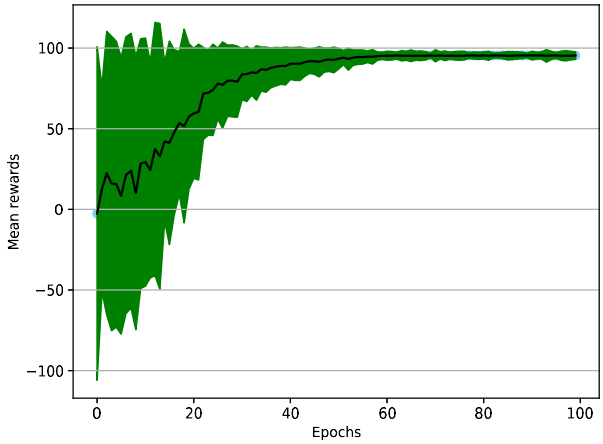

Figura 4.6: Média e desvio padrão das recompensas do treinamento do ambiente com alvo e agente fixos

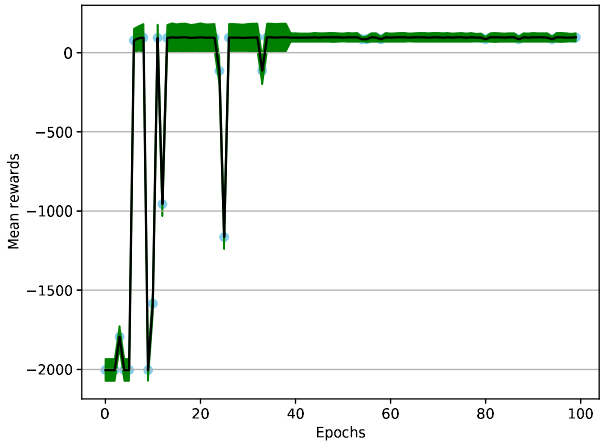

Figura 4.7: Média e desvio padrão das recompensas da validação do ambiente com alvo e agente fixos

4.2.2
Simulação inicializada com agente fixo e esfera em posições aleatórias

O cenário que foi utilizado no experimento anterior, nesta seção será treinado com uma inicialização diferente, com agente fixo e esfera em posições aleatórias (Figura 4.8), afim de generalizar mais o modelo.

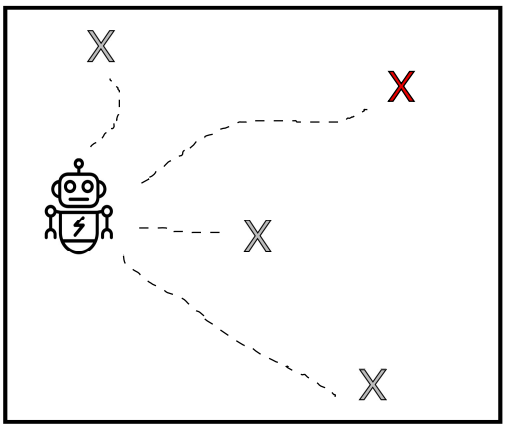

Figura 4.8: Inicialização do ambiente com alvo aleatório e agente fixo

Os resultados apresentados realmente foram melhores do que os obtidos no treino com inicialização de objeto e agente fixos, uma vez que generalizou o aprendizado praticamente no mesmo tempo de treinamento e número de épocas (30), como visto na tabela 4.3 e nos gráficos (Figuras 4.9 e 4.10). Além disso, ao se utilizar o modelo já treinado em 4.2.1, e modificando o ambiente de teste, para que a esfera seja colocada em outra posição, verificou-se que o agente seguia a mesma rota de onde estava a esfera no treinamento, nunca chegando até o objetivo.

Tabela 4.3: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente com alvo aleatório e agente fixo

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Alvo aleatório Agente fixo	13.60	94.46	2.95	74.56	24.67	-93.98	47.86

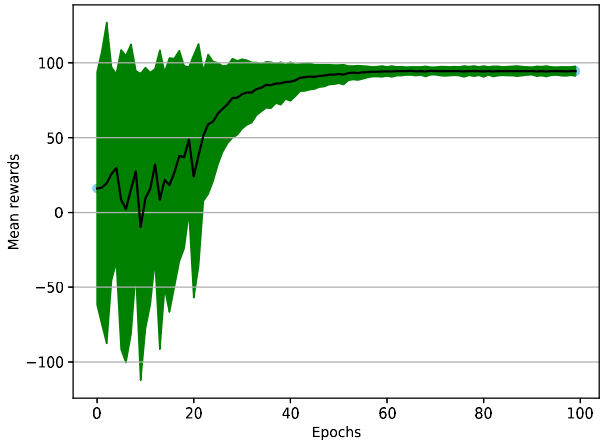

Figura 4.9: Média e desvio padrão das recompensas do treinamento do ambiente com alvo aleatório e agente fixo

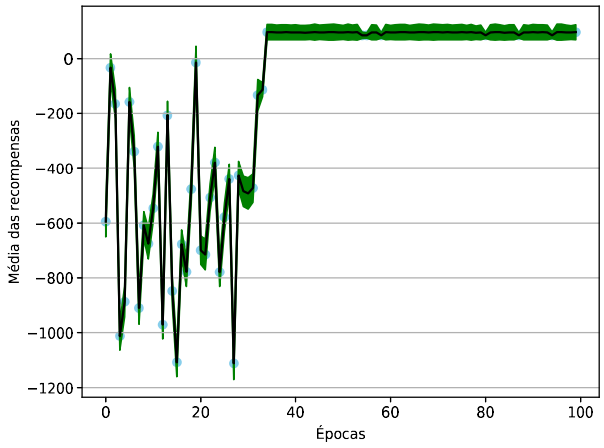

Figura 4.10: Média e desvio padrão das recompensas da validação do ambiente com alvo aleatório e agente fixo

4.2.3
Simulação inicializada com agente e esfera em posições aleatórias

For fim foi treinado um ambiente com inicializações aleatórias do agente e objetivo (Figura 4.11). Este foi o que apresentou maior capacidade de generalização do modelo, pois o agente era capaz de alcançar o objetivo em qualquer lugar que fosse colocado. Além disso, conluiu-se ser esta a melhor inicialização, pois a diferença de épocas - necessárias para o agente aprender - entre os dois primeiros experimentos feitos anteriormente é pequena, uma vez que neste experimento foram nessárias aproximadamente 40 épocas para para o agente ser treinado, como pode ser observado nos gráficos (Figuras 4.12 e 4.13) e na Tabela 4.12.

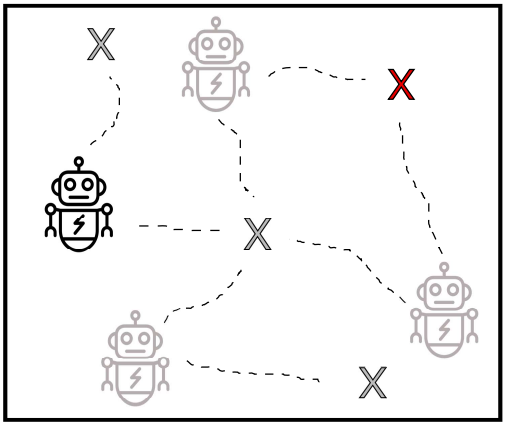

Figura 4.11: Inicialização do ambiente com alvo e agente aleatórios

Tabela 4.4: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente com alvo e agente aleatórios

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Alvo e agente aleatórios	18.02	94.43	5.93	56.49	45.53	-115.25	99.94

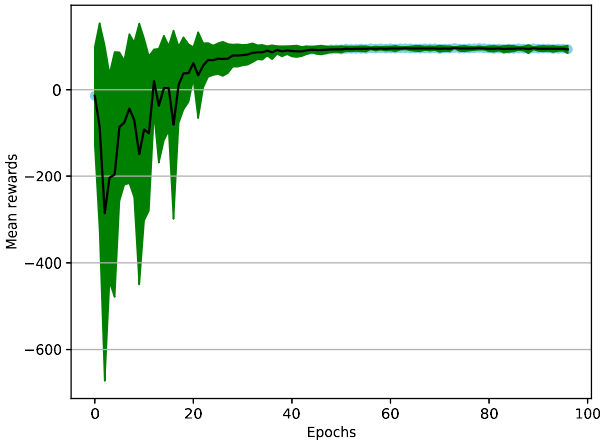

Figura 4.12: Média e desvio padrão das recompensas do treinamento do ambiente com alvo e agente aleatórios

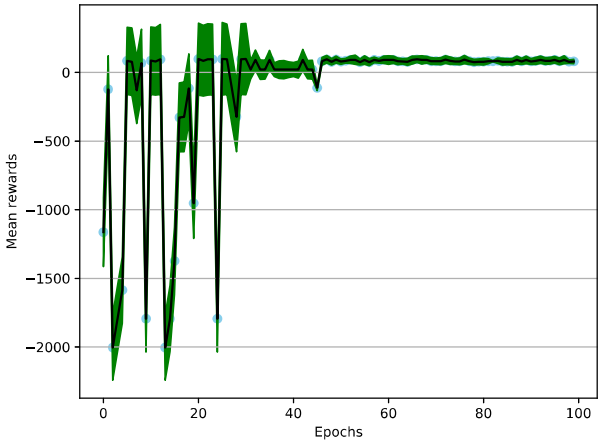

Figura 4.13: Média e desvio padrão das recompensas da validação do ambiente com alvo e agente aleatórios

Após se ter verificado a melhor inicialização da simulação, os próximos testes gerados seguindo ela, ou seja, utilizando agente e alvos aleatórios. Além disso, como os resultados foram bons, resolveu-se modificar o objeto, afim de verificar se os resultados seriam semelhantes. Nesse caso, um ambiente contendo uma caixa (*"medikit"*) como objetivo foi criado Figura 4.14 e então treinado com inicialização aleatória do agente e objeto. Os resultados exibidos na tabela 4.5 e gráficos (Figura 4.15 e 4.16) provaram ser parecidos com os do cenário treinados na seção anterior, porém com um *rise time* mais longo, demorando aproximadamente 45 épocas para ser treinado, uma vez que a caixa é um objeto um pouco menor que a esfera e pela sua cor, se destaca menos no ambiente.

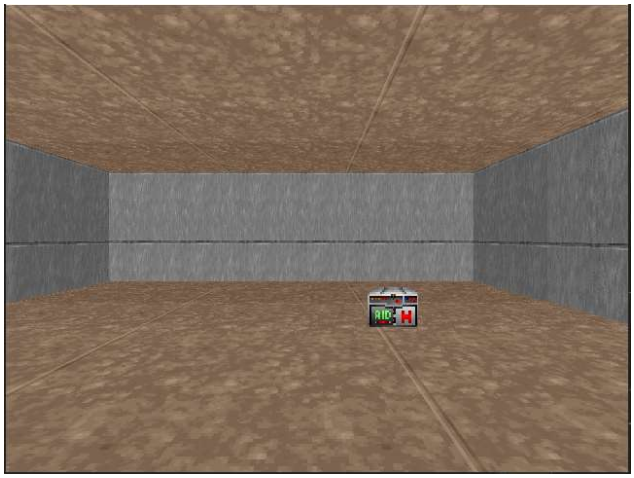

Figura 4.14: Ambiente com *'medikit'* como objetivo

Tabela 4.5: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente com *'medikit'* como objetivo

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
caixa ('medikit')	21.42	90.16	11.47	11.56	87.50	-514.06	218.59

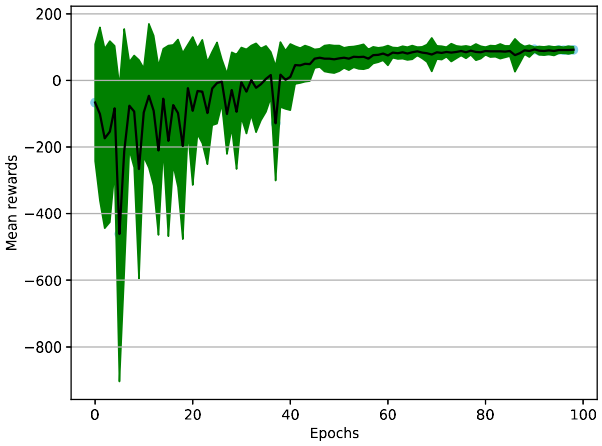

Figura 4.15: Média e desvio padrão das recompensas do treinamento do ambiente com *'medikit'* como objetivo

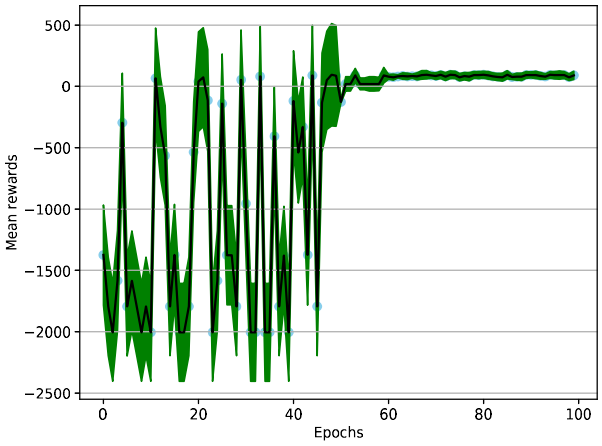

Figura 4.16: Média e desvio padrão das recompensas da validação do ambiente com *'medikit'* como objetivo

Visto os resultados, foi utilizado *transfer learning*, uma vez que o agente utilizando a esfera como alvo fora treinado em um ambiente muito semelhante

ao atual cenário, havendo nesse caso uma transferência de aprendizagem para uma nova tarefa. Os resultados podem ser vistos na próxima seção.

4.2.4
Transferência de aprendizagem para nova tarefa

Para a investigação da transferência de aprendizado, diferentes testes foram realizados. No primeiro teste a rede já treinada pode ter seus pesos atualizados, no segundo os pesos das duas camadas convolutivas foram congelados (Figura 4.21) e no terceiro teste foram congelados apenas os pesos da primeira camada convolutiva (Figura 4.26). O congelamento dos pesos nas camadas faz com que os mesmos não sejam retreinados. Em todos os esperimentos foi utilizada como rede pré-treinada a rede da seção 4.2.3 para que assim fosse possível treinar um ambiente cujo objetivo é alcançar uma caixa, seguindo o processo descrito acima.

4.2.4.1
Treinamento carregando todos os pesos da rede já treinada

Como neste experimento todos os pesos da rede foram reaproveitados e nenhum foi congelado, os resultados apresentaram maior diferença na validação, uma vez que o agente já começa o treinamento com algum conhecimento, recebendo maiores recompensas desde o início do treinamento, como pode ser observado na tabela 4.6 e gráficos (Figuras 4.17, 4.18, 4.19 e 4.20) a seguir.

Tabela 4.6: Tabela comparando valores das médias das recompensas e desvio padrão (DP) do treinamento sem utilização de *transfer learning* e com transfer learning da transferência de objetivos utilizando todos os pesos da rede já treinada

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Caixa ('medikit') sem TL	21.42	90.16	11.47	11.56	87.50	-514.06	218.59
Caixa ('medikit') com TL do ambiente esfera	18.02	91.25	5.59	48.51	47.05	-52.04	110.24

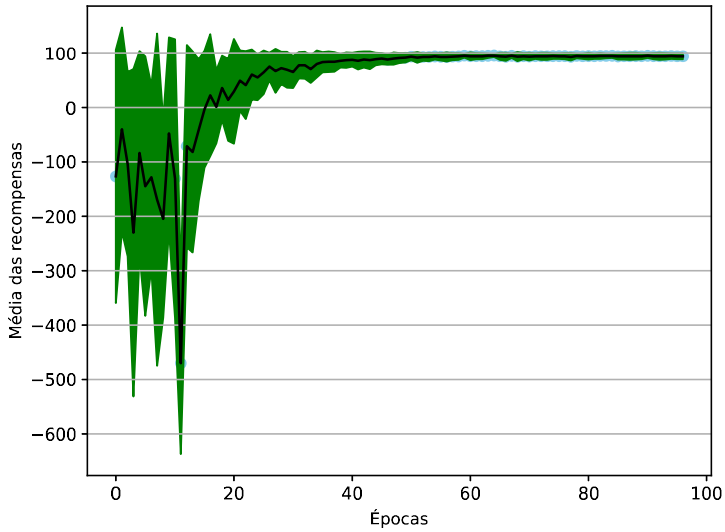

Figura 4.17: Média e desvio padrão das recompensas do treinamento utilizando *Transfer learning* com todos os pesos da rede

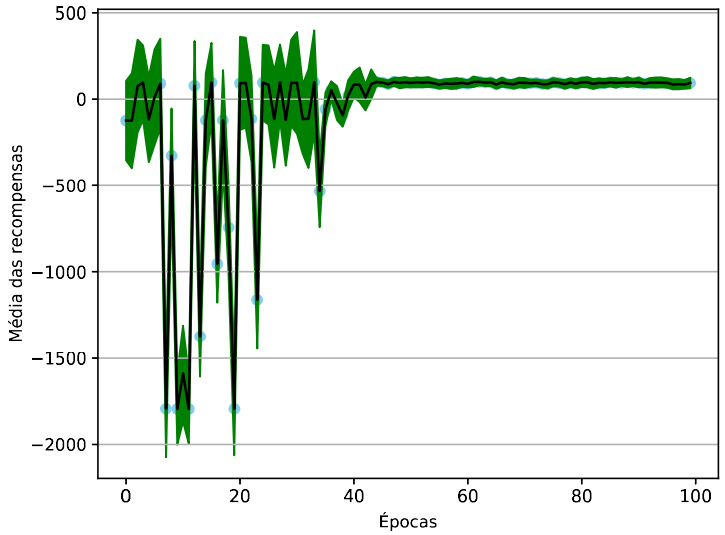

Figura 4.18: Média e desvio padrão das recompensas da validação utilizando *Transfer learning* com todos os pesos da rede

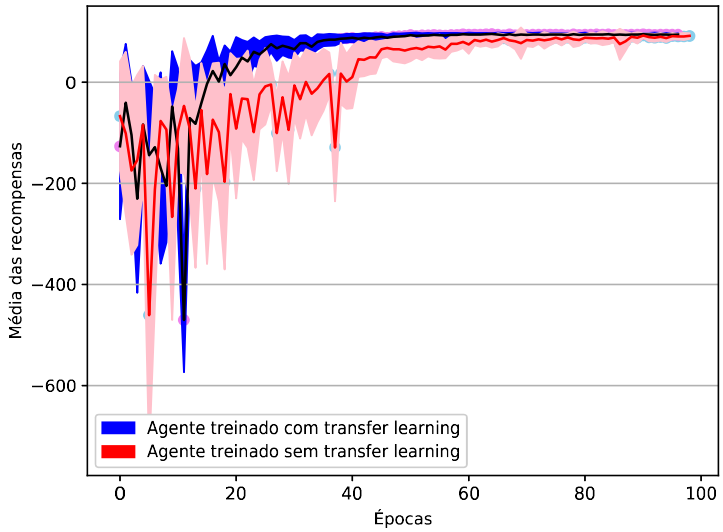

Figura 4.19: Comparação do treinamento da rede com e sem *transfer learning* do ambiente esfera - caixa, exibindo média com intervalo de confiança das recompensas

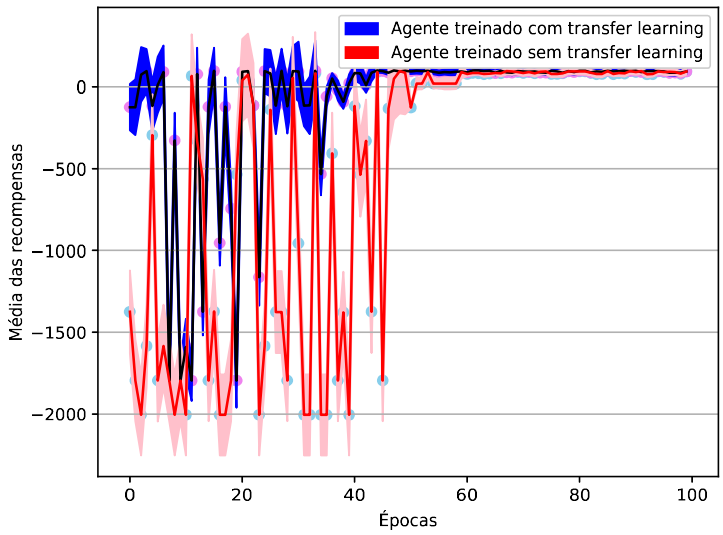

Figura 4.20: Comparação da validação da rede com e sem *transfer learning* do ambiente esfera - caixa, exibindo média com intervalo de confiança das recompensas

4.2.4.2 Transferência de aprendizagem utilizando duas camadas convolucionais congeladas

Neste experimento somente os pesos das camadas *fully connected* foram ajustados (Figura 4.21), ou seja, nenhuma característica nova de cada frame foi

utilizada no treino, pois todas as camadas convolucionais estão com seus pesos congelados, não sendo permitido os treinar novamente. Como os ambientes eram praticamente os mesmos, mudando apenas o objetivo (esfera - caixa), mesmo sem treinar as camadas convolucionais os resultados acabaram não sendo ruins, superando as expectativas, como é visto nas figuras 4.22, 4.23, 4.24 e 4.25 e Tabela 4.7.

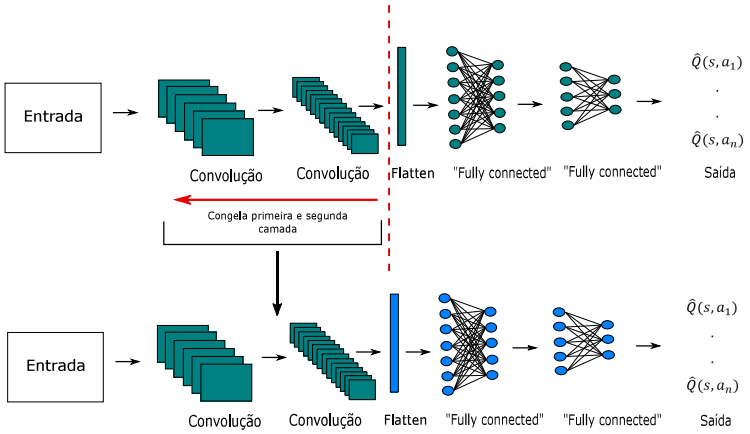

Figura 4.21: *Transfer learning* utilizando a primeira e a segunda camada já treinada, congelando o peso das duas camadas

Tabela 4.7: Tabela comparando valores das médias das recompensas e desvio padrão (DP) do treinamento sem e com utilização de *transfer learning* utilizando 2 camadas convolucionais congeladas

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Caixa ('medikit') sem TL	21.42	90.16	11.47	11.56	87.50	-514.06	218.59
Caixa ('medikit') com TL do ambiente esfera	21.76	67.29	30.28	7.23	95.24	13.81	24.95

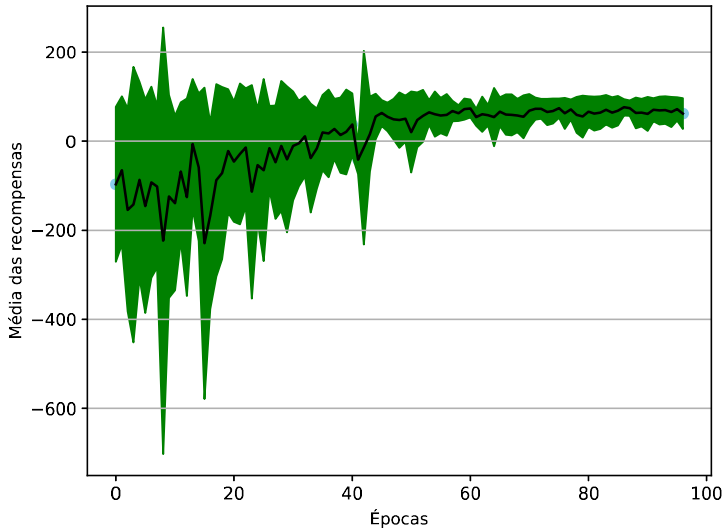

Figura 4.22: Média e desvio padrão das recompensas do treinamento com *Transfer learning* congelando a primeira e a segunda camada já treinada

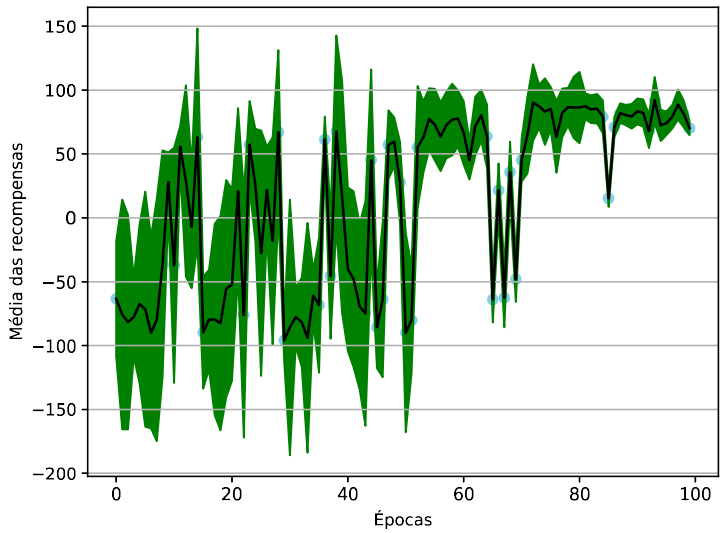

Figura 4.23: Média de desvio padrão das recompensas da validação da transferência de objetivos utilizando apenas a primeira camada

Figura 4.24: Comparação do treinamento da rede com e sem *transfer learning* (de duas camadas convolucionais), exibindo média com intervalo de confiança das recompensas

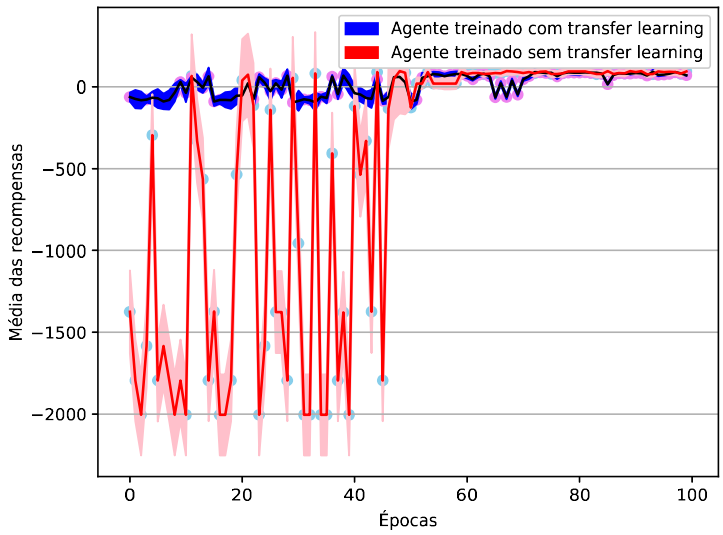

Figura 4.25: Comparação da validação da rede com e sem *transfer learning* (de duas camadas convolucionais), exibindo média com intervalo de confiança das recompensas

4.2.4.3 Transferência de aprendizagem utilizando a primeira camada convolucional congelada

Por último foi treinado o experimento utilizando apenas a primeira camada convolucional no *transfer learning* (Figura 4.26). Os resultados se

apresentaram melhores do que os anteriores, apesar de não haver muita diferença no *rise time*, entre o treinamento sem e com *transfer learning*, as recompensas recebidas desde o começo do treinamento foram maiores do que sem utilizar *transfer learning*, como pode ser observado nas figuras 4.27, 4.28, 4.29 e 4.30 e na Tabela 4.8.

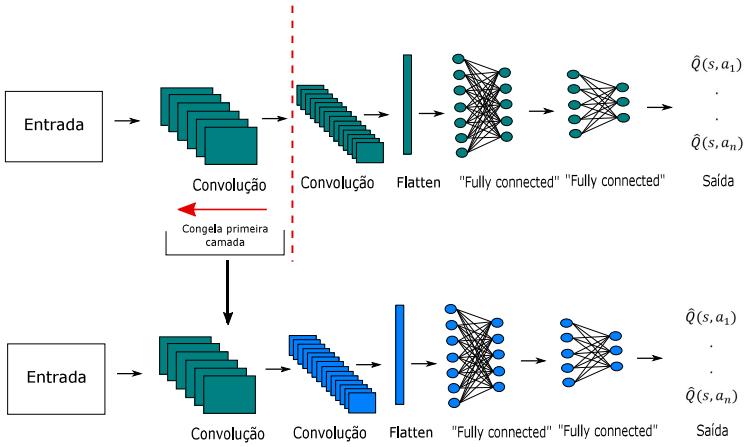

Figura 4.26: *Transfer learning* utilizando a primeira camada já treinada, congelando os pesos da primeira camada

Tabela 4.8: Tabela comparando valores das médias das recompensas e desvio padrão (DP) do treinamento sem utilização de *transfer learning* e com *transfer learning* utilizando 1 camada convolucional congelada

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Caixa ('medikit') sem TL	21.42	90.16	11.47	11.56	87.50	-514.06	218.59
Caixa ('medikit') com TL do ambiente esfera	13.26	92.62	5.67	35.38	64.43	52.04	16.52

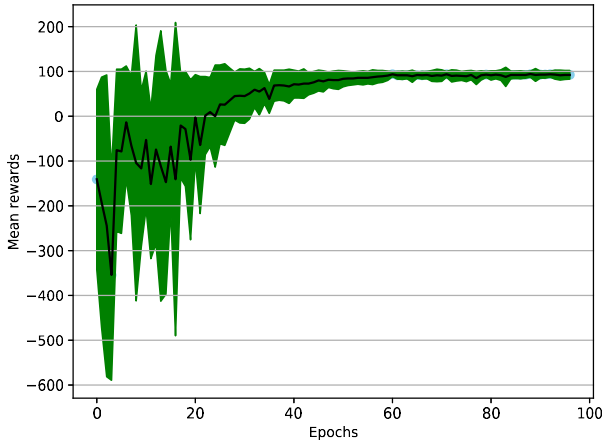

Figura 4.27: Média e desvio padrão das recompensas do treinamento com *Transfer learning* congelando a primeira camada convolucional já treinada

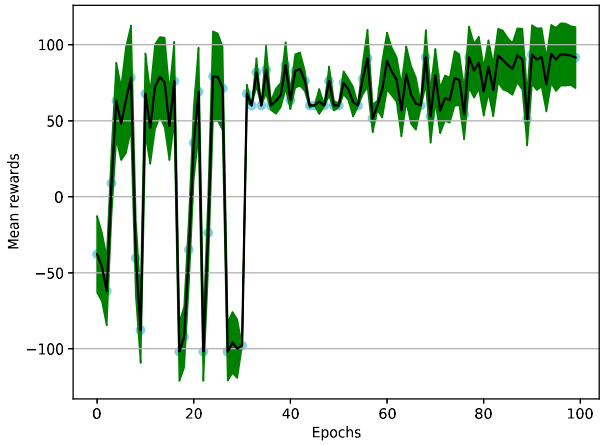

Figura 4.28: Média e desvio padrão das recompensas da validação com *Transfer learning* congelando a primeira camada convolucional já treinada

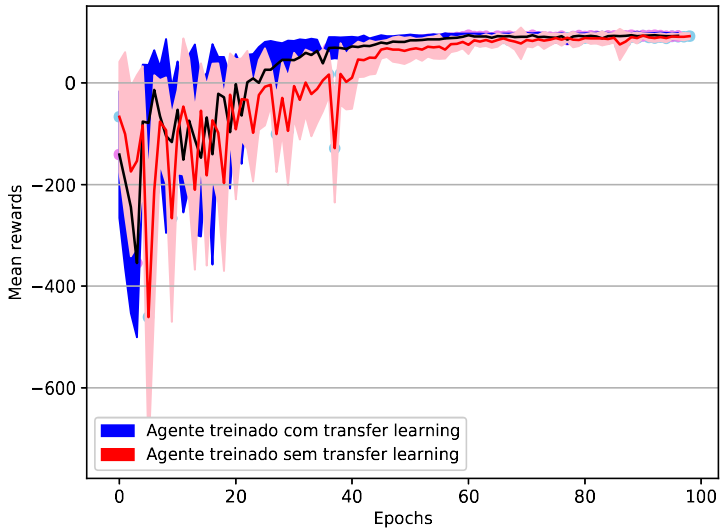

Figura 4.29: Comparação do treino da rede com e sem *transfer learning* (utilizando uma camada convolucional), exibindo média com intervalo de confiança das recompensas

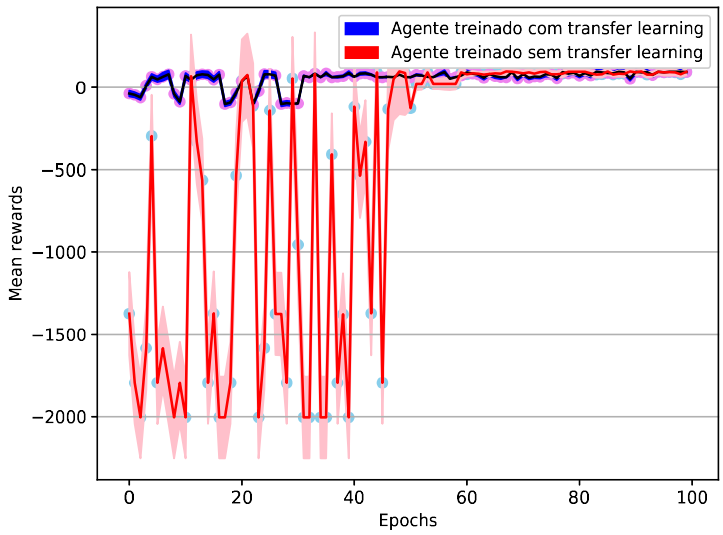

Figura 4.30: Comparação da validação da rede com e sem *transfer learning* (utilizando uma camada convolucional), exibindo média com intervalo de confiança das recompensas

Após esses testes, foi concluído que a melhor parametrização para se utilizar o *transfer learning* nos experimentos desse trabalho seria empregar apenas a primeira camada congelada, uma vez que os resultados foram melhores, pois foram necessárias menos épocas para o agente ser treinado.

4.3 DQN em um ambiente com barreiras

Dado que os resultados anteriores foram promissores, a complexidade do ambiente foi aumentada, colocando obstáculos no mesmo. O cenário é muito semelhante ao que foi utilizado nos testes anteriores, com apenas a diferença de ter uma barreira entre o agente e o objetivo (Figura 4.31).

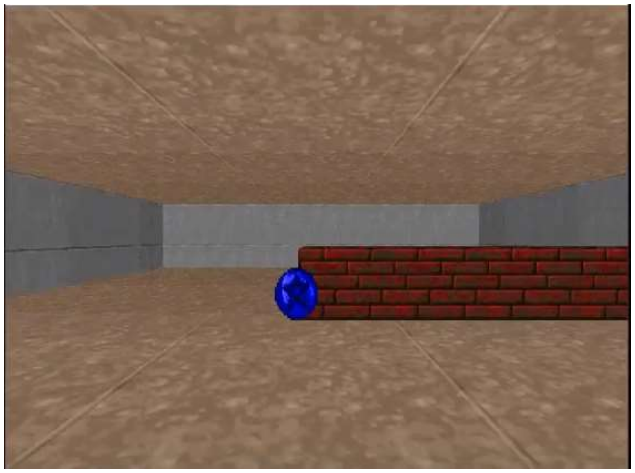

Figura 4.31: Ambiente simples com 'esfera' e barreira

Os resultados do treinamento dessa rede condizeram com o esperado, pois como o ambiente era mais complicado para o agente, era de se esperar que o mesmo demorasse a aprender a chegar ao alvo. No caso foram necessárias aproximadamente 72 épocas para que isso ocorresse, como pode ser visto nas figuras 4.32 e 4.33 e Tabela 4.9)

Tabela 4.9: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente com 'esfera' e barreira, exibindo média com intervalo de confiança das recompensas

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Esfera/Barreira	24.14	85.06	17.34	-38.75	114.13	-715.76	100.59

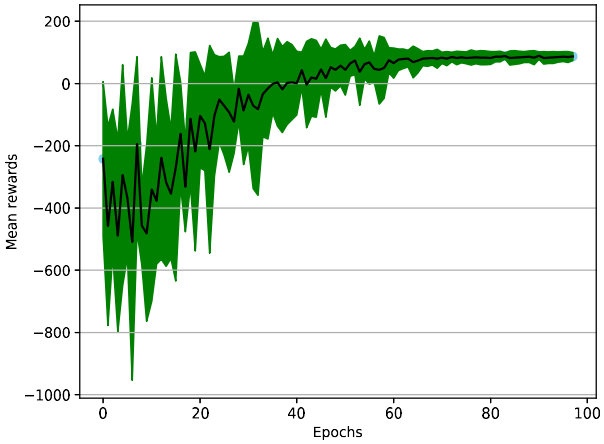

Figura 4.32: Média e desvio padrão das recompensas do treinamento do ambiente com 'esfera' e barreira

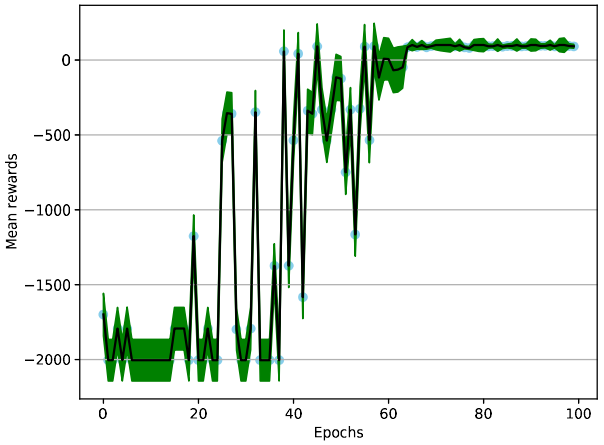

Figura 4.33: Média e desvio padrão das recompensas da validação do ambiente com 'esfera' e barreira

Do mesmo modo que foi realizado no experimento 4.2.4.3, houve a intenção de utilizar os pesos das redes já treinadas no ambiente mais simples, para verificar se o treinamento do cenário com barreira poderia ser mais eficiente, então novamente foi utilizado *transfer learning*, como pode ser conferido abaixo.

4.3.1 Transferência de aprendizagem para ambiente novo

Os pesos da rede treinada no experimento utilizando a esfera como objetivo foram carregados (4.2.4.3) e apenas os pesos da primeira camada convolucional foram congelados.

Após o treinamento, concluiu-se que os resultados foram melhores do que os sem utilizar *transfer learning*, o *rise time* foi um pouco menor e como no experimento 4.2.4.3, as médias das recompensas na validação foram maiores. Algo importante a dizer é que no *transfer learning* apesar do agente aprender mais rápido, as recompensas finais são um pouco mais baixas do que quando não se utiliza TL. Os resultados são melhor observados nas figuras 4.34, 4.35, 4.36 e 4.37 e na Tabela 4.10.

Tabela 4.10: Tabela comparando valores das médias das recompensas e desvio padrão (DP) do treinamento sem utilização de *transfer learning* e com transfer learning da transferência de aprendizado entre ambientes

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Esfera/Barreira sem TL	24.14	85.06	17.34	-38.75	114.13	-715.76	100.59
Esfera/Barreira com TL do ambiente esfera	20.10	81.94	23.34	-55.05	118.88	-5.2	39.63

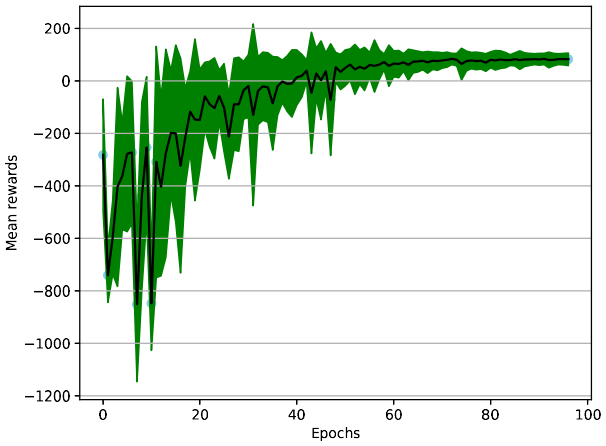

Figura 4.34: Média e desvio padrão das recompensas do treinamento utilizando *Transfer learning* entre os ambientes contendo esfera e do contendo esfera e barreira

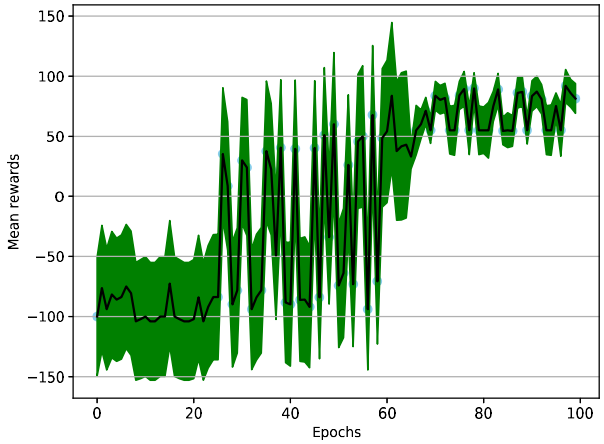

Figura 4.35: Média e desvio padrão das recompensas da validação utilizando *Transfer learning* entre os ambientes contendo esfera e do contendo esfera e barreira

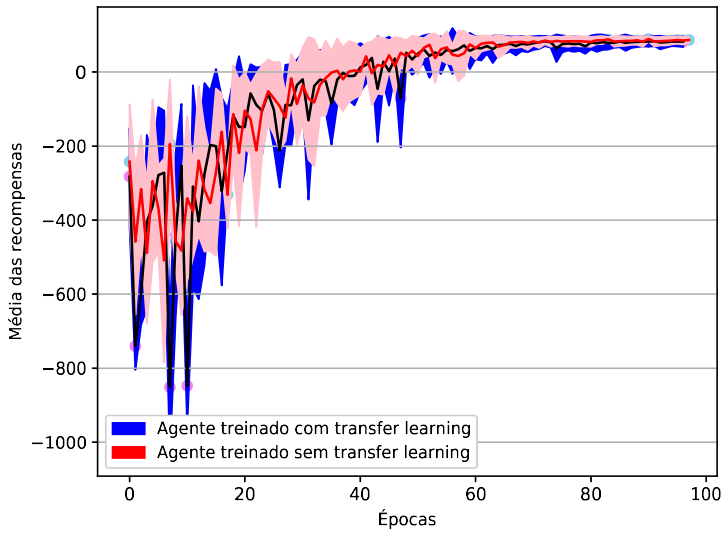

Figura 4.36: Comparação do treinamento da rede com e sem *transfer learning* entre os ambientes contendo esfera e do contendo esfera e barreira, exibindo média com intervalo de confiança das recompensas

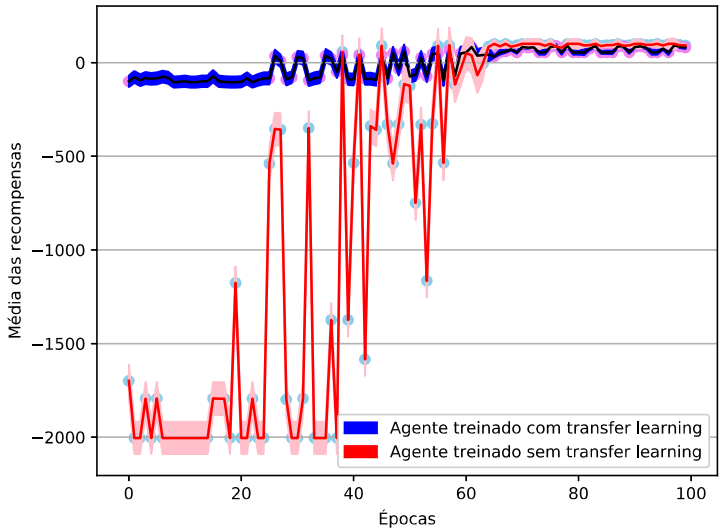

Figura 4.37: Comparação da validação da rede com e sem *transfer learning* entre os ambientes contendo esfera e do contendo esfera e barreira, exibindo média com intervalo de confiança das recompensas

4.4 DQN em um ambiente com portas

Afim de preparar os testes para o ambiente real simulado, foram criados ambientes com maiores diferenças, como por exemplo o ambiente utilizado neste experimento, onde o objetivo do agente é chegar até uma porta, como visto na figura 4.38.

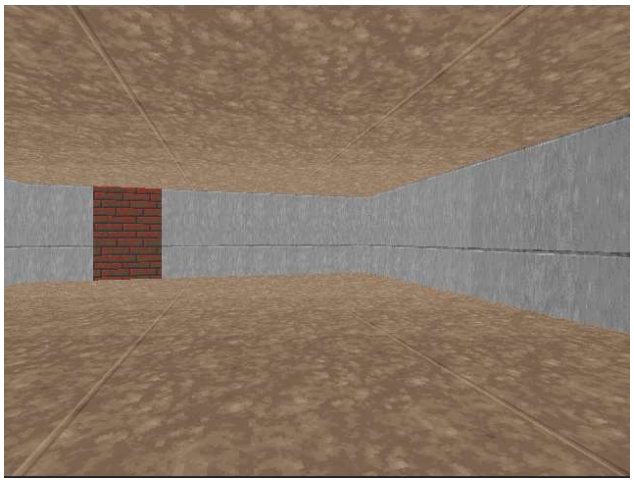

Figura 4.38: Ambiente com porta como objetivo

O agente foi treinado para essa nova tarefa e os resultados (Figuras 4.39, 4.40 e Tabela 4.11) novamente foram satisfatórios. Foram necessárias 50 épocas para o agente ser treinado.

Tabela 4.11: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente com porta como objetivo

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Porta	15.64	1.41	1.64	-11.47	5.69	-132.83	59.66

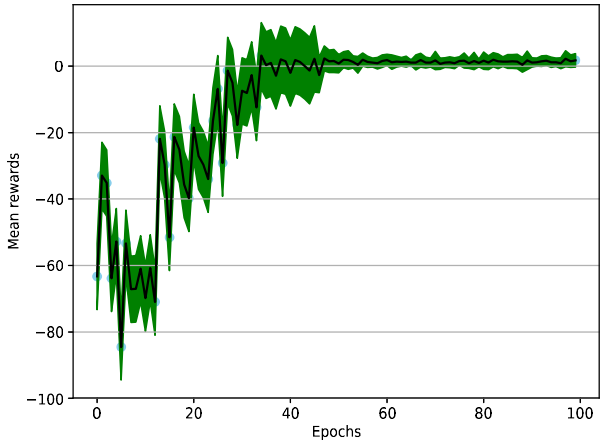

Figura 4.39: Média e desvio padrão das recompensas do treinamento do ambiente com porta como objetivo

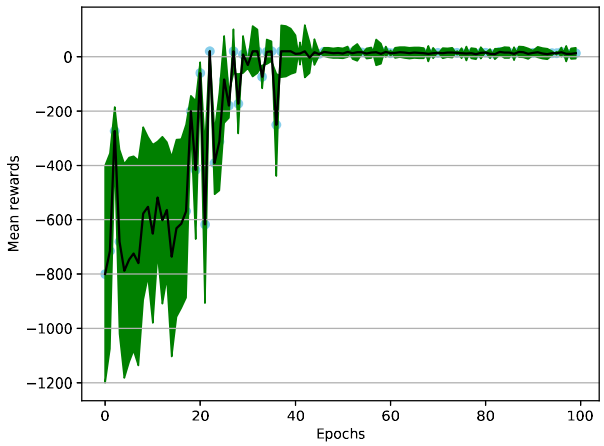

Figura 4.40: Média e desvio padrão das recompensas da validação do ambiente com porta como objetivo

Com o intuito de utilizar texturas que não fossem do Doom, o ambiente foi segmentado manualmente (Figura 4.41) e, então, treinado para conferir como o agente se comportaria.

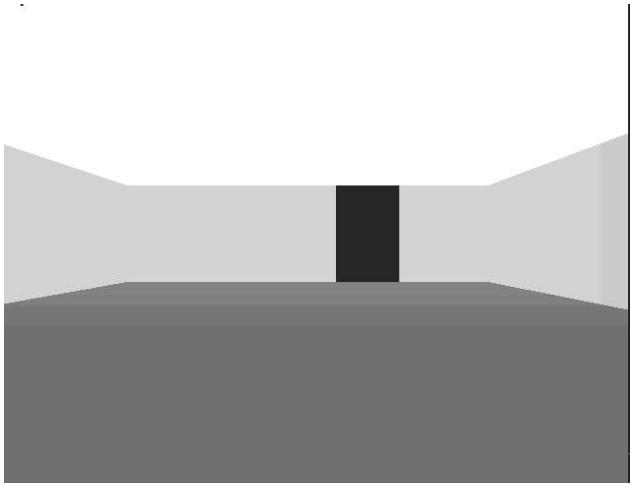

Figura 4.41: Ambiente segmentado com porta como objetivo

Os resultados são dados a seguir (Figuras 4.42 e 4.43 e Tabela 4.12) e se pode observar que como as texturas utilizadas são bem mais simples, o agente aprende mais rápido (com aproximadamente 40 épocas), apesar dos cenários terem o mesmo objetivo (a porta).

Tabela 4.12: Tabela com valores das médias das recompensas e desvio padrão (DP) do treinamento e validação do ambiente segmentado com porta como objetivo

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Porta segmentada	14.96	1.09	1.99	-12.66	5.50	-155.29	81.18

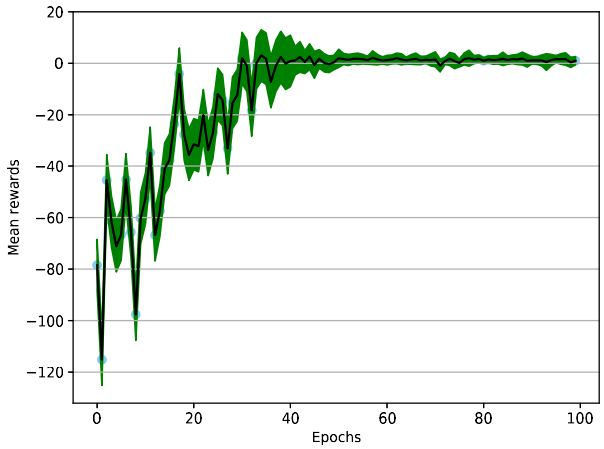

Figura 4.42: Média e desvio padrão das recompensas do treinamento do ambiente segmentado com porta como objetivo

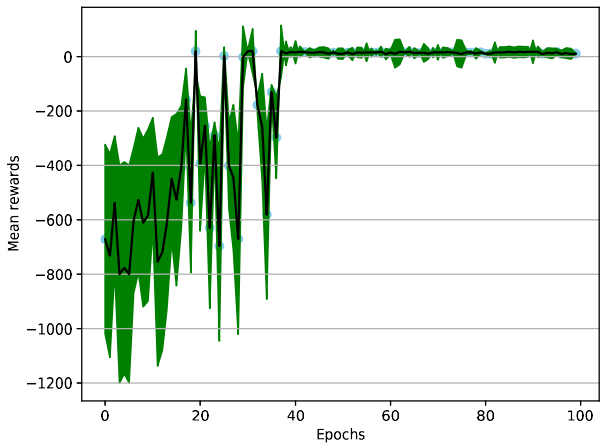

Figura 4.43: Média e desvio padrão das recompensas da validação do ambiente segmentado com porta como objetivo

Já que os ambientes são os mesmos, porém com aparência diferente - um ambiente segmentado e o outro não - resolveu-se testar se a rede treinada no cenário segmentado, tendo texturas mais simples, serviria no cenário original, tendo-se realizado, então, o experimento a seguir.

4.4.1 Transferência de aprendizagem para ambiente novo com textura diferente

O ambiente segmentado tem texturas mais simples, sem detalhes, ao contrário do ambiente não segmentado, o que causou uma diferença no aprendizado do agente, como pode ser analizado nos resultados (Figuras 4.44, 4.45, 4.46 e 4.47 e Tabela 4.13) provaram que a textura faz diferença no aprendizado,

uma vez que o *transfer learning* de um ambiente com textura mais simples para um cenário com textura mais complexa resultou em um treino mais rápido, fazendo com o que o agente aprenda apenas em 42 épocas.

Tabela 4.13: Tabela comparando valores das médias das recompensas e desvio padrão (DP) do treinamento sem utilização de *transfer learning* e com transfer learning da transferência de aprendizado entre ambientes com porta segmentada e não segmentada

Modelo	*Rise Time (s)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Porta sem TL	15.64	1.41	1.64	-11.47	5.69	-132.83	59.66
Porta com TL do ambiente Porta segmentada	11.02	1.24	1.92	-2.02	6.75	-98.24	71.44

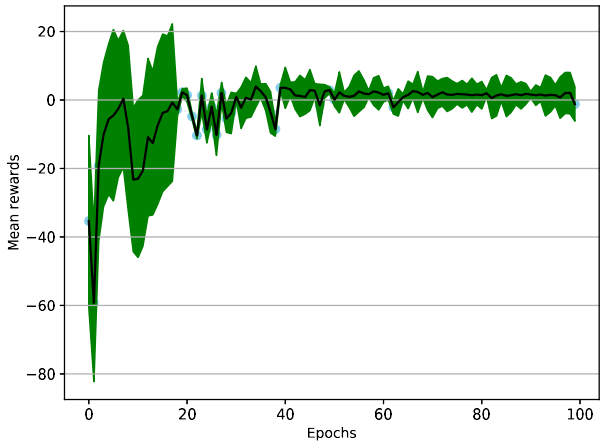

Figura 4.44: Média e desvio padrão das recompensas do treinamento da transferência de aprendizado entre ambientes com porta segmentada e não segmentada

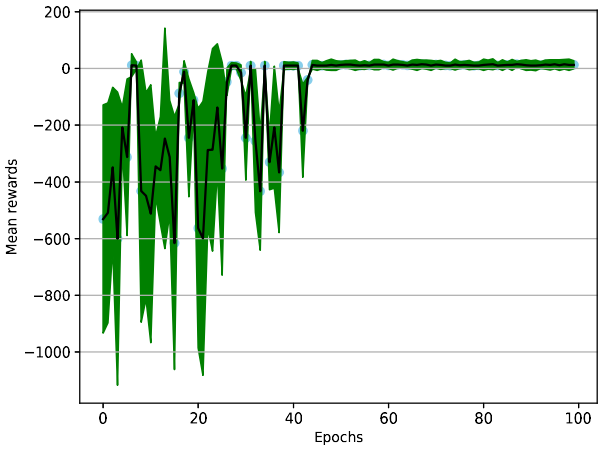

Figura 4.45: Média e desvio padrão das recompensas da validação da transferência de aprendizado entre ambientes entre ambientes com porta segmentada e não segmentada

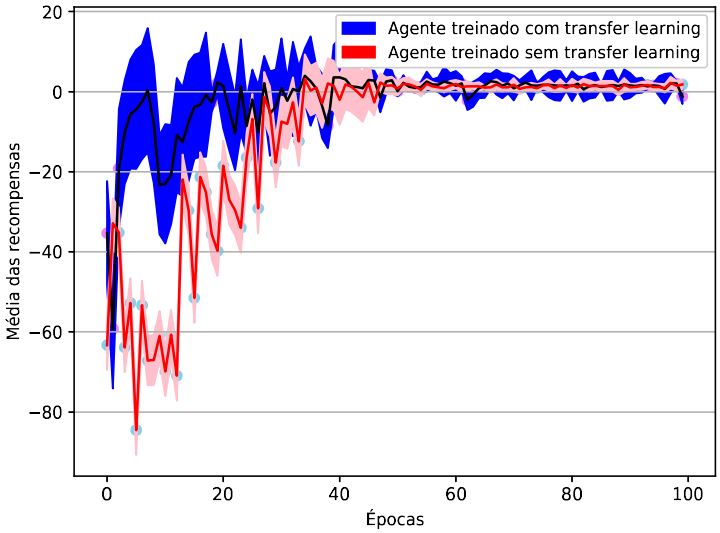

Figura 4.46: Comparação do treinamento da rede com e sem *transfer learning* entre ambientes com porta segmentada e não segmentada, exibindo média com intervalo de confiançadas recompensas

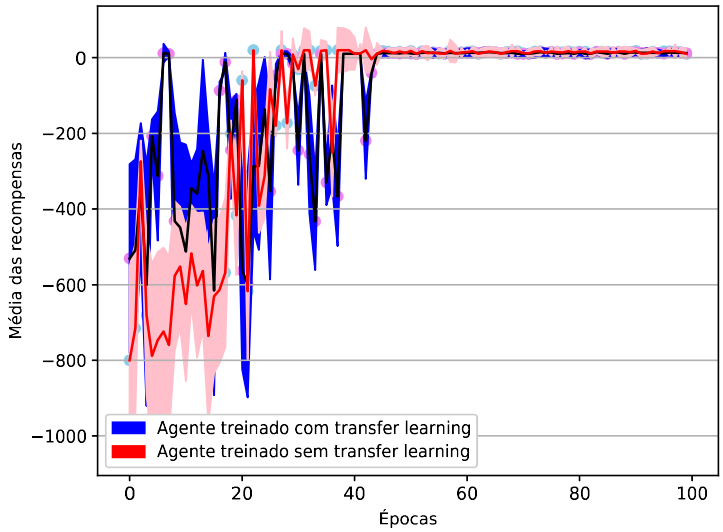

Figura 4.47: Comparação da validação da rede com e sem *transfer learning* entre ambientes com porta segmentada e não segmentada, exibindo média com intervalo de confiançadas recompensas

4.5 DQN em um ambiente real simulado

Como os resultados anteriores apresentaram-se muito bons, um ambiente mais complexo e similiar ao real foi utilizado (*Realistic Rendering* (44)). foi treinado. Ele apresenta muitas dificuldades, como móveis e plantas como exibido na Figura 3.2. O objetivo deste ambiente é encontrar uma porta, porém neste cenário existem três iguais, das quais apenas uma é a correta. Abaixo estão os resultados do treinamento.

Como o ambiente era muito mais complexo, foram necessárias mais épocas para o treinamento. No caso deste experimento foram necessárias aproximadamente 6255 épocas para o agente aprender.

Tabela 4.14: Tabela com valores das médias e desvio padrão (DP) das recompensas do treinamento e validação do ambiente *Realistic Rendering*

Modelo	*Rise Time (h)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Realistic Rendering	11.55	8.70	1.45	1.78	1.39	-0.91	1.66

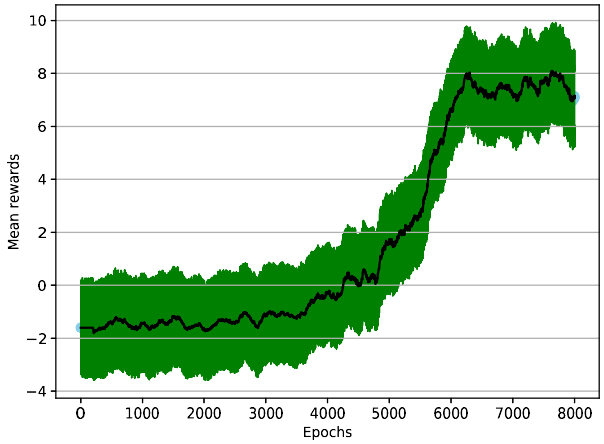

Figura 4.48: Média e desvio padrão das recompensas do treinamento do ambiente *Realistic Rendering*

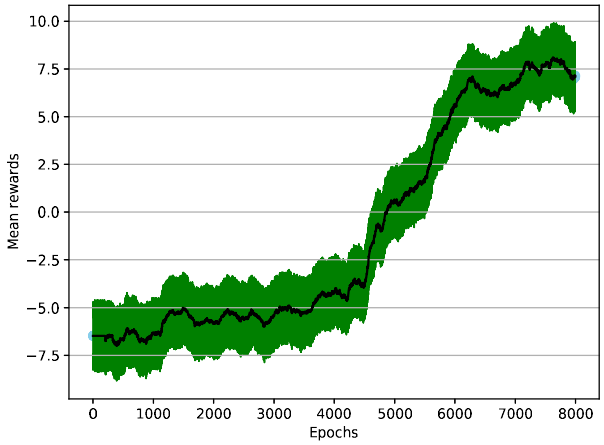

Figura 4.49: Média e desvio padrão das recompensas da validação o treinamento do ambiente *Realistic Rendering*

4.5.1 *Transfer learning* entre um ambiente simples e um ambiente real simulado utilizando segmentação

Devido ao ambiente ser mais complexo, o treinamento sem *transfer learning* demorou mais, portanto novamente foi utilizado *transfer learning* afim de diminuir o esforço computacional do treinamento. Para que isso fosse possível foi criado um ambiente (Figura 4.50) no VizDoom imitando alguns aspectos do cenário do *Realistic Rendering*, o mesmo possui segmentação semântica, uma vez que no experimento 5.4.1 houve um bom desempenho da rede. Os resultados do treinamento desse ambiente podem ser conferidos a seguir (Figuras 4.51 e 4.52 e tabela 4.15).

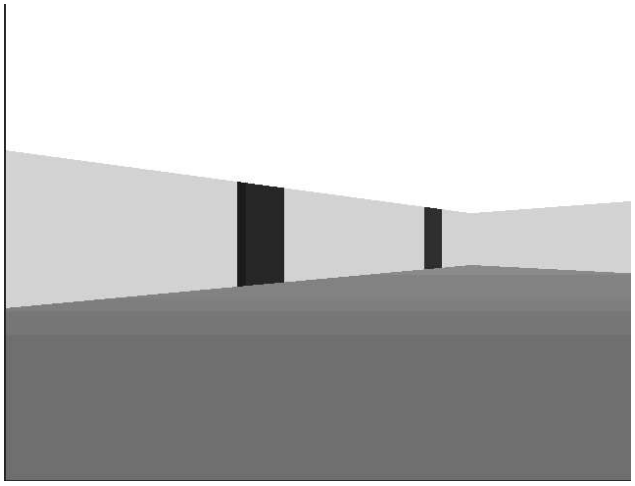

Figura 4.50: Ambiente segmentado inspirado no *Realistic Rendering*

Tabela 4.15: Tabela com valores das médias e desvio padrão (DP) das recompensas do treinamento e validação do ambiente com três portas

Modelo	*Rise Time (m)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Três portas	14.96	1.65	1.15	-2.15	6.46	-138.21	132.32

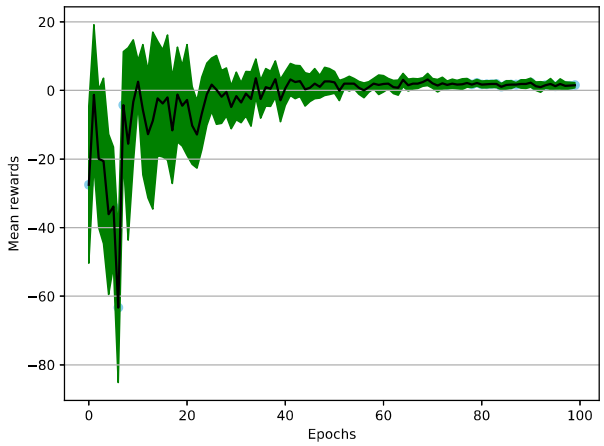

Figura 4.51: Média e desvio padrão das recompensas do treinamento do ambiente com três portas

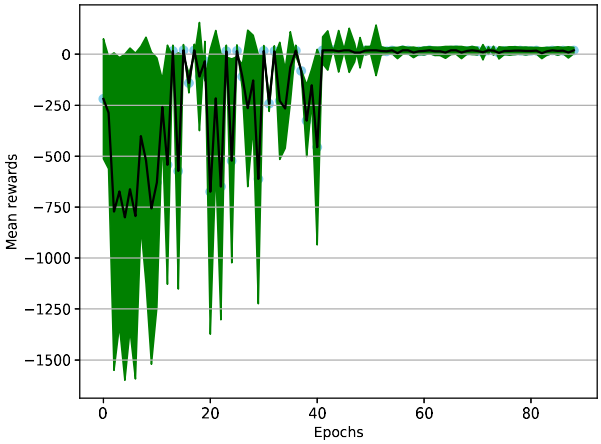

Figura 4.52: Média e desvio padrão das recompensas da validação do ambiente com três portas

Para que fosse possível a utilização de *transfer learning* a partir do ambiente treinado acima, as imagens do *Realistic Rendering* tiveram de ser segmentadas, para que as imagens ficassem parecidas com as que já foram treinadas, portanto a PSPNet, explicada no capítulo de *Deep Learning*, foi utilizada, a imagem da segmentação pode ser vista na Figura 4.53.

Figura 4.53: Imagens do ambiente *Realistic Rendering* antes e depois da segmentação

Os resultados deste *transfer learning* não foram bons como se esperava, isto ocorreu devido às imagens do pré-processamento, pois como nessa etapa foi usada a segmentação pela PSPNet, os resultados da segmentação em alguns

casos não foram corretos (Figura 4.54), principalmente quando havia pouca iluminação, desorientando o agente em diversos estados, assim o agente não conseguiu aprender, como pode ser visto nos resultados (Figuras 4.55 e 4.56 e Tabela 4.16).

Figura 4.54: Resultados ruins da segmentação pela PSPNet no ambiente *Realistic Rendering*

O que poderia ser feito nesse caso, seria um *transfer learning* da prórpia PSPNet, utilizando imagens segmentadas do *Realistic Rendering*, para que assim ela fosse treinada focando no ambiente desejado.

Tabela 4.16: Tabela comparando valores das médias e desvio padrão (DP) das recompensas do treinamento sem utilização de *transfer learning* e com transfer learning da transferência de aprendizado do ambiente segmentado com a rede PSPNet

Modelo	*Rise Time (h)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Realistic Rendering sem TL	11.55	8.70	1.45	1.78	1.39	-0.91	1.66
Realistic Rendering com segmentação PSPNet e TL do ambiente Porta segmentada	∞	-3.8	1.25	-4.01	0.74	-4.50	0.86

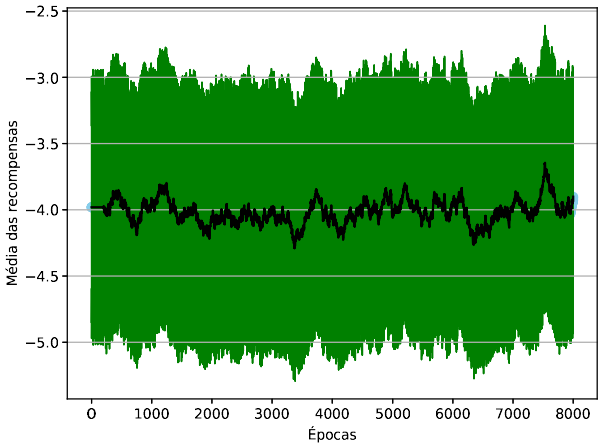

Figura 4.55: Média e desvio padrão das recompensas do treinamento do ambiente *Realistic Rendering* utilizando segmentação semântica da PSPNet

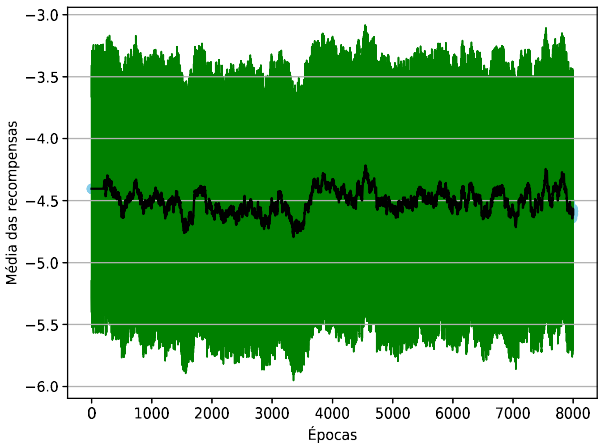

Figura 4.56: Média e desvio padrão das recompensas da validação do ambiente *Realistic Rendering* utilizando segmentação semântica da PSPNet

4.5.2 *Transfer learning* entre um ambiente simples e um ambiente real simulado utilizando segmentação padrão do próprio ambiente

Como os resultados anteriores não foram bons, optou-se por utilizar a segmentação do próprio ambiente, o qual o *Unreal* cria por padrão. Ou seja, ao invés de utilizar a PSPNet, foi utilizada a segmentação já dada pelo ambiente. Os resultados dessa vez foram animadores, apesar de o ganho não ser muito grande, como pode ser conferido nos gráficos (Figuras 4.57, 4.58, 4.59 e 4.60 e Tabela 4.17)

Tabela 4.17: Tabela comparando valores das médias e desvio padrão (DP) das recompensas do treinamento sem utilização de *transfer learning* e com transfer learning da transferência de aprendizado do ambiente segmentado utilizando a segmentação do próprio ambiente

Modelo	*Rise Time (h)*	Média Final Treino	DP. Final Treino	Média Treino	DP. Treino	Média Validação	DP. Validação
Realistic Rendering sem TL	11.55	8.70	1.45	1.78	1.39	-0.91	1.66
Realistic Rendering com segmentação do ambiente e TL do ambiente Porta segmentada	9.58	6.54	0.35	3.57	0.29	3.33	0.99

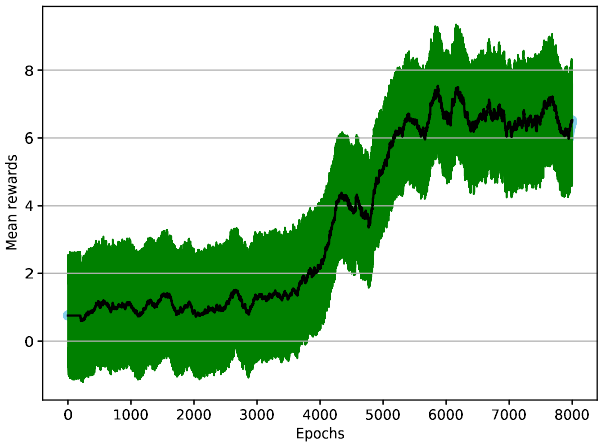

Figura 4.57: Média e desvio padrão das recompensas do treinamento utilizando *transfer learning* com o ambiente *Realistic Rendering*

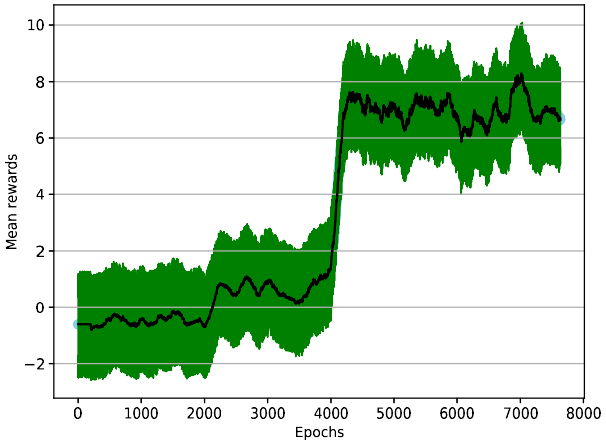

Figura 4.58: Média e desvio padrão das recompensas da validação o treinamento utilizando *transfer learning* com o ambiente *Realistic Rendering*

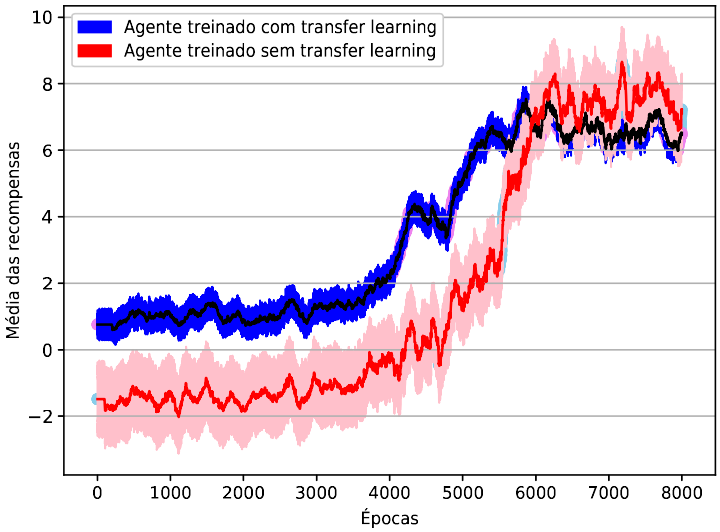

Figura 4.59: Comparação do treinamento da rede com e sem *transfer learning* do *Realistic Rendering*, exibindo média com intervalo de confiança das recompensas

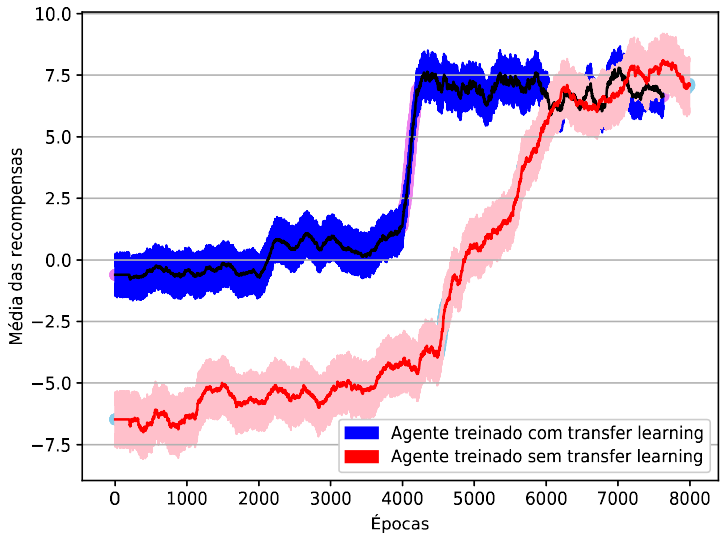

Figura 4.60: Comparação da validação da rede com e sem *transfer learning* do *Realistic Rendering*, exibindo média com intervalo de confiança das recompensas

Após esses experimentos, concluiu-se que no caso dos ambientes que tinham a porta como objetivo, o que o agente aprendeu foi a trajetória até a porta, diferente dos ambientes que ele tinha que pegar a esfera ou a caixa, nos quais ele realmente aprendeu que o objetivo era chegar até esses objetos onde quer que o agente estivesse.

5
Conclusões e Trabalhos futuros

Este trabalho apresentou um estudo sobre simulação de um robô autônomo interagindo com um ambiente através de uma câmera. Para isso foi utilizado o algoritmo de aprendizado profundo por reforço visual (*Deep Q-Learning*) e técnicas de transferência de conhecimento em redes convolutivas. O principal objetivo desta dissertação foi testar estes algoritmos e métodos de forma conjunta para simular o robô em diferentes ambientes e aproveitar a experiência em ambientes anteriores para poder alcançar os objetivos de maneira mais rápida em ambientes complexos.

A escolha de *Deep Q-Learning* como base da simulação se fez após realizada a etapa de levantamento bibliográfico e foi motivada pelo grande crescimento na área de *Deep Learning* que permite através de redes convolutivas o processamento rápido de imagens, depois de uma etapa de treinamento. Isto permite com que robôs guiados por câmera estendam cada vez mais sua autonomia e consigam superar limitações presentes em algoritmos de aprendizado por reforço tradicionais.

Por precisar de grande processamento computacional e muito tempo de treinamento as técnicas de transfer learning permitem que o robô consiga utilizar experiências de ambientes passados em novos ambientes mais complexos podendo cumprir seus objetivos de maneira mais rápida.

Nos primeiros experimentos foi testado qual seria a melhor inicialização para o agente a objetivo e foi concluído que inicializando os dois aleatoriamente se obtinha melhores resultados, uma vez que as épocas de treinamento eram quase as mesmas, porém o aprendizado era era mais generalizado, fazendo com que o agente encontrasse o objetivo em qualquer lugar do cenário.

Após utilizar *transfer learning* em ambientes diversos, concluíu-se que o seu uso é eficiente, tanto para transferência de aprendizagem para novas tarefas, quanto para novos ambientes, tenham eles texturas diferentes ou não. Notou-se que o uso de *transfer learning* torna o aprendizado mais rápido e com menos custo computacional porém que em contrapartida, as recompensas finais acabam sendo ligeiramente menores do que quando não se utiliza esta técnica e que em ambientes reais simulados foram necessárias um maior número de camadas convolutivas, uma vez que as imagens capturadas são mais complexas,

havendo mais caracteríticas parada a rede ser treinada.

As principais contribuições deste trabalho foram:

• A introdução da metodologia de *tranfer learning* em ambientes dinâmicos e complexos mostrando que é uma técnica possível de ser testada em robôs reais trazendo benefícios no tempo de processamento.

• Desenvolvimento de aplicações em ambientes realistas de simulação para robôs deste tipo de técnicas mostrando como fazer de forma eficiente simulações antigamente muito complexas de realizar.

Como trabalho futuro é proposto o uso de *deep Q-learning* em ambientes reais, utilizando um robô real, para verificar quão semelhantes os resultados seriam dos que foram obtidos neste trabalho, uma vez que já se conhece a eficiencia do *transfer learning*.

Referências bibliográficas

[1] DAY, M. A.; CLEMENT, M. R.; RUSSO, J. D.; DAVIS, D. ; CHUNG, T. H.. **Multi-UAV software systems and simulation architecture**. In: 2015 International Conference ON Unmanned Aircraft Systems (ICUAS), p. 426–435, Denver, CO, USA, June 2015. IEEE.

[2] MIYAHARA, K.. **Prototype of ARM processor-based robot module for a multi-agent mobile robot system**. In: 2017 14TH International Conference ON Ubiquitous Robots AND Ambient Intelligence (URAI), p. 629–631, Jeju, June 2017. IEEE.

[3] SZEGEDY, C.; WEI LIU; YANGQING JIA; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V. ; RABINOVICH, A.. **Going deeper with convolutions**. In: 2015 IEEE Conference ON Computer Vision AND Pattern Recognition (CVPR), p. 1–9, Boston, MA, USA, June 2015. IEEE.

[4] NGUYEN, V.; KIM, O. T. T.; PHAM, C.; OO, T. Z.; TRAN, N. H.; HONG, C. S. ; HUH, E.-N.. **A Survey on Adaptive Multi-Channel MAC Protocols in VANETs Using Markov Models**. IEEE Access, 6:16493–16514, 2018.

[5] BISHT, M.; ABBOTT, J. ; GAFFAR, A.. **Social dilemma of autonomous cars a critical analysis**. In: 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet OF People AND Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), p. 1–3, San Francisco, CA, Aug. 2017. IEEE.

[6] MEGGITT, D.; ROPER, C.; HENSON, J. ; WICKLUND, D.. **Autonomous Underwater Vehicle Intervention for Advanced Undersea Networks**. In: OCEANS 2016 MTS/IEEE Monterey, p. 1–5, Monterey, CA, USA, Sept. 2016. IEEE.

[7] CHEN, Y.; WANG, W.; ABDOLLAHI, Z.; WANG, Z.; SCHULTE, J.; KROVI, V. ; JIA, Y.. **A Robotic Lift Assister: A Smart Companion for Heavy Payload Transport and Manipulation in Automotive**

Assembly. IEEE Robotics & Automation Magazine, 25(2):107–119, June 2018.

[8] RAVINDRAN, R.; MILLS, J. P. ; KRISHNAN, M.. **Autonomous Multi-Robot Platoon Monitoring**. In: 2018 IEEE 61ST International Midwest Symposium ON Circuits AND Systems (MWSCAS), p. 328–331, Windsor, ON, Canada, Aug. 2018. IEEE.

[9] LY, O.; GIMBERT, H.; PASSAULT, G. ; BARON, G.. **A Fully Autonomous Robot for Putting Posts for Trellising Vineyard with Centimetric Accuracy**. In: 2015 IEEE International Conference ON Autonomous Robot Systems AND Competitions, p. 44–49, Vila Real, Portugal, Apr. 2015. IEEE.

[10] NAIK, N. S.; SHETE, V. V. ; DANVE, S. R.. **Precision agriculture robot for seeding function**. In: 2016 International Conference ON Inventive Computation Technologies (ICICT), p. 1–3, Coimbatore, India, Aug. 2016. IEEE.

[11] ALBERRI, M.; HEGAZY, S.; BADRA, M.; NASR, M.; SHEHATA, O. M. ; MORGAN, E. I.. **Generic ros-based architecture for heterogeneous multi-autonomous systems development**. In: 2018 IEEE INTERNATIONAL CONFERENCE ON VEHICULAR ELECTRONICS AND SAFETY (ICVES), p. 1–6. IEEE, 2018.

[12] SUTTON, R. S.; BARTO, A. G.. **Reinforcement Learning: An Introduction**. p. 352.

[13] FORERO, L. A.; VELLASCO, M. M. B. R.; FIGUEIREDO, K. ; SILVA, E.. **Intelligent Multiagent Coordination Based On Neuro-Fuzzy Models With Hierarchical Reinforcement Learning**. In: ANAIS DO 11. Congresso Brasileiro DE Inteligência Computacional, p. 1–7, Porto de Galinhas, Pernambuco, Mar. 2016. SBIC.

[14] JEERIGE, A.; BEIN, D. ; VERMA, A.. **Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing**. In: 2019 IEEE 9TH Annual Computing AND Communication Workshop AND Conference (CCWC), p. 0366–0371, Las Vegas, NV, USA, Jan. 2019. IEEE.

[15] WU, Z.; KHAN, N. M.; GAO, L. ; GUAN, L.. **Deep Reinforcement Learning with Parameterized Action Space for Object Detection**. In: 2018 IEEE International Symposium ON Multimedia (ISM), p. 101–104, Taichung, Dec. 2018. IEEE.

[16] ID SOFTWARE, M. T.. **Doom**. 1993.

[17] BELLMAN, R.; KALABA, R.. **Dynamic programming and statistical communication theory**. Proceedings of the National Academy of Sciences of the United States of America, 43(8):749, 1957.

[18] BUSONIU, L.; BABUSKA, R.; DE SCHUTTER, B. ; ERNST, D.. **Reinforcement learning and dynamic programming using function approximators**. CRC press, 2017.

[20] WATKINS, C. J.; DAYAN, P.. **Q-learning**. Machine learning, 8(3-4):279–292, 1992.

[21] KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E.. **Imagenet classification with deep convolutional neural networks**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2012.

[22] LIN, M.; CHEN, Q. ; YAN, S.. **Network in network**. arXiv preprint arXiv:1312.4400, 2013.

[23] **University of standford. convolutional neural networks for visual recognition.** `http://cs231n.github.io/convolutional-networks/`. Acessado em: 2018-09-30.

[24] SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V. ; RABINOVICH, A.. **Going deeper with convolutions**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 1–9, 2015.

[25] GUO, Y.; LIU, Y.; OERLEMANS, A.; LAO, S.; WU, S. ; LEW, M. S.. **Deep learning for visual understanding: A review**. Neurocomputing, 187:27–48, 2016.

[26] LAVIN, A.; GRAY, S.. **Fast algorithms for convolutional neural networks**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 4013–4021, 2016.

[27] VAN DER AALST, W. M.; RUBIN, V.; VERBEEK, H.; VAN DONGEN, B. F.; KINDLER, E. ; GÜNTHER, C. W.. **Process mining: a two-step approach to balance between underfitting and overfitting**. Software & Systems Modeling, 9(1):87, 2010.

[28] BADRINARAYANAN, V.; KENDALL, A. ; CIPOLLA, R.. **Segnet: A deep convolutional encoder-decoder architecture for image segmentation**. IEEE transactions on pattern analysis and machine intelligence, 39(12):2481–2495, 2017.

[29] LONG, J.; SHELHAMER, E. ; DARRELL, T.. **Fully convolutional networks for semantic segmentation**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 3431–3440, 2015.

[30] ZHAO, H.; SHI, J.; QI, X.; WANG, X. ; JIA, J.. **Pyramid scene parsing network**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 2881–2890, 2017.

[31] CHAURASIA, A.; CULURCIELLO, E.. **Linknet: Exploiting encoder representations for efficient semantic segmentation**. In: 2017 IEEE VISUAL COMMUNICATIONS AND IMAGE PROCESSING (VCIP), p. 1–4. IEEE, 2017.

[32] PASZKE, A.; CHAURASIA, A.; KIM, S. ; CULURCIELLO, E.. **Enet: A deep neural network architecture for real-time semantic segmentation**. arXiv preprint arXiv:1606.02147, 2016.

[33] ZHOU, B.; ZHAO, H.; PUIG, X.; XIAO, T.; FIDLER, S.; BARRIUSO, A. ; TORRALBA, A.. **Semantic understanding of scenes through the ade20k dataset**. International Journal of Computer Vision, 127(3):302–321, 2019.

[34] PAN, S. J.; YANG, Q.. **A survey on transfer learning**. IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2010.

[35] MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D. ; RIEDMILLER, M.. **Playing atari with deep reinforcement learning**. arXiv preprint arXiv:1312.5602, 2013.

[36] **Breakout**. `https://www.digitalspy.com/videogames/a790432/atari-breakout-40-today-all-gamers-need-to-know-how-it-came-to-be/`. Acessado em: 2019-03-13.

[37] ASAWA, C.; ELAMRI, C. ; PAN, D.. **Using transfer learning between games to improve deep reinforcement learning performance and stability**.

[38] YIN, H.; PAN, S. J.. **Knowledge transfer for deep reinforcement learning with hierarchical experience replay**. In: THIRTY-FIRST AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2017.

[39] PARISOTTO, E.; BA, J. L. ; SALAKHUTDINOV, R.. **Actor-mimic: Deep multitask and transfer reinforcement learning**. arXiv preprint arXiv:1511.06342, 2015.

[40] KEMPKA, M.; WYDMUCH, M.; RUNC, G.; TOCZEK, J. ; JAŚKOWSKI, W.. **ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning**. arXiv:1605.02097 [cs], May 2016. arXiv: 1605.02097.

[41] **Slade 3**. `http://slade.mancubus.net/`. Acessado em: 2018-06-18.

[42] **Zdoom**. `https://zdoom.org/index`. Acessado em: 2018-01-25.

[43] FANGWEI ZHONG, WEICHAO QIU, T. Y. A. Y. Y. W.. **Gym-unrealcv: Realistic virtual worlds for visual reinforcement learning**. Web Page, 2017.

[44] **Realistic rendering**. Web Page, 2019.

[45] ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANE, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIEGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y. ; ZHENG, X.. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems**. arXiv:1603.04467 [cs], Mar. 2016. arXiv: 1603.04467.