Training an Agent to Find and Reach an Object in Different Environments using Visual Reinforcement Learning and Transfer Learning

Evelyn Conceição Santos Batista[1], Wouter Caarls[1], Leonardo A. Forero[2] and Marco Aurélio C. Pacheco[1]
[1]*Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil*
[2]*Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, Brazil*

Keywords: Autonomous Agent, Transfer Learning, Reinforcement Learning, Deep Learning, Vizdoom, DQN.

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

1 INTRODUCTION

The learning process of autonomous agents is a field of research increasingly developed and explored because of the constant improvements in sophisticated algorithms and specialized hardware (Day et al., 2015), (Miyahara, 2017), (Szegedy et al., 2015). These improvements have contributed to the development of applications in the area of autonomous simulation, which include autonomous cars (Nguyen et al., 2018), (Bisht et al., 2017), aerial and marine autonomous vehicles (Meggitt et al., 2016), (Chen et al., 2018), and autonomous robots (Ravindran et al., 2018), (Ly et al., 2015). In each of these cases, techniques are studied to allow robots to acquire innovative abilities or to adapt to an environment through learning algorithms, making it possible for them to constantly learn and develop new abilities for different purposes (Naik et al., 2016) (Alberri et al., 2018).

In this context, an autonomous robot must interact with its environment to achieve its goals. It must be capable of gathering information about its environment, making decisions based on this information and initiating a specific action based on those decisions.

In recent years, there has been a need for more autonomous robotic systems in several areas. More specifically, there has been a need for robots to develop the ability to make decisions without human interference in order to better achieve pre-established goals. Because of that, most research tends to focus on developing applications that give autonomy to robots, as the agents recognize the environment through sensors and act independently in those environments through actuators. The autonomous system allows robots to make decisions quickly, taking into consideration the diverse variables of an environment. Consequently, the system has several applications, such as in autonomous cars, intelligent drones, and robots with agrarian or safety applications.

With that in mind, the objective of this paper is to simulate autonomous robots in complex environments in order to develop their autonomy further and raise the level of complexity of their tasks. In order to do so, a visual reinforcement technique, known as deep reinforcement learning, is used. Because it is very difficult to train an automaton in the real world, this works aims at performing simulations so that the agent does not have to learn everything in a real environment. In addition, transfer learning is used in order to allow the transference of the simulated learning to the real world.

This paper is organized as follows: Section 2 describes the concepts of Reinforcement Learning and Deep Q-networks. Section 3 presents the

Batista, E., Caarls, W., Forero, L. and Pacheco, M.
Training an Agent to Find and Reach an Object in Different Environments using Visual Reinforcement Learning and Transfer Learning.
DOI: 10.5220/0010236807320741
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 732-741
ISBN: 978-989-758-484-8

architecture implemented, introduce the environments and the proposed method. Section 4 is the results obtained and finally in section 5 we talk about the conclusions and future implementations.

2 THEORETICAL BACKGROUND

2.1 Autonomous Agent

Agent-based processing can be characterized by an input state which produces an output after being processed. The input state is determined by the environment in which the agent acts and the output is determined by the effects caused by the agent in the environment. Despite the various similarities between agent theory and control theory, agents are best applied to specific situations which demand autonomy and intelligence, such as complex and dynamic environments.

2.2 Q-Learning

Q-Learning (Watkins and Dayan, 1992) is a reinforcement learning algorithm (Sutton and Barto, 1998), which learns the value of state-action pairs. In it, the state-action value function $Q^{\pi}(s, a)$ must be estimated for the policy π, for all states *s,* and for actions *a.* This algorithm is defined mainly by the updates of the function shown in equation 1.

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

In which an agent in time *t,* situated in state $s_t \in S$, appoints one of the possible actions, $a_t \in A_{st}$, in a state according to the selection policy (ϵ-greedy (Sarkar et al., 1994)). Equation 1 updates the evaluation function for this state-action pair considering the value of the obtained reward,r_t and the max evaluation function for all possible actions in future state *s.* In order to do so, the discount factor γ is used. This factor determines how the reinforcement learning agent's future reward will impact the immediate reward. In addition, the learning rate α defines the extent to which newly acquired information replaces old information.

The goal of the Q-Learning algorithm is to estimate the Q function for all visited states and taken actions, recording it in a table. For each action taken, a reward (r) will be awarded - which can be favourable or not - so that on subsequent visits to the same state (s), the most appropriate action can be determined.

2.3 Deep Q-Networks (DQN)

As previously explained, Q-Learning uses a table to represent the Q function. This allows for a precise representation of the value of the long-term rewards, which can be obtained in every state for each of the actions available. However, this representation also presents a very important problem: increasing the complexity of the state also increases the size of the Q table exponentially.

In order to create a model capable of solving a complex problem while guaranteeing that the size of the Q table is still able to be stored in memory a function approximator is required. In order to do so, we use an approximation of the Q function using the neural network employed in 2013 by the DeepMind team in the project Playing Atari with Deep Reinforcement Learning (Mnih et al., 2013).

In order to use Deep Q-Learning, it is first necessary to initialize a *replay memory* (Mnih et al., 2013), which stores previous transitions, and initialize the Q function with random weights. For each epoch, a state is initialized. The state consists of a sequence of images taken from the simulator and preprocessed. For each step an action is chosen in accordance to a greedy policy. Then, the action is executed in the simulator and the reward is obtained, along with the next image. The image is processed and integrated into the state. The transitions are stored in the replay memory and a mini-batch of transitions is chosen from that memory. The mini-batch is used to calculate a future reward and a neural network is used to estimate the Q reward.

2.4 Transfer Learning

Transfer learning implies the reuse of a pre-trained model to solve a new problem, that is, the use of a trained neural network on another set of data, usually bigger and more complex, to solve a new problem.

The only trained layer is the added one. Therefore, in order to save training time, the output layer of the last trained layer is calculated for all training images. This output vector is known as bottleneck features. When training the network, the bottleneck features are passed on to the added layers, which are trained normally. Consequently, the training is faster and usually has good results.

Since Deep Q-Learning algorithms require great computational power to be trained, Transfer Learning techniques are used to decrease the training time and processing for agents. Accordingly, Transfer Learning is used throughout this study in order to reduce the time spent on training without

compromising the results. Some examples of Transfer Learning in Deep reinforcement learning can be seen in (Asawa et al., 2017), (Yin and Pan, 2017), (Parisotto et al., 2015).

3 PROPOSED METHOD

In accordance with this study's objective, many environments were created in order to test the learning capabilities of autonomous robots using transfer learning techniques in simple and complex environments. Additionally, many scenarios were created in order to investigate the transfer learning in DQNs.

3.1 Environments Developed through the Game Doom

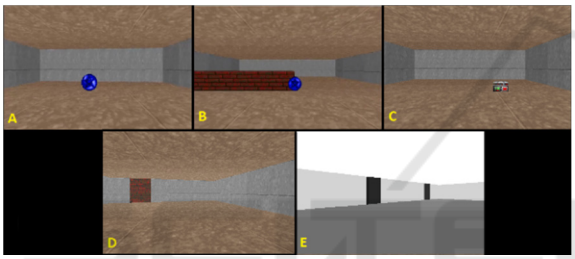

Figure 1: Doom Environments (A: environment with the task of collecting spheres, B: environment with barriers, C: environment with the task of colecting a box, D: environment with non-segmented doors, and E: environment with segmented doors).

Some environments were developed in the Doom game setting using API ViZDoom (Kempka et al., 2016) and the open source Doom editor, Slade 3 (Slade 3). The API from ViZDoom gives direct access to the ZDoom game engine (Zdoom), which allows to send commands to the game agent synchronously and to receive input from the current state of the game. The interaction with the Doom game engine was done using ACS scripts within the Doom editor to calculate rewards for all scenarios.

The Doom environment (Figure 1) was used on the first batch of tests. In this case, four scenarios with different difficulty levels were created. In the first scenario, the agent had to find and collect a sphere (Figure 1.A). In the second scenario (Figure 1.B), the agent had to go through a path with a barrier in order to find and collect the sphere. In the third scenario, the agent had to retrieve a box (Medkit) (Figure 1.C). The fourth scenario included non-segmented doors (Figure 1.D) and the agent had to locate a door. Lastly, the fifth scenario had segmented doors (Figure 1.E) and the agent had to locate the correct door amongst three existing ones.

3.2 Unreal Environment

Apart from ZDoom, this study included a platform for creating games called Unreal (Zhong et al., 2017). The template Realistic Rendering (Abadi et al., 2016) was taken from Unreal. This simulator was chosen because its images resembled reality (Figure 2) and fit with the objective of increasing the level of complexity for every environment created for the tests, along with including obstacles such as tables and plants. In this scenario, there are three doors and the agent must choose between them in order to reach the correct one.

Figure 2: Unreal Environment - Realistic Rendering.

3.3 Reward Value

The reward values for the environments of object collection are 100 on reaching the goal and -1 for each step taken to reach the destination. Meanwhile, the rewards for the navigation environments are 200 if the agent finds the door and -10 in case of collisions with the wall.

3.4 The Architecture of the Network Chosen

For simpler environments such as Doom, a neural network model was created through *Tensorflow* [24] and used as base. The model was slightly modified. It is a simple model, with 2 convolutional layers and 2 full connected, as shown in Figure 3 and Table 1.

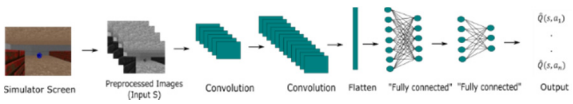

Figure 3: Neural network architecture used in VizDoom training environments.

Table 1: Neural network architecture used in VizDoom training environments.

Layer	Output	Kernel size	Stride	Activation
Conv2d	8	(6,6)	(3,3)	ReLU
Conv2d	16	(3,3)	(2,2)	ReLU
Fully Connected	128			ReLU
Fully Connected	8			

The hyperparameters used in the training of the environments created with the use of *ZDoom* are shown in *Table 12*.

Table 2: Doom Hyperparameters.

Hyperparameters	Values
Epochs	100
steps per epoch	2000
batch size	128
discount rate	0.99
learning rate	0.0002
replay memory size	10000
test episodes per epoch	20

For the Realistic Rendering environments, a deeper network (Figure 4 and Table 3) was used. This choice was made because the environment was more complex than the ones created through ZDoom. It presented a texture with more details and more objects scattered around, making it difficult for the agent to navigate to the goal.

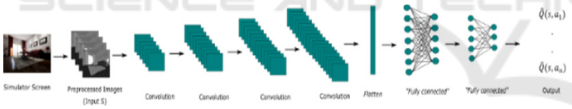

Figure 4: Neural network architecture used in Unreal training environments.

Table 3: Neural network architecture used in Unreal training environments.

Layer	Output	Kernel size	Stride	Activation
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			

The hyperparameters used in the training are shown in Table 4.

Table 4: Unreal Hyperparameters.

Hyperparameters	Values
Epochs	8000
steps per epoch	2000
batch size	128
discount rate	0.95
learning rate	0.0001
replay memory size	10000
test episodes per epoch	20

3.5 Training Summary

The objective of this work was to make an automaton learn to navigate through an unknown environment. In order to do so, a neural network was trained. The network input, described above, only required 4 frames (Mnih et al., 2013). Each frame needed a preprocessing before it was given as input to the neural network. The preprocessing environment created with ZDoom required a resizing of each image to (30, 45), a grayscale, and a normalization.

The environments created with Unreal were semantically segmented before they were resized. The segmentation was necessary because the previously trained network learned from images of a segmented environment. Consequently, images from the Realistic Rendering environment should look like the images previously trained. In some trainings, transfer learning is used to avoid reworking and retraining of already trained weights. After preprocessing the images, they are given as input to the network that was previously described. Through this process, the agent is trained.

3.6 Graphics Creation

The graphs were designed to display the results of each epoch as follows: 1) each epoch can consist of multiple episodes; 2) an epoch has a maximum of 2000 steps and the agent must reach his goal within this limit of steps; 3) each time a goal is reached, an episode is completed; 4) an epoch can be equal to one episode if the agent does not reach the goal during the 2000 steps.

Considering the stochasticity of the algorithm, the same experiment was run 10 times and the average rewards accumulated in each epoch were made. To obtain the average of each epoch, the accumulated rewards of each episode within an epoch were averaged.

4 RESULTS

The results of the training are reported below.

The experiments were separated as follows:

- Best initialization for the agent and the object;
- Transfer learning to new tasks by testing layer cutting for transfer learning;
- Transfer learning to a new environment;
- Transfer learning for environments with different textures;
- Transfer learning from a simple environment to a complex one.

4.1 Best Initialization for the Agent and the Object

In this experiment, the environment with the objective of collecting spheres (Figure 1.A) was used to train the agent. Some changes were made in the agent's and the sphere's initialization An example would be making the agent start the simulation in a fixed place and the sphere in different locations or having the sphere and the agent start in the fixed places. The results show that these different types of initialization affect the results of the agent's training.

4.1.1 Starting the Simulation with the Agent and the Sphere in Fixed Positions

In this environment, the agent and the object were created in the same position for all episodes. The results are shown in Figure 5 and Table 5.

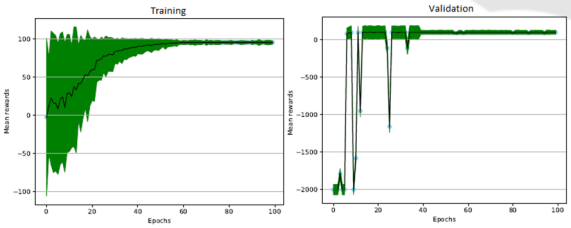

Figure 5: Training results for agent and object initialization in fixed positions.

The agent learned to reach the objective within approximately 30 epochs. It is important to notice that the reward difference between Figure 1.A and Figure 1.B occurs because, once the validation takes place, the agent has to execute up to 2000 steps. If it does not reach the objective, the 2000 steps take place. If it reaches the objective, less steps are given, and the reward is greater. The idea was to transfer from fixed to random positions. However, random positions learns faster than fixed positions, so therefore it does not make sense to try to transfer

4.1.2 Starting the Simulation with the Agent in a Fixed Position and the Sphere in Random Positions

In this experiment, the agent was placed in a fixed position and the sphere in random positions so that the model could gain a greater generalization capacity.

The results obtained in this case were better than the ones achieved through the fixed agent and sphere initialization, since this experiment generalized learning at practically the same training time and number of epochs (30), as seen in *Table 5* and *Figure 6*.

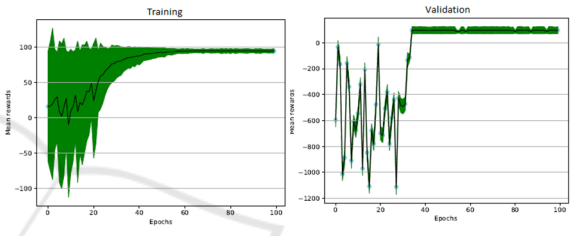

Figure 6: Result of the start training for the agent in a fixed position and for the object in random positions.

4.1.3 Starting the Simulation with the Agent and the Sphere in Random Positions

Finally, an environment with random initialization of the agent and objective was trained. This environment presented the highest capacity to generalize the model, as the agent was able to reach the objective anywhere it was placed. In addition, it was concluded that this was the best initialization due to the epoch difference needed for the agent to learn. Even though the results were similar, the agent's learning was generalized to find a sphere from any position. The difference between the two first experiments performed previously was small, since approximately 40 epochs were needed for the agent to learn in this experiment, as shown by Figure 7 and Table 5.

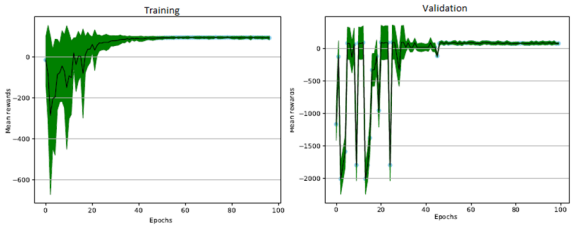

Figure 7: Result of agent and object initialization training at random positions.

The next tests used the simulation initializations considered the best. In other words, they used random agents and targets.

Table 5: Best Agent and Goal Initialization Test Results.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Fixed agent and environment	13.26	95.37	2.60	78.45	21.19	-176.2	71.7
Random Target Fixed Agent	13.60	94.46	2.95	74.56	24.67	-93.98	47.86
Random Target and Agent	18.02	94.43	5.93	56.49	45.53	-115.2	99.94

4.2 Transfer Learning to New Tasks by Testing Layer Cutting for Transfer Learning

In order to test the robustness of transfer learning, the target object was modified, verifying if the training results would be similar. In this case, an environment containing a ("medikit") box as its objective (Figure 1.C) was used.

This environment was trained without the use of transfer learning, obtaining results (Table 6) which proved to be similar to the scenario which had the sphere as its objective. However, it had the longest rise time, taking up to 45 epochs to be trained, since the box is smaller than the sphere and, due to its color, does not stand out in the environment.

Table 6: Results of agent training in the box-picking environment.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Box without TL	21.42	90.16	11.47	11.56	87.50	-514.0	218.5

Different tests were performed to test the transfer learning. In the first test, the trained net could have its weights updated. In the second test, the weights of the two convolutive layers were fixed. In the third test, only the weights of the first convolutive layer were fixed, that is, the weights were not retrained. All experiments used an environment-trained network as a pre-trained network. That was done in order to retrieve spheres so that the environment could be trained. The goal of the environment, in this case, was to reach the box, as mentioned previously.

4.2.1 Training Which Combined All the Network Weights Previously Trained

In this experiment, all net weights were reused and none were fixed. The greatest difference shown by the results was in the validation, as the agent already has some knowledge prior to the training and received greater rewards once the training process began. This is shown by *Table 7* and *Figure 8*.

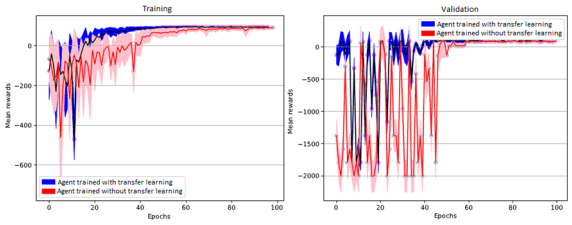

Figure 8: Results of the training which combined all the network weights previously trained.

Table 7: Table with a comparison between the presence and absence of transfer learning in the training which combines all transfer weights previously trained.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Box without TL	21.42	90.16	11.47	11.56	87.50	-514.06	218.59
Box with TL of sphere environment	18.02	91.25	5.59	48.51	47.05	-52.04	110.24

4.2.2 Applying Transfer Learning to Two Convolutional Layers with Fixed Weight

In this experiment, only the weights of the fully connected layers were adjusted, that is, no new frame elements were used during the training, as all convolutional layers had their weights fixed and could not be trained again. As the environments were practically the same - the only change was the goal (sphere - box) - the results exceeded the baseline even without training the convolutional layers, as seen by *Table 8* and *Figure 9*.

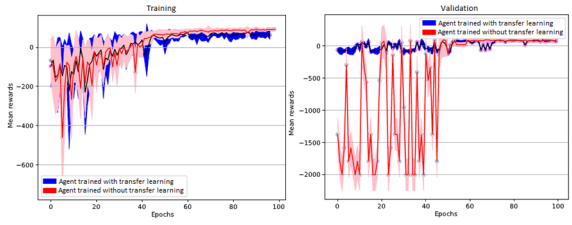

Figure 9: Training output using two-layer convolutional transfer learning with fixed weights.

Table 8: Comparison table showing the use of transfer learning in the training using two fixed weight convolutional layers.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Box without TL	21.42	90.16	11.47	11.56	87.50	-514.0	218.59
Box with TL of sphere environ-ment	21.76	67.29	30.28	7.23	95.24	13.81	24.95

4.2.3 Using Transfer Learning in the First Convolutional Layer with Fixed Weights

The last experiment was trained using the first convolutional layer of the transfer learning. The results obtained were better than those of the previous experiment (*Table 7* and *Table 8*), although the rise time did not present a significant difference. Overall, the results of the training with transfer learning were better than those without transfer learning, as shown by *Table 9* and *Figure 10*.

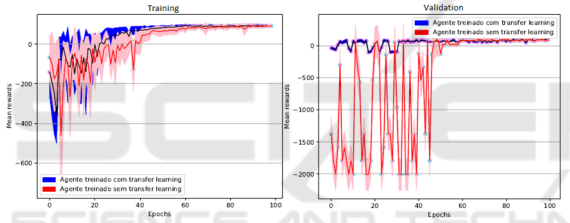

Figure 10: Results of the training with transfer learning using the first convolutional layer with fixed weights.

Table 9: Comparison table of the training with and without transfer learning, using the first convolutional layer with fixed weights.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Box without TL	21.42	90.16	11.47	11.56	87.50	-514.06	218.59
Box with TL of sphere environ-ment	13.26	92.62	5.67	35.38	64.43	52.04	16.52

Upon analyzing the results, it was concluded that the first layer with fixed weights was the best parameterization method when it came to the use of transfer learning in the experiments. After all, the results produced by it were better because less epochs were required for the agent to be trained.

4.3 Using Transfer Learning in a New Environment

Given the previous results, obstacles were inserted in the environment to make it more complex. Instead of now changing the target, we change the environment instead. The scenery chosen was similar to the one used in previous experiments. The only difference was that a barrier was placed between the agent and the object (Figure 1.B).

The environment was initially trained without transfer learning in order to allow a comparison between results with and without transfer learning.

The results shown by the network were as expected. The level of complexity of the environment increased; therefore, the agent took longer to learn how to reach the target. It needed approximately 72 epochs in order to succeed, as shown by *Table 10* and *Figure 11*.

The agent was trained once more using transfer learning by applying the weights of the trained net in the environment with the objective of collecting spheres (Figure 1.A).

The results shown by the training with transfer learning were better than those without transfer learning. The rise time was smaller, and the averages of the validation rewards were higher. It is important to mention that, although the agent learns faster with transfer learning, the final rewards were slightly lower as the weights of the first convolution layer were fixed.

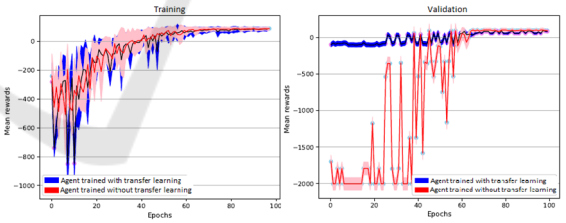

Figure 11: Training results of the new environment using transfer learning.

Table 10: Comparison table of the results shown by the new environment with and without transfer learning.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Sphere / Barrier without TL	24.14	85.06	17.34	-38.75	114.13	-715.76	100.59
Sphere / Barrier with TL of Sphere Environ-ment	20.10	81.94	23.34	-55.05	118.88	-5.2	39.63

4.4 Using Transfer Learning in Environments with Different Textures

In order to prepare the tests for the simulated real environment, environments with significant differences between each other were created. An example are the experiments described in this paper, where the agent's objective was to reach a doorway in an environment that is segmented (Figure 1.E) and one that is not segmented (Figure 1.D).

The agent was trained separately in each of the experiments in order to compare results. The first environment trained was not segmented (Figure 1.D) and its results (*Table 11*) were satisfactory. Overall, the agent required 50 epochs to be successfully trained.

Table 11: Table with the results of the agent's training in the environment with a non-segmented door.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Door without TL	15.64	1.41	1.64	-11.47	5.69	-132.83	59.66

In order to use textures that were not from the Doom, the environment was manually segmented (Figure 1.E) and trained to discover how the agent would behave. The results are shown in *Table 12*. Even though the objective of the previous environments was also a door, the fact that the textures were simpler in this experiment made it possible for the agent to learn quicker, with approximately 40 epochs.

Table 12: Table with the results of the agent's training in an environment with the segmented door.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Segmented Door	14.96	1.09	1.99	-12.66	5.50	-155.29	81.18

Once the training was over, the transfer learning was used to transfer the knowledge of the trained network with a segmented environment to the non-segmented environment.

Contrary to the non-segmented environment, the segmented environment had simpler textures, without details. That divergence resulted in a learning difference for the agent, shown by Figure 12 and *Table 13*. The results indicate that the texture affects the learning process. Accordingly, the use of transfer learning in an environment with a simpler texture and then the transference of this learning to a more complex texture resulted in a faster training. Consequently, the agent learned after 42 epochs.

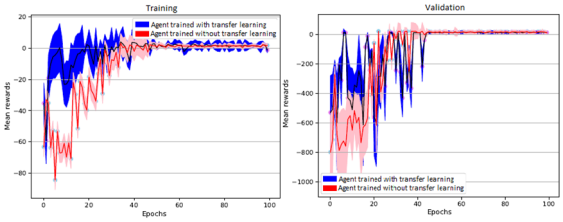

Figure 12: Results of the training with transfer learning in environments with different textures.

Table 13: Table comparing the learning rate of different textured environments.

Model	Rise Time (s)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Door without TL	15.64	1.41	1.64	-11.47	5.69	-132.83	59.66
Door with TL of Door Segmented environment	11.02	1.24	1.92	-2.02	6.75	-98.24	71.44

4.5 Transfer Learning: From a Simple Environment to a More Complex Environment

After the results were obtained in the Doom environment, a more complex environment (*Figure 2*) was developed similar to the real one. It presented several challenges, such as furniture, plants and intricate lighting. The objective of the environment was to find the door. However, in this case, three identical doors were created and only one of them was considered correct.

The agent was trained without the use of transfer learning. The training required more epochs because the environment was more complex. It took 6255 epochs for the agent to learn (Figure 14 and *Table 14*).

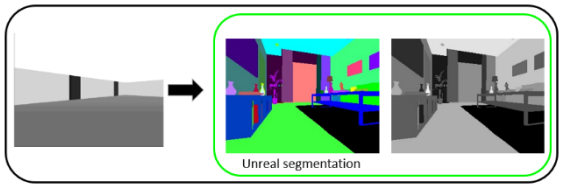

Figure 13: Pre-processing of the frame in the Realistic Rendering environment.

In order to use transfer learning in the environment trained with segmentation (Figure 1.E), the images from the Realistic Rendering were semantically divided using the segmentation for patterns created by

Unreal. After this process, the images were placed in scales of gray so that the frames could match the trained environment (*Figure 13*).

Regarding the use of transfer learning, the results (Figure 14 and Table 14) showed that the technique made a difference. Although the environment was significantly more complex, the fact that the network had previous knowledge, even from a distinct environment, made the rewards higher from the beginning of the training and decreased the rise time.

Upon analyzing the experiments, it was concluded that, in the environments where the door was the main objective, the agent learned the trajectory to the door. Meanwhile, in the environments where the objective was to retrieve a box or a sphere, the agent learned that the objective was to reach objects regardless of where they were placed.

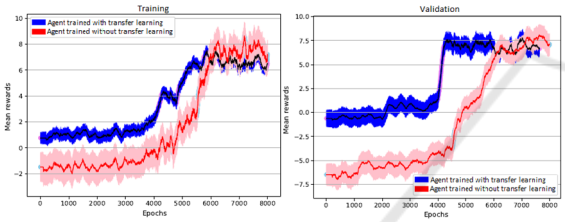

Figure 14: Results regarding the application of transfer learning from a simple environment to a complex one.

Table 14: Table comparing the application of transfer learning from a simple environment to a complex one.

Model	Rise Time (h)	Mean Final Train	Std. Final Train	Mean Train	Std. Train	Mean Val.	Std. Val.
Realistic Rendering without TL	11.55	8.70	1.45	1.78	1.39	-0.91	1.66
Realistic Rendering with TL of Segmented Port Environment	9.58	6.54	0.35	3.57	0.29	3.33	0.99

The training of each environment can take time depending on the difficulty level. For example, the training of an agent in the Realistic Rendering environment using transfer learning in a Xeon Platinum 8160 processor takes about 18.24h. Using the same processor with an optimized *Tensorflow* for the Intel® processor, the training time will decrease, becoming 9.58h (Table 15).

Table 15: Time Comparison.

DQN – Realistic Rendering using transfer learning	Xeon Platinum 8160 Processor	Xeon Platinum 8160 Processor with optimizer Tensorflow
Training time	18.24h	9.58h

5 CONCLUSION AND FUTURE WORK

This work presented a study on the simulation of an autonomous robot that interacted with the environment through a camera. In order to do so, a deep learning algorithm by visual reinforcement (Deep Q-Learning) was used, along with knowledge transfer techniques in convolutive networks. The main objective was to test the methods and algorithms together in order to simulate the robot's performance in different environments and to use the experience from the previous environments to reach the objective quicker in complex environments.

The Deep Q-Learning choice as basis for the simulation was based on a review of literature and was motivated by major advances in Deep Learning, which permit fast image processing through convolutive network after a training stage. As a result, camera guided robots increasingly extend their autonomy and overcome limitations in traditional reinforcement learning algorithms.

The transfer learning methodology was applied to complex and dynamic environments, proving that it can be used to test real robots. In addition, it improved the processing time and allowed the development of applications in realistic robot simulation environments for these transfer learning techniques. As a result, the methodology presents an efficient way of performing simulations.

The performance of the methodology was superior to that presented by other researches, showing the stability and robustness of the robotic system. These characteristics are essential to real robots. The scenarios used in the simulation are highly complex and dynamic, demonstrating that the methodology conceived in this work works for robots and real scenarios.

As future research, the authors propose using real robots to test Deep Q-Learning in real environments and extend the methodology to more specific tasks, in order to verify how different the results would be from the ones achieved in this study, as the efficacy of transfer learning is already known.

REFERENCES

Day, M. A., Clement, M. R., Russo, J. D., Davis, D., & Chung, T. H. (2015, June). Multi-uav software systems and simulation architecture. In 2015 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 426-435). IEEE.

Miyahara, K. (2017, June). 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) (pp. 629-631). IEEE.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Nguyen, V., Kim, O. T. T., Pham, C., Oo, T. Z., Tran, N. H., Hong, C. S., & Huh, E. N. (2018). A survey on adaptive multi-channel MAC protocols in VANETs using Markov models. IEEE Access, 6, 16493-16514.

Bisht, M., Abbott, J., & Gaffar, A. (2017, August). 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) (pp. 1-3). IEEE.

Meggitt, D., Roper, C., Henson, J., & Wicklund, D. (2016, September). Autonomous Underwater Vehicle Intervention for Advanced Undersea Networks. In OCEANS 2016 MTS/IEEE Monterey (pp. 1-5). IEEE.

Chen, Y., Wang, W., Abdollahi, Z., Wang, Z., Schulte, J., Krovi, V., & Jia, Y. (2018). A Robotic Lift Assister: A Smart Companion for Heavy Payload Transport and Manipulation in Automotive Assembly. IEEE Robotics & Automation Magazine, 25(2), 107-119.

Ravindran, R., Mills, J. P., & Krishnan, M. (2018, August). Autonomous Multi-Robot Platoon Monitoring. In 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS) (pp. 328-331). IEEE.

Ly, O., Gimbert, H., Passault, G., & Baron, G. (2015, April). A fully autonomous robot for putting posts for trellising vineyard with centimetric accuracy. In 2015 IEEE International Conference on Autonomous Robot Systems and Competitions (pp. 44-49). IEEE.

Naik, N. S., Shete, V. V., & Danve, S. R. (2016, August). Precision agriculture robot for seeding function. In 2016 International Conference on Inventive Computation Technologies (ICICT) (Vol. 2, pp. 1-3). IEEE.

Alberri, M., Hegazy, S., Badra, M., Nasr, M., Shehata, O. M., & Morgan, E. I. (2018, September). Generic ROS-based Architecture for Heterogeneous Multi-Autonomous Systems Development. In 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES) (pp. 1-6). IEEE.

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 2, No. 4).

Sarkar, U. K., Chakrabarti, P. P., Ghose, S., & DeSarkar, S. C. (1994). Improving greedy algorithms by lookahead-search. Journal of Algorithms, 16(1), 1-23.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Asawa, C., Elamri, C., & Pan, D. (2017). Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability.

Yin, H., & Pan, S. J. (2017, February). Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In Thirty-First AAAI Conference on Artificial Intelligence.

Parisotto, E., Ba, J. L., & Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. arXiv preprint arXiv:1511.06342.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016, September). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In 2016 IEEE Conference on Computational Intelligence and Games (CIG) (pp. 1-8). IEEE.

Slade 3. http://slade.mancubus.net/. Accessed em: 2019-06-18

Zdoom. https://zdoom.org/index. Accessed em: 2019-01-25

Zhong, F., Qiu, W., Yan, T., Alan, Y., & Wang, Y. (2017). Gym-UnrealCV: Realistic virtual worlds for visual reinforcement learning.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.