Investigating Deep Q-Network Agent Sensibility to Texture Changes on FPS Games

Paulo Bruno de Sousa Serafim
Instituto Atlântico
Fortaleza, Brazil
paulo_serafim@atlantico.com.br

Yuri Lenon Barbosa Nogueira
Department of Computing (DC)
Federal University of Ceará (UFC)
Fortaleza, Brazil
yuri@dc.ufc.br

Creto Augusto Vidal
Department of Computing (DC)
Federal University of Ceará (UFC)
Fortaleza, Brazil
cvidal@dc.ufc.br

Joaquim Bento Cavalcante-Neto
Department of Computing (DC)
Federal University of Ceará (UFC)
Fortaleza, Brazil
joaquimb@dc.ufc.br

Rômulo Férrer Filho
Teleinformatics Engineering Department (DETI)
Federal University of Ceará (UFC)
Fortaleza, Brazil
romulofffufc@gmail.com

***Abstract*—Graphical updates are very common in modern digital games. For instance, PC game versions usually receive higher resolution textures after some time. This could be a problem for autonomous agents trained to play a game using Convolutional Neural Networks. These agents use the pixels of the screen as inputs and changing them could harm their performance. In this work, we evaluate agents' sensibility to texture changes. The agents are trained to play a First-Person Shooter game and then are presented to different versions of the same scenario, in which the only difference among them is texture changes. As the testbed, we use a ViZDoom scenario with a static monster that should be killed by the agent. Four agents are trained using Deep Q-Networks in four different scenarios. Then, every agent is tested in all four scenarios. We show that although every agent can learn the behaviors to win the game when playing the same version in which it was trained, they cannot generalize to all other versions. Only in one case, the agent had a good performance in a different scenario. Most of the time, the agent moved randomly or just stood still, and shot continuously, indicating that it could not understand the current screen. Even when the background textures were kept the same, the agent could not identify the enemy. Thus, to ensure proper behavior, an agent needs to be retrained not only if the problem changes, but also when only the visual aspects of the problem are modified.**

***Keywords*-autonomous agents, deep reinforcement learning, digital games, first-person shooter games**

I. Introduction

Since the start of Deep Reinforcement Learning (DRL) as a research area creating agents to play Atari games [1], [2], we have seen many achievements in autonomous game agents. They cover a broad range of challenges, like solving dozens of Atari games [2], specific harder Atari games [3], all Atari games [4], First-Person Shooters (FPS) [5], board games [6], [7], [8], Multi-Agent games [9], [10], and Real-Time Strategy games [11].

The vast majority of DRL agents are based on visual input, specifically the game screen. The agent usually consists of a Convolutional Neural Network (CNN) which input is the current frame, i.e. the matrix of pixel values. Being an image-based machine learning technique, the game screen approach can suffer from a bias problem towards the dataset images. If not handled carefully, the model would correctly evaluate images similar to the ones used in its development but can have poor performance with other samples.

This is not a mere overfitting problem since the model probably performed well in the test dataset. In this work, we focus on the issues that arise when using external data. For example, models that tend to classify sheep as dogs[1]. Or, if we can make a deeper analysis of the model, we could see that just deactivating a single pixel is enough to fool the neural network, as in one-pixel attack [12]. Also, as we are aware, CNNs tend to classify the images in accordance with their texture content rather than in accordance with their shape content [13]. Those examples show how vision-based models are very dependent on the training data.

When talking about games, there is a related problem. We expect the players to be able to play the same game on different platforms with approximately the same skill. For example, a chess player should be as good in real life as in an digital version of chess. If we learn to play a game like Tetris[2] in a game console, we would also know how to play it in a handheld version, or at least its basic rules. However, when the player is a vision-based DRL agent, this could not be the case.

One example is if an agent learns how to play a game and after that the game is updated. Commonly, there are changes to the game, like increasing the health points of an enemy or decreasing the power of a weapon. We call "buff" when there is an increase in the power of any character, and "nerf" when there is a decrease. This newer version of the game with buffs

[1]https://aiweirdness.com/post/171451900302/do-neural-nets-dream-of-electric-sheep

[2]Tetris is a property of The Tetris Company, LLC (TTC)

and nerfs is new to the agent and it may not perform as good as before.

Although this is by itself a concern, there is also a more subtle change that can affect the agent as much. If the patch has changes in textures, the agent would not recognize the frames to be similar to the ones it was trained. As such it has to be retrained in this newer version. This is not unusual, as we can see in recent works [11], where the agent was trained in a specific version of the game. To have a good performance in a newer version, it must be retrained. This is a very costly and time-consuming process.

In this work, we investigate the sensibility of DRL-based agents, training using Deep Q-Networks (DQN) [1], to texture changes and evaluate the effects in their performance. We train it in a scenario of a Doom-based FPS environment called VizDoom [14], create three versions of the same scenario with different textures, and then test the agent in all four versions (Fig. 1). We answer three questions: (i) can a DRL-based autonomous agent generalize its behavior for scenarios with different textures without retraining? (ii) How much changing only the textures affect the results? (iii) Does the training version affect the performance when testing in unseen textures? We show that an agent capable of solving an FPS environment has a poor performance when playing the same scenario with different textures. The results would help us to move towards creating agents that can learn to play the games without relying on specific textures.

This paper is organized as follows. In Section II, we summarize some works that evaluate the influence of changing input image textures. Section III brings a brief description of DRL. Section IV presents the testbed scenarios used, neural network parameters, and describes the settings of the experiments performed. In Section V, we show the obtained results and discuss them. Finally, in Section VI, we present closing remarks and suggest some future works.

II. Related Works

In this section, we present two works that indicate the effects in performance when testing with unseen input images for classification tasks and briefly describe three works that change the texture of the environment using DRL-based agents. Two of them train the agents in multi-texture scenarios, both with better performance than single-texture training. The last work makes a series of ablation studies on the impact of human priors in the performance, comparing the results obtained by agents and humans in different versions of a game.

One work that demonstrated a problem with CNN-based image classification was one-pixel attack, from Su et al. [12]. For each image, the pixel that contributed the most to the classification was computed and "removed" (turned into white or black). The authors showed that several images in three different datasets are misclassified by modifying only a single pixel. A general problem with CNN recognition, exposed by Geirhos et al. [13], is that these networks are biased towards texture. The authors modified the input images to maintain the original shape but with texture from other images. The results showed that the network tends to classify the inputs in accordance with the image from which the texture was taken.

Training DRL with different textures is not common. Chaplot and Lample [15] train agents on ViZDoom environments using random textures. After training, the agent is tested with unseen textures on the same map. After training with several random textures, the agents are not affected by texture changes when tested. This strategy clearly improves the agent's performance, but no further analysis of the impact of the random textures training is performed. In this paper, we also use ViZDoom as the environment and analyze the effects of texture changes in the performance of the agents, but training the agents in a single texture version.

Polvara et al. [16] present another work that trains DRL-based agents using different textures. The authors train an agent to control a quadrotor unmanned aerial vehicle (UAV) to find a landing marker and land correctly. To ensure that the agent is robust to different types of asphalt, they train it with different random textures among 71 possibilities. An agent trained in a single texture scenario is also trained, but the performance of the multi-texture one is significantly better. This indicates that indeed training with a single texture is worse in this situation, but there is no further analysis of the impact of multi-texture training.

The closest work to our proposal is the work of Dubey et al. [17]. In this work, the authors investigate the importance of human priors in performance when playing games. They create a 2D platform game and modify its textures in order to make it more difficult in the human perspective. The authors compared human gameplay against a curiosity-driven agent [18] to solve the game. Although the performance of humans decreases after changes, for the agents, however, the game is not harder to learn, independently of the textures used. The results indicate that autonomous agents can learn much better than humans if we could remove prior knowledge.

Nevertheless, in our work, we are interested in analyzing the performance of an agent compared to other agents, not humans. This would be useful to evaluate DRL-based agents sensibility to texture changes, ignoring human learning characteristics completely. To do this, we are going to train the agents using a traditional method called Deep Q-Networks.

III. Background

Reinforcement Learning tasks are sequential decision problems in which an agent should take actions in an environment while trying to maximize the sum of received rewards. The agent analyzes the current state s in the set of states S and decides what action a in the set of actions A to take under a policy $\pi\colon S \times A \rightarrow \mathbb{R}$. The policy function gives the probability of taking action a given that the agent is in state s. If the policy is deterministic, we can say that the policy returns the action selected for a given state, that is $a = \pi(s)$.

Therefore, the goal of the agent is to find the actions that

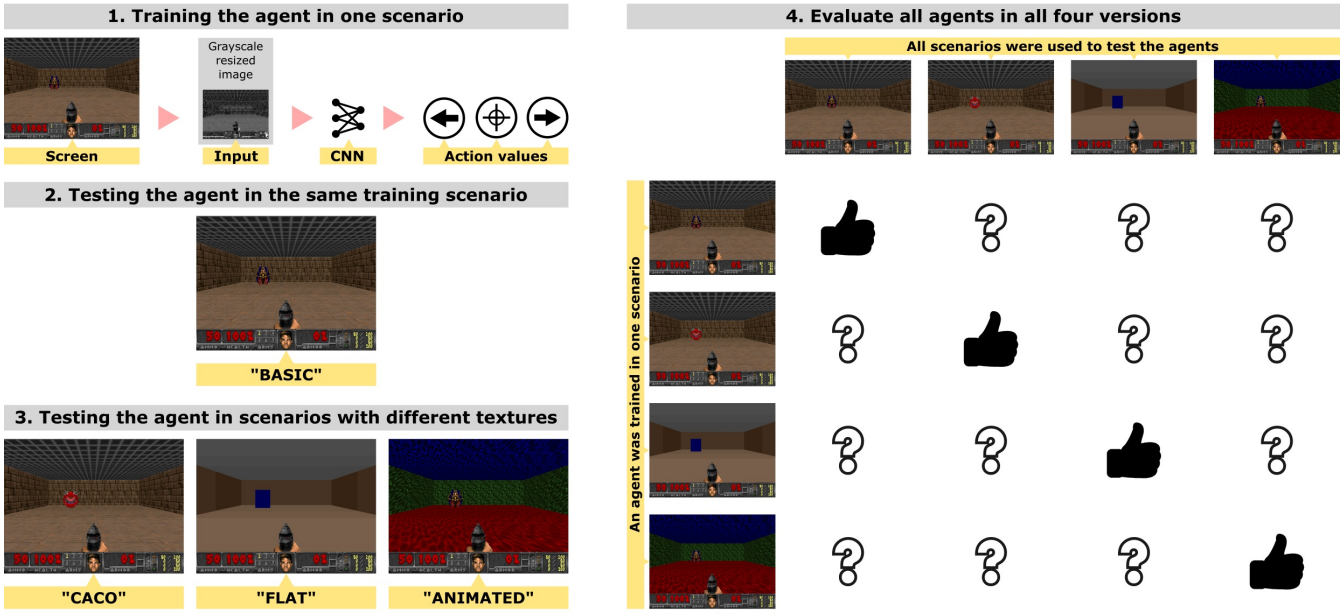

Fig. 1. Schematic view of the evaluation process. (1) One agent is trained in a single version using a CNN to learn the best action for every screen input. (2) The agent is tested in the same version it was trained. (3) Test the agent in the other scenarios. All four agents go through steps 1, 2, and 3, each one trained in a different version. (4) Finally, we evaluate the performance of all agents in all versions, already knowing that they will perform well in the scenarios they were trained.

maximize the expected sum of discounted rewards,

$$R_t = \mathbb{E}\Big[\sum_{i=t}^{T} \gamma^{\,i-t} r_i\Big]. \tag{1}$$

In this equation $T \in \mathbb{N}$ is the total amount of interactions, $r_i \in \mathbb{R}$ is the reward at iteration i, and $\gamma \in [0, 1)$ is the discount factor.

The discount factor γ not only ensures that (1) will converge eventually, but also determines how fast future rewards decay in accordance with the current reward, making immediate rewards more important than the later ones.

In this work, we use the DQN method [1], [2]. It combines traditional Q-Learning [19] with Deep Neural Networks as a function approximation for the values of states and actions experienced by the agent.

A. Q-Learning

Q-Learning [19] is a way of solving Reinforcement Learning tasks. It consists of the computation of a scalar value for each possible action for every state, which is called Q-value, $Q(s, a)$. This function is defined as the expected sum of rewards received from starting in state s and taking action a. It can be computed using the iterative process

$$Q(s, a) \leftarrow Q(s, a) + \alpha\,[y - Q(s, a)]\,, \tag{2}$$

where

$$y = r + \gamma \max_{a'} Q(s', a') \tag{3}$$

is called the target, r is the reward received, and α is the learning rate. This is the Bellman equation for Q-values [20].

The optimal Q-function, Q^*, gives the maximum value for each state, as defined by:

$$Q^*(s, a) = \max_{a} Q(s, a) \text{ for each } s \in S. \tag{4}$$

In practice, we do not have the exact Q-function. Instead, we approximate the Q-values in accordance with agent-environment interactions. After enough amount of interactions, $Q \to Q^*$.

The traditional approach to update the Q-value is to create a table with $|S|$ rows by $|A|$ columns. Each entry in the table is iteratively updated following the current values of (s, a). This solution is simple and converges to the optimal values [20]. However, when the number of states and/or actions is very large, using a table becomes too expensive. A solution is to use a function approximation.

B. Deep Q-Networks

One possible approximation is to use a neural network [21], [22]. The neural network with weights θ, $Q_\theta(s, a, \theta)$, approximates the Q-values $Q(s, a)$. The solutions of DRL use a Deep Neural Network, usually a CNN [1]. After training the network, the learned weights make $Q_\theta(s, a) \approx Q^*(s, a)$.

To adapt the tabular Q-Learning to use neural networks, we need to change (2) from an update rule to a loss function. The loss function of a parameterized value function Q_θ is given by the mean square error:

$$L_t(\theta_t) = \mathbb{E}\Big[(y_t - Q_{\theta_t}(s, a))^2\Big]. \tag{5}$$

Then we can compute the gradient of the above loss function

with

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}\Big[(y_t - Q_{\theta_t}(s,a))\nabla_{\theta_t} Q_{\theta_t}(s,a)\Big]. \quad (6)$$

With the loss function and its gradient defined, we can use backpropagation and Stochastic Gradient Descent to train the network.

C. Exploration vs. Exploitation

A crucial aspect of Reinforcement Learning is that the agent should favor the actions with higher values along the learning process, exploiting the best ones. However, it should also explore the environment to look for potentially better options than the ones currently considered. Although studied for decades [23], [24], this trade-off, widely known as the Exploration vs. Exploitation dilemma, is still an active research field with several methods trying to handle it [20].

In this work, we use an ε-decay strategy. The agent explores the environment with an initial probability ε_0 of taking random actions for the first episodes. Then, the value of ε decreases linearly until it reaches a final ε_T, which is maintained for the last episodes. At all times, ε should be greater than 0 to ensure that exploration is always possible. This annealing helps the learning process to explore more in the first interactions when the information was not properly evaluated, and favor exploitation when there is more information available.

D. Experience Replay

If we consider the input pairs (s, a) from the sequence in which they were generated, there would be a strong correlation between consecutive pairs. That would make the network to bias the learning process towards the expected sequences. We use a technique known as Experience Replay [25] to minimize this problem. Instead of learning directly from the state experienced, we store the current transition tuple (s, a, r, s) in a data structure called *replay memory*. When there are enough transitions stored, the network is updated using a minibatch of random samples from replay memory.

With inputs taken randomly, when the size of the memory is large enough, there is a small chance to give the network two consecutive transitions in sequence. This process effectively breaks the correlation between the samples. Another advantage of using a replay memory is that the same input could be used more than once to update the weights, increasing data efficiency. In this work, we use the DQN method with Experience Replay to train the agents in the experiments described below.

IV. Texture sensibility analysis

We created the following experiments to evaluate texture sensibility in DQN agents playing FPS games, that is, if they can perform properly when only the textures of a game are changed, and how much this change affects their performance. To test these cases, we compare the performance of the agent in one ViZDoom [14] scenario (Fig. 1.2) and in other three custom environments (Fig. 1.3), designed to emulate the characteristics of different game versions. Then, we compare the agents' performance on all versions (Fig. 1.4). The agent is composed of a Deep CNN, trained with no prior information (Fig. 1.1).

A. Environments

The agent was trained and evaluated in four different scenarios, which follow the same general gameplay. The scenario is a square room with only one enemy. It is static and remains in the same position after spawned. An episode starts with the agent on the opposite side of the monster. The agent always starts in the center of his side, while the enemy starts in a random position. Important aspects of the environment are described in the following.

1) Scenarios: Four different scenario versions are used to train and evaluate the agents. Each one is a variation of the standard Basic scenario. All scenarios are shown in Fig. 2.

Basic. This is the standard basic scenario present in ViZDoom. The monster, walls, ceiling, and floor have the default sprite versions which come by default in the library.

Caco. In the first alternative scenario, we want to simulate a different appearance of the monster. This is a very common technique used in games. The textures of the floor, ceiling, and walls remain the same. However, the monster texture was changed to Doom's original Cacodemon sprite.

Flat. To emulate an older, low-resolution version of the game we change all the sprites. Instead of more detailed textures, we have only flat colors for every part. The walls have a dark brown texture, the floor is light brown, the ceiling is gray, and the monster is a dark blue rectangle.

Animated. Finally, we want to emulate a new version of the game, in which the environment is more sophisticated. In it, all the floor, ceiling, and walls are animated textures with colored patterns. The monster, however, is the default version.

2) Action Space: The agent has three possible actions: move horizontally to the left, move horizontally to the right, and shoot. All of these actions are binary choices, the agent can perform an action or not, with no intermediate values. Thus, the full action space has eight possible discrete elements, comprising all possible combinations of the three binary actions.

3) Reward Distribution: Following the definition of Reinforcement Learning, as described in Section III, we evaluate the performance of the agent in accordance with the sum of received rewards. All the following values are already present in ViZDoom's default scenario, and we use them in all other versions. At every timestep, the agent receives a reward of -1. This negative score makes the agent try to kill the monster as soon as possible. For every shoot, the agent receives a reward of -5. A single shot is enough to kill the enemy. Thus, the agent also tries to kill the monster with a single shot. After killing the enemy, the agent receives a score of 100.

In an optimistic scenario, the monster spawns right in front of the agent, which shoots immediately. Since the agent did not move or missed a shot, its total score will be 95, discounting only the killing shot. However, if the agent moves randomly and shoots aimlessly, its total score can be a large negative

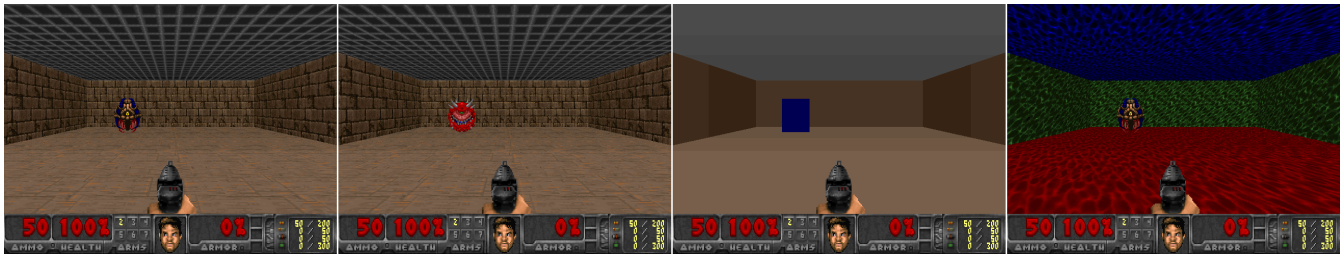

Fig. 2. All four scenarios used in this work. From left to right: Basic, the default scenario; Caco, which replaces the monster's sprite with its original version; Flat, in which the sprite textures are replaced by flat textures; and Animated, in which the static textures are replaced by animated sprites.

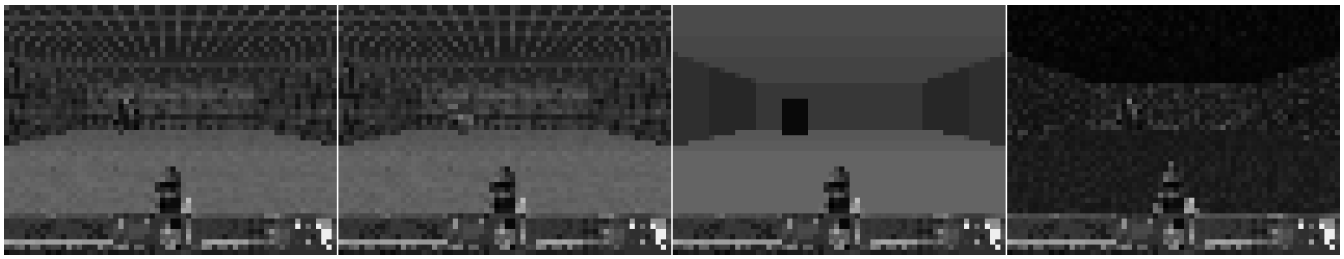

Fig. 3. Input samples. These are the agent's point of view of the same frames from Fig. 2.

number. In our experiments, we observed that a mean score of around 75 points indicates an agent with optimal behavior, i.e., immediately follows the direction of the monster and kill it with a single shot.

B. Neural Network settings

We based our settings in [26] since it obtained good results for the scenario used in this work. Although mostly the same, there are some modifications. For reproducibility reasons, all the settings are described below.

1) Inputs and outputs: The input is a grayscale image of size (64×48) pixels (Fig. 3). Usually, gray images are used to decrease the input size. In this work, however, we use grayscale images also to avoid a network bias towards the colors of them. Every pixel of the image is a floating-point value in the range $[0.0, 1.0]$, in which 0.0 is a pure black pixel, and 1.0 is pure white. The outputs are 8 neurons, one for each possible action in the action space, as described in Section IV-A2. Every output returns a float value that represents the action-value Q of each action.

2) Architecture: The network architecture is the same as in [26]. It consists of two convolutional (conv) layers followed by two fully connected layers. The first conv layer has 32 filters with a kernel of size (4×4) with stride (2×2). The second conv layer has 64 filters also with kernel a of size (4×4) and stride (2×2). In all conv layers, we use ReLU [27], [28] as the activation function, and the weights are initialized using Glorot uniform initialization [29]. Next, the output of the second conv layer is flattened into an array of 8960 neurons. They are fully connected with 512 neurons, which are fully connected with the 8 output neurons. The model uses Adam optimizer [30], and a mean squared error loss given by (5). The architecture is summarized in Fig. 4.

C. Hyperparameters

We defined an ε-decay strategy, as described in Section III-C. It starts with a purely random action selection, $\varepsilon_0 = 1.0$ for the first 10% epochs. Then, ε decays linearly for half of the epochs. Finally, $\varepsilon_T = 0.1$ for the last 40% epochs. The values of ε per epoch are shown in Fig. 5. We use a replay memory, as described in Section III-D, to hold the 10000 most recent samples. At every step, a minibatch of 64 samples is randomly retrieved from the memory and passed to the network. For the update rule of (2), we use a discount factor of 0.99 and a learning rate of 0.0001.

When an agent performs one action per frame, the difference between the consecutive frames is so subtle that almost identical states are saved in replay memory, which makes the learning process more difficult. To minimize this problem, the agent skips some frames before going to the next state [31]. This happens by choosing an action and repeating it through the next frames, and only then another action is chosen. In this work, we use a frame repeat value of eight.

D. Training Regime

Training is the stage in which the agent actually learns. In the experiments, we observed that 25 epochs is enough to stabilize the results. In each epoch the agent trains for 200 episodes. An episode ends when the monster is killed, or in timeout after 300 timesteps. After training, the agent is tested in another 500 episodes, to keep track of the learning process, and to check if its performance is increasing. During tests, the network is not updated and the agent performs the action with the highest Q-value for the current screen.

A very important adjustment to keep the fairness of the test, among all scenarios, is that the agents should be tested in the exact same episodes. In ViZDoom, we can achieve this by setting the seed of every episode. We created a list of seeds

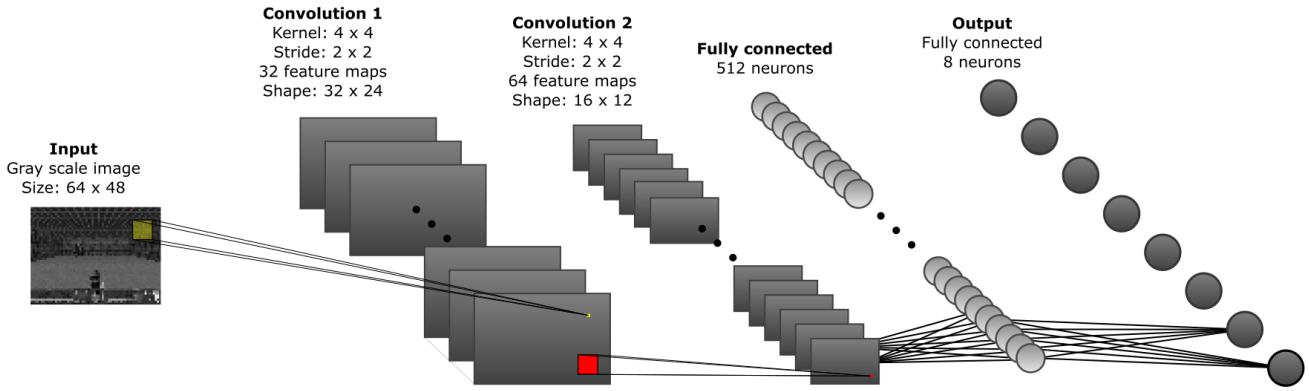

Fig. 4. Neural network architecture.

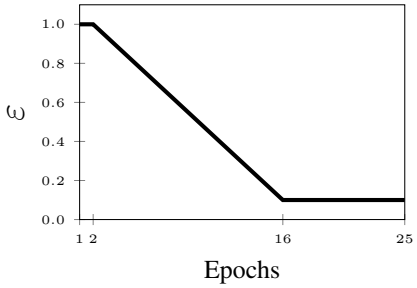

Fig. 5. ε-decay strategy: ε starts in 1.0 for the first 10% epochs, decays linearly for half of the epochs, until it reaches 0.1 for the last 40% epochs.

for test episodes, which is shared in all scenarios. The tests during learning are also made in the same scenarios.

V. Results and Discussion

In this section, we present the results obtained after training four agents in the scenarios described in Section IV-A1, and the results of evaluating them in all the scenarios they were not trained. We also describe and discuss the behavior of the agents in all of the tests.

A. Learning in a single scenario

First, we have to make sure the agents can learn in a single environment. There are three general behaviors to be learned. The first occurs when the monster is spawned directly in front of the agent. In this case, the agent only needs to shoot, killing the monster immediately. When the monster spawns to the left of the screen, the agent must move to the left and shoot when in front of the monster. Similarly, when the monster spawns on the right side, the agent must move to the right and shoot when in front of the monster. As presented in Section IV-A3, a mean score of around 75.0 indicates an optimal behavior.

All four agents were able to learn the expected behaviors after 25 epochs of 200 episodes. The agent trained in the Basic scenario achieved a mean score of 75.96. The agent trained in the Caco scenario achieved a mean score of 80.01. The agent trained in the Flat scenario achieved a mean score of 77.17. Only for the agent trained in the Animated scenario achieved a mean score of less than 75.0, achieving 70.33. However, even this agent learned to identify the monster, move to the correct side, and try to kill it as soon as possible. This lower result is probably due to the increased difficulty of understanding animated background textures.

B. Evaluation in other scenarios

1) Basic: First, we evaluate the agent trained in the Basic scenario when tested in the other three.

Caco. When tested in the Caco scenario, the agent trained in the Basic scenario achieves a mean score of 30.59. Although this is the second-highest score in a different evaluation environment, the behavior of the agent is not close to the optimal. The agent seems to randomly move left and right, shooting repeatedly, and only eventually hits the monster. The agent clearly cannot identify the enemy, however, in contrast to the other tests, it moves to both sides, which indicates that it at least recognizes the other elements of the map.

Flat. This is the only case in which an agent was able to succeed in a scenario with different textures than it was trained. The mean score of the Basic agent tested in the Flat scenario was 73.35. Not only the score indicates that the agent actually played well, but its behavior also confirms. As expected in a learned environment, the agent follows the enemy and tries to kill it with a single shot. This result indicates that, in some scenarios, we could reuse a trained agent and have a good performance.

Animated. The agent does not perform well in this test, it cannot understand the screen of the Animated environment. In every episode the agent just moves to the left, shooting constantly, and remains at the leftmost side. Since the enemy can spawn on the left side of the screen, the agent kills it sometimes. It achieves a mean score of -152.01, which agrees with its behavior. This scenario is indeed the most different and indicates that a generalization of performance with arbitrary textures is very difficult.

2) Caco: Now, we analyze the behavior and score of the agent trained in the Caco scenario.

Basic. This is one of the three cases with a mean score higher than zero, specifically 15.29. Not only the score is one

of the highest, but the agent's behavior also stands out. Similar to the opposite test, which achieved the second high score, this agent also moves alternately from left to right, shooting constantly. After a while, it goes towards the monster and kills it. Again, this behavior indicates that the agent seems to recognize the map, although it clearly cannot identify the monster.

Flat. In this test, the behavior of the agent varies. Usually, it goes to the left, but sometimes it just stands in the center of the screen, shooting repeatedly in both cases. It achieves a mean score of -103.22, showing that it can kill the monster. However, its behavior shows that the agent recognizes neither the enemy nor the background of the map.

Animated. A mean score of -233.39 already shows to us that the agent cannot kill the monster. In fact, the agent just keeps standing in the center of the screen without shooting at all. We can then conclude that the agent had problems with the animated background and also could not identify the enemy. This pattern is common for the agents tested in Animated. It shows that the change from static background to moving ones is not dealt properly by the agents.

3) Flat: Next, we present the results of the agent trained in the Flat scenario.

Basic. Similar to the last test case described above, the agent remained in the center, and again without shooting. The mean score of the agent was very low, -239.87, which happens when the agent cannot kill the enemy in almost any episode. This indicates that the agent could not understand the scenario. This result goes against a possible generalization of learning since the Flat textures were based in the Basic scenario.

Caco. Again, the agent just stood still and without shooting, and with a very low mean score, -240.92. Since the background textures are the same as the previous test case, this result corroborates the hypothesis that the agent could not understand them. And similar to the previous case, the agent also could not identify the monster.

Animated. For the third time, the agent trained in the flat scenario just keep itself standing in the center of the screen. However, in this test, it keeps shooting repeatedly, which implies a slightly higher mean score of -224.08. Still, remaining in the starting position is a strong indication that the agent could not understand the input.

4) Animated: Finally, we evaluate the performance of the agent trained in the Animated scenario.

Basic. Among the tree evaluations of the agent trained in the Animated scenario, this one presents the highest score of -67.57. A score below 0 indicates an agent unable to kill the monster regularly, which is also true here. Usually, the agent alternate moves to the left and the right, always shooting. However, in some episodes it just keeps standing in the center, shooting repeatedly. Thus, the agent could not identify the enemy and had trouble when dealing with a very different background.

Caco. The behavior of the agent in this test is a little odd. At the start of the episode, it remains in the center of the screen, shooting constantly. Then, it moves to the right, still shooting. Finally, it gets stuck on the rightmost side of the screen. With this behavior, the agent kills the enemy sometimes, achieving a mean score of -117.82, but clearly unable to understand the input screen.

Flat. In this evaluation, an agent trained with animated textures is presented with flat ones. The difficulty of presenting a suitable behavior also appears here. The agent moves randomly from left to right, shooting repeatedly, without showing signs of identifying the enemy. In the end, the agent achieves a mean score of -104.91.

C. Discussion

All four agents were able to learn and perform well in their trained scenarios. The agent trained in Animated textures was the only one with sub-optimal performance, which indicates that using animated textures increases the learning difficulty. All other three agents achieve optimal behavior in their respective scenarios.

When tested in different scenarios, only the agent trained in the Basic scenario and tested in Flat textures was able to perform as expected, moving towards the enemy and trying to kill as soon as possible. In no other test it showed any indication of being able to fully understand the inputs, which include the enemy and the background textures. All scores are condensed in Table I.

TABLE I. MEAN EVALUATION SCORE.
ROW: TRAINED SCENARIO. COLUMN: EVALUATED SCENARIO.

	Basic Test	**Caco Test**	**Flat Test**	**Anim. Test**
Basic Train	**75.96**	30.59	**73.35**	-152.01
Caco Train	15.29	**80.01**	-103.23	-233.39
Flat Train	-239.87	-240.92	**77.17**	-224.08
Anim. Train	-67.57	-117.82	-104.91	**70.33**

When trained in the Flat scenario, the agent just remains standing in the center of the screen. In two of the tests, it did not perform any action, while in one case it shot continuously. These three tests show that the agent when trained in simple flat color textures is not able to understand any other texture, not even the ones in which they were based on, the Basic scenario.

The Animated scenario is the most difficult and all agents trained in other scenarios had trouble with it. In the case of the agent trained in the Basic scenario, it moved to the leftmost side of the screen and in the other two tests the agent remained at the center. Indeed, this was the most challenging scenario, due to the animated versions of the background, in both training and evaluation of the agents.

Caco and Basic scenarios results showed the agent moving to the left and to the right, which was uncommon. This indicates that the agent could have recognized the background, which shared the same textures but had trouble identifying the enemy. If the agent could identify the monster, the agent could move to kill it, which does not happen. And if it did not understand the scenario at all, it would probably remain in the center or move to a side of the screen, as in most of the tests.

VI. Conclusion

Although there are prior works that change the texture of the environment using DRL-based agents, they do not perform further analysis of its impact on the performance of the agents when presented with unseen textures. In this work, only a single agent was able to play another scenario version with good performance. This indicates that even in the simplest cases a CNN-based autonomous agent is not able to generalize its understanding of inputs from different textures consistently. For example, although there is a penalty for every shot in all environments, in most evaluations the agent kept shooting continuously. These results of DRL follow the ones obtained in classification tasks presented in Section II which are very sensitive to input changes. Thus, to ensure proper behavior, an agent needs to be retrained not only if the problem changes, but also when only the visual aspects are modified.

Now, we address the questions raised in Section I directly. First, although all four agents are able to learn suitable behaviors in their respective trained scenarios, in general, they did not have a good performance in versions with different textures. Answering the second question, we also observed that in a single case the agent understood the inputs, but in all other eleven tests changing the textures made the performance much worse. Finally, the results differ by a large margin when comparing different training scenarios, which lowers confidence in trying to avoid retraining. These results indicate that, to achieve a proper performance after texture changes, a CNN-based autonomous agent trained using DQNs should be retrained.

As future works, we want to look further into how the agents learn from screen inputs and to evaluate if training with different textures could be a starting point for the network weights. We also want to look for new insights in how the screen representation affects learning and performance in unseen texture in known environments, and to make the network discard the unnecessary information. For example, comparing Basic and Animated scenarios, the network should focus only on the monster. Since the monster has the same pixels, the performance should be the same in both scenarios, which does not happen.

Acknowledgment

The authors would like to thank the Instituto Atlântico, the Department of Computing (DC), and the Teleinformatics Engineering Department (DETI), both at the Federal University of Ceará (UFC), for their support.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv e-prints*, pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[3] H. van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 5398–5408.

[4] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," *ArXiv e-prints*, pp. 1–30, 2020. [Online]. Available: https://arxiv.org/abs/2003.13350

[5] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 2140–2146.

[6] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *ArXiv e-prints*, pp. 1–21, 2019. [Online]. Available: https://arxiv.org/abs/1911.08265

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[8] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *ArXiv e-prints*, pp. 1–19, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[9] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *International Conference on Learning Representations (ICLR)*, 2020, pp. 1–28. [Online]. Available: https://arxiv.org/abs/1909.07528

[10] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *ArXiv e-prints*, pp. 1–66, 2019. [Online]. Available: https://arxiv.org/abs/1912.06680

[11] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1724-z

[12] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, p. 828–841, Oct 2019.

[13] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," in *International Conference on Learning Representations (ICLR)*, 2019, pp. 1–22.

[14] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based AI research platform for visual reinforcement learning," *ArXiv e-prints*, pp. 1–8, 2016. [Online]. Available: http://arxiv.org/abs/1605.02097

[15] D. S. Chaplot, G. Lample, K. M. Sathyendra, and R. Salakhutdinov, "Transfer deep reinforcement learning in 3 d environments: An empirical study," in *30th Conference on Neural Information Processing Systems (NIPS)*, 2016, pp. 1–9.

[16] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Autonomous quadrotor landing using deep reinforcement learning," *ArXiv e-prints*, pp. 1–8, 2017. [Online]. Available: https://arxiv.org/abs/1709.03339

[17] R. Dubey, P. Agrawal, D. Pathak, T. L. Griffiths, and A. A. Efros, "Investigating human priors for playing video games," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 1–9.

[18] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017, pp. 1–12.
[19] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf
[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.
[21] G. Tesauro, "Temporal difference learning of backgammon strategy," in *Proceedings of the Ninth International Workshop on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, p. 451–457.
[22] ——, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.
[23] J. C. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 41, no. 2, pp. 148–177, 1979.
[24] S. B. Thrun, "Efficient exploration in reinforcement learning," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1992.
[25] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1993, uMI Order No. GAX93-22750.
[26] P. B. S. Serafim, Y. L. B. Nogueira, C. A. Vidal, and J. B. Cavalcante-Neto, "On the development of an autonomous agent for a 3d first-person shooter game using deep reinforcement learning," in *Anais do XVI Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)*, 2017, pp. 477–483.
[27] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, pp. 947 EP –, Jun 2000. [Online]. Available: http://dx.doi.org/10.1038/35016072
[28] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323.
[29] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.
[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015, pp. 1–15.
[31] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.