Assessing the Robustness of Deep Q-Network Agents to Changes on Game Object Textures

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

Yuri Lenon B. Nogueira
Department of Computing
Federal University of Ceará
Fortaleza, Brazil
yuri@dc.ufc.br

Joaquim B. Cavalcante-Neto
Department of Computing
Federal University of Ceará
Fortaleza, Brazil
joaquimb@dc.ufc.br

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

***Abstract*—The research in autonomous agents aspires to achieve Artificial General Intelligence, where agents, like humans, are able to understand concepts and learn how to solve tasks. We would like to observe this ability on game agents as well. Recent research on autonomous agents for game playing uses a combination of Deep Neural Networks and Reinforcement Learning algorithms. Commonly, Neural Networks present vision-based models, usually Convolutional Neural Networks (CNN). However, those models can undergo performance degradation when dealing with different pixel patterns, an issue that also happens with vision-based autonomous agents in games. Prior works have shown that CNN-based autonomous agents cannot reproduce the behavior learned in one scene when they are placed into a brand new version with different textures. In this work, we evaluate whether the agents educe high-level elements, such as enemy, foreground, and background. Instead of testing the agent in a completely different scene, we designed two experiments based on slight changes. In the first experiment, we change only a subset of the game objects. In the second experiment, the agents play in an interpolated version of two scenes. Even when changing only a single game object texture, the agents are not guaranteed to present good behavior. We show that, depending on the training scenario, the agents are not fully robust to generalize a high-level concept of game objects.**

***Index Terms*—autonomous agents, deep reinforcement learning, digital games, first-person shooter games**

I. Introduction

In the search for Artificial General Intelligence (AGI), we would like to develop autonomous agents capable of recognizing high-level concepts inside their testing environments. For example, in vision-based tasks, humans can easily identify objects. That ability helps us to understand what we are seeing. We do not look at a forest and see green and brown dots, we see trees. The modern vision-based models used in autonomous agents are also interested in that kind of semantic segmentation.

In the last decade, we witness important advances concerning agents in the domain of games. First, autonomous agents learned how to master several Atari games [1], [2]. Now, the agents can master all Atari games [3]. The defeat of the world's champion of the highly complex table game Go by an autonomous agent was greatly publicized [4]. That agent was later generalized to play Go, Chess, and Shogi [5]. The agents were also able to achieve high performance in 3D games like ViZDoom [6]. And, more recently, highly complex competitive multiplayer games, like Dota [7] and StarCraft II [8].

Autonomous agents developed in those works have one feature in common: they are all Deep Reinforcement Learning (DRL) agents. As the name suggests, this method uses Deep Neural Networks combined with Reinforcement Learning algorithms. One of the most common types of Neural Networks for vision-based tasks, like the examples presented above, is Convolutional Neural Network (CNN).

Since we are interested to move towards AGI, we would like to develop agents capable of segmenting the visual aspects of the games in a way similar to humans. Today, we know that CNNs can present some limitations, which lowers our confidence in the current methods. For example, CNNs are very sensitive to specific changes, which could lead from good to poor performance when changing only a single pixel [9]. Prior works have also shown that CNNs used in DRL are not robust to texture changes in games as well [10]. However, as far as we know, nobody is aware of the full extent of this limitation.

In this work, we evaluate the limitations of DRL agents when facing game objects with different textures. To assess the magnitude of those issues, we answer four questions: (i) is the agent able to maintain its performance when we change the texture of a single object? (ii) is there a correlation between the performance of the agent and the game objects whose textures were changed? (iii) is the amount of performance degradation related to the number of pixels changed? (iv) can we observe any pattern of performance degradation when the textures are changed gradually?

To answer those questions, we extend the work of Serafim *et al.* [10] to change the textures of all game objects. In [10], four agents were trained in four different scenes of ViZDoom [11], a Doom-based First-Person Shooter research platform, and tested in different scenes. This was enough to verify that the agents are not able to generalize their behavior. However, to better understand this limitation, we designed more elaborated experiments, in which each agent is tested in different scene configurations.

We designed two distinct experiments. The first one tests the agents in scenes where at least one game object texture was replaced with a different version. Running this experiment

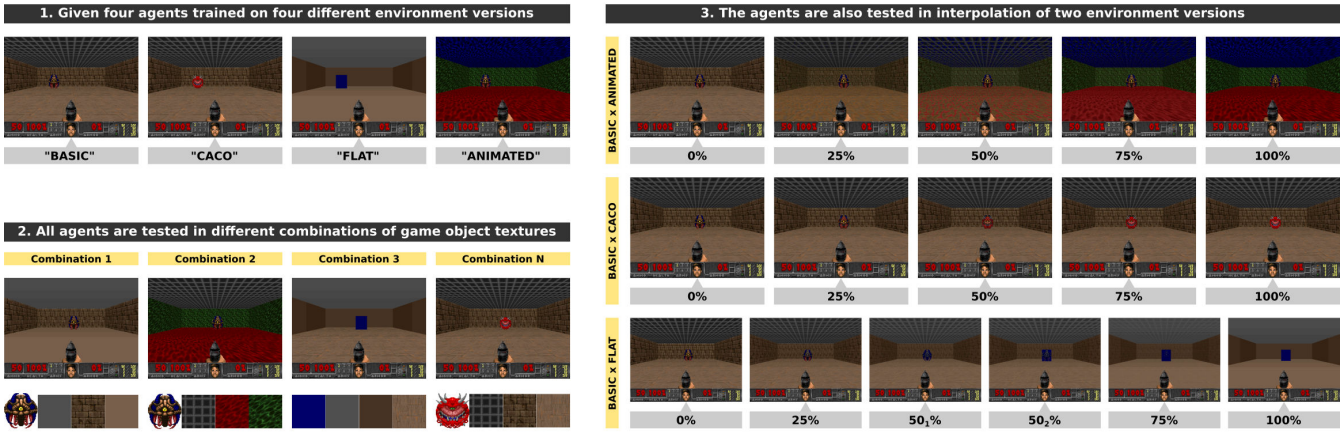


Fig. 1. Summary of the experiments performed. (1) Four agents were previously trained in four different versions of a ViZDoom environment. (2) To evaluate agents' performance with unseen versions of the same game objects, we test them across all possible texture combinations. (3) To verify if the performance degradation follows any pattern, the agents are tested in scenes with an interpolation of pixel values.

with all possible combinations of texture changes, we are able to answer questions (i) and (ii). To answer questions (iii) and (iv), we created custom scenes by interpolating two of the four original scenes. With the interpolations, we can evaluate the gradual performance changes of each agent. These experiment processes are summarized in Fig. 1.

We show that, depending on the scene where the agents are trained, they present a different sensibility to single game object texture changes. Therefore, the choice of the training scene can have a huge impact on the robustness of the model. Moreover, we observed that the agents present a performance degradation pattern that follows the number of changes in pixel values. The results would assist us to encounter specific model improvements that can support the future development of agents fully robust to texture changes.

This paper is organized as follows. In Section II, we present the three works that inspired the creation of the experiments presented in this paper. In Section III, we present a brief description of DRL. In Section IV, we present the environment, test scenes, model parameters, and describe the configurations of the performed experiments. In Section V, we present the results and discuss them. Finally, in Section VI, we present closing remarks and propose future works.

II. Related Works

In this section, we present three works that evaluate the performance of DRL autonomous agents when placed in environments with texture changes. They all inspired the development of this work and the designed experiments.

Chaplot and Lample [12] used a general approach to train an agent on ViZDoom environments with random textures. After the training is done, the agent is tested with unseen random textures on the same map. When training with a low amount of randomness, the agents do not present a robust performance. However, after training with several random textures, the agents are not affected by texture changes anymore when tested. Although effective, this strategy relies on the fact that multiple random textures are available to the training phase. In our work, we would like to consider a more human-like learning, evaluating whether or not an agent can generalize its behavior training in a single scene with given textures.

Moreover, the authors do not perform further analysis of the random texture training and its impact on the results. For example, we do not know how each texture change impacts the learning, which could be useful to reduce the necessary training data. In this paper, we try to evaluate those questions by testing all combinations of textures and also creating interpolation versions of two given scenes.

Dubey *et al.* [13] make a series of ablation studies on the impact of human priors on the performance of an agent. The authors compared the results obtained by agents and humans in different versions of a game. They create a 2D platform game and modified its textures increasing the difficulty to the human players. The goal of the paper was to investigate the importance of human priors in performance on gameplay. They found that the agents can learn the game with the same effort, regardless of texture version. However, the human players' performance decreased significantly when the textures were not related to known objects.

The results indicate that prior human knowledge is not a factor for the agents. Therefore, they could learn much easier than human players when there is no visual aspect to indicate the role of each game object. Instead of comparing the performance of agents and human players, we want to evaluate the ability of agents when trained and tested with different texture versions. This could help us to assess the robustness of DRL autonomous agents to brand-new versions of the same game objects.

Serafim *et al.* [10] present the closest work to our proposal. In fact, our evaluation is an extension of their experiments, in which they tested four custom versions of a ViZDoom environment. First, they trained one agent in each version.

Then, all agents were tested in all scenes. Therefore, if the agents were robust to texture changes, all of them would present good performance in all tested cases.

The authors found that a CNN-based agent cannot generalize its behavior for the scenes in which they were not trained. Considering the twelve test cases in different scenes, only one agent was able to succeed. Their experiments indicate that the agents are not able to generalize what they learned. However, to have more insights into how to construct better agents, we have to investigate what the agents can understand or not. In our work, we want to expand those findings and evaluate whether or not the agents can focus on specific game objects, which is not possible when testing in a completely different scene. The results could lead us to appropriate changes in the agents' settings towards the direction of achieving generalization. Another important aspect to highlight is that the authors did not consider possible internal correlations of the scenes in order to find any possible pattern in the performance difference, which we also evaluate in the following sections.

III. Background

In this section, we describe the methods used to allow the agent to learn good behaviors through interaction with the environment.

A. Reinforcement Learning

Reinforcement Learning is an area of Machine Learning that presents techniques to solve interactive problems based on trial-and-error. Unlike Supervised Learning, Reinforcement Learning methods do not rely upon a correct answer, instead, we have a response from a critic. Moreover, unlike Unsupervised Learning, we do have a response, but in the form of a scalar signal, called reward.

The classic interaction dynamics involve two main entities: an agent and an environment. The agent receives the current state, S_t, from the environment, executes an allowed action, A_t, obtains a reward, R_{t+1}, and goes to a new state, S_{t+1}, with a given probability, P_{t+1}. Formally, Reinforcement Learning is typically stated as Markovian Decision Processes, defined by the state space, S, the action space, A, the probability distribution, $P : S \times A \times S \rightarrow [0, 1]$, and the reward distribution, $R : S \times A \times S \rightarrow \mathbb{R}$.

In Reinforcement Learning, the goal is to find a function that returns the probability of executing each possible action for every state. This function is called policy, $\pi : S \times A \rightarrow [0, 1]$, which effectively describes the behavior of an agent. Ideally, we would like to find the optimal policy, π_*, for each problem.

Generally speaking, in order to achieve good behavior, the agent has to try all possible states and actions. In fact, several Reinforcement Learning algorithms are only guaranteed to find an optimal policy if every state is visited at least once. In practice, however, most methods will converge to the optimal policy if the agent visits all states multiple times [14].

A fully greedy agent, with a deterministic policy, would follow the same path every time, which prevents it to test all possibilities. In this case, we say that the agent is exploiting the environment. However, we would like to observe the agent visiting all paths. In other words, the agent should explore all the possibilities. Thus, we must ensure that the agent always has a positive probability of executing every action.

There are several exploration strategies, but the most commons are the ones called ε-strategies [14]. In those approaches, the agent has a big probability of choosing the greedy action and a small probability of choosing a random action, which is enough to allow a full environment exploration. The agent used in this paper was trained with an ε-decay strategy. In the first epochs, the knowledge of the environment is still unknown, therefore, it always executes a random action. Then, the probability of executing a random agent decays while the policy is converging. In the last epochs, the agent executes the greedy action with a high probability.

B. Q-Learning

Q-Learning [15] is the Reinforcement Learning algorithm used in this work. In its traditional definition, Q-Learning uses a tabular representation to relate the state-action pairs with the rewards. Each entry in the table stores the current Q-values, $Q_\pi(A_t \mid S_t)$, the expected total reward after executing A_t in S_t and then following π. After each iteration, the Q-values are updated according to the rule that

$$Q_\pi(A_t \mid S_t) \leftarrow Q_\pi(A_t \mid S_t) + \alpha\Big[y_t - Q_\pi(A_t \mid S_t)\Big], \quad (1)$$

where

$$y_t = R_{t+1} + \gamma Q_\pi(A_{t+1} \mid S_{t+1}), \quad (2)$$

and γ is the discount factor. Note that $0 \leq \gamma \leq 1$, such that γ decreases the value of future rewards, which gives more importance to immediate rewards and ensures the convergence in continuous tasks.

As the action and state spaces grow, the table size grows exponentially. Therefore, using a tabular method becomes unfeasible. One alternative to this problem often used recently is the approximation of Q-values in the form of Deep Neural Networks.

C. Deep Q-Networks

Using Neural Networks with Reinforcement Learning algorithms is not a new approach [16]. However, in the last years, with the popularization of Deep Neural Networks, a novel method called Deep Q-Networks was developed using Convolutional Neural Networks and Q-Learning to solve several Atari 2600 games [2].

Instead of using a table to store the Q-values, the Deep Neural Network with weights θ approximates $Q_\pi(A_t \mid S_t)$. After training, $Q_{\theta_t}(A_t \mid S_t) \approx Q_{\pi_*}(A_t \mid S_t)$. To train the model using Stochastic Gradient Descent, we need the loss function $L_t(\theta_t)$ and its gradient $\nabla_{\theta_t} L_t(\theta_t)$. The loss is a version of the update rule (1) and they are defined such that

$$L_t(\theta_t) = \mathbb{E}\Big[(y_t - Q_{\theta_t}(A_t \mid S_t))^2\Big] \quad (3)$$

and

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}\Big[(y_t - Q_{\theta_t}(A_t \mid S_t))\nabla_{\theta_t} Q_{\theta_t}(A_t \mid S_t)\Big]. \quad (4)$$

One problem with the use of Neural Networks is that while the agent is playing the game, the frames are closely related. This causes the Neural Network weights to change towards the current images, essentially "forgetting" the past experiences. To break this correlation, Deep Q-Networks use Experience Replay [17]. The frames are first stored in a structure called Replay Memory, then a sample from the memory is taken and passed as input to the model. Since the Neural Network is now being updated with images from different interaction moments, which are not correlated, the trained model is robust to frame changes.

IV. Methodology

Inspired by the works presented in Section II, we designed two experiments to evaluate the extension of the limitations of the agents when faced with different textures of the same game objects in which they were trained. We use the agents developed by [10] and create custom versions of ViZDoom's scenes [11] to execute the tests.

A. Environment

The agents were trained in four different scenes: the original Basic environment from ViZDoom [11] and three variations of it. All versions follow the same general gameplay as the Basic version. The environment is a four-side room with only one static monster enemy. An episode starts with the agent on the opposite side of the enemy. The agent always starts at the center, while the monster starts at a random position. After being spawned, the monster will not move until the end of the episode. Next, we describe the specific scenes' characteristics.

1) scenes: Each scene is a variation of the standard Basic scene. All of them are shown in Fig. 1-1.

Basic. It is the standard basic scene present in ViZDoom. All sprite textures are the default monster, ceiling, walls, and floor versions from the library.

Caco. It is a scene in which the textures used on the ceiling, on the walls, and on the floor are identical to those on the Basic version, but the monster's texture was changed to the original Doom's Cacodemon sprite. This scene simulates a different visual appearance of an enemy with the same behavior.

Flat. In this scene, all textures are replaced by flat colors for every game object. The ceiling is gray, the walls have a dark brown texture, the floor is light brown, and the monster is a dark blue rectangle. This scene simulates an older, low-resolution version of the environment.

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

2) Different Textures: There are exactly three texture versions for each of the four game objects.

Monsters. The original monster from ViZDoom is used in Basic and Animated scenes (Fig. 2a). The Flat version is a full dark blue sprite (Fig. 2b), which relates with the blue patches in the original version. The Caco version is mostly red (Fig. 2c) and very different from the Basic monster.

(a) Basic and Animated. (b) Flat. (c) Caco.

Fig. 2. Monster texture used in each scene.

Ceiling. The Basic and Caco ceiling have a gray grid pattern (Fig. 3a). The Flat version is a purely gray texture (Fig. 3b). The Animated version presents blueish patches (Fig. 3c).

(a) Basic and Caco. (b) Flat. (c) Animated.

Fig. 3. Ceiling texture used in each scene.

Walls. The Basic and Caco walls have a brick texture (Fig. 4a). The Flat version is a purely dark brown texture (Fig. 4b). The Animated version presents greenish patches (Fig. 4c).

(a) Basic and Caco. (b) Flat. (c) Animated.

Fig. 4. Wall texture used in each scene.

Floor. The Basic and Caco floor have a wooden texture (Fig. 5a). The Flat version is a purely light brown texture (Fig. 5b). The Animated version presents reddish patches (Fig. 5c).

(a) Basic and Caco. (b) Flat. (c) Animated.

Fig. 5. Floor texture used in each scene.

3) Action Space: There are three possible actions: move horizontally to the left, move horizontally to the right, and shoot. ViZDoom considers an action as a binary value. Therefore, at each step, the output should tell us what actions should be executed or not. In the environment used in this work, the action space comprises eight discrete combinations of three binary choices.

4) Reward Distribution: We use the sum of rewards received in each step to evaluate the performance of the agents. The reward values are already defined in ViZDoom and in the

custom scenes. The agent receives a score of -1 at every step, which encourages it to kill the monster faster. When the agent shoots, it receives a score of -5. Since a single shot is enough to kill the monster, the agent will also be encouraged to kill it by shooting only once. When the monster is killed, the agent receives a score of 100.

B. Neural Network Settings

We use the trained agents made available by [10]. Therefore, the agents have the same settings. Although we do not train the Neural Network, we briefly describe its settings in this section, which also helps with the reproduction of our work.

1) Inputs and outputs: The network input is a grayscale image of (64×48) pixels. Besides the reduction in the input size, the use of grayscale images also helps to avoid a network bias towards the pixel colors. Every pixel 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. Following the action space described in Section IV-A3, there are 8 output neurons, one for each combination of actions. Every output returns a float value that represents the action-value Q of the corresponding action.

2) Architecture: The first two layers of the Neural Network are convolutional (conv). The first conv layer has 32 filters, a kernel of size (4×4), and a stride size of (2×2). The second conv layer has 64 filters, also with a kernel of size (4×4) and stride of size (2×2). In all conv layers, the activation function is ReLU [18], [19] with weights initialized using Glorot's uniform initialization [20]. The output of the second conv layer is flattened into an array of 8960 neurons, which are fully connected with 512 neurons. These neurons are fully connected with the output of 8 neurons. The model uses Adam optimizer [21] and a mean squared error loss given by (3). An illustration of the architecture is shown in Fig. 6.

C. Experiments

We developed two kinds of experiments to better understand the behavior of the agents. They were designed to answer the four questions proposed in Section I.

1) Combinations: To evaluate if the agents can maintain the same performance when changing specific textures, not the full scene, we create custom scenes combining all game objects with different textures. For example, one scene combines the monster and the ceiling from the Basic version with the walls and the floor from the Flat scene. This experiment will also tell us whether there is any correlation of the game objects with the agent's performance. If any game object has a higher impact on the agent's performance, we will see it in the results.

In the case of Basic and Caco, the only possible combination is the full Caco scene (Fig. 1-1). There are 14 possible combinations of Basic and Flat scenes, which are presented in Fig. 7. For the Basic and Animated combinations, there are six possible combinations, as shown in Fig. 8. This adds up to a total of 24 tested combinations, including the original versions of the four scenes.

2) Interpolation: We created the second part of the experiments making an interpolation of each custom scene with Basic. The interpolation uses a percentage of the pixel values from the Basic scene with the complementary percentage of pixel values coming from the other scene. For example, in an interpolation version of Basic and Animated with 25% of Animated, 75% of every pixel value comes from the Basic scene, and 25% of the values come from the Animated scene.

ViZDoom does not accept translucent pixels. Therefore, when interpolating non-overlapping pixels, they will have the color of the front texture or fully transparent. This is not apparent in the Basic and Caco interpolation, since the textures of both monsters are similar. However, when interpolating the original monster with the blue square texture of the Flat version, the difference is clear. For the 25% Flat interpolation, the majority of transparent pixels of the Basic version make the border around the transparent monster. Analogously, the majority of pixels from the Flat texture in the 75% version creates a blue border around the monster. For the 50% version, in which the proportion of transparency and opacity is equal, we decided to create two versions: one without a blue border and another with a blue border.

All interpolations of Basic and Caco are shown in Fig. 9. There are six interpolations of Basic and Flat, counting the

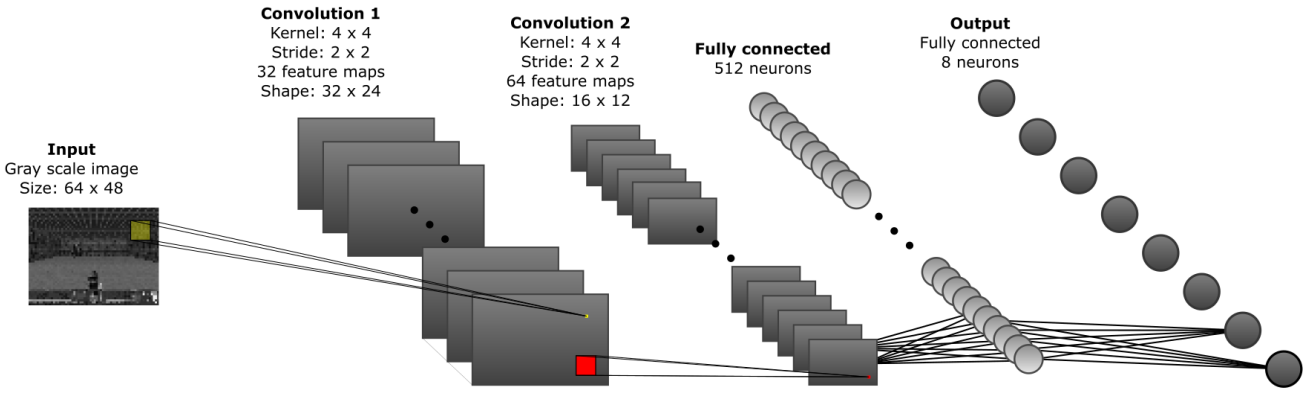
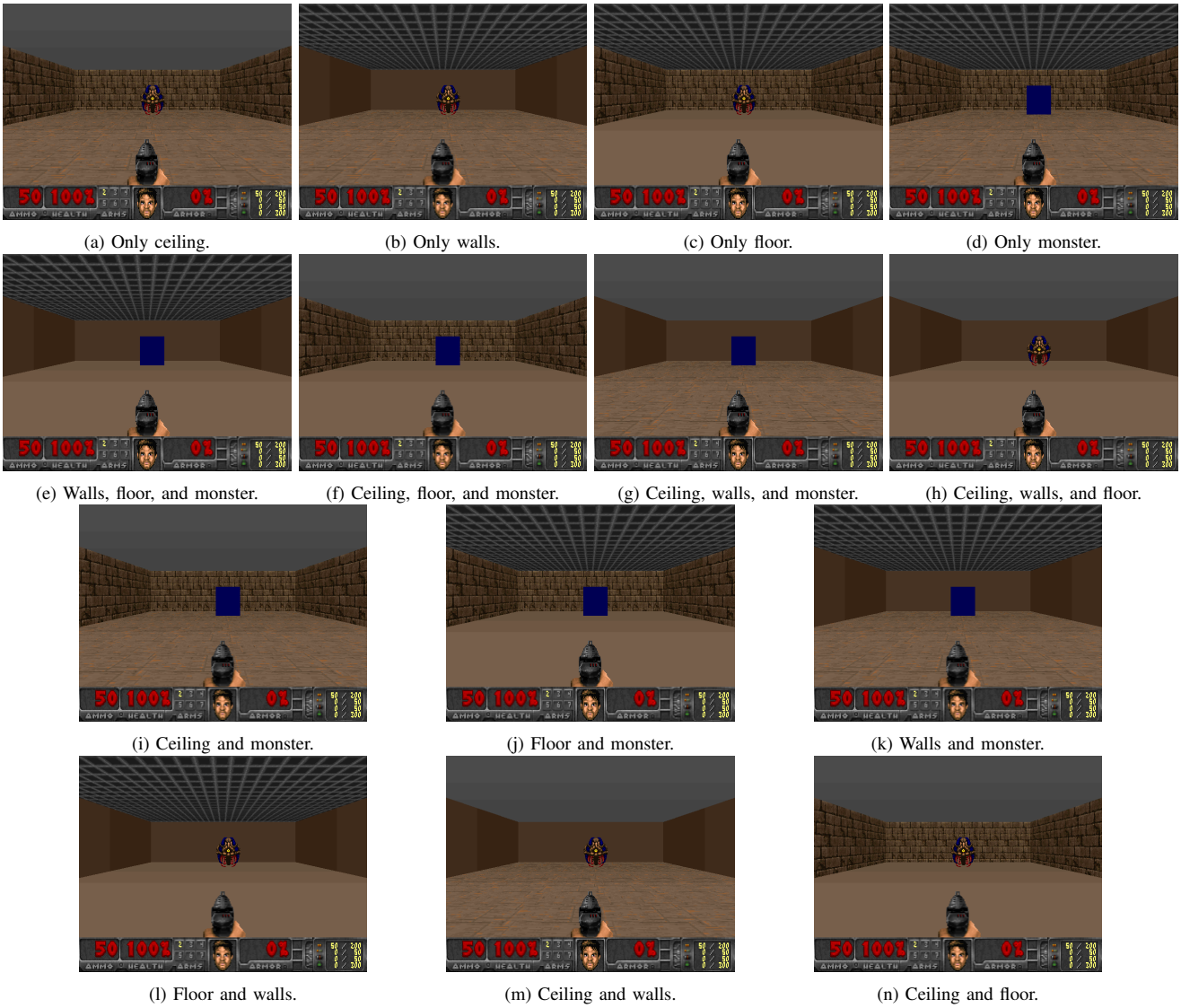


Fig. 6. Neural Network architecture. Reproduced with authorization of Serafim *et al.* [10].



(a) Only ceiling. (b) Only walls. (c) Only floor. (d) Only monster.

(e) Walls, floor, and monster. (f) Ceiling, floor, and monster. (g) Ceiling, walls, and monster. (h) Ceiling, walls, and floor.

(i) Ceiling and monster. (j) Floor and monster. (k) Walls and monster.

(l) Floor and walls. (m) Ceiling and walls. (n) Ceiling and floor.

Fig. 7. All combinations of Basic and Flat scenarios.

two versions of 50%, one without blue borders, and another with blue borders, which are presented in Fig. 10. The five interpolations of Basic and Animated are shown in Fig. 11.

D. Testing Regime

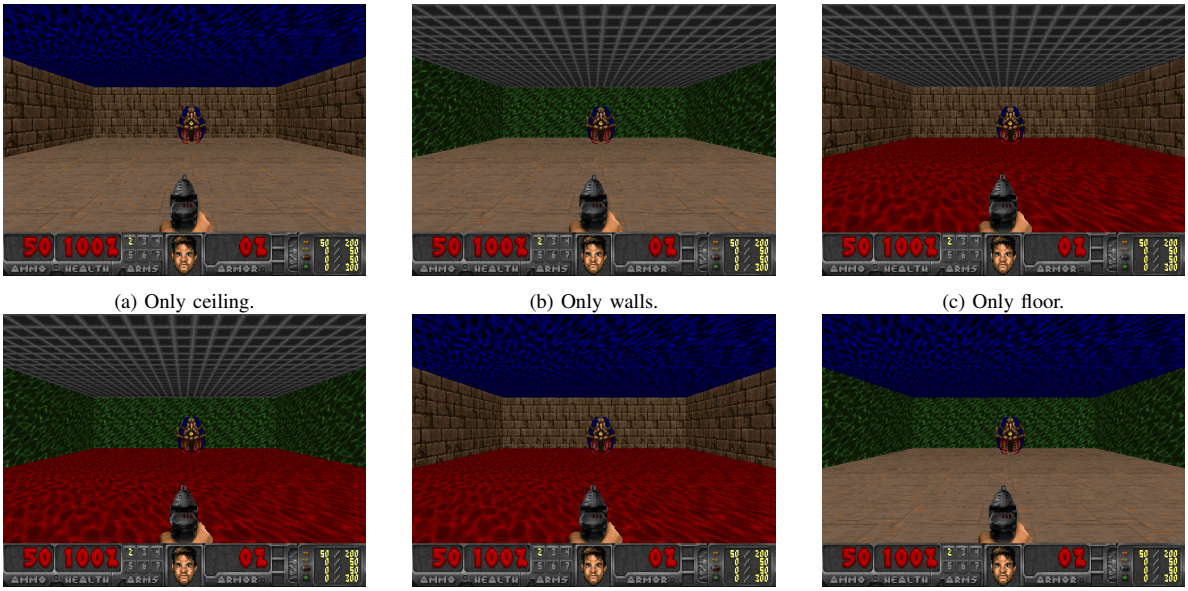
All agents are tested for 200 hundred episodes and the presented results are the average scores. In each episode, the monster appears in a different position. However, to ensure a fair comparison between the agents, the initial positions are the same across all the experiments. We can guarantee that all agents face the same problem using a list of episode seeds, which is shared among all agents independently of the scene.

E. Running an agent

The frame returned by ViZDoom is preprocessed and passed to the Neural Network. The model is executed and, for each input, the agents choose the action with the highest Q-value. Note that all the agents were already trained, thus there is no model update. Since the difference of consecutive frames is very subtle, we repeat an action for four frames, which increases the testing speed without any noticeable performance loss. Then, the action is executed and this process is repeated until the end of the episode.

V. Results and Discussion

This section presents the scores of each agent in all tested cases. We also discuss the results that presented the most notable behaviors. Note that when the monster appears in the middle of the screen and the agent shoots immediately, the total score will be 95. If the agent does not move or shoot at all, the total reward will be -240. If the agent moves randomly and shoots aimlessly, the score will be a large negative value, depending on whether or not it killed the monster. In the worst case, the agent receives a score of -345.



(a) Only ceiling. (b) Only walls. (c) Only floor.

(d) Floor and walls. (e) Ceiling and floor. (f) Ceiling and walls.

Fig. 8. All combinations of Basic and Animated scenarios.

Fig. 9. Basic and Caco interpolation scenes. From left to right: 0% Caco, 25% Caco, 50% Caco, 75% Caco, and 100% Caco.

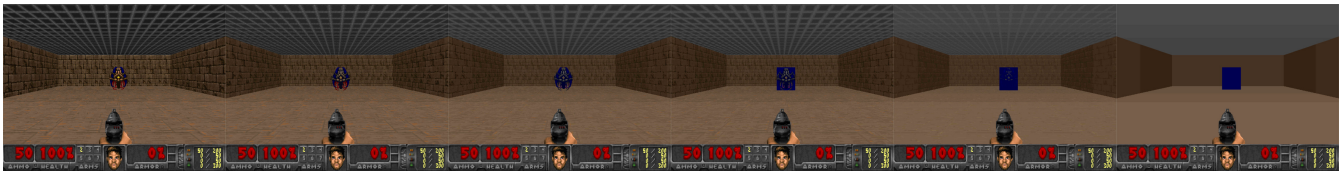


Fig. 10. Basic and Flat interpolation scenes. From left to right: 0% Flat, 25% Flat, 50% Flat without blue border, 50% Flat with blue border, 75% Flat, and 100% Flat.

Fig. 11. Basic and Animated interpolation scenes. From left to right: 0% Animated, 25% Animated, 50% Animated, 75% Animated, and 100% Animated.

As a reference, we used the fact that a score of around 70 or greater means that the agent presents good behavior, and is able to kill the monster in all cases. That was chosen based on observations of the experiments. We usually refer to that case as good or winning behavior. In these cases, the agent immediately moves towards the monster and kills it with a single shot. However, any negative score means that the agent does not kill the monster consistently. So, the agent does not present good behavior.

A. *Combinations*

In the first set of experiments, we tested the agents with all possible combinations of the Basic scene and a testing scene.

We grouped the results according to the pair formed by the Basic agent and the other agent. We present the results in tables and highlight the results that stand out, positively or negatively.

In all tables, the "Difference from Basic" column informs the game objects whose textures were replaced by the testing versions. A value of "None" represents the full original Basic scene. A value of "All" represents the full custom test scene, which can be Caco, Flat, or Animated.

1) Basic and Caco: Since the single difference from Basic to Caco scenes is the monster, there are only two possible combinations: the full Basic case and the full Caco case. The Basic's agent achieved a score of 76.92 in Basic's scene, which is expected since both training and testing scenes are the same. However, in Caco's scene, Basic's agent achieved a score of 29.46 (Table I).

TABLE I. BASIC'S AGENT SCORES IN BASIC AND CACO COMBINATIONS.

Difference from Basic	Mean Score
None	76.92
All	29.46

Similarly, Caco's agent presented a winning behavior in Caco's scene, with a score of 81.00. In Basic's scene, Caco's agent achieved a score of 16.55, showing that it was not able to consistently kill the monster (Table II). These results follow the ones presented in [10].

TABLE II. CACO'S AGENT SCORES IN BASIC AND CACO COMBINATIONS.

Difference from Basic	Mean Score
None	16.55
All	81.00

2) Basic and Flat: From [10], we know that the Basic's agent presents good behavior in both scenes, thus we would expect it to obtain good scores in all test cases. The results presented in Table III show that this is what happens with some variations, although the two highlighted cases have scored a little under 70.

TABLE III. BASIC'S AGENT SCORES IN BASIC AND FLAT COMBINATIONS.

Difference from Basic	Mean Score
None	76.92
Ceiling	**64.25**
Ceiling, Floor	**64.71**
Ceiling, Floor, Monster	74.32
Ceiling, Monster	73.85
Floor	76.74
Floor, Monster	76.93
Floor, Monster, Wall	75.90
Floor, Wall	80.26
Monster	77.79
Monster, Wall	78.14
Monster, Wall, Ceiling	72.40
Wall	80.16
Wall, Ceiling	79.85
Wall, Ceiling, Floor	79.99
All	72.70

This experiment presents remarkable findings. The Flat agent is capable of achieving a winning behavior when the monster is the blue texture (Table IV). However, when the monster has the original texture, the agent does not move or shoot at all. In this case, we can see that this agent learned to recognize the full blue texture pattern as target. However, this is not enough for us to affirm that it learned the concept of "monster" game object because the agent did not execute any action when the monster was presented with a different texture.

TABLE IV. FLAT AGENT SCORES IN BASIC AND FLAT COMBINATIONS.

Difference from Basic	Mean Score
None	-240.90
Ceiling	-237.79
Ceiling, Floor	-237.64
Ceiling, Floor, Monster	**75.50**
Ceiling, Monster	**75.45**
Floor	-240.04
Floor, Monster	**75.48**
Floor, Monster, Wall	**76.39**
Floor, Wall	-239.61
Monster	**75.57**
Monster, Wall	**76.24**
Monster, Wall, Ceiling	**76.52**
Wall	-239.87
Wall, Ceiling	-238.85
Wall, Ceiling, Floor	-238.76
All	**76.71**

3) Basic and Animated: When we changed any of the game objects, the Basic agent could not achieve a winning behavior. As seen in Table V, it presented the highest score when the walls were changed, with a mean score of 57.28. However, even in this case, it fires shots constantly, no matter the monster's position. Nevertheless, it still achieved scores greater than the minimum, with the lowest one being -152.02. This happens because the agent keeps shooting endlessly, which grants it some random kills.

TABLE V. BASIC AGENT SCORES IN BASIC AND ANIMATED COMBINATIONS.

Difference from Basic	Mean Score
None	76.92
Ceiling	-88.53
Ceiling, Floor	-126.31
Floor	**-152.02**
Floor, Wall	-137.80
Wall	**57.28**
Wall, Ceiling	-55.76
All	-140.79

The Animated agent can achieve scores close to optimal in the versions with an animated floor. Analyzing the behavior of the agent, it correctly learned to move towards the monster. However, the agent sometimes shoots when the monster is not yet in front of it, which makes it miss some shots, and decreases its mean score.

B. Interpolation of scenes

In the second part, we tested the agents in the interpolated versions of Basic with the three custom scenes. The results are presented for each pair, according to the experiment.

TABLE VI. ANIMATED AGENT SCORES IN BASIC AND ANIMATED COMBINATIONS.

Difference from Basic	Mean Score
None	-54.40
Ceiling	-50.23
Ceiling, Floor	**62.20**
Floor	**53.49**
Floor, Wall	**63.63**
Wall	-24.18
Wall, Ceiling	-33.94
All	**71.33**

1) Basic and Caco: With 25% Caco, we can see already a considerable performance improvement. The red pixels are present and the agent is able to correctly identify the monster in all cases. Starting from the 50% Caco version, most pixels of the monster are reddish and there are barely any blue pixels, which leads to a good performance. In fact, in all of the last three versions, Caco's agent obtained winning scores. Basic's agent, however, had an almost linear performance degradation. This indicates that its ability to identify the monster decreases almost in pair with the increasing percentage of changes (Table VII).

TABLE VII. MEAN SCORES IN BASIC AND CACO INTERPOLATION.

Agent	0%	25%	50%	75%	100%
Basic	76.92	68.14	51.58	48.54	29.46
Caco	16.55	63.45	75.42	78.69	81.00

2) Basic and Caco: The Flat agent had a very poor performance in the version with 25% Flat interpolation, and also with the 50% interpolation without the blue border. When the monster has a distinct blue border, the Flat agent can perform well. However, if the monster does not have a clear blue border, the agent did not move or shoot at all. These results, which can be found in Table VIII, corroborate the findings presented in Section V-A and demonstrate the importance of creating the two 50% versions. The Basic agent presented a winning behavior in all cases, as we expected since in all cases of combinations (Section V-A) it achieves good results.

TABLE VIII. MEAN SCORES IN BASIC AND FLAT INTERPOLATION. 50_1 DOES NOT HAVE BLUE BORDERS. 50_2 HAS BLUE BORDERS.

Agent	0%	25%	50_1%	50_2%	75%	100%
Basic	76.92	80.11	81.26	78.98	75.34	72.70
Flat	-240.90	-222.51	28.23	75.34	76.56	76.71

3) Basic and Animated: Regarding the Animated agent, there is a spike from 25% to 50% of the Animated scene, which can be explained by the presence of a more noticeable animation pattern (Table IX). In the 25% version, after the preprocessing step to a low-resolution grayscale image, the animation is almost imperceptible. Starting from the 50% version, we can see that the animation is a little more noticeable after the preprocessing step. This result indicates that besides learning how to play the Animated scene, the performance of the Animated agent is closely related to the animation patterns. The Basic agent can achieve winning behavior in the first three versions but suffers a big drop from 50% to 75%. This performance degradation follows the inverse path of the Animated agent, which indicates that the Basic agent cannot differentiate the animations from the monster.

TABLE IX. MEAN SCORES IN BASIC AND ANIMATED INTERPOLATION.

Agent	0%	25%	50%	75%	100%
Basic	76.92	79.36	73.15	-133.25	-140.79
Animated	-54.40	-51.60	22.47	64.74	71.33

VI. CONCLUSION

Inspired by prior works that evaluate the performance of DRL agents when trained and tested with different textures of a game scene, we designed several experiments to evaluate the full dimension of the difference in performance. The experiments were created to evaluate whether there are patterns in the agents' behavior when testing in slightly modified versions of its training scenes. We observed that CNN-based autonomous agents will not necessarily have good performance even if we change only a single game object's texture. Moreover, its performance is closely related to how similar the tested scene is when compared to the training scene. Therefore, instead of learning high-level concepts of game objects, the agents learn patterns according to the pixel values.

It is important to answer the four questions raised in Section I to ensure that the experiments gave us meaningful results. (i) Is the agent able to maintain its performance when we change the texture of a single object? Firstly, the performance difference when changing the textures depends on the training scene. However, in most cases, the agents were not able to maintain their scores. (ii) Is there a correlation between the performance of the agent and the game objects whose textures were changed? The agents present a different sensibility to different game objects. For example, the agent trained in the Flat scene presents good results when the monster is the blue texture and poor behavior when the monster is the original version, no matter the other textures. On the other hand, the agent trained in the Animated scene presented its best results when the floor was animated. We could not observe any correlation with a specific game object across all agents.

To answer the last two questions, we can use the results obtained from the interpolated experiments. (iii) Is the amount of performance degradation related to the number of pixels changed? In general, we observed that the performance degradation is related to the interpolation values. Summing up, the higher the proximity of the trained scene, the better the results. (iv) Can we observe any pattern of performance degradation when the textures are changed gradually? Although there is no defined pattern across all agents, we can still observe that the amount of pixel changes is directly related to the performance degradation. For example, when Caco's agent runs in a scene with some red pixels in the monster, its performance is much better.

The results obtained with the experiments performed in this work suggest that although the agents cannot generalize

under basic settings it is possible to avoid the performance degradation using settings focused more on the game objects, not on the pixel values. For example, the first experiment indicates that some of the objects have a greater influence on performance than others. Therefore, if the agents are trained to learn general concepts like enemy or background, they could be able to achieve comparable performance in all scenes.

With these insights, we suggest as future works to use specific architectures that could be used for understanding the game objects. For example, one can use Autoencoders, which are responsible to deliver a semantic segmentation of the objects, or training the agents to look at the border of the objects. Although that would be more complex to train, that kind of model may lead to more robust agents. Another possibility would be to try to interpret how the agents see the screen, i.e. how the weights of the Neural Network are related to the outputs, using recent methods of Neural Network Interpretability. Then, we could use that new information to create mechanisms to eliminate the performance degradation based only on texture changes.

ACKNOWLEDGMENT

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

REFERENCES

[1] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *ArXiv e-prints*, pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] ——, "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] A. P. Badia *et al.*, "Agent57: Outperforming the atari human benchmark," *ArXiv e-prints*, pp. 1–30, 2020. [Online]. Available: https://arxiv.org/abs/2003.13350

[4] D. Silver *et al.*, "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

[5] ——, "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

[6] 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.

[7] C. Berner *et al.*, "Dota 2 with large scale deep reinforcement learning," *ArXiv e-prints*, pp. 1–66, 2019. [Online]. Available: https://arxiv.org/abs/1912.06680

[8] O. Vinyals *et al.*, "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

[9] 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.

[10] P. B. S. Serafim, Y. L. B. Nogueira, C. A. Vidal, J. B. Cavalcante-Neto, and R. F. Férrer Filho, "Investigating deep q-network agent sensibility to texture changes on FPS games," in *Proceedings of the XIX Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2020, pp. 311–319.

[11] 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

[12] 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.

[13] 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.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[15] 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

[16] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, p. 58–68, 1995. [Online]. Available: https://doi.org/10.1145/203330.203343

[17] 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.

[18] 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

[19] 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.

[20] 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.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015, pp. 1–15.