university of groningen / faculty of science and engineering / artificial intelligence

Out of Distribution Detection in a DQN using Uncertainty Quantification Methods

Bachelor's Project Thesis

Dhruvs Sharma, s3812596, d.sharma.3@student.rug.nl,
Supervisor: Matias Valdenegro Toro

Abstract: In the current times one can see Reinforcement Learning (RL) models being applied to a variety of problems. These include robotics, industrial automation and even video games. The concerned models are not well suited for Out-Of-Distribution (OOD) inputs where they can make false predictions with high confidences. Although OOD detection is a well-researched topic in Deep Learning, OOD Detection in RL has had a lack of emphasis in terms of research until recently. In this report we take a deep Q-Network and modify it to output confidences with uncertainty using dropout and ensembles. The models are trained on the basic scenario (ID environment) from VizDoom, an API that allows one to train RL agents on preexisting game scenarios in the Doom video game. The scenario is edited to look different, say environment B, where the textures and target monster sprite are dissimilar to the training environment. After testing the models on environment B, the confidences produced show that dropout is somewhat suitable for OOD detection in the current task, while an ensemble fails to do so with higher standard deviation in the ID environment compared to the OOD environment.

1 Introduction

The advancements in the field of **Machine Learning** have been extensive in recent years, and it has evolved into many sub-paradigms, with one of them being **Reinforcement Learning** (RL). RL allows an agent to learn from its interactions with the environment and make better decisions over time in contrast to Supervised Learning. In the latter case an agent is trained on a pre-existing dataset, while an RL agent learns through trial and error. This is achieved through rewarding or punishing the agent based on its actions in an environment. RL has been applied in the fields of game playing, robotics, and autonomous vehicles (Shao et al., 2019); (Kiran et al., 2022); (Kober et al., 2013), to name a few.
However, it is important to note that the performance of RL policies is quite dependent on the training data/experience during training. It is often assumed that the test data follows the same distribution as training data, but this assumption does not hold true in many real-world scenarios. These samples/observations from a different underlying distribution are known as **Out-of-Distribution (OOD)** samples. Hence, one of the major challenges in RL is ensuring that the trained agent can generalize to these new, unseen situations. This is known as OOD detection (Sedlmeier et al., 2019).

The task of OOD detection is crucial in safety-critical applications, such as autonomous systems and robotics, where the decisions made by an RL agent have the potential to cause harm to human beings or damage to property. For example, an RL-controlled robot that is deployed in an industrial setting may not be able to perform well when it experiences a new or unseen scenario, oftentimes in which the robot has an imperfect and limited perception (Botteghi, 2021).
But OOD detection is not limited to these domains of application. As another example of OOD detection in RL, but in a virtual use-case scenario, is an agent being trained on a video game level and then evaluated on an unseen level. This poses a difficult challenge for current RL algorithms and both the quality and quantity of the training data matter for generalization (Balla et al., 2020). A

lack of training data, in this case, gives rise to uncertainty in the model itself.

Moving on to the topic of modern advancements in RL, which have utilized deep neural networks (DNNs). These DNNs have been shown to be susceptible to out-of-distribution (OOD) data (Hendrycks et al., 2020) which can result in wrong decisions leading to higher costs and sometimes, critical failures.
One of the problems in current RL practice is that researchers focus only on the metrics, namely the mean squared error (MSE) and the accuracy of the model. Since standard DNNs are used in common practice, point-wise predictions are generated as outputs instead of a prediction distribution that would be generated by a model that outputs uncertainty. This very uncertainty can then be used to assess how much confidence one has in the predictions of an RL model, with the predictions with higher uncertainty being a tell for humans to intervene in the process.

In this work we will explore detecting OOD samples in the context of an RL policy trained on a video game environment. Although most of the focus in research on OOD detection has been on image and text recognition tasks, recently there has been a benchmark that extends it to deep Reinforcement Learning (Mohammed & Valdenegro-Toro, 2021).
In the mentioned work, the author corrupts the states and changes some physical parameters to create OOD environments. We modify a game environment directly by changing its sprites and textures which enables us to create a visually different environment B. The confidence (mean and standard deviation distribution) output of this environment B then enables us to evaluate if B is Out-of-distribution or not. The model output being a distribution instead of point-wise predictions is achieved through modifying it to incorporate two sampling-based Uncertainty Quantification Techniques, namely MC Dropout and Ensembles, (further described in Section 3 (Gal & Ghahramani, 2015), (Lakshminarayanan et al., 2016). Hence, our Research Question being the following:
A Reinforcement Learning agent is trained on a video game environment A. Does the standard deviation output produced by testing it on a visually different environment B support B being OOD?

2 OOD Detection in Deep Learning

In **Deep Learning** (DL, an umbrella field for both classification and regression problems), when it comes to low-dimensional feature spaces, out-of-distribution (OOD) detection has been a well-researched topic. Pimentel et al. (2014) provide a survey distinguishing between probabilistic, domain-based, reconstruction-based, distance-based and information-theoretic methods.
With regard to higher dimensional feature cases, OOD detection has had a recent surge in interest, especially classification problems. For example, Hendrycks & Gimpel (2016) use predicted class probabilities of a softmax classifier to propose a baseline for OOD samples in Neural Networks. This baseline is enhanced via using temperature scaling and adding perturbations to the input (Liang et al., 2017).
But since our task is to do a regression of the state Q-Values of an agent in a virtual environment using a DQN, what is done here is not an application of the methods described above. One might glance at Generative-neural-network (GAN) based techniques (Schlegl et al., 2017) as they can be helpful to model the distribution of the training data. Although utilizing GANs would be computationally expensive and require a lot of data, which makes them less suitable for our purpose.
Our approach is based on treating OOD Detection as a classification problem, in which we have two classes, one for in-distribution data (ID) and another for out-of-distribution data (OOD). It is based on the UBOOD framework (Uncertainty Based Out-of-Distribution Detection) proposed by Sedlmeier et al. (2019). The idea is that the uncertainty in the model for ID samples is lower than OOD samples. Akin to this framework Lütjens et al. (2018) propose an algorithm that uses uncertainty information to cautiously avoid dynamic obstacles in novel scenarios, a risk-sensitive approach that can be used to make an RL policy sensitized to novel data, although not explored here.
Based on Mohammed & Valdenegro-Toro (2021), we chose MC Dropout and Ensembles as our choice for Uncertainty Quantification methods as they generally perform the best for OOD Detection according to the authors.

3 Out-of-Distribution Detection and Uncertainty

It is necessary to have an understanding of **Uncertainty Quantification** (UQ) in order to comprehend how a model can produce uncertainty. UQ is the process of determining and measuring the uncertainty in the predictions made by a model. As previously noted, uncertainty in predicitons can be particularly helpful in the field of OOD detection, and can be categorized into two types: Epistemic uncertainty and Aleatoric uncertainty (Kiureghian & Ditlevsen, 2009); (Gal & Ghahramani, 2015). Aleatoric uncertainty is uncertainty that arises from the inherent noise in the data, while epistemic uncertainty arises from the model's lack of knowledge about the data distribution.
Below we focus on two popular sampling methods of UQ both of which play a crucial role in detecting OOD samples and providing reliable predictions in uncertain scenarios. The first one is Monte Carlo Dropout (MC Dropout), which is a technique that involves randomly dropping out neurons (Figures 3.1, 3.2) during the forward pass of a neural network, on top of doing it during training (Gal & Ghahramani, 2015). This results in a distribution of predictions, rather than a single prediction. By taking the mean and standard deviation across these predictions produced over multiple forward passes, we can estimate the model's uncertainty. This is akin to Bayesian Model Averaging, where multiple models are trained with different weight values (Kendall & Gal, 2017).

On the other hand, using an Ensemble in the context of UQ is to train multiple models (estimators) and combine their predictions (Figure 3.3) by taking the mean and standard deviation across their outputs. The output standard deviation serves as an uncertainty measure in both UQ methods. Furthermore, ensembles can be used to estimate both aleatoric and epistemic uncertainty (Lakshminarayanan et al., 2016). An advantage of using an ensemble is that the diversity of the models in the ensemble can help to identify when the input data is outside of the distribution the ensemble was trained on. The idea behind the aforementioned UQ methods is that model's uncertainty is typically higher for OOD samples than for in-distribution samples.

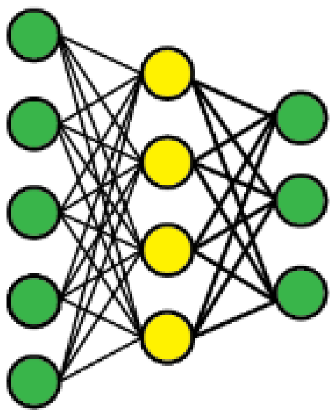


Figure 3.1: An example figure of a standard neural network (Krause, 2016).

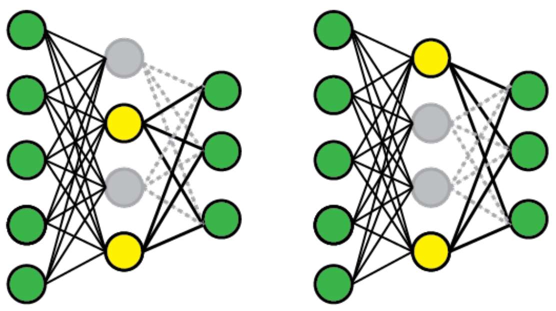


Figure 3.2: An example figure of Dropout in a neural network (Krause, 2016).

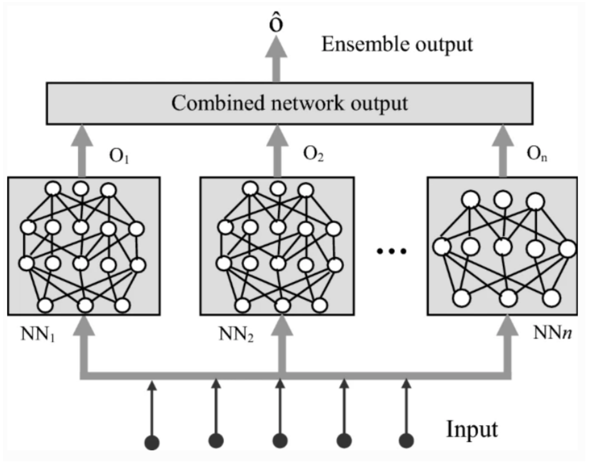


Figure 3.3: A figure which shows training an ensemble (Alam et al., 2019).

It is reasonable to inquire about which one of epistemic and aleatoric uncertainty matters more in OOD detection, with the answer to that being epistemic uncertainty. This is because epistemic uncertainty captures the model's uncertainty about the true underlying data distribution. In contrast, aleatoric uncertainty captures the inherent noise in the data, which does not affect the model's uncertainty about the data distribution. Leading from this, epistemic uncertainty is a better indicator of OOD samples as it captures the model's lack of knowledge about the input data. In conclusion, both MC Dropout and Ensembles are viable choices to estimate epistemic uncertainty.

4 Methods

4.1 The Game Scenario

VizDoom is a reinforcement learning platform built on top of the Doom game engine (Wydmuch et al., 2019). In this work, we used the VizDoom API to train an RL agent in one of the pre-existing 'basic' scenario (game environment level) provided by the authors.
In the scenario, the player agent would spawn with a gun at the center of a square room along a wall. A monster would spawn somewhere alongside opposite room wall, the aim/goal being to shoot the monster (one shot enough to reach the end goal). The player would be allowed to move left, move right or shoot, allowing for a combination of two actions at once, totaling $2^3 = 8$ possible actions. A living reward of -1 would be given for every time step, with the reward for shooting the monster being +101, and the reward for every bullet missed being -5. The episode would end on monster death or t = 300 timeout (t is the number of steps in a game episode).

4.2 Model Architecture

A Deep Q-Network (DQN) is a specific type of Q-learning algorithm that uses a neural network, called the Q-network, to approximate the Q-value function (Mnih et al., 2013), (Mnih et al., 2015). In our case, we used the DQN to do a regression of the state values for all possible actions, enabling us to choose the best action during training via the epsilon-greedy policy.

According to Mnih et al. (2013) the optimal strategy for selecting action a' in a Markov Decision Process (MDP, playing a game scenario in our case is an MDP with a state, actions and rewards tied to it) while maximizing the expected value of $r + \gamma Q^*(s', a')$ is given by the following equation (Bellman Equation):

$$Q^*(s,a) = \mathbb{E}s' \sim \mathcal{E}\left[r + \gamma \max a' Q(s',a') \mid s,a\right]$$

where the optimal value $Q^*(s', a')$ of the next time-step sequence s' is known for all possible actions a'.

The author further mentions that a Q-network with weights θ can be used as a non-linear functional approximator to estimate the action-value function ($Q(s,a;\theta) \approx Q^*(s,a)$). The network can be trained by minimizing a series of loss functions $L_i(\theta_i)$ with the loss function changing at each iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

In the above equation, the author expresses that $y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a\right]$ is the target for iteration i, while $\rho(s,a)$ being the probability distribution over sequences and actions (s,a). Here, the previous iteration's (θ_{i-1})

parameters are fixed when optimizing the loss function $L_i(\theta_i)$. This optimization is preferably done through stochastic gradient descent (also in our case).

The driver/example code (using Tensorflow) for a Deep Q-Network with an epsilon-greedy policy was taken from the VizDoom repository on github*, made for the purpose of learning the 'basic' game scenario.
The design of the neural network included the incorporation of convolutional layers to extract features from the input image, a flattening operation to convert the output of these layers into a one-dimensional array, and two dense layers to produce the final output. Below the structure of the DQN is described with hyper-parameters:

1. It begins with an input layer which takes in an image with shape `(30, 45, 1)` as input: `Input(shape=(30, 45, 1))`.
2. The first convolutional layer is applied to the input image from the `Input()` layer: `Conv2D(filters = 8, kernel_size = 6, stride = 3, input_shape = (30, 45, 1))`. `BatchNormalization()` and `ReLU()` activation function are applied in order right after the convolution step.
3. Then a second convolutional layer is applied on the output of the first convolutional layer: `Conv2D(filters = 8, kernel_size = 3, stride = 2), input_shape = (9, 14, 8))`. Similarly here, `BatchNormalization()` and `ReLU()` follow the convolutional step.
4. The flatten layer (`Flatten()`) is applied on the output of the second convolutional layer, which flattens the output into a 1D array.
5. The output of `Flatten()` layer is split into two parts, x1 and x2. x1 corresponds to the first 96 elements, and x2 corresponds to the remaining elements.
6. x1 is passed through a dense layer with one neuron named `state_value`:
   `state_value = Dense(1)`
   `x1 = state_value(x1)`

*https://github.com/Farama-Foundation/ViZDoom/blob/master/examples/python/learning_tensorflow.py

Table 4.1: A table with the hyper-parameter values used to train all models.

Hyperparameter	Value
Learning rate	0.00025
Discount factor	0.99
Replay mem. size	10000
#Training epochs	15
#Learning steps per epoch	2000
Target net update steps	1000
Batch size	64

7. x2 is passed through the `advantage` dense layer, which has `num_actions` number of neurons, which is 8:
   `advantage = Dense(num_actions)`
   `x2 = advantage(x2)`
8. `x1` and `x2` are then combined. `x2` is adjusted by subtracting the mean value from all the predicted advantages, thus making the mean of the advantages zero. `x1` is added to the adjusted `x2`, thus combining the state value and the advantages:
   `x = x1 + (x2 - tf.reshape(tf.math.reduce_mean(x2, axis=1), shape=(-1, 1)))`
9. The final output (`x`) of the model is the combined state value and advantages:

4.3 Training and Hyper-parameters

The agent was given a set of initial actions and allowed to explore the environment. While playing, it collected experiences (i.e., state, actions, rewards, next state), stored the states in a replay buffer and then used these experiences to update its Q-values and improve its policy. Three instances of the model were trained, first a baseline model, then a model with Dropout and finally a model with a number of estimators (an Ensemble). All had the same following hyper-parameters throughout, in Table 4.1

Additionally, in the case of Dropout, we replaced the `Dense()` layers with `StochasticDropout()` layers (with a probability of 0.2) from the package `keras_uncertainty`[†], and then passed the model to a `StochasticRegressor()` model class (also from `keras_uncertainty`). The `predict()` method of this class uses Monte Carlo Sampling to produce a mean and standard deviation (the uncertainty) output of the state Q-values (for each action) learned during training, which was used for OOD Detection.
In a similar fashion, we passed the model to the `SimpleEnsemble()` class from `keras_uncertainty`, with 5 ensemble members. The `predict()` method of this class also outputs the learned state q-value distribution (mean and standard deviation distribution of each possible action along an episode). This standard deviation output was used as a confidence measure for OOD evaluation, just like in the case of MC Dropout.

4.4 ID vs. OOD Environment

To understand how the environment was modified, one needs to be familiar with the concept of .WAD files (i.e. "Where's All the Data?"), which are files used in the Doom video game's engine to store game data, which includes levels, graphics, and sound effects. In VizDoom, .WAD files are used to provide the game environment for the AI agent to interact with. One can use their own custom .WAD files, or use the ones provided by VizDoom to create a variety of different game environments for the AI agent to train on.
By default, VizDoom uses an open source archive 'freedoom2.WAD'[‡] as an alternative to the original archive 'DOOM2.WAD' to make game levels. It includes custom textures, sounds etc. We use this as our environment A (ID environment, Figure 4.1).

On the other hand, for the (Out-of-distribution) OOD environment B we used the original Doom 2 game's assets (when initiliazing the environment) by using an inbuilt method in VizDoom to utilize 'DOOM2.WAD' as the base archive:
`vizdoom.game.set_doom_game_path(scenario_path, "DOOM2.WAD")`.

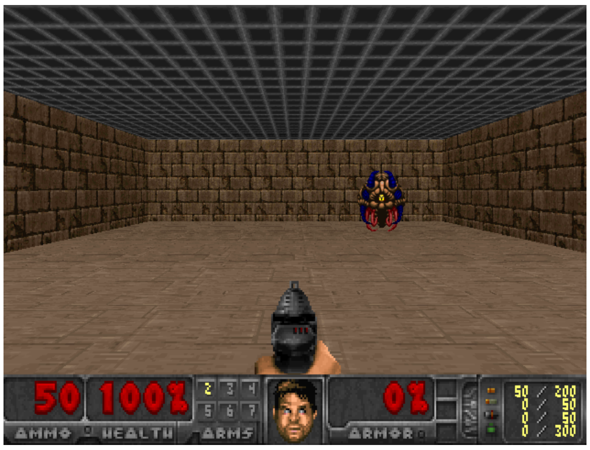


Figure 4.1: A screenshot of the In-Distribution (ID) environment A. Pay attention to the monster, floor and the wall textures

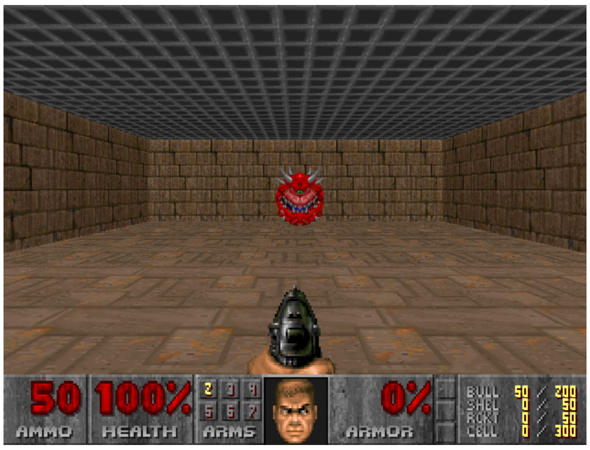


Figure 4.2: A screenshot of the Out-on-Distribution (OOD) environment B. It has different textures for the monster, floor and the walls utilizing assets from the Doom 2 game.

[†]`https://github.com/mvaldenegro/keras-uncertainty`
[‡]`https://freedoom.github.io/`

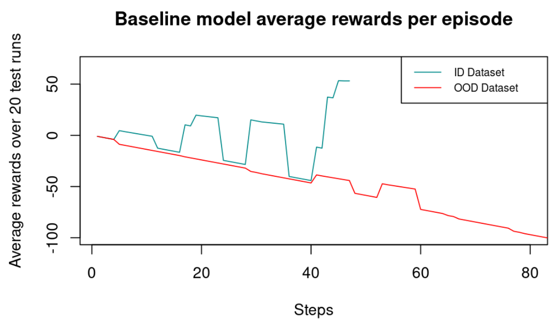


Figure 5.1: Baseline model average rewards per episode (ID vs. OOD)

Refer to Figure 4.2 for the OOD environment. Since 'DOOM2.WAD' is licensed by the developers of Doom 2 game, the authors had to purchase the game itself and get original .WAD file from the installation directory of the game.

5 Experimental Results and Analysis

In this work, the data collected from the baseline, MC Dropout, and ensemble models was analyzed across 20 test runs or episodes, with the results being averaged over the number of steps per episode. First, the examination of the baseline model's performance is conducted using the ID and OOD datasets (as depicted in Figure 5.1).
The results show higher rewards in the ID setting for episodes lasting no more than 50 steps, which is consistent with the expectation as the model was trained on the ID dataset. Conversely, the rewards in the OOD setting demonstrate the agent's inability to defeat the monster within the first 40 steps, with a declining trend observed until the episode times out (which is limited to 120 steps on the x-axis). This trend suggests that the baseline model experienced significant challenges in the OOD setting.

5.1 Dropout Results

Again the examination of the reward plot is done, but for the Dropout model on the ID and OOD datasets (5.2). In contrast to the baseline model,

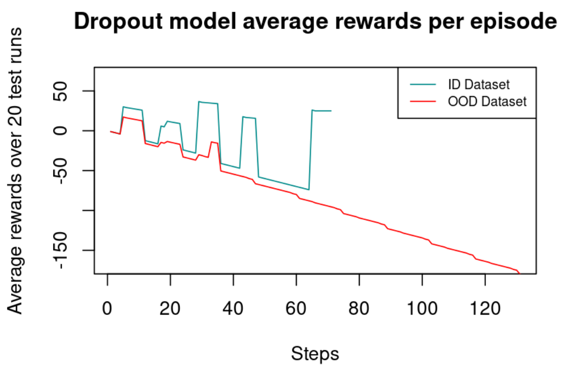


Figure 5.2: Dropout model average rewards per episode (ID vs. OOD)

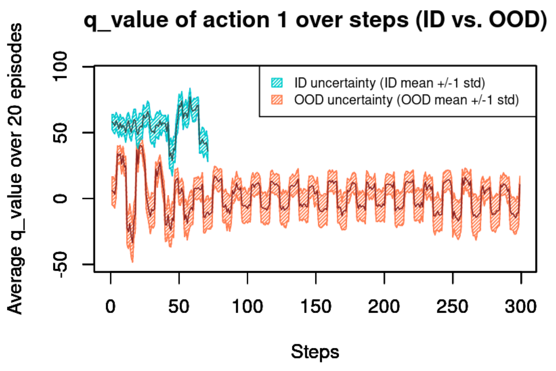


Figure 5.3: Dropout model mean and standard deviation of Q-value for action 1 (ID vs. OOD)

we see reward spikes in some episodes during the initial steps in the OOD setting, which suggest that the dropout model is more successful, although the episodes in the ID setting end sooner. One interesting thing to notice is that the agent misses their shots on the monster via the little dips in the reward curve in the OOD setting (check A.2 to observe an example run), with the trend going downwards without any spikes until episode timeout here as well.

Moving onto the plots of the mean and standard deviations (uncertainty) of the Q-value for each one of the 8 possible action combinations (Figure 5.3 - Figure 5.10).

One can observe that the dropout model has

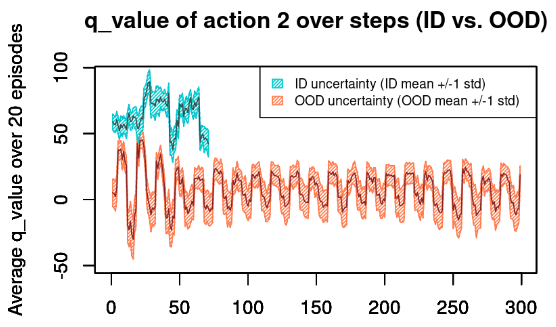


Figure 5.4: Dropout model mean and standard deviation of Q-value for action 2 (ID vs. OOD)

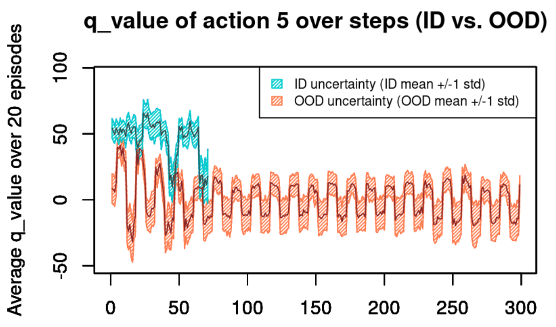


Figure 5.7: Dropout model mean and standard deviation of Q-value for action 5 (ID vs. OOD)

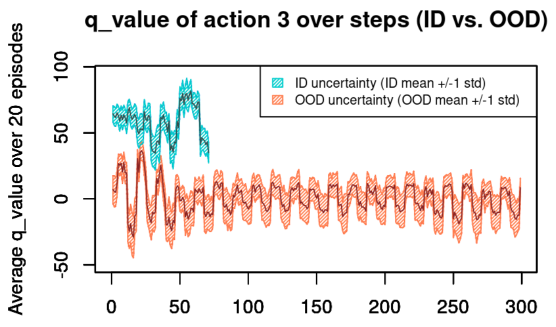


Figure 5.5: Dropout model mean and standard deviation of Q-value for action 3 (ID vs. OOD)

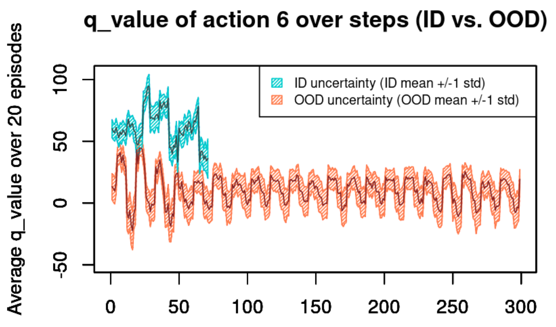


Figure 5.8: Dropout model mean and standard deviation of Q-value for action 6 (ID vs. OOD)

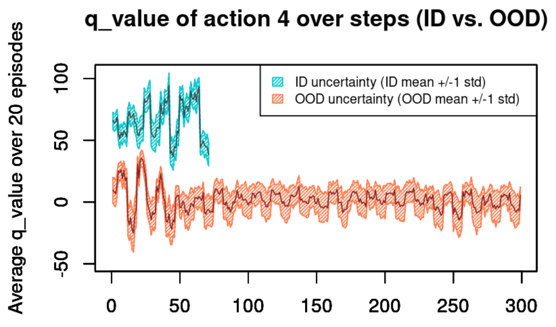


Figure 5.6: Dropout model mean and standard deviation of Q-value for action 4 (ID vs. OOD)

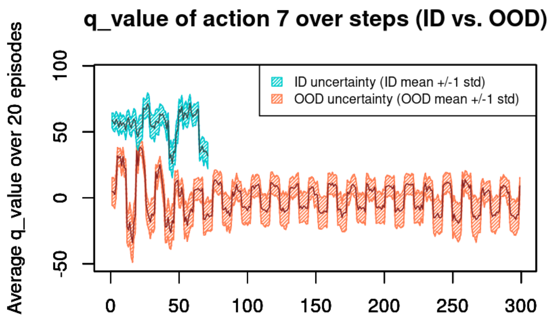


Figure 5.9: Dropout model mean and standard deviation of Q-value for action 7 (ID vs. OOD)

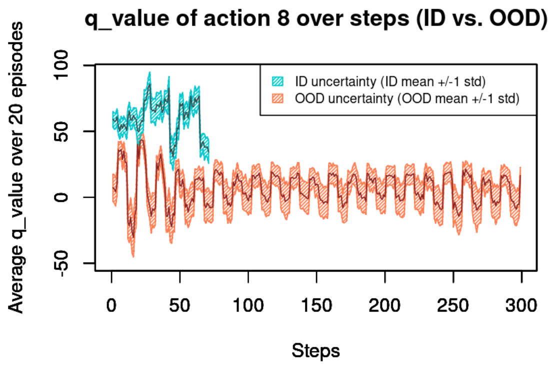


Figure 5.10: Dropout model mean and standard deviation of Q-value for action 8 (ID vs. OOD)

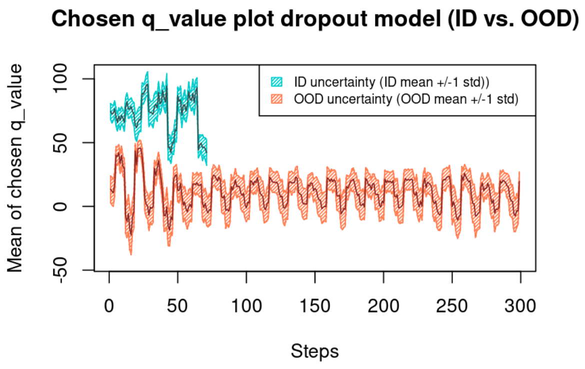


Figure 5.11: Dropout model mean and standard deviation of Q-value for the chosen action (ID vs. OOD)

higher ID mean with less spikes in all actions when compared to the OOD mean. The OOD mean varies more during the initial phase of an episode with less number of fluctuations when compared to later on in an episode. Whereas the ID mean varies less during the start of an episode, where the quick test runs end. With the different colored shaded regions one can see the uncertainty in both settings. As mentioned earlier, OOD setting episodes run until the 300 step timeout, in contrast with 71 steps for the ID setting. Overall, we see a similar trend across all actions for both the settings.

Moving forward, a plot of the mean and standard deviation for the Q-value corresponding to the chosen action is generated (Figure 5.11). The plot reflects the same trend as the previous action plots, however, for the purpose of OOD detection, it is crucial to determine which setting, either ID or OOD, has higher uncertainty. Unfortunately, the current graph does not provide a clear distinction in this regard.

Up next, the plots for the standard deviations for the ID and OOD datasets are observed (Figure 5.12). Here, one can see the difference between the ID and OOD dataset standard deviation, and see that OOD setting has a trend of having a higher standard deviation overall, which is what one would anticipate when the dropout model is tested on the OOD environment.

To evaluate how the standard deviation outputs for the ID and OOD setting are separated, one can

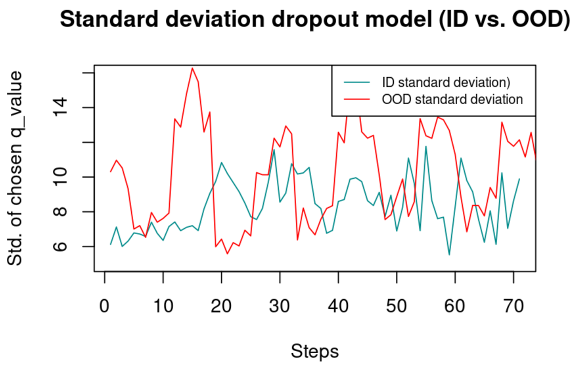


Figure 5.12: Dropout model standard deviation of Q-value for the chosen action (ID vs. OOD)

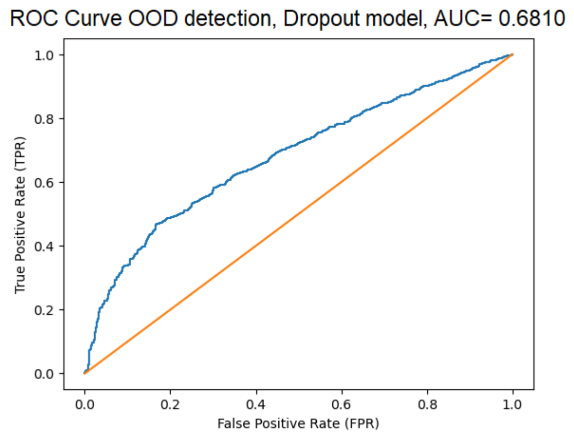


Figure 5.13: ROC curve for the dropout model, for OOD Detection. The orange line is for a random classifier which corresponds to an AUC of 0.5

use an ROC curve which acts as a binary classifier (Figure 5.13). We instantly observe that it performs better than chance, and an AUC of 0.6810 portrays a 68% chance that the positive class (OOD) has a higher predicted standard deviation (uncertainty) than the negative class (ID). Hence, one can say that the dropout model somewhat succeeds in separating in ID and OOD samples.

5.2 Ensemble Results

Commencing with the ensemble model results, we analyze the reward plot for the ID and OOD setting (Figure 5.14).

An interesting trend is shown by the OOD setting, matching and even passing the rewards with the ID setting during the first few steps, but having lower rewards in the subsequent steps of an average episode. The test runs in the ID setting end sooner when compared to the OOD setting, in which some runs hit the time out. We also see big spikes in the rewards in the OOD setting in the subsequent steps, unlike the dropout model which suggests that the agent is still killing the monster, even though much later. Like in the Dropout model, we don't see any missed shots (little dips) in the reward curves here. Overall, suprising results for the OOD setting are seen here.

Stepping ahead, we analyze the ensemble model

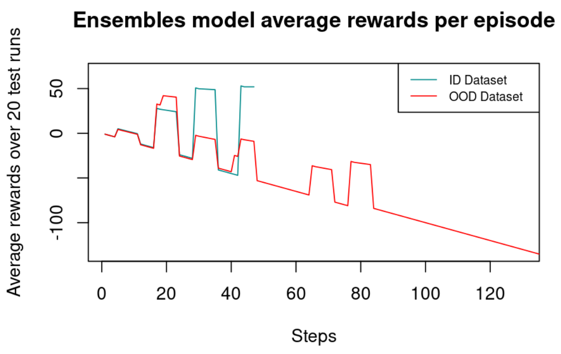


Figure 5.14: Ensemble model average rewards per episode (ID vs. OOD)

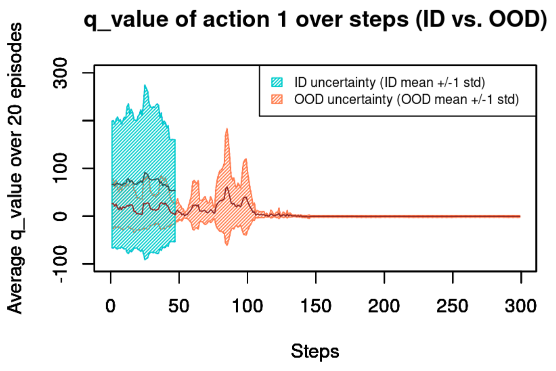
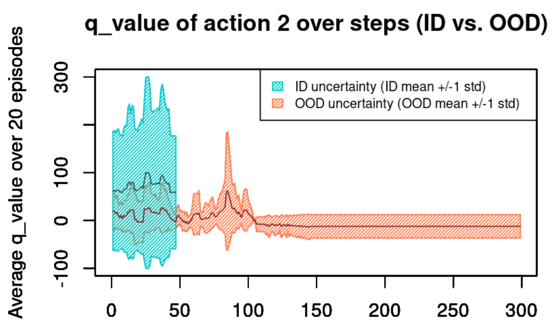


Figure 5.15: Ensemble model mean and standard deviation of Q-value for action 1 (ID vs. OOD)

plots which have the mean and standard deviations of the Q-values for all possible actions (Figures 5.15 - 5.22).

One noticeable trend in all action plots for the ensemble model which can be seen here is that the ID setting has a visibly higher uncertainty (blue shaded region) compared to the OOD setting (brown shaded region), even though the mean of the Q-Value for all actions in the ID setting is higher than in OOD setting. The former result is quite unexpected, and the authors are not sure why that is the case. One can also notice spikes in the uncertainty in the OOD setting in the subsequent steps for all actions.



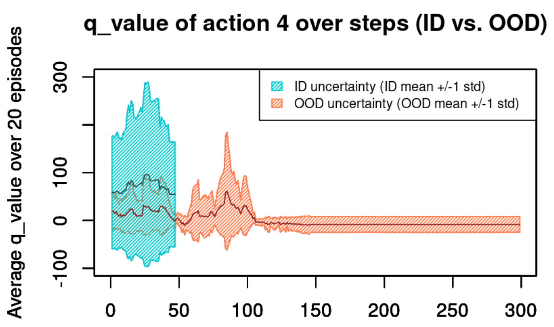
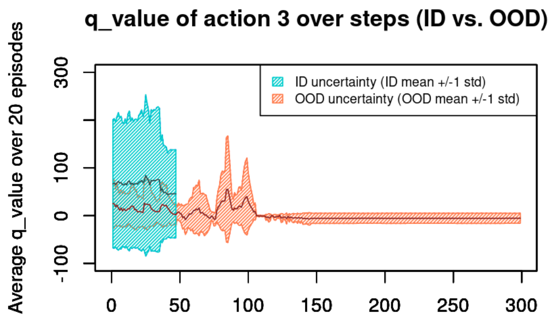


Figure 5.16: Ensemble model mean and standard deviation of Q-value for action 2 (ID vs. OOD)

Figure 5.18: Ensemble model mean and standard deviation of Q-value for action 4 (ID vs. OOD)



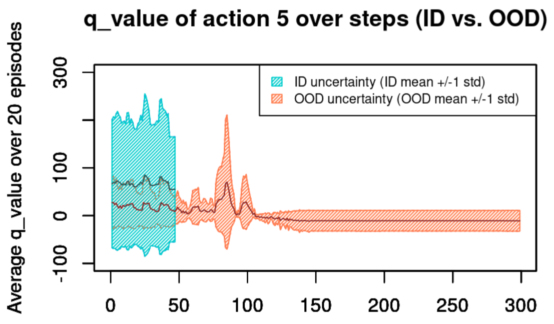


Figure 5.17: Ensemble model mean and standard deviation of Q-value for action 3 (ID vs. OOD)

Figure 5.19: Ensemble model mean and standard deviation of Q-value for action 5 (ID vs. OOD)

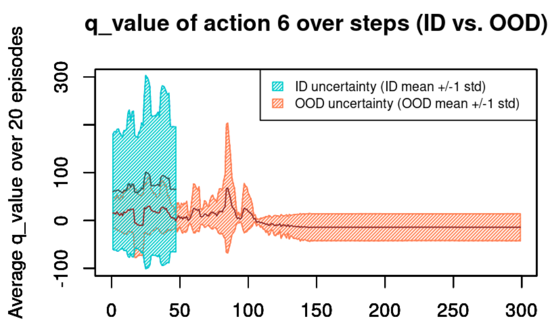


Figure 5.20: Ensemble model mean and standard deviation of Q-value for action 6 (ID vs. OOD)

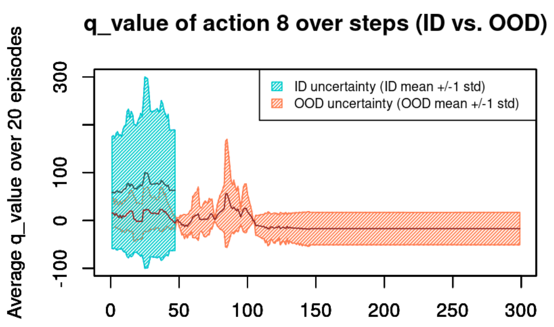


Figure 5.22: Ensemble model mean and standard deviation of Q-value for action 8 (ID vs. OOD)

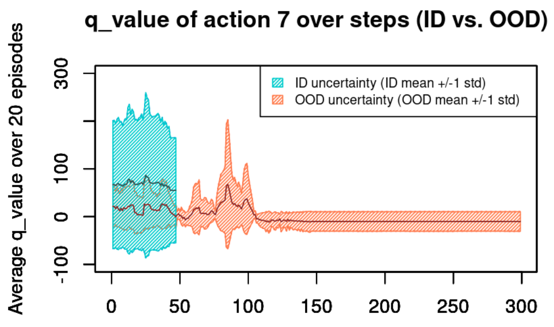


Figure 5.21: Ensemble model mean and standard deviation of Q-value for action 7 (ID vs. OOD)

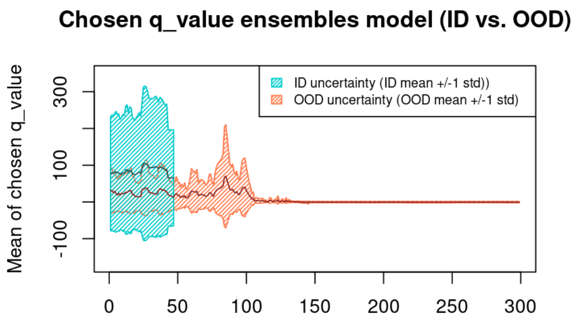


Figure 5.23: Ensemble model mean and standard deviation of the Q-value for the chosen action (ID vs. OOD)

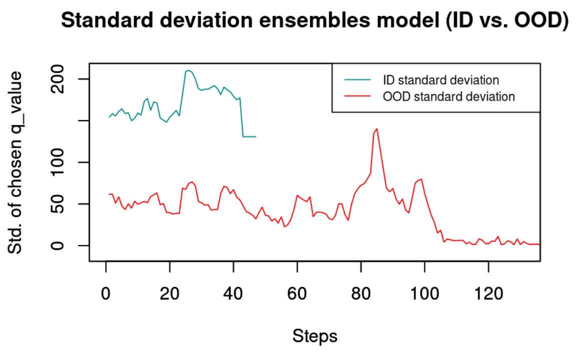


Figure 5.24: Ensemble model standard deviation of Q-value for the chosen action (ID vs. OOD)

Furthermore, the ensemble plot of the mean and standard deviation of the chosen Q-value is examined (Figure 5.23). This plot shows similar trends as the plots for all actions. Even though apparent through the current plot, we check the standard deviation (uncertainty) plots for the ensemble model (Figure 5.24) and observe that the ID setting has a way higher uncertainty compared to the OOD setting. As previously noted, these are unexpected results for the ensemble model.

Furthermore, an ROC curve is plotted to analyse how the ensemble model detects OOD samples (Figure 5.25). We instantly see that it completely fails, with an AUC of 0.0009, which means that there is a 0.09 % chance that the OOD class/setting has a higher standard deviation than the ID class/setting.

Last but not the least, two tables are made for the scores of both the models in the ID and OOD settings (Tables 5.1 and 5.2). These scores give a rough idea to the reader of how the models do in both the ID and OOD environments, with the ROC AUC being the most important one, as it conveys the ability of the models to separate ID and OOD samples.

6 Conclusions and Future Work

In the current work, two Uncertainty Quantification methods (MC Dropout and Ensemble) are

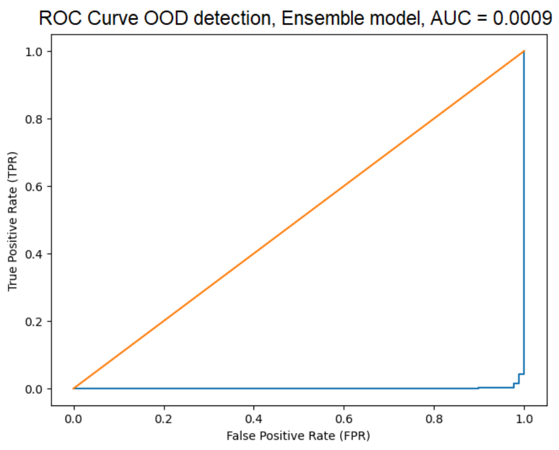


Figure 5.25: ROC curve for the ensemble model, for OOD Detection. The orange line is for a random classifier which corresponds to an AUC of 0.5

Table 5.1: A table with the averaged scores and results from the Dropout model.

Avg. reward ID setting	-12.3
Avg. reward OOD setting	-201.7
Avg. mean of chosen Q-value ID	72.6
Avg. mean of chosen Q-value OOD	10.8
Avg. std. of chosen Q-value ID	8.3
Avg. std. of chosen Q-value OOD	10.2
ROC AUC	0.68

Table 5.2: A table with the averaged scores and results from the Ensemble model.

Avg. reward ID setting	6.1
Avg. reward OOD setting	-145.7
Avg. mean of chosen Q-value ID	84.2
Avg. mean of chosen Q-value OOD	8.9
Avg. std. of chosen Q-value ID	168.6
Avg. std. of chosen Q-value OOD	20.0
ROC AUC	0.09

compared to detect OOD samples in a video game scenario/level of the Doom 2 game. The task is to kill a monster in the scenario, with the original environment having a different distribution than the OOD environment. This difference in the OOD setting is achieved through changing the archive that the game level loads it assets from, to the original Doom 2 game, in contrast with the default textures the level uses from an open-source archive. The performance discrepancy between the trained models in their original environment and the customized versions illustrates the models' vulnerability to variations in the environment. One of the difficulties faced was to change the original environment and the target not so much, which would result in a complete failure of the models. Another challenge faced by the authors was to train the ensemble members independently, which did not result in rewards that converged.

MC Dropout is able to somewhat distinguish ID samples from OOD samples, with ensembles completely failing to do so. The reason why this happens in our case can be explored in future research (if interested, one can infer with Appendix A to see examples of agent gameplay in the OOD setting). Hence, we can answer our Research Question by saying yes, MC Dropout somewhat supports the visually different environment B being Out-of-distribution, while an Ensemble fails to do so. New game scenarios can also be tested alongside more UQ methods, such as Bayes by Backpropogation, Flipout, DropConnect etc.

References

Alam, K. M. R., Siddique, N., & Adeli, H. (2019, July). A dynamic ensemble learning algorithm for neural networks. *Neural Computing and Applications*, *32*(12), 8675–8690. Retrieved from `https://doi.org/10.1007/s00521-019-04359-7` doi: 10.1007/s00521-019-04359-7

Balla, M., Lucas, S. M., & Pérez-Liébana, D. (2020). Evaluating generalisation in general video game playing. *CoRR*, *abs/2005.11247*. Retrieved from `https://arxiv.org/abs/2005.11247`

Botteghi, N. (2021). *Robotics deep reinforcement learning with loose prior knowledge* (Doctoral dissertation, University of Twente, Netherlands). doi: 10.3990/1.9789036552165

Gal, Y., & Ghahramani, Z. (2015). *Dropout as a bayesian approximation: Representing model uncertainty in deep learning.* arXiv. Retrieved from `https://arxiv.org/abs/1506.02142` doi: 10.48550/ARXIV.1506.02142

Hendrycks, D., Basart, S., Mu, N., Kadavath, S., Wang, F., Dorundo, E., . . . Gilmer, J. (2020). The many faces of robustness: A critical analysis of out-of-distribution generalization. *CoRR*, *abs/2006.16241*. Retrieved from `https://arxiv.org/abs/2006.16241`

Hendrycks, D., & Gimpel, K. (2016). A baseline for detecting misclassified and out-of-distribution examples in neural networks. *CoRR*, *abs/1610.02136*. Retrieved from `http://arxiv.org/abs/1610.02136`

Kendall, A., & Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? *CoRR*, *abs/1703.04977*. Retrieved from `http://arxiv.org/abs/1703.04977`

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., & Pérez, P. (2022). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, *23*(6), 4909-4926. doi: 10.1109/TITS.2021.3054625

Kiureghian, A. D., & Ditlevsen, O. (2009, March). Aleatory or epistemic? does it matter? *Structural Safety*, *31*(2), 105–112. doi: 10.1016/j.strusafe.2008.06.020

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238-1274. Retrieved from `https://doi.org/10.1177/0278364913495721` doi: 10.1177/0278364913495721

Krause, M. (2016). *What is the difference between dropout and drop connect?* Retrieved from `https://rb.gy/uhunab`

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2016). *Simple and scalable predictive uncertainty estimation using deep ensembles.* arXiv. Retrieved from `https://arxiv.org/abs/1612.01474` doi: 10.48550/ARXIV.1612.01474

Liang, S., Li, Y., & Srikant, R. (2017). Principled detection of out-of-distribution examples in neural networks. *CoRR*, *abs/1706.02690*. Retrieved from `http://arxiv.org/abs/1706.02690`

Lütjens, B., Everett, M., & How, J. P. (2018). Safe reinforcement learning with model uncertainty estimates. *CoRR*, *abs/1810.08700*. Retrieved from `http://arxiv.org/abs/1810.08700`

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, *abs/1312.5602*. Retrieved from `http://arxiv.org/abs/1312.5602`

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015, February). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. Retrieved from `https://doi.org/10.1038/nature14236` doi: 10.1038/nature14236

Mohammed, A. P., & Valdenegro-Toro, M. (2021). Benchmark for out-of-distribution detection in deep reinforcement learning. *CoRR*, *abs/2112.02694*. Retrieved from `https://arxiv.org/abs/2112.02694`

Pimentel, M. A., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014, June). A review of novelty detection. *Signal Processing*, *99*, 215–249. Retrieved from `https://doi.org/10.1016/j.sigpro.2013.12.026` doi: 10.1016/j.sigpro.2013.12.026

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., & Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *CoRR*, *abs/1703.05921*. Retrieved from `http://arxiv.org/abs/1703.05921`

Sedlmeier, A., Gabor, T., Phan, T., Belzner, L., & Linnhoff-Popien, C. (2019). Uncertainty-based out-of-distribution detection in deep reinforcement learning. *CoRR*, *abs/1901.02219*. Retrieved from `http://arxiv.org/abs/1901.02219`

Shao, K., Tang, Z., Zhu, Y., Li, N., & Zhao, D. (2019). *A survey of deep reinforcement learning in video games.* arXiv. Retrieved from `https://arxiv.org/abs/1912.10944` doi: 10.48550/ARXIV.1912.10944

Wydmuch, M., Kempka, M., & Jaśkowski, W. (2019). ViZDoom Competitions: Playing Doom from Pixels. *IEEE Transactions on Games*, *11*(3), 248–259. (The 2022 IEEE Transactions on Games Outstanding Paper Award) doi: 10.1109/TG.2018.2877047

A Appendix

Here, the screen buffer screenshots are provided of runs in which the agent fails to kill the monster, hence the runs in the OOD environment for both the models.

A.1 Ensemble

We look at an Ensemble model's test episode in which the agent fails to kill the monster in the OOD environment, with an ending score of -300 (Figures A.1 - A.5). Each screenshot is taken after every 12 frames. Here, we can clearly see that the agent is not moving or firing the weapon at all before time out as seen in Figure 5.14. This behaviour is also consistent across all failed runs exhibited by the Ensemble agent.

A.2 MC Dropout

Lastly, we look at a Dropout model's test episode in which the agent fails to kill the monster in the OOD environment, with an ending score of -410 (Figures A.6 - A.10). Each screenshot is taken after every 12 frames here as well. One notices that the monster has spawned on the right side, and the agent exhibits similar behaviour whenever this happens, that is completely ignoring the target monster and moving towards the left wall of the room (away from the monster), whilst firing missed shots. The agent stays at the left wall and ignores the monster, still firing missed shots until timeout. This would explain the little dips in the rewards because of missed bullets in the OOD setting (Figure 5.2). This behavior is seen in all the failed runs by the MC Dropout agent.

Figure A.1: Steps 0 - 5 failed run Ensemble

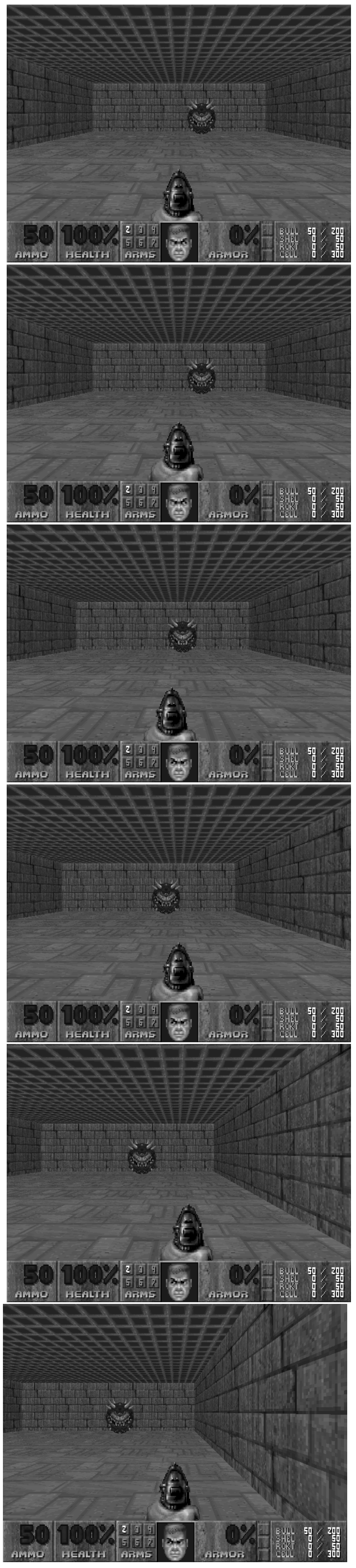


Figure A.2: Steps 6 - 11 failed run Ensemble

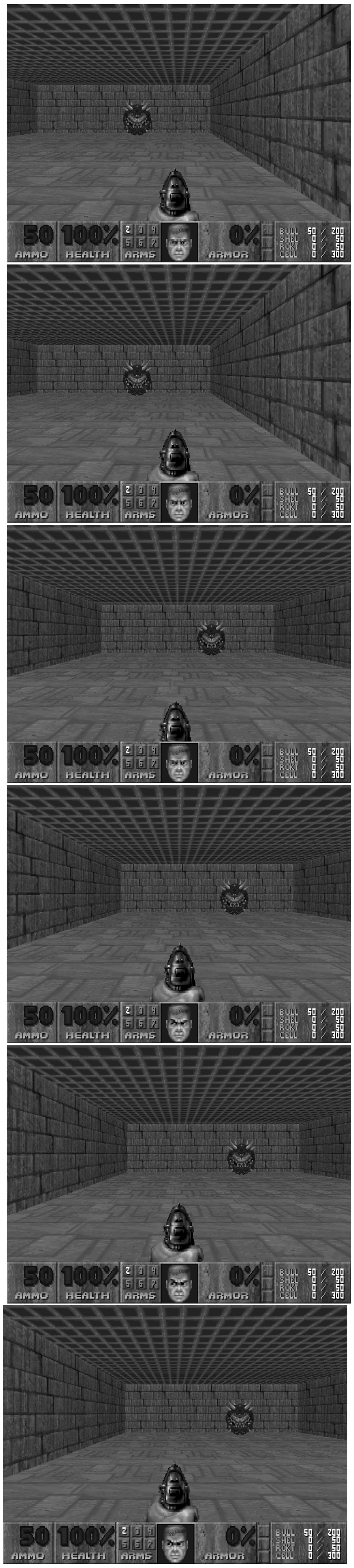


Figure A.3: Steps 12 - 17 failed run Ensemble

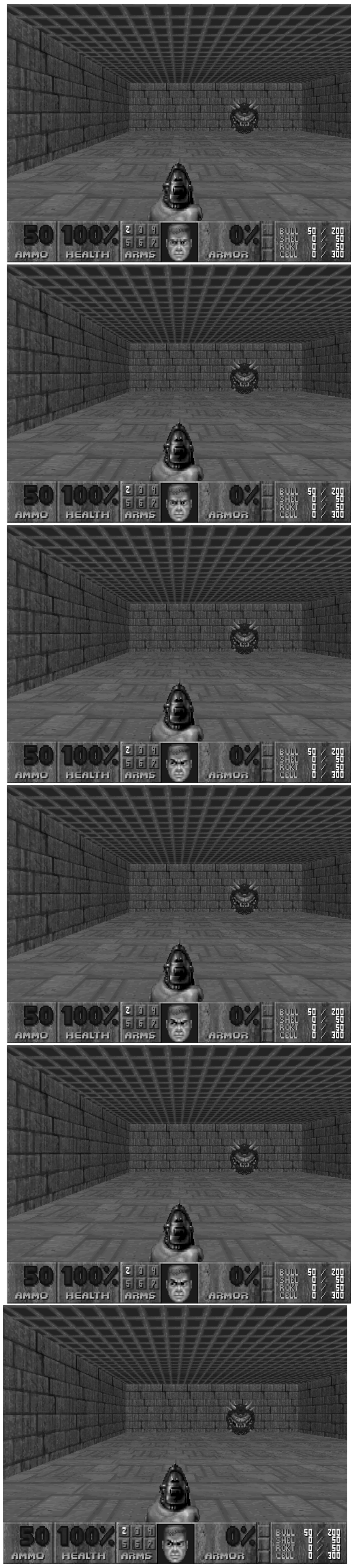


Figure A.4: Steps 18 - 23 failed run Ensemble

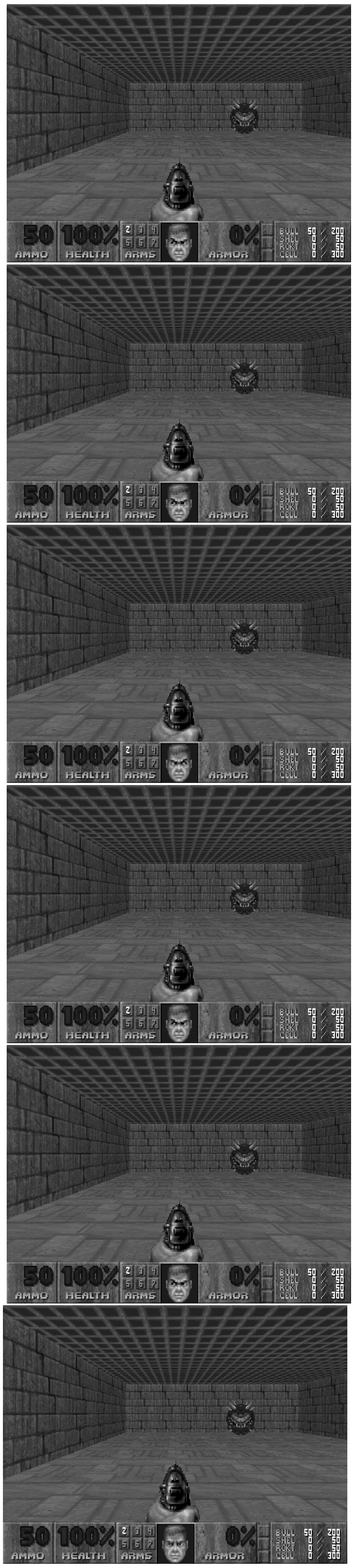


Figure A.5: Step 24 failed run Ensemble

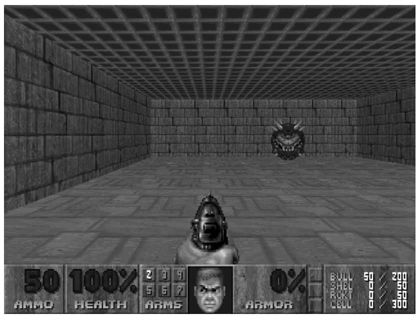
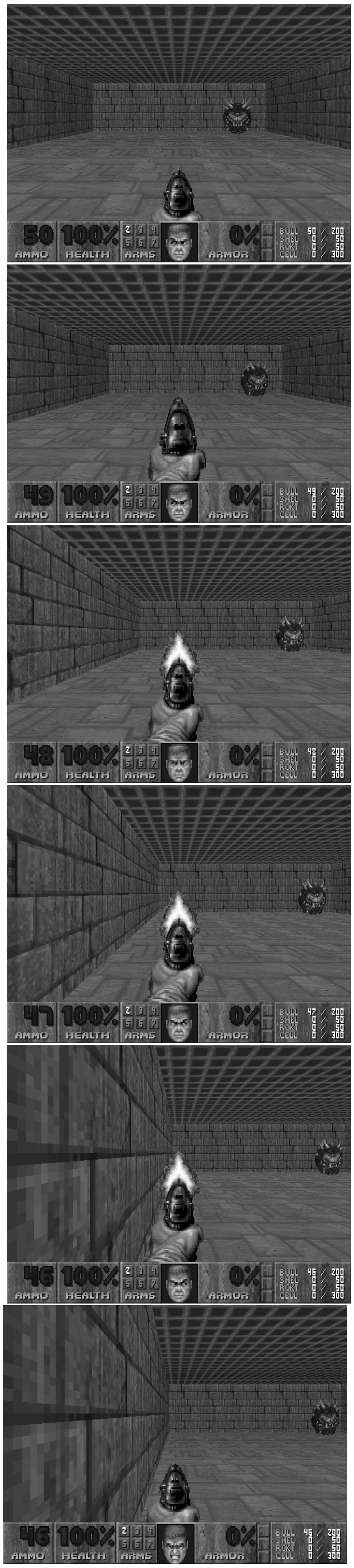


Figure A.6: Steps 0 - 5 failed run MC Dropout **Figure A.7: Steps 6 - 11 failed run MC Dropout**



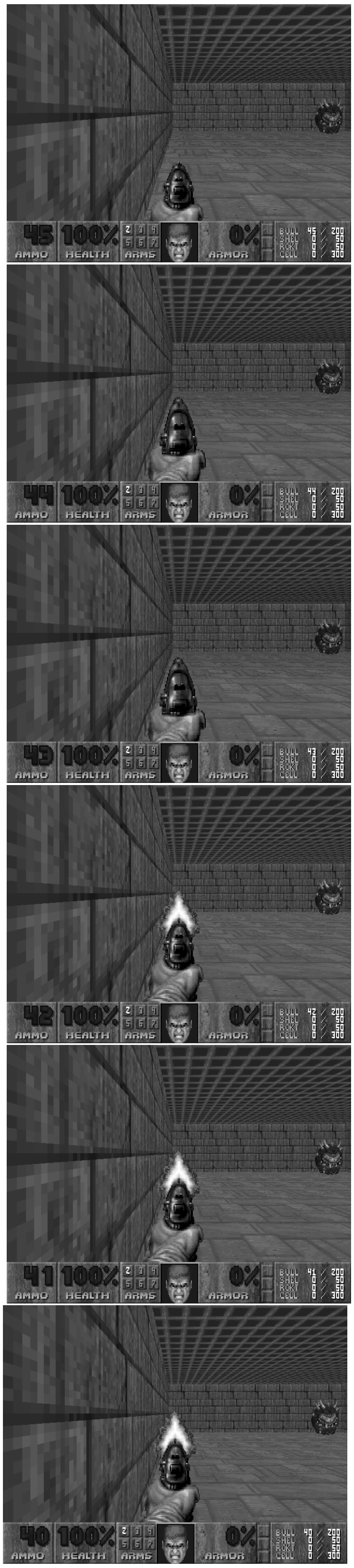


Figure A.8: Steps 12 - 17 failed run MC Dropout

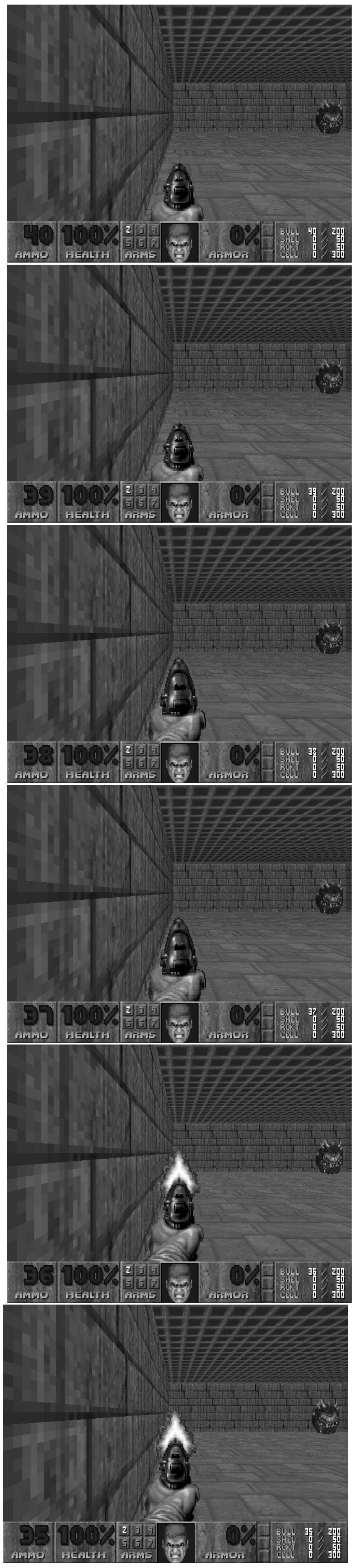


Figure A.9: Steps 18 - 23 failed run MC Dropout

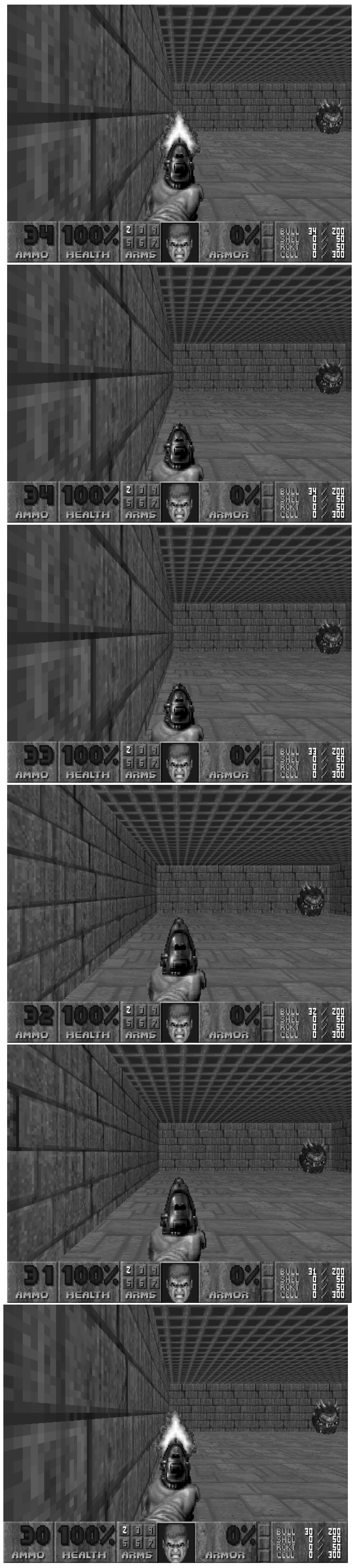
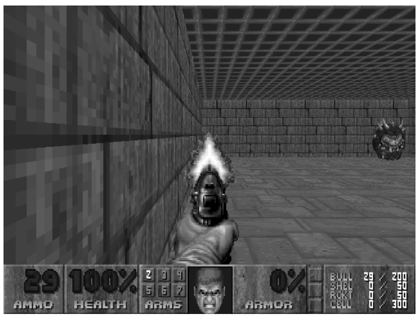


Figure A.10: Step 24 failed run MC Dropout


29
AMMO
100%
HEALTH
ARMS
0%
ARMOR