Improving Generalization in Game Agents with Data Augmentation in Imitation Learning

Derek Yadgaroff[1], Alessandro Sestini[2], Konrad Tollmar[2], Ayca Ozcelikkale[1], Linus Gisslén[2]
[1]*Uppsala University*, [2]*SEED - Electronic Arts (EA)*
derekyadgaroff@gmail.com, ayca.ozcelikkale@angstrom.uu.se, {asestini, ktollmar, lgisslen}@ea.com

***Abstract*—Imitation learning is an effective approach for training game-playing agents and, consequently, for efficient game production. However, generalization—the ability to perform well in related but unseen scenarios—is an essential requirement that remains an unsolved challenge for game AI. Generalization is difficult for imitation learning agents because it requires the algorithm to take meaningful actions outside of the training distribution. In this paper we propose a solution to this challenge. Inspired by the success of data augmentation in supervised learning, we augment the training data so the distribution of states and actions in the dataset better represents the real state–action distribution. This study evaluates methods for combining and applying data augmentations to observations, to improve generalization of imitation learning agents. It also provides a performance benchmark of these augmentations across several 3D environments. These results demonstrate that data augmentation is a promising framework for improving generalization in imitation learning agents.**

***Index Terms*—imitation learning, data augmentation, reinforcement learning, game AI**

I. Introduction

Playtesting is an essential component of modern video game production. Gameplay issues and bugs can significantly reduce the quality of the user experience. The process of game testing is usually done by designated human testers, both internal and external to the development team. However, due to the increased size and complexity of modern video games, manual playtesting is becoming expensive in regard to time, manpower, resources, and budget. Stakeholders in academia and industry have recently proposed automated testing solutions that could free up human resources to focus on more meaningful tasks, such as evaluating gameplay balance, difficulty, and player engagement.

Current automated testing techniques rely primarily on model-based automated playtesting [1, 2], but recent developments in Reinforcement Learning (RL) [3, 4, 5] have demonstrated that it is possible to train game Artificial Intelligence (AI) agents to reach human-level performance in complex video games, requiring however a vast amount of resources to achieve such training. These techniques can also train agents to explore game scenes and exploit environments to detect bugs [6, 7], thereby improving automated playtesting.

However, most techniques used for automated game testing suffer from an efficiency–generalization trade-off. Model-based and RL policies are known to easily overfit [8, 9], and achieving greater generalization requires numerous training samples. Imitation Learning (IL) was proposed to reduce the sample-inefficiency problem of RL [8, 10]. An IL agent learns from demonstrations rather than experience, allowing game designers to use prior knowledge to guide the agent towards its goal without additional knowledge of programming or machine learning. Even though IL can mitigate the sample-inefficiency problem, the generalization issue persists. To develop an IL agent that can address situations beyond the demonstration set and minimize the distributional shift issue—-the phenomenon where the distribution of states and actions encountered by the learning agent differs from those in the expert demonstration data, leading to sub-optimal performance—a significant number of datasamples are needed. However, producing a large demonstration set can be costly. Additionally, even with a considerable amount of data, IL might still overfit to the training environment. Ideally, we should test the IL policy across various versions of the game, simulating game developers modifying it. Incorporating priors for new environments may be challenging, as developers may not anticipate future changes. Hence, it is crucial to have a model that can generalize beyond the training environment.

In this paper, we investigate how to improve the generalization of IL agents by reducing sample-inefficiency via data augmentation. While data augmentation is commonly used to address generalization and efficiency in computer vision problems [11, 12], recent applications of image-based [13] and feature-based [14] state spaces have shown promising results in both online and offline RL. Consequently, we propose a comprehensive study of the different data augmentation techniques in IL, with a focus on augmented feature-based state spaces, as game AI typically operates with feature values instead of images. We address the following research questions: Which augmentation techniques perform best in a game AI IL setting? Which consistently perform well in different use cases? And which are ineffective in a game AI IL setting? To answer these questions, we train agents with different augmentation combinations in a training environment and evaluate their generalization across four test environments. We do not propose any new data augmentation techniques; instead, we conduct a comprehensive study of existing ones used in various domains and present an analysis of the combination of augmentations that yield the best results. This study shows that certain combinations are more suitable for this setting than others, with the best achieving a performance 1.6 times higher than that of the non-augmented agent.

Our main contribution aims to provide guidance to those seeking to use data augmentation to increase generalization in IL agents, particularly in the game testing context. It can also be used as a potential starting point for improving generalization of any feature-based game AI agent.

II. Related Work

While generalization has been widely discussed in reinforcement learning literature [9, 15], only a few works have focused on generalization and imitation learning. In this section, we review the most relevant literature related to this work.

A. Imitation Learning and Games

Imitation learning aims to distill, from a dataset of demonstrations, a policy that mimics the behavior of an expert demonstrator. It is assumed that the demonstrations come from an expert exhibiting near-optimal behavior. Standard approaches are based on Behavioral Cloning (BC), and mainly employ supervised learning [16, 17]. Generative Adversarial Imitation Learning (GAIL) is a recent IL technique based on a generator–discriminator approach [18].

IL has emerged as a widely accepted method for creating agents to perform tasks in 3D video games without RL [8, 19]. Amiranashvili et al. [20] used an agent trained solely through BC to play the game Minecraft. Chang et al. [21] used demonstrations to guide exploration, while Zhao et al. [22] employed a BC approach to teach agents based on game designers' expertise. Harmer et al. [23] trained agents using a combination of IL and RL with multi-action policies for a first person shooter game. Additionally, Tucker et al. [24] used an inverse RL technique to train agents capable of playing a variety of games in the Arcade Learning Environment suite [25]. Other notable examples include: the work of Sestini et al. [10], who combined RL, IL, and curiosity-driven learning to train playtesting agents; the work of Ferguson et al. [26], who proposed the use of dynamic time warping IL to imitate distinct playstyles; and the work of Pearce et al. [27], who introduce several innovations to make diffusion models suitable for imitating human behaviors in a first-person shooter game. However, all of these models must address the efficiency–generalization trade-off: higher levels of generalization require more demonstrations or more interactions with the environment. In this paper we examine the latest data augmentation techniques used in RL to increase generalization and efficiency and apply them in IL.

B. Generalization in Reinforcement Learning

In several use cases, policies produced with RL, IL or inverse RL have overfit to their training environments [15, 28]. Techniques like domain randomization and procedural content generation improve generalization and help prevent overfitting [29, 30]. In inverse RL, Sestini et al. [28] proved that with the correct training setup, domain randomization can mitigate overfitting and improve efficiency. However, domain randomization requires having a procedurally ready game or access to the environment code to implement randomized elements. In practical settings such as video game production, domain randomization is not always feasible.

Data augmentation has also been investigated in the context of RL and generalization [31]. Laskin et al. [32] proposed a thorough survey regarding data augmentation and RL for image data, while Zolna et al. [33] combined data augmentation and the GAIL algorithm to train a sample-efficient IL agent in a robotic setting. The present work takes inspiration from the S4RL algorithm of Sinha et al. [14]. These authors investigated the efficacy of using data augmentation in offline reinforcement learning on a feature-based state space. They proposed 7 different augmentation schemes and analyzed their performance with existing offline reinforcement learning algorithms. Furthermore, they combined the most successful data augmentation scheme with a state-of-the-art offline RL technique. Similarly, this present work investigates the effects of 6 feature-based data augmentation techniques for training game-playing agents with IL.

III. Method

Here we first review the main IL technique, behavioral cloning, and discuss the rationale behind its selection. Second, we review data augmentations previously studied in offline RL research and explain how we adapt these techniques to improve sample efficiency and generalization in our use case.

A. Behavioral Cloning

We begin by introducing the main algorithm used in this study. Since the focus is on data augmentation rather than selecting the optimal algorithm, we choose one of the simplest IL algorithms, BC. Recent literature has shown BC to be a good algorithm for IL in 3D games [8, 19]. Formally, we describe our domain as a Markov Decision Process (MDP) consisting of a state space $S \in \mathbb{R}^n$, an action space $A \in \mathbb{R}^m$, a transition dynamic function $p(s'|s,a) : S \times A \rightarrow p(S)$ and a parameterized policy $\pi_\theta : S \rightarrow p(A)$ that is a mapping from the state space S to a probability distribution over the set of actions A. Section IV provides a complete description of the particular state- and action-space used in this work. Note that in our setting we do not have a reward function since the agent is learning from expert demonstrations. Given a set of optimal demonstrations $\mathcal{D} = \{\tau_i \mid \tau_i = (s_0^i, a_0^i, ..., s_{T_i}^i, a_{T_i}^i),\ i = 1, .., N\}$ of N trajectories τ_i, each composed of a sequence of state-action pairs (s_k^i, a_k^i), we define the BC objective used to update the network as:

$$\arg\max_\theta \mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a|s)]. \tag{1}$$

This equation follows the maximum entropy objective: by increasing the log-probability of the policy $\pi(a_t|s_t)$ for a given $(a_t, s_t) \sim D$ pair sampled from the expert dataset $\mathcal{D}$, we enhance the probability of sampling that action in that specific state or similar states in proximity to s_t. At optimality, the policy π mimics the expert behavior represented by $\mathcal{D}$.

B. Data Augmentations

Inspired by the success of data augmentation for computer vision and online and offline RL, we apply a set of data augmentation techniques to our data. As we previously mentioned, our dataset consists of (s, a) pairs, but we augment only the state s and retain the original action a for the augmented state. To accomplish this, as noted by Sinha et al. [14], we assume that applying a small transformation to an input state s results in a physically realizable state $\hat{s}$, and that the original action a remains a valid action for the state $\hat{s}$. For the experiments detailed in Section V-B, to ensure this assumption holds, we use our prior knowledge of the environment to define our augmentations and select their hyperparameters.

We denote a data augmentation transformation as $\tau_i(\hat{s_t}|s_t)$, where $s_t \sim D$ is an original state sampled from the dataset D at timestep t, $\hat{s_t}$ is an augmented state, and $\tau_i \in \mathcal{T}$ is a stochastic augmentation with $\mathcal{T}$ representing the set of all available augmentations defined below. In contrast to Sinha et al. [14], who used only 1 augmentation per dataset, we apply from 1 to 3 sequential augmentations (see Section V-B). In this way we assess whether a combination of augmentations performs better than individual ones. For instance, if we augment the data with τ_1, τ_2, and τ_3, the resulting state will be $\hat{s_t} = \tau_3(\tau_2(\tau_1(s_t)))$.

As we see in Section IV-A, our state space is composed of both continuous and categorical values, and we choose to keep these separate when applying our augmentations. In particular, Gaussian noise, uniform noise, scaling, state mix-up and continuous drop-out are only applied to the continuous values of the state vector whereas semantic dropout is only applied to the categorical values. With that in mind, we next define the set of augmentations $\mathcal{T}$. For the sake of convenience, we represent vectors with i.i.d. components with a certain distribution with $s \sim X()$, e.g. $s \sim N(\mu, \sigma)$ represents a vector of i.i.d. Gaussian variables with mean μ and standard deviation σ.

Gaussian Noise: We sample $\epsilon \sim N(\mu, \sigma)$ and let $\hat{s}_t = \tau_{\text{gauss}}(s_t) = s_t + \epsilon$. We set $\mu = 0$ and experiment with different values of σ from $\{0.03, 0.003, 0.0003, 0.00003\}$.

Uniform Noise: We sample $\epsilon \sim U(-\lambda, \lambda)$ and let $\hat{s}_t = \tau_{\text{uni}}(s_t) = s_t + \epsilon$. We set $\lambda = 0.0003$.

Scaling: We sample $\epsilon \sim U(\alpha, \beta)$ and let $\hat{s}_t = \tau_{\text{sm}}(s_t) = s_t * \epsilon$ where $*$ represents element-wise multiplication. We set $\alpha = 0.0003$ and $\beta = 0.0006$. [32].

State-mixup: We sample $\epsilon \sim \beta(\alpha, \alpha)$ with $\alpha = 0.4$ and let $\hat{s}_t = \tau_{mixup}(s_t) = \epsilon * s_t + (1 - \epsilon) * s_{t+1}$ [34].

Continuous Dropout: We zero-out values of randomly chosen subset of continuous-valued elements of s_t. In particular, let ϵ be a vector of 1's with the same size as s_t. Let $\mathcal{S}$ be the indices of the elements of continuous-valued part of s_t. We sample n random and unique indices from $\mathcal{S}$ and set the elements of ϵ corresponding to these indices to zero. The resulting state is defined as $\hat{s}_t = \tau_{\text{drc}}(s_t) = s_t * \epsilon$. For our experiments, we set $n = 3$.

Semantic Dropout: We zero-out some random values of the *categorical* set of values in the state s_t. The procedure is the same as for continuous dropout, but we use $n = 12$. The resulting state is defined as $\hat{s}_t = \tau_{\text{drs}}(s_t) = s_t * \epsilon$.

IV. Experimental Setup

Here we detail the main components used in our evaluation, including the environment, the state space, the neural network architecture, and the training setup.

A. Environment and State Space

The environment used in this study is the same as used by Sestini et al. [8]. The game is an open-world city simulation, as illustrated in Figure 1(a) and Figure 1(b). The agent has a discrete action space of size 9, consisting of moving forward, moving backward, right rotation, left rotation, jumping, shooting, right sidestep, left sidestep, and no action. In our experiments, as we eliminate the presence of enemies from the environment, we do not utilize the shooting action. The state space is the same proposed by Sestini et al. [8] and consists of a goal position, represented as the $\mathbb{R}^2$ projections of the agent-to-goal vector onto the XY and XZ planes, normalized by the gameplay area size, along with game-specific state observations such as the agent's climbing status, contact with the ground, presence in an elevator, jump cooldown, and weapon magazine status. Observations includes a list of entities and game objects that the agent should be aware of, e.g., intermediate goals, dynamic objects, enemies, and other assets that could be useful for achieving the final goal. For these entities, the same relative information from agent-to-goal is referenced, except as agent-to-entity. For our use case, the entities are the button to open the door and the goal itself. Plus, a 3D semantic map is used for local perception. This map is a categorical discretization of the space around the agent. Each voxel in the map carries a semantic integer value describing the type of object at the corresponding game world position. We use a semantic map of size $5 \times 5 \times 5$. In this environment, an episode consists of a maximum of 750 steps. An episode is marked a success if the agent reaches the goal before the timeout. As we discuss in Section V-B, from this game we create a training environment and different testing ones.

B. Neural Network

We use the same neural network as in the work by Sestini et al. [8]. First, all the information about the agent and a goal is fed into a linear layer with ReLU activation, producing the self-embedding $x_{\text{a}} \in \mathbb{R}^d$, with $d = 128$. The list of entities is passed through a separate linear layer with shared weights, producing embeddings $x_{\text{e}_i} \in \mathbb{R}^d$, one for each entity e_i in the list. Each embedding vector was concatenated with the self-embedding, producing $x_{\text{ae}_i} = [x_{\text{a}}, x_{\text{e}_i}]$, with $x_{\text{ae}_i} \in \mathbb{R}^{2d}$. The list of vectors is then input to a transformer encoder with 4 heads and average pooling, producing a single vector $x_{\text{t}} \in \mathbb{R}^{2d}$. In parallel, the semantic occupancy map $M \in \mathbb{R}^{5\times5\times5}$ is first input into an embedding layer of size 8 with $\tanh$ activation and then into a 3D convolutional network with three convolutional layers, with 32, 64, and 128 filters, stride 2, and leaky ReLU activation. The output of the latter component is a vector embedding $x_{\mathrm{M}} \in \mathbb{R}^d$ that is concatenated with x_{t}, producing x_{Mt}. We then input x_{Mt} through one linear layer of size 256 and ReLU activation, and one final linear layer of size 9 and softmax activation, resulting in the action probability distribution.

C. Training Setup

Training is a supervised learning problem where the input data for each model is the unique dataset described in Section V-A. We use a neural network, as described in Section IV-B, built with the TensorFlow framework. Each model is trained for 300 epochs on a single machine with an AMD Ryzen Threadripper 1950X 16-Core Processor and an NVIDIA GV100 GPU, with an average training time of 336 seconds.

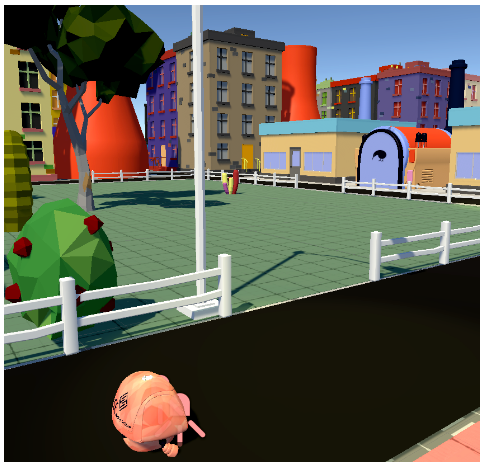
V. Experiments and Results

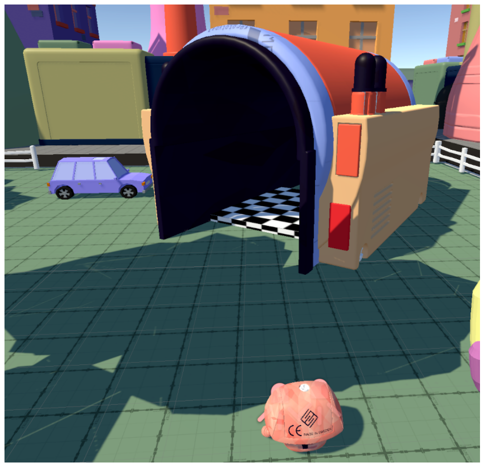
To evaluate the effect of data augmentation on an IL agent's generalization performance, we performed an exhaustive study of unique combinations of different data augmentations. We first collected a set of demonstrations using a training environment as described in Section IV-A. We then trained a model for each augmentation combination using this set. Subsequently, we tested our augmented models in four different test environments with modified designs compared to the training environment. We assessed the augmented models against each other and against a baseline model, which was trained only with the original dataset without augmentation.

In the training environment, we began by creating a set of demonstrations presenting the agent with the desired goal: the agent must reach a building, press a button to open a door, and enter the building before the door closes in order to reach the goal position. This task is the same Use Case 1 in the work by Sestini et al. [8]. The complexity was relatively low, with few obstacles between the agent's starting position and the goal. For the human player, there were two natural paths to reach the goal: diagonally through the park, or down the road and to the left. We used both paths to increase the range of possible states that the agent could encounter. We alternated between these two paths to maintain a balanced dataset for supervised learning. We provided human demonstrations totaling 78 episodes, or $15,380$ samples.

For the test environments, we created four distinct new environments from the initial training environment by making design changes to the original. This included moving and rotating the goal building, increasing the number of obstacles, and obstructing one of the direct paths to the goal building. For each environment, the agent had to complete the same task of navigating to the building, opening the door, and entering the goal position. Herein, we describe the list of modifications for each test environment:

Test Environment 1 - the position of the building containing the goal is slightly different from the training environment, with a slight rotation to the left. Additionally, we introduce 5


(a)


(b)

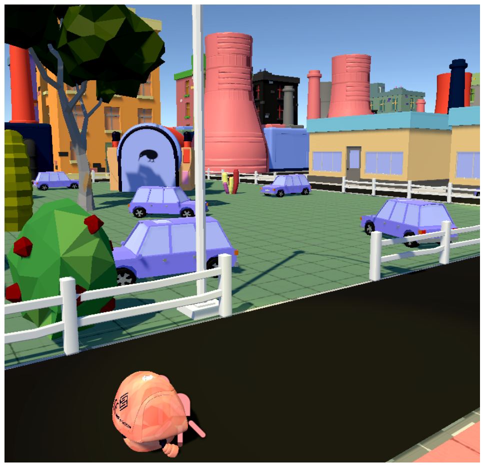

(c)

Fig. 1: (a) The original environment where human demonstrations were created and used for training models. (b) The player/agent must navigate to the building, press the button to open the door, and enter the goal state before the door closes. (c) A modified version of the original environment used for testing the model's performance and ability to generalize in a new environment.

TABLE I: An overview of the changes made to the training environment to create the 4 distinct test environments. These changes were implemented to evaluate the agent's generalization performance.

Environment	***Goal Position***	***Obstacles***	***Subjective Difficulty***
Original (Train)	x=0, $z = 0$, $rot = 0°$	1 obstacle	easy
Test 1	$x = 2$, $z = 8$, $rot = -17°$	6 obstacles	easy
Test 2	$x = 3$, $z = 0$, $rot = 0°$	17 obstacles	medium
Test 3	$x = -9$, $z = -25$, $rot = -21°$	5 obstacles	medium
Test 4	$x = 65$, $z = -45$, $rot = 150°$	5 obstacles	hard

more obstacles in the path from the agent's starting position to the goal. We categorize the subjective difficulty of this variation as *easy*, consistent with the training environment;

Test Environment 2 - The position of the building containing the goal differs slightly along the x-axis, while the rotation remains the same as in the training environment. However, a total of 17 obstacles obstruct the path from the agent's starting position to the goal. We categorize the subjective difficulty of this variation as *medium*;

Test Environment 3 - The position of the building is significantly different from the one in the training environment, as can be seen by comparing Figure 1(a) and (c). Furthermore, the building has a different rotation, with a higher degree compared to Test Environment 1. This environment also includes a total of 5 obstacles. We categorize the subjective difficulty of this variation as *medium*; and

Test Environment 4 - The position of the building is entirely different from the training environment. In contrast to the training environment, where the goal is in front of the agent facing its direction, the goal in this test environment is behind the agent's starting position, facing its back. This configuration requires the agent to follow a completely different trajectory to reach the goal, one that is not in the expert dataset. Moreover, there are a total of 5 obstacles. Due to the significant difference in the goal position, we categorize the subjective difficulty of this variation as *hard*.

A summary of design changes and subjective complexity is available in Table I and a visual for the modified environment is presented in Figure 1(c).

A. Augmented Models

As a baseline, we used the dataset consisting of the original 78 episodes of data without any applied augmentations. Using this data and the augmentations, we created different datasets, each featuring one or more augmentations. We refer to the models trained using these datasets as augmented models, and they are labeled with the corresponding dataset's augmentations. We applied all the augmentations listed in Section III-B and all their combinations, up to 3 augmentations, i.e, one dataset could be augmented with 1 augmentation, 2 augmentations, or 3 augmentations. Preliminary experiments showed that combining Gaussian and uniform noise together decreased the overall performances. For this reason, we excluded the datasets that have both Gaussian and uniform noise. This process resulted in a total of 38 different combinations of data augmentations.

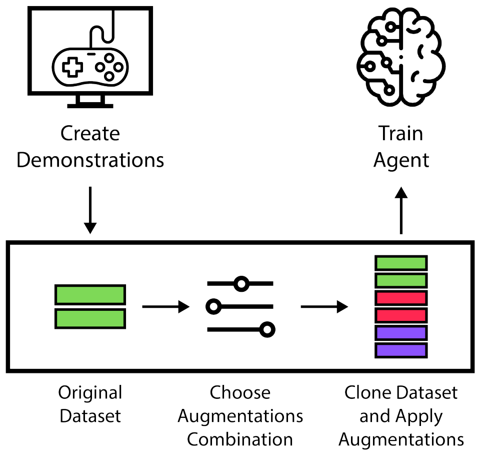


Fig. 2: Overview of the augmentation process. We start with the original set of demonstrations. Then, depending on the number and type of the selected augmentations, we create a new, unique dataset. For example, if we select two augmentations (Gaussian noise and semantic dropout), we start from the original dataset, apply Gaussian noise to it, and then augment the resulting dataset with semantic dropout.

We initialized a unique training dataset for each model by starting with the original demonstration data. We then cloned the original data, sequentially applied each augmentation from the combination, and added the resulting data to the dataset. Additional details of how we applied the augmentations are provided in Section III. We performed the augmentation step three times, resulting in a dataset four times as large as the original data. Augmentations were applied to the data in the order listed in III-B. This ordering was chosen to maximize the effect of changing the data by first performing translations followed by scaling, and to preserve dropout by performing those augmentations last in the composition. The process is outlined in Figure 2.

Dataset size and standard deviation of Gaussian noise are two significant parameters. To better understand their effects, we conduct a hyperparameter study over these parameters. The tested ranges were selected following preliminary experiments. All hyperparameters together with the variations are summarized in Table II. With these variations, we trained a total of 228 different models. Each model is a different combinations of augmentations, varying their hyperparameters. For instance, one model could be applying Gaussian noise with $\theta = 0.03$ plus state mixup to 50% of the original data. As we previ-

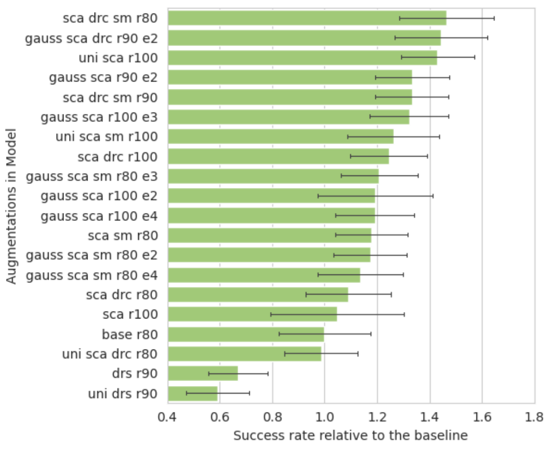


Fig. 3: Relative performance of the top 20 models, averaged over all experiments (i.e., the 4 test environments) with standard error. Abbreviations: gauss (Gaussian noise), uni (uniform noise) sca (scaling), sm (state-mixup), drc (continuous dropout), drs (semantic dropout), rX (X percentage of the original dataset used), and eX ($\sigma = 3 \cdot 10^{-X}$). Additional details about the augmentations are provided in Section III.

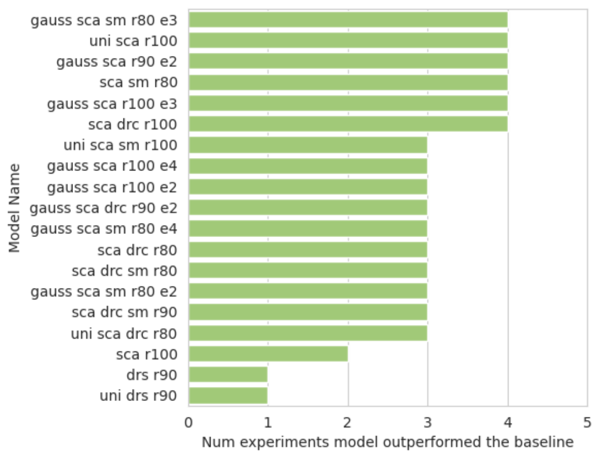


Fig. 4: Models that consistently outperform the baseline in the 4 different testing environments plus the training environment. For the abbreviations legend, see Figure 3.

ously said, we have a total of 38 combinations times the 6 different sizes of the dataset. For each model, we repeated the experiments for 10 different seeds.

Figure 3 shows the relative performance of each of the top 19 models, averaged over all experiments. The figure illustrates that augmentations can yield large improvements over the baseline model. However, the large standard deviation indicates that models may be sensitive to training parameters. In some instances, this can be interpreted as noise added to the data sending the agent into an unknown state, from which it cannot recover. We conduct further experiments to examine the standard deviation of the augmented models. Our findings suggest that the standard deviation of these models increases as the dataset size increases. This particular effect cannot be seen in the figure due to the fact that only a subset of the models are shown. For instance, the standard deviation of the model augmented with Gaussian noise ranges from 0.07 when using 50% of the dataset to 1.11 when using 100%. Similar trends were observed with all the other augmentations.

B. Evaluation

During testing, agents run in each environment for 100 episodes. Episodes are reset after a timeout of 750 steps. The episode is recorded as a success if the agent reaches the goal before the timeout. For an initial evaluation, we run the full set of 228 models in the initial training environment. Using the mean success rate over 10 seeds of each model as a selection criterion, we narrow the set of models down to 40 to evaluate generalization performance in the testing environments. This set of models consists of the following two groups: 1) the 19 top-performing models and 2) a selection of 20 of the lowest-performing models. For the lowest-performing group, we selected models with relative success rates ranging from 0.00 to 0.39 to obtain a mix of both low-performing and failing models. In addition, we selected the highest-performing baseline model and included it in the top-performing group. Our main focus is to evaluate how data augmentation can impact the generalization performance of the trained agents. For this reason, we evaluate only the top 19 models found in the initial evaluation. We are interested in finding the relative success rate of each augmented model compared to the baseline for each test environment. This provides a quantitative measure of the improvements, if any, that are achieved through the data augmentation combination, so we use this as our primary benchmark. We also seek to identify consistency in model performance across environments, as it might suggest a clear set of combinations to use—or to avoid—in similar scenarios, as a starting point for other game environments.

Our first assessment is whether data augmentations could improve an imitation learning agent's ability to generalize to unseen changes in an environment. The results, summarized in Table III, show that these augmented models outperform the baseline in each of the test environments. Note that a mean relative success rate of 1.X corresponds to an improvement of X%. Hence, the mean relative success rates showed an improvement of 20% up to 80% for different environments but with a relatively high variance over different models.

The next assessment is to determine whether a combination of augmentations consistently improves generalization regardless of the environment. Figure 4 shows that 6 different models outperform their respective baselines in 4 of the 5 environments. These 6 models each includes a combination of at least 2 augmentations and use at least 80% of the data. While these models are the most consistent in outperforming the baseline, none of them have the highest relative success rates, but 3 of them are in the top 6 in the same evaluation (see Figure 3). This suggests that there is a trade-off between best achievable generalization performance and consistency over

TABLE II: Summary of available hyperparameters. Values in square brackets were experimentally varied; all others remained fixed. The ranges were selected following preliminary experiments.

Hyperparameter	*Values*
Gaussian noise	$\mu = 0, \sigma = [0.03, 0.003, 0.0003, 0.00003]$
Data size	$[50\%, 60\%, 70\%, 80\%, 90\%, 100\%]$
State mixup	$\alpha = 0.4, \beta = 0.4$
Uniform noise	$\text{low} = -0.0003, \text{high} = 0.0003$
Dropout, num. continuous features	3
Dropout, num. semantic features	12
Max. augmentation combinations	3
Num. data clones	3

TABLE III: Summary of average performance for models which outperformed the baseline. Models shown are taken from the set of 19 top-performing models. Additional details of how the models were chosen are provided in Section V-A.

Environment	*Mean relative success rate*	*STD relative success rate*	*Number of models which outperform the baseline in each environment*
Train	1.210187	0.854240	19
Test 1	1.765625	1.214967	16
Test 2	1.197297	0.976769	5
Test 3	1.800574	1.525612	17
Test 4	1.234450	1.028380	1

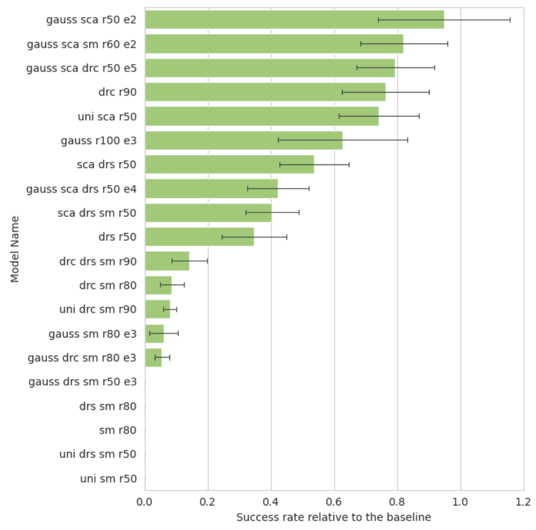


Fig. 5: Relative performance, averaged over all environments, is consistently poor or failing for the bottom group of models. Abbreviations: gauss (Gaussian noise), uni (uniform noise) sca (scaling), sm (state-mixup), drc (continuous dropout), drs (semantic dropout), rX (X percentage of the original dataset used), and eX ($\sigma = 3 \cdot 10^{-X}$).

all testing environments.

For the 20 lowest-performing models in the training environment defined above, we assess whether their poor relative performance is consistent between the training environment and across the testing environments. Figure 5 shows the average success rate relative to the baseline across all environments remains quite low and these results are consistent with the results from the training environment.

Finally, this study aims to determine if there is a single most effective augmentation for agent performance and if it exists, to identify it. To do this, we grouped all models tested in the testing environment by augmentation, and computed the average success rate relative to the baseline. For example, in the Gaussian noise group, we include all models containing *gauss* in their set of augmentations. From this analysis, we found the following average relative success rates: scaling (1.27), state mixup (1.26), continuous dropout (1.26), Gaussian noise (1.25), uniform noise (1.02), and semantic dropout (0.50). These investigations, combined with the consistency study, highlight scaling as one of the most promising augmentations. State mixup is one of the most effective augmentations for generalization in certain environments but it did not outperform the baseline as consistently as models augmented with scaling. Gaussian noise and continuous dropout follow the same trend, with the latter exhibiting the lowest consistency. Lastly, uniform noise had the least impact on generalization while semantic dropout clearly had a negative effect on generalization. Nevertheless, due to the sensitivity to the training parameters previously mentioned, particularly for noise-based augmentations, these results should be considered as promising starting points for further studies to cover a wider range of the possible augmentations with different parameters.

VI. Conclusion and Discussions

In this paper we have conducted a comprehensive study of data augmentation in imitation learning. Training self-learning agents with either reinforcement learning or imitation learning, involves a generalization–efficiency trade-off: we can increase generalization, but at the cost of sample efficiency. Data augmentation can help improve generalization while maintaining sample efficiency. Building on the success of data augmentation in supervised learning and reinforcement learning, we investigated data augmentation techniques for feature-based imitation learning and their effects on the training of imitative agents. We evaluated our agents in four distinct 3D test environments. The main findings of this paper are twofold. First, data augmentations can indeed improve performance in an imitation learning setting. The imitative agents demonstrated improved generalization in previously unseen situations when trained on an augmented dataset. Second, our study indicates that certain combinations of data augmentations consistently enhance performance across a variety of generalization environments, even if the performance of individual augmentations may vary depending on the parameters used.

This improvement in generalization with augmented data is not surprising. Data augmentation in supervised learning involves manipulating the images to resemble a visual system. Rotation, cropping, translation, noise, etc., are augmentations that can be translated into game-state observations. The challenge lies in identifying which augmentations contribute to generalization and which do not. Our preliminary study revealed that combining multiple augmentations yields better results than using single augmentations alone. At the same time, of the single augmentations tested, scaling emerged as one of the most consistently performing throughout our study, followed by continuous dropout and Gaussian noise.

This study provides a promising research direction for data augmentation and imitation learning. However, several limitations warrant consideration. Our preliminary results indicate that certain augmentations are particularly sensitive to the training parameters, thus requiring a more thorough study to assess their contribution. Furthermore, we tested these augmentations only in an internal game environment, and exclusively on navigation and interaction tasks. Exploring these augmentations in different environments, and with different tasks, should prove valuable. Furthermore, although most game AI agents use a similar state space, alternative solutions exist. Hence, testing these augmentations in a larger range of environments, task and observation spaces is a promising future research direction.

References

[1] S. Stahlke, A. Nova, and P. Mirza-Babaei, "Artificial playfulness: A tool for automated agent-based playtesting," in *CHI Conference on Human Factors in Computing Systems*, 2019.

[2] C.-S. Cho, K.-M. Sohn, C.-J. Park, and J.-H. Kang, "Online game testing using scenario-based control of massive virtual users," in *International Conference on Advanced Communication Technology*, 2010.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] 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, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint 1912.06680*, 2019.

[5] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs *et al.*, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[6] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, "Augmenting automated game testing with deep reinforcement learning," in *Conference on Games (CoG)*, 2020.

[7] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," in *Conference on Games (CoG)*, 2021.

[8] A. Sestini, J. Bergdahl, K. Tollmar, A. D. Bagdanov, and L. Gisslén, "Towards informed design and validation assistance in computer games using imitation learning," in *NeurIPS workshop on Human In the Loop Learning*, 2022.

[9] N. Justesen, R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating generalization in deep reinforcement learning through procedural level generation," *NIPS Workshop on DRL*, 2018.

[10] A. Sestini, L. Gisslén, J. Bergdahl, K. Tollmar, and A. D. Bagdanov, "Automated gameplay testing and validation with curiosity-conditioned proximal trajectories," *IEEE Transactions on Games*, 2022.

[11] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[12] Q. Xie, Z. Dai, E. Hovy, T. Luong, and Q. Le, "Unsupervised data augmentation for consistency training," *Advances in neural information processing systems*, vol. 33, pp. 6256–6268, 2020.

[13] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Mastering visual continuous control: Improved data-augmented reinforcement learning," *arXiv preprint arXiv:2107.09645*, 2021.

[14] S. Sinha, A. Mandlekar, and A. Garg, "S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics," in *Conference on Robot Learning*. PMLR, 2022, pp. 907–917.

[15] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *International conference on machine learning*. PMLR, 2020, pp. 2048–2056.

[16] M. Bain and C. Sammut, "A framework for behavioural cloning." in *Machine Intelligence 15*, 1995, pp. 103–129.

[17] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *2011 International Conference on Artificial Intelligence and Statistics (ICAIS)*, 2011.

[18] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *International Conference on Neural Information Processing Systems*, 2016.

[19] T. Pearce and J. Zhu, "Counter-strike deathmatch with large-scale behavioural cloning," in *2022 IEEE Conference on Games (CoG)*. IEEE, 2022, pp. 104–111.

[20] A. Amiranashvili, N. Dorka, W. Burgard, V. Koltun, and T. Brox, "Scaling imitation learning in minecraft," *arXiv preprint arXiv:2007.02701*, 2020.

[21] K. Chang, B. Aytemiz, and A. M. Smith, "Reveal-more: Amplifying human effort in quality assurance testing using automated exploration," in *2019 Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[22] Y. Zhao, I. Borovikov, F. D. M. Silva, A. Beirami, J. Rupert, C. Somers, J. Harder, J. Kolen, J. Pinto, R. Pourabolghasem *et al.*, "Winning isn't everything: Enhancing game development with intelligent agents," *Transactions on Games*, 2020.

[23] J. Harmer, L. Gisslén, J. del Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjöö, and M. Nordin, "Imitation learning with concurrent actions in 3d games," in *Conference on Computational Intelligence and Games*. IEEE, 2018.

[24] A. Tucker, A. Gleave, and S. Russell, "Inverse reinforcement learning for video games," in *NIPS Workshop on Deep Reinforcement Learning*, 2018.

[25] 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.

[26] M. Ferguson, S. Devlin, D. Kudenko, and J. A. Walker, "Imitating playstyle with dynamic time warping imitation," in *International Conference on the Foundations of Digital Games*, 2022, pp. 1–11.

[27] T. Pearce, T. Rashid, A. Kanervisto, D. Bignell, M. Sun, R. Georgescu, S. V. Macua, S. Z. Tan, I. Momennejad, K. Hofmann *et al.*, "Imitating human behaviour with diffusion models," *arXiv preprint arXiv:2301.10677*, 2023.

[28] A. Sestini, A. Kuhnle, and A. D. Bagdanov, "Demonstration-efficient inverse reinforcement learning in procedurally generated environments," in *2021 Conference on Games (CoG)*. IEEE, 2021, pp. 1–8.

[29] ——, "Deepcrawl: Deep reinforcement learning for turn-based strategy games," *arXiv preprint arXiv:2012.01914*, 2020.

[30] C. Romac, R. Portelas, K. Hofmann, and P.-Y. Oudeyer, "Teachmyagent: a benchmark for automatic curriculum learning in deep rl," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9052–9063.

[31] A. Mumuni and F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, p. 100258, 2022.

[32] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in neural information processing systems*, vol. 33, pp. 19 884–19 895, 2020.

[33] K. Zolna, S. Reed, A. Novikov, S. G. Colmenarejo, D. Budden, S. Cabi, M. Denil, N. de Freitas, and Z. Wang, "Task-relevant adversarial imitation learning," in *Conference on Robot Learning*, 2021.

[34] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.