AI

MDPI

Article

Evaluating the Perception, Understanding, and Forgetting of Progressive Neural Networks: A Quantitative and Qualitative Analysis

Lucía Güitta-López, Jaime Boal and Álvaro J. López-López *

Institute for Research in Technology (IIT), ICAI School of Engineering, Comillas Pontifical University, Rey Francisco, 4, 28008 Madrid, Spain; lucia.guitta@iit.comillas.edu (L.G.-L.); jaime.boal@iit.comillas.edu (J.B.)
* Correspondence: alvaro.lopez@iit.comillas.edu

Abstract

The use of virtual environments to collect the experience required by deep reinforcement learning models is accelerating the deployment of these algorithms in industrial environments. However, once the experience-gathering problem is solved, it is necessary to address how to efficiently transfer the knowledge from the virtual scenario to reality. This paper focuses on examining Progressive Neural Networks (PNNs) as a promising transfer learning technique. The analyses carried out range from studying the capabilities and limits of the layers responsible for learning the state representation from a pixel space, which could arguably be the convolutional blocks, to the forgetting agents suffer when learning a new task. Introducing controlled visual changes in the environment scene can lead to a performance degradation of 50.3% in the worst-case scenario. These visual discrepancies significantly impact the agent's learning time and accuracy when using a PNN architecture. Regarding the PNN forgetting assessment, partial forgetting occurs in two of the three environments analyzed, those where the agent masters its new task. This could be due to a balance between the relevance of the new features learned and the ones inherited from the teacher agent.

Keywords: deep reinforcement learning; progressive neural networks; sim-to-real; sample efficiency; representation learning

1. Introduction

To address sequential decision problems requiring interaction between the process and the controlled system, Deep Reinforcement Learning (DRL) [1] emerges as a prominent tool, combining principles from Reinforcement Learning (RL) [2] and Deep Learning [3,4]. DRL overcomes RL limitations in high-dimensional state spaces and exhibits representation learning capabilities [5], reducing the need for manual feature engineering. Agents using DRL learn from experience through interaction with the environment, enabling applications in diverse situations, including non-stationary scenarios or cases where an accurate model is unavailable. However, a drawback is the substantial amount of experience needed [1], known as the *sample efficiency problem*. This issue can be addressed through virtual environments and physics engines simulating behavior of the asset or process under varying circumstances, efficiently acquiring more experience without significant time and financial outlays.

Several transfer learning techniques aim to bridge the reality gap, also known as the *sim-to-real problem*, arising from discrepancies between virtual and real environments.

Academic Editors: Ergina Kavallieratou, Nikolaos Vasilopoulos and Paraskevas Diamantatos

Received: 18 December 2025
Revised: 20 February 2026
Accepted: 12 March 2026
Published: 31 March 2026

Progressive Neural Networks (PNNs), introduced by Rusu et al. [6], have brought attention for their modular and adaptable architecture. In essence, PNNs transfer knowledge from a teacher neural network, proficient in a virtual task, to a shallower student network adapting to a different task or setup [7]. The approach involves training a large model with synthetic experience and then transferring the acquired knowledge through learned layer connections to a smaller model, refining results in the real setup with reduced experience.

The training approach of the teacher network significantly influences the performance of the DRL agent. This component of the architecture must efficiently share knowledge among multiple specialized student networks. The synthetic experience it incorporates should be sufficiently diverse for the agent to handle dynamic scenarios, yet specific enough to reduce the required experience to achieve fast learning in the student setup. Currently, to the best of our knowledge, this aspect remains an open research topic, with no dedicated papers addressing this concern. In our proposal, a DRL agent trained under a PNN architecture confronts challenging environments, whose design is based on the performance of a pre-trained baseline teacher. These adversarial scenarios contribute to three key aspects: (1) assessing the robustness of the teacher agent to specific changes in visual features; (2) evaluating its learning dynamics when fine-tuned for setups with varying difficulty levels; and (3) analyzing the agent's forgetting related to the teacher task. Throughout this paper, we will understand *adversarial environments* as static environments designed to maximize or cause a performance degradation in the agent.

To properly analyze and evaluate the learning mechanisms of the PNN architecture, all experiments in this work are conducted under a sim-to-sim approach, where the drivers that may lead to discrepancies between environments are fully controlled.

The remainder of the paper is structured as follows: Section 2 describes some of the most relevant state-of-the-art transfer learning and sim-to-real techniques. Section 3 goes through the fundamentals of PNNs and the details of our implementation. If the reader is familiar with this topic, they can safely skip the section. Section 4 presents the problem solved and the setup used as the baseline. Section 5 introduces the methodology followed to design and evaluate the adversarial environments and PNN-like agents. The results of all the experiments conducted are discussed in Section 6. Finally, Section 7 summarizes the essential takeaways and lays the groundwork for future research.

2. Related Work

This paper focuses on the same task as in ref. [7], approaching a robotic arm to a static target randomly placed in the workspace. This section delves into model-free proposals, particularly applied to industrial processes [8], cyber-physical systems [9], or robotic grasping and control [10–12].

As outlined in the introduction, one alternative to address the sim-to-real problem is to find an efficient transfer learning technique that enables the reuse of existing knowledge between domains.

Domain Adaptation (DA) [13] is a transfer learning technique based on translating data from a source domain to a target domain, unifying both feature spaces. Ref. [14] proposed a zero-shot approach, employing a multi-stage RL strategy to first learn the vision task and then the acting policy. Ref. [15] developed a model to learn a latent state representation in simulation and, through its combination with real data, adapts the representation to the real domain. Ref. [16] suggested a pixel-level DA model to reduce the number of real samples needed to train a model in a grasping task, trained with both adapted images from a Generative Adversarial Network (GAN) [17] and real ones. Ref. [18] reversed the flowchart, translating data from real images to the synthetic domain (real-to-sim) to "make the robot feel at home" during deployment. Ref. [19] combined RL

https://doi.org/10.3390/ai7040120

under a CycleGAN architecture, transforming images from simulation to reality and then back to simulation while maintaining the scene consistency loss. Finally, ref. [20] combined DA with Meta Reinforcement Learning (MetaRL) [21,22] to train a model under different robot dynamic conditions. The main weakness of DA in comparison to our work is the need for an additional model that has to be trained.

Domain Randomization (DR) [23] is a well-known sim-to-real technique that involves training an agent over a distribution of environments with randomized features. DR is widely used for bridging the reality gap due to its ease of implementation and adaptability to various DRL problems. An alternative to the vanilla DR approach, where all environment features are randomized, is to selectively sample attributes [24]. Ref. [25] combined distributed DRL and DR to learn optimal policies for dexterous in-hand manipulation in a virtual environment with randomized physical properties and appearance features. Ref. [26] presented a methodology to automatically set the tunable features of the virtual environment. A Bayesian approach proposed in [27,28] is used to select parameters instead of the common uniform distribution, which might yield sub-optimal results. Ref. [29] implemented active DR, identifying the most relevant environment variations by leveraging discrepancies between the current policy outcomes in the randomized scene and the baseline instance. Conversely, ref. [30] proposed a structured DR alternative where scene information was used to generate synthetic data that were more similar to the real scenario, avoiding infeasible or unreal situations. In a similar vein, [31,32] advocated using knowledge about the real environment to adapt randomized images during training in the virtual world. Ref. [31] utilized real images processed with an image segmentation model to create an initial dataset to which DR will be applied, whereas [32] adapted the simulation parameter distribution based on results obtained after a few roll-outs in the real environment. Although DR can significantly improve agent performance, as shown in [33], it may not be entirely suitable on its own if there are several sources of discrepancy between environments, as randomizing all these attributes comes with a high computational cost.

Other approaches utilized Imitation Learning [34] to address the reality gap by leveraging samples from a human expert or other policies. Ref. [35] employed behavior cloning to reduce agent exploration in sparse rewards environments, and Ref. [36] applied Inverse Reinforcement Learning [37,38] to learn various motion types across a wide range of assets and scenarios. Ref. [39] utilized Generative Adversarial Imitation Learning (GAIL) [40], a combination of imitation learning and GANs, to successfully train end-to-end visuomotor policies in a virtual robotic arm, later transferring it to a real robot. Conversely, ref. [41] employed human demonstrations to learn a DR distribution used for policy training. The primary drawback of these techniques is the reliance on access to an expert, which is not always feasible.

Alternative solutions transfer knowledge through the policy, either via distillation or reuse. Ref. [42] introduced policy distillation based on the knowledge distillation term proposed in [43] as a framework consisting of a student network learning from a teacher net. Ref. [44] applied this concept to learn two tasks in simulation and combined them into a single policy deployed in the real scenario. Ref. [45] combined DR and policy distillation into a cyclic policy distillation framework, creating a local policy per randomized parameter. Ref. [6] proposed PNNs, in which knowledge is transferred between a teacher column and a student column as learned representations. Each PNN column represents a (deep) neural network model, and an agent solving a particular Markov Decision Process (MDP) [46]. Both columns are connected through learnable lateral connections that enable knowledge sharing. PNNs fundamentals and our particular implementation are explained in Section 3.

PNNs are framed in the continual or lifelong learning paradigm [47,48]. This involves mastering a specific task and continuously learning other tasks by reusing previously

acquired knowledge [49,50]. Within this context, the experience and know-how assimilated must not be forgotten. PNNs mitigate catastrophic forgetting through their architecture and learning procedure, where parameters from previous columns are kept frozen. However, this does not ensure zero knowledge loss. To quantify and analyze this potential drop, we assessed and analyzed the forgetting agents' experience under two different PNN architectures. Ref. [7] applied PNNs to solve the sim-to-real problem. The task they solved was reaching a target with a robotic arm using a 64×64 pixel RGB image as the model input. Although they achieved great results, the similarities between their virtual and real setups left the question of what would have happened if both environments had visual mismatches open. Ref. [51] used PNNs to learn the state transition distribution in a model-based approach. They solved the manipulation problem using the robot joints' state, not images, and adapted the formulation presented in [6] to their problem. To compare the results and conclusions obtained, we proposed a PNN architecture deeply based on the concepts explained in [7], and we solved the same task but under different scenes. On the other hand, we implemented and tested an additional architecture to study the forgetting that might appear and evaluate whether it was dependent or not on the approach.

In our previous work [33], we demonstrated that in a PNN-like architecture, the performance of the first-column agent could significantly improve within the same training effort by applying DR to the camera pose and preserving the rest of the visual features. In the present paper, we keep the camera pose fixed and investigate, firstly, how the agent's behavior can be influenced when solving the approach to a target using only visual inputs under challenging scenarios with feature discrepancies, and secondly, assess its forgetting after learning under these new conditions.

Generally, DRL agents rely on high-dimensional visual inputs. Ref. [52] provided a comprehensive survey of state representation learning methods, highlighting how structured latent embeddings can improve sample efficiency and robustness in DRL settings. These approaches are especially relevant when visual observations contain variations that are not directly related to the task objective. In a similar vein, methods that incorporate auxiliary objectives or learn joint state-action representations have shown improvements in continuous control and visual RL tasks by enriching the learned embedding space [53]. Moreover, recent benchmarks designed to explicitly evaluate visual state representation learning in RL environments reveal the challenges arising from distribution shifts and visual discrepancies [54]. These findings further motivate the analysis conducted in this work, where the robustness of learned visual representations is assessed under controlled environment variations.

3. Background

This section provides a description of PNNs plus a detailed explanation of our implementation. Readers familiar with this topic can safely skip the section.

Progressive Neural Networks (PNNs)

As briefly introduced in Section 2, PNNs [6] provide a promising solution to avoid catastrophic forgetting and to leverage knowledge learned from one task when transferring it to an agent performing another related task. The representations learned by one or more teacher networks are shared with a smaller student network. These networks, often referred to as columns due to their graphical layout (Figure 1), represent a DRL agent solving an MDP, which is in some way related to the MDPs solved by other columns. The knowledge transfer occurs through trainable lateral connections that link the layers of these networks. During the training process of the second column, the learnable parameters are the weights of the second column and the weights of the lateral connections. The weights

of the first column remain frozen. This approach helps address the catastrophic forgetting issue by preserving the knowledge encoded in the first agent. However, we argue that this does not entirely prevent the second agent from forgetting some of the previously knowledge learned.

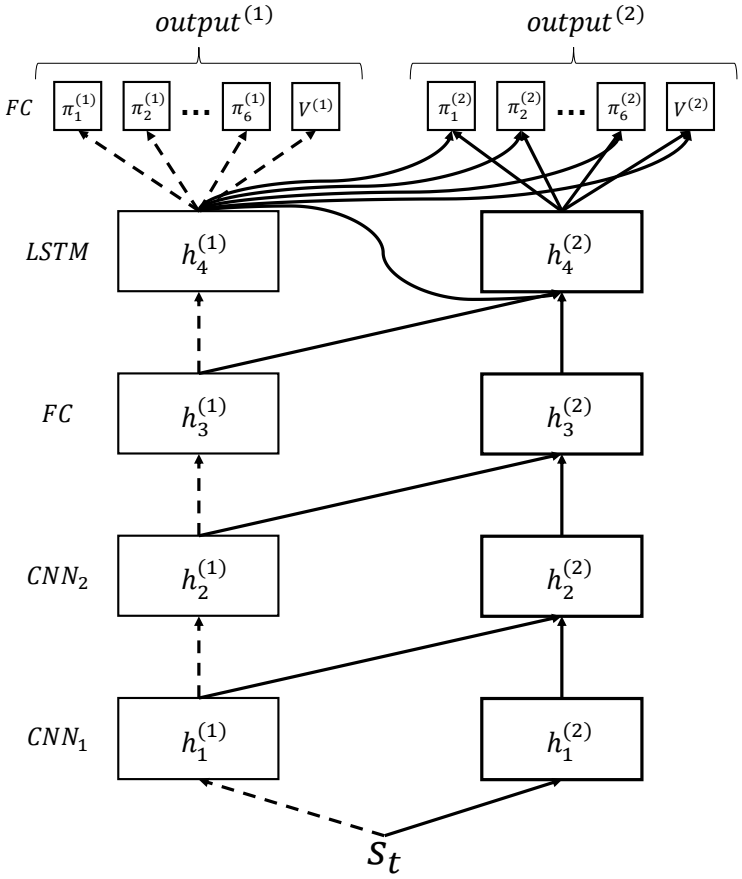

Figure 1. Schematic diagram of the PNN architecture implemented. This layout has been modified with respect to the one presented in [7] by adding lateral connections to the output layer, as mentioned in their text. The output layer has an FC block per each robot joint policy, and one for the state-value function. The dashed lines represent frozen parameters.

Our PNN implementation is based on the architecture suggested by Rusu et al. in [7], where they apply the PNN theoretical background explained in [6] to the sim-to-real problem. Although the agent's goal is the same, the MDP definition and the environment setup differ. See Section 4 and [33] for more details.

Figure 1 presents the general structure of the implemented PNN architecture. It consists of two columns with identical layer topology but different capacities. The first column (teacher) contains a larger number of parameters, while the second column (student) is shallower. We argue that the convolutional layers (CNN_1 and CNN_2) are primarily responsible for learning the state representation, whereas the subsequent blocks (FC, LSTM, and output layers) focus on policy and value estimation. The task solved is approaching a target (Section 4.1), so the policy is estimated by the actor using six FC blocks, one per robot joint. We used the A3C [55] as the learning algorithm due to its well-known results and parallelization capabilities and to ensure consistency in our analyses with the original PNN implementation proposed by [7]. Note that Figure 1 includes the lateral connections links between the LSTM and the output's FC layers, while in ref. [7], they were specified in the text description, but were not shown in their diagram.

The lateral connections between columns follow the formulation originally introduced in [42]. For a given layer i in column k, the hidden activation $h_i^{(k)}$ is defined as follows:

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right) \tag{1}$$

where $W_i^{(k)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the layer weight matrix, $h_{i-1}^{(k)}$ is the hidden activation of the previous layer, $U_i^{(k:j)} \in \mathbb{R}^{n_i \times n_j}$ are the lateral connection weights from previous columns, and $f(\cdot)$ is a non-linear activation function (ReLU in our case). Bias terms are omitted for clarity.

In our specific two-column setup ($k = 2$), Equation (1) reduces to:

$$h_i^{(2)} = f\left(W_i^{(2)} h_{i-1}^{(2)} + U_i^{(2:1)} h_{i-1}^{(1)}\right) = f\left(B_i^{(2)} + A_i^{(1)}\right). \tag{2}$$

Figure 2 provides a layer-wise visualization of this formulation, where each block explicitly shows the intra-column transformation ($W_i^{(2)} h_{i-1}^{(2)} = B_i^{(2)}$) and the lateral contribution ($U_i^{(2:1)} h_{i-1}^{(1)=A_i^{(1)}}$). This figure is a graphical instantiation of Equation (2).

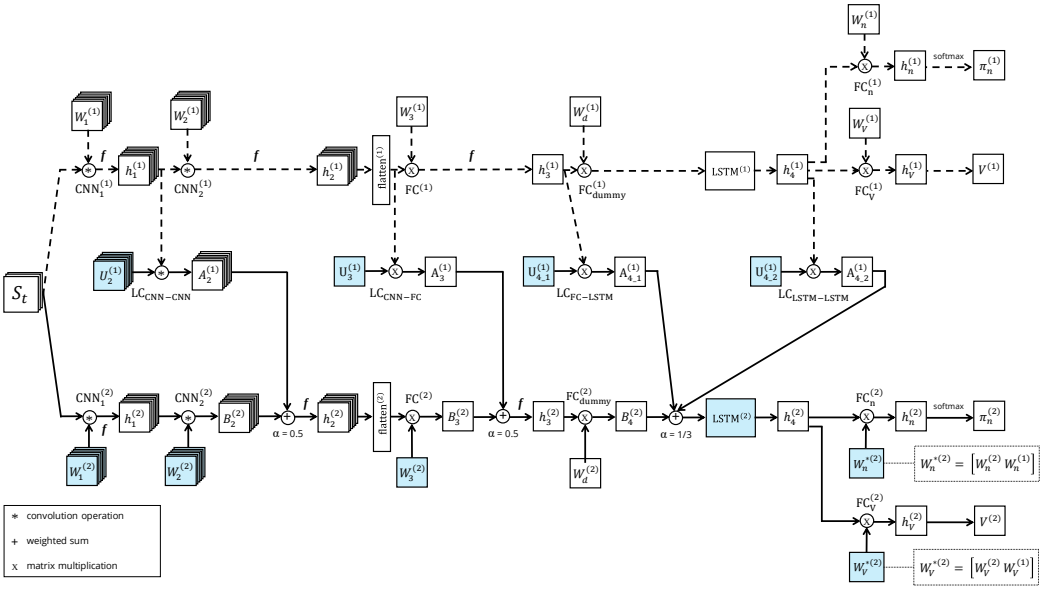

Figure 2. Detailed diagram of the PNN architecture implemented. The top stream corresponds to the first column (teacher), and the bottom is the student. There are four explicit lateral connections plus the connections in the output layers. All the connections, except the last one, which is a concatenation, are implemented as weighted sums. The dashed lines indicate frozen parameters during the student training phase. The blue blocks are learnable parameters. f represents the ReLU function.

The dashed lines represent frozen parameters during the training phase of the second-column (student) layers, while the blue blocks are the learnable parameters. The tensors $W_d^{(1)}$ and $W_d^{(2)}$ are auxiliary identity mappings introduced solely to ensure dimensional consistency in the LSTM lateral connections. They contain constant values and are not trainable parameters. Their purpose is to maintain compatible tensor dimensions when combining LSTM hidden and cell states across columns, without altering the learned policy. They do not contribute to the model capacity nor affect gradient updates. Lateral connections are implemented as weighted feature combinations. When a block receives a single lateral input, both intra-column and lateral contributions are weighted by $\alpha = 0.5$. When two lateral inputs are present, all contributions are uniformly weighted by $\alpha = 1/3$.

This normalization prevents dominance of one signal over the other and stabilizes feature aggregation across columns. The actor and critic lateral connections do not follow Equation (1), as there is no trainable matrix. Their weight matrices, $W_n^{*(2)}$ for the actors, and $W_V^{*(2)}$ for the critic, are the result of the concatenation of the learnable parameters from the second column, $W_n^{(2)}$ and $W_V^{(2)}$, respectively, with the $\text{FC}_n^{(1)}$ and $\text{FC}_V^{(1)}$ weight matrices, i.e., $W_n^{(1)}$ and $W_V^{(1)}$, respectively. This layout allows the initialization of the second-column output with the policies and state-value function from the first column. These initial values provide a crucial guide to the agent in its learning process. We applied the softmax function to translate the FC outputs' activations into the probability that defines policy $\pi_n^{(k)}$, where $n \in \{1, 2, ..., 6\}$.

Table 1 presents the parameter breakdown per layer, including the lateral connections. For the CNN blocks, the expression corresponds to (number of filters × depth × width × height) + bias. For the FC blocks, it is (input size × output size) + bias. The LSTM has four gates, and their parameters are multiplied by two, as the number of hidden and cell states is the same. Although the first-column parameters are frozen during the second-column training, we also provided them to demonstrate the capacity difference between columns.

Table 1. PNN trainable parameters per column and layer. Each row corresponds to one blue block from Figure 2. For the CNN layers, the information is given as (number of filters × depth × width × height) + bias, whereas it follows (input size × output size) + bias in the FC blocks. The LSTM has four gates.

	First Column (Teacher)	**Second Column (Student)**
$\text{FC}_V^{(2)}$	$(128 \times 1) + 1$	$(144 \times 1) + 1$
$\text{FC}_n^{(2)}$	$6 \times [(128 \times 7) + 7]$	$6 \times [(144 \times 7) + 7]$
$\text{LSTM}^{(2)}$	$2 \times (512 \times 128) + 2 \times 512$	$2 \times (64 \times 16) + 2 \times 64$
$\text{LC}_{LSTM-LSTM}$	-	16×128
$\text{LC}_{FC-LSTM}$	-	16×128
$\text{FC}^{(2)}$	$(128 \times 1152) + 128$	$(16 \times 288) + 16$
LC_{CNN-FC}	-	16×1152
$\text{CNN}_2^{(2)}$	$(32 \times 16 \times 5 \times 5) + 32$	$(8 \times 8 \times 5 \times 5) + 8$
$\text{LC}_{CNN-CNN}$	-	$(8 \times 16 \times 6 \times 6)$
$\text{CNN}_1^{(2)}$	$(16 \times 3 \times 8 \times 8) + 16$	$(8 \times 3 \times 8 \times 8) + 8$
Total	**301,147**	**43,323**

We chose to analyze this architecture thoroughly because it was the one selected in [7] to perform their main experiments due to its results. However, we designed and trained one more PNN architecture to carry out the forgetting analysis. This architecture is based on the discarded proposal presented in [7], and it consists of removing the LSTM layer. With this change, we decided to skip also the lateral connection that goes between FC layers, keeping only the ones between the CNN layers and the CNN with the FC. We believe that it is interesting to evaluate if the agents' forgetting persists even when the core of the architecture is the same, but the layers and the lateral connections change.

4. Materials, Problem Definition, and Baseline Model (BM)

This section first introduces the task solved and then the Baseline Model (BM) used as the first-column agent.

4.1. Materials and Problem Definition

The task the agent should learn corresponds to the first part of an industrial pick and place operation, namely, the approach to a target. The virtual setup for our experiments consists of a 6 degrees of freedom (DoF) IRB120 robotic arm from ABB (https://search.abb.com/library/Download.aspx?DocumentID=9AKK107045A1509, accessed on 18 December 2025) an externally mounted monocular camera with a fixed position during all tests, and a red 3 cm cube target that the robot must reach. The robot is equipped with a two-finger gripper that is not controlled since we focus only on the approaching stage. As stated in [7], we argue that even though it seems an easy task, the complexity lies in the fact that the agent must learn to infer the complete state information from a high-dimensional state-space, which is the RGB image. Moreover, the state-space perceived by the agent is a 2-dimensional representation, while the robot movement and target take place in a 3-dimensional space.

The virtualization was performed in MuJoCo [56]. As mentioned earlier, the model input is the 64 × 64 RGB environment observation. The images were captured using Matplotlib v3.10.8 [57] to speed up rendering and avoid a strong dependency on the visual characteristics of the scenario. As shown in [16], using a realistic virtual environment and high-quality images does not always lead to better results or a more efficient sim-to-real transfer.

The MDP is modeled as an episodic setup where the agent has up to 50 steps to reach the goal. The episode finishes either if the maximum number of steps per episode is achieved or if the gripper is closer than 5 cm to the target. This rewarding distance increases to 10 cm during the post-training model evaluation. The actions command the joints' velocity as in [7]. For more details on the choices made in the MDP design, refer to Appendix A Table A1, which is a summarized description of the MDP configuration adopted in this work. A detailed justification of the state space, action set, and reward function—including comparative analyses of alternative action discretizations and reward formulations—is presented in our previous work [33]. At the beginning of each episode, the initial robot pose is sampled from a uniform distribution. On the other hand, the target position is sampled from two uniform distributions, one for each x and y coordinate with the workspace area being a rectangle of 60 × 20 cm^2, same width as [7], but 20 cm larger. The z-coordinate is kept constant at ground level.

4.2. Baseline Model (BM)

We refer to the agent that has already been trained for the first column of the PNN architecture as BM. Its training process was carried out using the A3C [55] algorithm for 70 M steps, achieving the steady-state regime in approximately 40 M steps. For more details about the BM training hyperparameters, please refer to Appendix A Table A2 and [33]. Figure 3 exhibits an example of the 64 × 64 pixel RGB model input (i.e., the environment observation).

Figure 3. Sample of the BM environment at the episode's beginning. This 64 × 64 pixel RGB image is the model input. The initial joints and target positions are sampled from two uniform distributions.

Regarding the environment features, we focus on the background, divided into the ground and sky, the robot that can be split into its joints, the gripper and its different parts, and the target. Specifically, we study variations in the color of these elements and how they affect the agent's performance compared with the BM outcomes. Table 2 presents the environment features in which the BM agent was trained and evaluated. These characteristics are the ones that were used as the reference during the adversarial environment experiments.

Table 2. BM environment features. The RGB code is provided to supplement the color name.

Background	Robot	Gripper	Target
Green (ground) (0.19, 0.30, 0.23)	Orange (1.0, 0.5, 0.0)	Yellow (connector) (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)
Gray (sky) (0.67, 0.71, 0.75)	Black (joint 6) (0.0, 0.0, 0.0)	White (fingers) (1.0, 1.0, 1.0)	

5. Method

The general overview of the proposed method is depicted in Figure 4. First, we designed a set of experiments to determine which scenarios lead to a decrease in agent performance. The differences that these new scenarios present with respect to the baseline are changes in the colors of the background, the robot, and the target. Once we chose three environments based on the agent's perceived complexity (i.e., easy, intermediate, and complex), we trained and evaluated three PNNs to assess the knowledge transfer between the baseline agent and the one learning in the new environment. Finally, we selected the best model of each PNN-like agent and evaluated its forgetting with respect to the baseline.

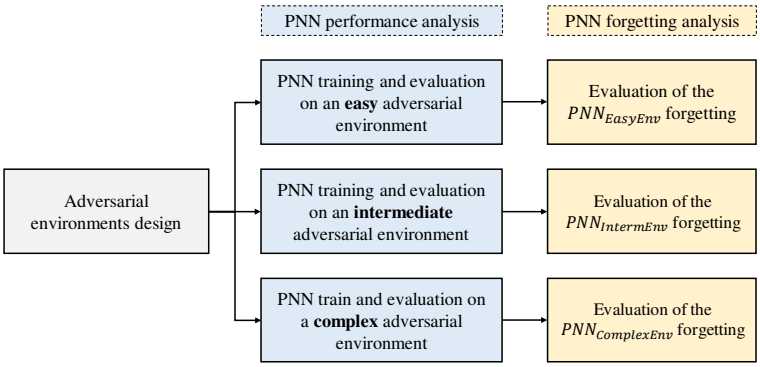

Figure 4. Diagram of the methodology followed.

In this section, we begin with a detailed explanation of the adversarial environment's design and evaluation (Section 5.1), and continue with the definition of the training characteristics and evaluation benchmark of each PNN agent and its forgetting (Section 5.2).

5.1. Adversarial Environments

The real setup built in [7] is almost a perfect replica of the virtual scenario. We argue that a thorough analysis of the PNNs' capability concerning their state representation is needed to plan the correct deployment in a real environment, as it will certainly present visual discrepancies.

To study the limitations of PNN-like agents and evaluate their robustness and forgetting, we designed a set of virtual adversarial environments. As aforementioned, we understand *adversarial environments* as static environments designed to maximize or cause a performance degradation in the agent. We simulate the deployment of synthetic experience into a second modified virtual environment to control the mismatches between both settings, and better identify and analyze the drivers that lead to learning inefficiencies. We consider adversarial environments those that result in less than 90% accuracy on the BM post-training evaluation. We use the outcomes obtained in this study to establish an evaluation benchmark against which we can compare the results obtained after training PNN-like agents.

During the design of the adversarial environments, we applied DR to different colors of the virtual scenario, as we have already studied the effect of the camera pose [33]. Based on the scenario elements, we have arranged the subsequent categories: (1) the background, which can be divided into ground and sky, (2) the robot, which can be split per joint, (3) the gripper, which is separated into the fingers and the connector (i.e., the union between the fingers and the robot's last joint), and (4) the target.

The procedure to define the environment characteristics of the second-column agent is depicted in Figure 5. We started by evaluating the BM under a set of scenarios, changing only one of its features. We selected the color variation deterministically. Appendix B Table A4 summarizes the simple changes performed to each environment feature. After analyzing the results, we designed an additional set of evaluation tests in which we combined several changes per environment (Table A5). The final adversarial environments are chosen according to the performance decay results, which correlate with the difficulty perceived by the agent, including easy, intermediate, and complex.

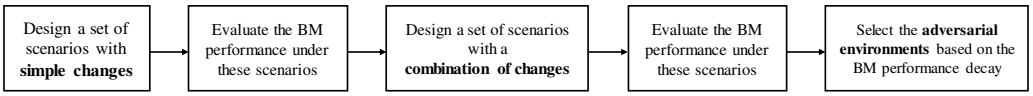

Figure 5. Procedure followed for the selection of the adversarial environments.

Throughout all the steps, the performance of the agents is mainly measured by their accuracy, which is defined as the percentage of episodes in which the agent reached the goal (3). The evaluations are carried out across 1000 episodes with random initial target and robot poses. We complement the assessment by measuring the mean and standard deviation of the return, the mean and standard deviation of the episode length, and the mean, standard deviation, and maximum failure distance.

$$\text{Accuracy (\%)} = \frac{\sum_{i=1}^{N} \mathbb{I}(\text{dist}_i \leq 10\,\text{cm})}{N} \times 100 \tag{3}$$

where N is the total number of evaluated episodes, dist_i is the relative distance between the gripper and the target in the ith episode, and $\mathbb{I}$ is the indicator function.

Additionally to these metrics offering a quantitative perspective, we use saliency maps [58] in an attempt to explain the rationale behind the agent's decisions [59] and identify focal image areas. This provides a qualitative perspective that complements the quantitative analysis. Saliency methods are typically used in image-based problems to highlight features based on activation maps of convolutional layers. In this paper, we opt for the gradient saliency map due to its straightforward implementation and because it passes the sanity checks proposed in [60]. These sanity checks are particularly relevant in our case, as they verify that the saliency maps are sensitive to model parameters and learned representations rather than merely reflecting input structures or random patterns. This ensures that the highlighted regions genuinely correspond to the agent's learned decision-

making process, which is essential for a reliable qualitative analysis of its perception mechanisms. Equation (4) describes the saliency calculation. The score that corresponds to the output of the last CNN layer is aggregated, and then it is partially derived with respect to each pixel, identifying the pixels with higher gradient values per channel. Finally, the outcomes are normalized to reduce the variance. This normalization occurs prior to visualization to ensure comparability across images and environments. Consequently, the absolute gradient magnitudes are not interpreted directly; instead, the analysis focuses on the relative spatial distribution of high-response regions. This normalization enables consistent qualitative comparison of attention patterns, even when underlying gradient scales differ between scenarios. Therefore, high saliency in image regions indicates that changes in those areas will have a greater impact on the output.

$$M_{i,j} = \max_{c \in \{R,G,B\}} \left| \frac{\partial S(I)}{\partial I_{i,j,c}} \right| \tag{4}$$

$$S(I) = \sum_{u} f_u(I)$$

where $I_{i,j,c}$ denotes the pixel intensity at spatial position (i, j) and color channel $c \in \{R, G, B\}$. The term $S(I)$ corresponds to the aggregated activation of the last convolutional layer, computed as the sum of its feature map activations $f_u(I)$, where u indexes the feature maps. The saliency value $M_{i,j}$ is obtained by taking the maximum absolute gradient across the three color channels, indicating how sensitive the network's output is to perturbations at that pixel location. High values of $M_{i,j}$ therefore highlight image regions that have a stronger influence on the agent's learned representation.

5.2. PNN Agent Training and Evaluation Methodology

5.2.1. Training Procedure

The second-column agents have been trained for 30 M steps using the A3C algorithm. For all intermediate layers and lateral connections, weight matrices are randomly set using orthogonal initialization. In contrast, lateral connections and weights of the second-column output are initialized so that in the first episode, the agent follows the policy set by the first column (Section 3).

Every 50 k steps, we conducted an interim evaluation for the instantiated models, testing them across 40 episodes with fixed initial configurations (robot and target poses). The models share its weights with the global shared network asynchronously. Network updates continue until the next interim evaluation. Throughout this process, the optimizer used is Root Mean Square Propagation (RMSprop) [61]. Appendix A Table A2 gathers the training hyperparameters of the teacher and student networks. Note that during training and interim evaluation, success is determined by a relative distance between the gripper and the target of 5 cm. However, in the post-training evaluation, we increase it to 10 cm, as in [7], arguing that training for a more challenging objective will yield better post-training evaluation statistics.

5.2.2. Post-Training Evaluation Procedure

The *post-training* evaluation assesses the accuracy of the trained PNN-like agent across 1000 episodes using a different seed than the one for training and random initial poses. This ensures a more reliable comparison between models without trusting solely the average return obtained during training. To evaluate the goodness of a model, we consider both the agent's performance and the time needed to achieve it.

Evaluation metrics for the 1000 episodes are the same as those introduced in Section 5.1. For a detailed overview of the post-training hyperparameters, refer to Appendix A, Table A3.

5.2.3. Forgetting Evaluation Procedure

To evaluate the forgetting the student agent suffers with respect to the teacher's task, we assess its performance in the first-column environment with a post-training evaluation. Our goal with this experiment is to determine how much the student can remember about the teacher's task. The metrics employed during the forgetting assessment are the same as those introduced in Section 5.1. These experiments are conducted using two different PNN architectures, whose designs are made according to the two alternatives described in [7] (Section 3).

6. Results and Discussion

In this section, we present (1) the results of the BM evaluation under the three adversarial environments chosen, (2) the outcomes of PNN-like agents in the three adversarial environment conditions, and (3) the forgetting that these three agents suffer when they face the first-column environment again.

The experiments were conducted on a PC running Ubuntu 20.04 equipped with an Intel Core i9-10900KF processor at 3.70 GHz, 64 GB of DDR4 RAM, an NVIDIA GeForce RTX 2080 Ti GPU, and a 2 TB M.2 SSD. The algorithms were programmed in Python v.3.8 using PyTorch v.1.13.1 [62].

6.1. Results in the Adversarial Environments

Figure 6 presents the BM agent saliency map for two states of the same episode. The left group (a) corresponds to an episode start, whereas (b) is an intermediate step. Based on the gradient values, the saliency map shows the agent's attention is on the target and the robot base at the beginning (a), while in (b) it is relocated to a broader area that wraps the gripper, the last joint surroundings, the target, and the robot's base. The agent is able to focus on the environment elements whose relative pose leads to higher rewards and a successful episode. However, these attention zones are not a perfect semantic segmentation of the environment, as there are regions of the background where the agent also *looks* to decide how to act efficiently.

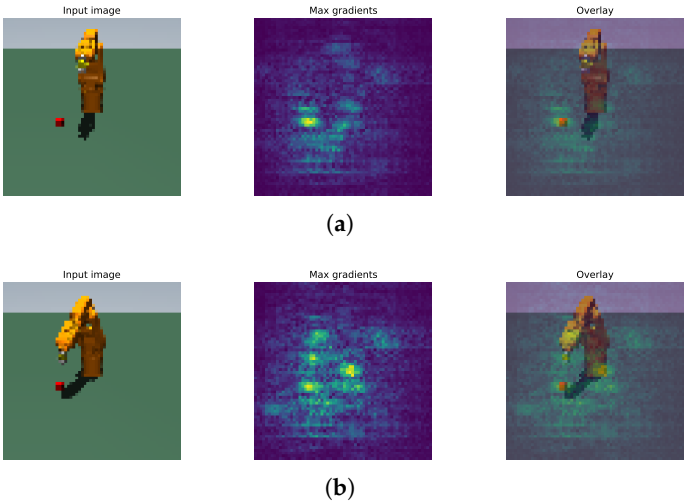

Figure 6. Environment observation and saliency map obtained in the episode beginning (**a**) and in an intermediate step (**b**) for the BM agent in its original setup. In both moments, the gradients show that the agent's attention is mainly on the target and the robot pose. The episode finished with a success.

We evaluated the BM under the scenarios gathered in Appendix B Tables A4 and A5 to determine which of them are adversarial environments (i.e., those that lead to an accuracy lower than 90% in the post-training evaluation) and, thus, the best options to set controlled but challenging conditions. Appendix C Table A6 presents the evaluation results of the environments qualified as adversarial. The BM agent performance decreases from 30% to 40% when one or both background elements are modified. The reason may be the aforementioned partial attention of the agent to background regions. Something similar happens to the combinations where the last joints or the end-effector are changed. In these last cases, since the number of pixels is narrower, the accuracy reduction is only between 20% and 30%.

In those scenarios where the robotic arm has a different color, the decay is even higher, with a drop up to 45%. For the target's color variation, it is interesting that the reduction is between 20% and 25% for black and white. In comparison, blue results in a 50.3% deterioration, the highest of all the evaluations. For these two changes, the saliency map inspection should be complemented with an analysis of the RGB channels. Regarding the robot, most of its pixels are orange (1.0, 0.5, 0.0). Therefore, the effect of the blue channel on the CNN activation maps is negligible. The lowest performance for these cases is achieved when the robot is entirely blue. The target results can also be examined under this hypothesis. With the red target (1.0, 0.0, 0.0), none of the green and blue channels are likely to influence the CNN activation maps, so if image has changes in that area (Figure 6), the agent may loose part of its perception capacity. We argue that this outcome is enhanced when the target is blue due to the poor contrast between the cube, the green ground, and the black robot shadow.

Table A6 summarizes the final three adversarial environments selected: (1) an easy scenario, with the target color changed to white (78.8% accuracy); (2) an intermediate setup with both the ground and sky colors changed to white (This scenario is especially interesting because it can be regarded as if a mask, i.e., the elimination of all the background keeping just the main scenario elements) (69.8% accuracy); and (3) the hardest adversarial environment which is the one with the blue target (49.7% accuracy). Table 3 gathers the environments' features, and Figure 7 displays the BM saliency map achieved in these scenarios.

To ease comparison, Figures 6 and 7 show the same initial poses. In Figure 6, the attention is predominantly concentrated in the central area of the scene, with salient regions clearly focused on the target position at the beginning of the episode and on the robot joints and the target during intermediate steps. The gradients in Figure 7 suggest that in the adversarial environments, the attention is spread over a wider area. On the one hand, the saliency map (a) shows the attention mainly around the robot and its shadow and, to a lesser extent, in part of the background. On the other hand, (b) and (c) display scenes where the saliency maps cover apparently useless areas, especially (b), without noticing either the target or the robot. This shift in attention distribution suggests a degradation in the learned state representation when significant visual discrepancies are introduced. With these experiments and remaining under the hypothesis that the state representation is embedded in the convolutional layers, we demonstrate the relevance of the first two CNN blocks in the PNN architecture.

Regarding the PNN's strengths and weaknesses as a DRL sim-to-real method, we argue that, *a priori*, its perception (i.e., the CNN layers) cannot efficiently handle changes in the setup. For an efficient transfer learning process, the visual appearance of the virtual environment should be similar to the real scenario, as [7] exhibits in their work, or adequately randomized in the virtual training phase. Concerning Appendix C Table A6, we argue that the agent will need a large amount of real experience to learn the new features. However, we also hold that, if the virtual training phase is improved by adding enough

diversity, the probabilities of success in the transfer learning process will increase. We leave the verification of this hypothesis for future work.

Table 3. Adversarial environments selected. The RGB code is provided to supplement the color name. Features highlighted in bold are the ones that change with respect to the BM.

	Background		Robot	Gripper			Accuracy
	Ground	Sky	Link n	Fingers	Connection Part	Target	(%)
Env_{Easy}	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	**White (1.0, 1.0, 1.0)**	78.8
$\text{Env}_{\text{Interm}}$	**White (1.0, 1.0, 1.0)**	**White (1.0, 1.0, 1.0)**	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)	69.6
$\text{Env}_{\text{Complex}}$	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	**Blue (0.0, 0.0, 1.0)**	49.7

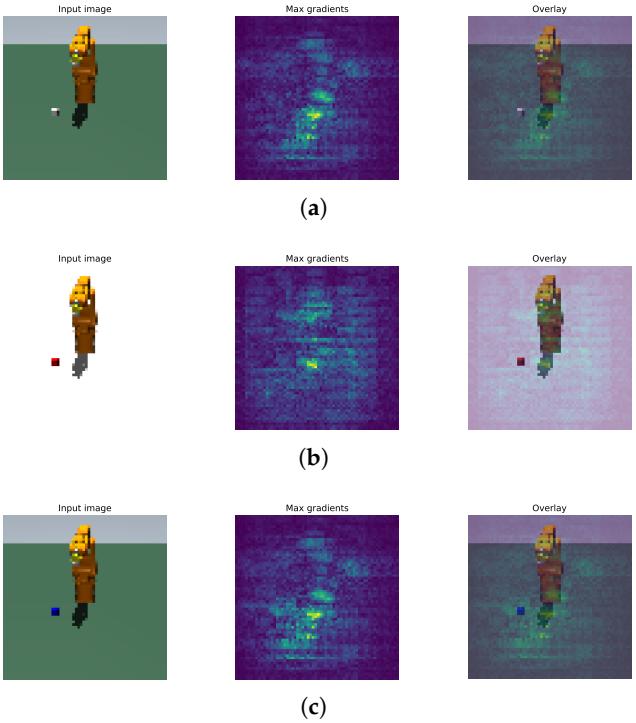

Figure 7. Environment observations and saliency maps obtained in an intermediate step for the easy (**a**), the intermediate (**b**), and the complex (**c**) adversarial environments. The agent's attention is not completely focused on either the robot or the target. It is mainly in the robot's shadow. The three situations ended in failure.

6.2. PNN Agent Training, Evaluation and Forgetting Results

Figure 8 presents the training curves for the BM (orange) and the three student agents: the $PNN_{EasyEnv}$ (blue), where the target is white, the $PNN_{IntermEnv}$ (green), where the background is white, and the $PNN_{ComplexEnv}$ (purple), where the target is blue. To ease the comparison, we only present the first 30 M steps of the BM training. The three agents surpass the BM from the beginning, thanks to the initialization with the first-column policy. The PNNEasyEnv trendline shows minimal growth, reaching an average return of nearly 50. This rapid achievement of the steady state, almost three times faster than the BM, can be attributed to the knowledge transfer from the BM and the reduced complexity of the environment, that resulted in only a 20% performance drop compared to the teacher. In contrast, the PNNIntermEnv agent reaches the steady state with the same average return as the PNNEasyEnv, but takes around 3 M steps longer. Lastly, the PNNComplexEnv initially outperforms the BM, but after 7 M steps, it plateaus at an average return of just 15, indicating difficulties in handling this more visually distinct and challenging environment. The agent fails to surpass the BM's performance later, implying that additional architectural modifications or deeper feature extraction may be required to manage such complex scenarios effectively.

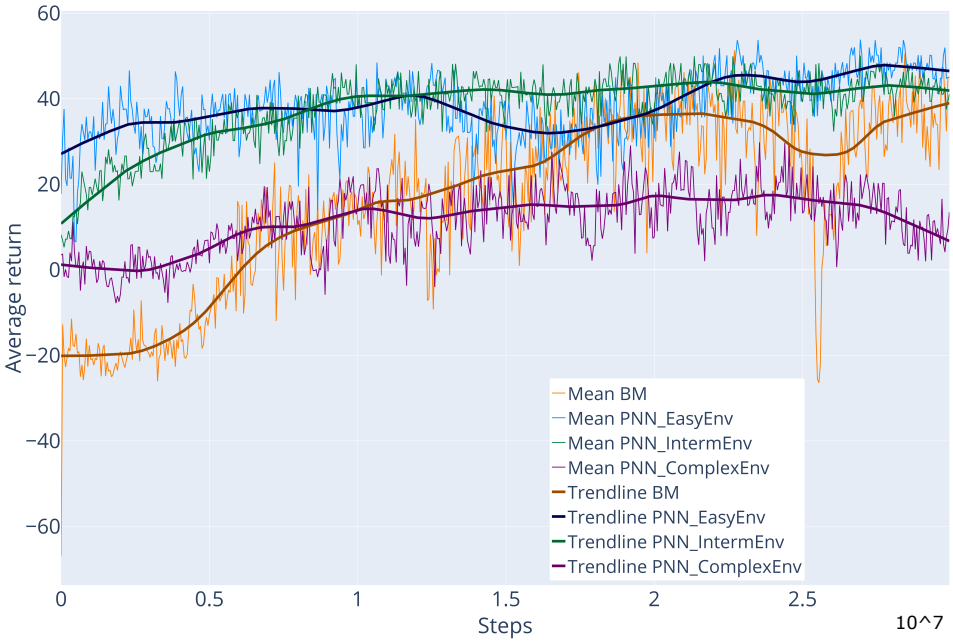

Figure 8. Average returns in the interim evaluation for $PNN_{EasyEnv}$ (blue), the $PNN_{IntermEnv}$ (green), the $PNN_{ComplexEnv}$ (purple), and the BM (orange) agents over 30 M steps (note the 10^7 scaling factor on the right-hand corner of the figure). While the $PNN_{EasyEnv}$ and $PNN_{IntermEnv}$ agents present better results than the BM, the $PNN_{ComplexEnv}$ cannot achieve the same performance.

Table 4 summarizes the main findings from the post-training evaluations for each agent. The PNNEasyEnv agent reaches 99% accuracy at 1.6 M steps, requiring only a few additional steps to improve from the initial 78.8%. This demonstrates that the agent quickly adapts to the new environment using the transferred knowledge. Similarly, the PNNIntermEnv agent achieves 94.7% accuracy after 5.65 M steps. While this agent requires more training steps than the PNNEasyEnv due to the increased complexity of the environment, it still significantly outperforms the BM in this setup, needing minimal experience. Since the differences are primarily based on visual features, the learning challenges appear to stem from the model's state representation layers, supporting the assumption made when analyzing the saliency map results. In contrast, the PNNComplexEnv agent struggles more with its challenging environment, achieving a maximum accuracy of 85.6%

at 10.65 M steps, after which the performance plateaus. This is likely due to limitations in the state representation layers, making it hard to obtain a reliable state approximation when visual differences cause a significant drop in the teacher's performance.

Table 4. $PNN_{EasyEnv}$ agent, $PNN_{InterimEnv}$ agent, and $PNN_{ComplexEnv}$ agent results in the post-training evaluation. The evaluation interval is not exactly 50 k because a global step counter is shared among all the instantiated agents.

Agent	Evaluation Step	Mean Return	Std. Dev. Return	Mean Episode Length	Std. Dev. Episode Length	Mean Failure Distance (cm)	Std. Dev. Failure Distance (cm)	Max Failure Distance (cm)	Accuracy (%)
$PNN_{EasyEnv}$	0	39.20	4.78	32.85	1.30	0.20	0.03	0.26	78.8
	1,600,031	54.32	9.22	27.60	5.46	0.27	0.10	0.41	99.0
$PNN_{InterimEnv}$	0	32.58	5.12	34.34	1.39	0.18	0.02	0.22	69.8
	5,650,232	50.85	18.08	29.49	7.32	0.17	0.04	0.30	94.7
$PNN_{ComplexEnv}$	0	17.43	6.18	39.67	1.63	0.20	0.01	0.22	49.7
	10,650,675	44.59	25.92	33.09	9.80	0.21	0.11	0.50	85.6

Finally, Table 5 shows the forgetting assessment results for the three PNN agents evaluated with both architectures, with and without LSTM. This evaluation measures how well the agents retain knowledge from the BM environment after learning in a different environment. Regarding the architectures with the LSTM, the outcomes indicate that both the PNNEasyEnv and PNNIntermEnv agents experience partial forgetting when tested in the original BM environment. The PNNEasyEnv agent retains 83.7% accuracy, while the PNNIntermEnv agent shows a similar pattern with an accuracy of 80.9%. This suggests that although they learn the new tasks, some of the previously acquired knowledge from the BM environment is lost, likely due to *overfitting* on the second task. Interestingly, the $PNN_{ComplexEnv}$ agent retains the highest performance from the original BM environment, achieving 99.8% accuracy when tested in the red target scenario. This may imply that, despite struggling with the complex environment, the agent can still retain and leverage knowledge from the simpler BM environment, likely because the lateral connections have a stronger influence than the weights learned in the second column. Analyzing the forgetting results of the agents trained without the LSTM, we observe that the PNNEasyEnv agent shows almost no forgetting with respect to the teacher's task. We suggest that this may occur because the student agent has not fully mastered its task. It achieves only 32.7% accuracy, allowing the lateral connections to still exert a significant influence on the output. A similar effect is seen in the PNNComplexEnv agent, which, in fact, performs around 12% better on the teacher's task than on its own. Regarding the $PNN_{IntermEnv}$, the agent's performance is so low that the forgetting results suggest the lateral connections have not had sufficient time to properly update their values since initialization.

Table 5. Forgetting assessment for the $PNN_{EasyEnv}$ agent, the $PNN_{InterimEnv}$ agent and the $PNN_{ComplexEnv}$ agent. The students are evaluated on the teacher's task.

Agent	Architecture	Training Environment	Evaluated Environment	Mean Return	Std. Dev. Return	Mean Episode Length	Std. Dev. Episode Length	Mean Failure Distance (cm)	Std. Dev. Failure Distance	Max Failure Distance (cm)	Accuracy (%)
$PNN_{EasyEnv}$	LSTM	White target	White target	54.32	9.22	27.60	5.46	0.27	0.10	0.41	99.0
		White target	Red target	44.19	26.57	34.84	9.46	0.18	0.10	0.55	83.7
	FC	White target	White target	2.13	38.96	43.05	10.56	0.22	0.08	0.46	32.7
		White target	Red target	1.37	38.24	43.24	10.36	0.22	0.07	0.41	31.6
$PNN_{InterimEnv}$	LSTM	White ground & sky	White ground & sky	50.85	18.08	29.49	7.32	0.17	0.04	0.30	94.7
		White ground & sky	Green ground & grey sky	40.82	29.65	32.24	9.62	0.21	0.05	0.35	80.9
	FC	White ground & sky	White ground & sky	−4.85	37.75	44.30	9.69	0.23	0.08	0.42	27.5
		White ground & sky	Green ground & gray sky	−39.98	19.96	49.80	1.39	0.32	0.09	0.60	2.4
$PNN_{ComplexEnv}$	LSTM	Blue target	Blue target	44.59	25.92	33.09	9.80	0.21	0.11	0.50	85.6
		Blue target	Red target	54.88	5.65	26.54	3.95	0.21	0.05	0.27	99,8
	FC	Blue target	Blue target	21.26	39.04	37.53	11.63	0.22	0.06	0.38	56.7
		Blue target	Red target	29.69	37.18	35.23	10.85	0.23	0.07	0.40	68.3

7. Conclusions and Future Work

This paper addresses the limitations of PNNs as a transfer learning framework under a sim-to-sim approach, with all the environment features properly characterized and shows the replicability of [7] on another setup. Three adversarial scenarios of varying difficulty that represent different setups are first designed and evaluated on a Baseline Model (BM). These virtual scenarios with controlled discrepancies with respect to the teacher's setup allow to test the efficiency of the PNN architecture without sources of noise or uncertainty. Finally, the forgetting these PNN agents suffer is assessed with respect to the teacher's task under two different PNN architectures.

The results from the evaluation of the adversarial environments under the BM agent show that, since the input is an RGB image, there is a relevant dependency on the representation learning blocks, mainly the convolutional layers. Apparently, when the representation of the problem learned by the teacher column does not perform well under an adversarial scenario, it seems difficult for the shallower convolutional layers of the student column to learn a good representation of the problem. Small visual mismatches between the trained and the evaluated scenarios can lead to dramatic accuracy decays of 50.3% in the worst-case environment.

The outcomes obtained after training three PNN agents in the selected adversarial scenarios confirm the hypothesis that the more significant the visual differences between setups, the more complicated, inefficient, and poor the learning of the student column agent is. According to the sim-to-sim analyses, the current PNN proposal might not be enough to address the sim-to-real problem efficiently if the virtual and the real scenarios present appearance discrepancies. Since the agent relies exclusively on RGB images, the learned state representation depends entirely on convolutional filters trained on a specific color distribution. Significant shifts in color statistics across environments may alter the activation patterns of these filters, degrading the extracted features. In such cases, the fine-tuning of the student column may not be sufficient to compensate for the mismatch, potentially limiting the effectiveness of knowledge transfer within the PNN framework. However, future work is required to deploy the PNN in a real sim-to-real setting in order to fully validate this hypothesis.

The last finding is related to the quantification of the PNN agents' forgetting. Although catastrophic forgetting is avoided by design, if the student column agent achieves a high accuracy in the new setup, it cannot completely remember the task the first column mastered. This partial amnesia might be caused by *overfitting* in the second environment. Besides, the relative importance (i.e., the weight value) given to the lateral connections with respect to the student's layers can also play a key role. If the student agent neglects the lateral connections and learns to master the new task on its own, as happens in the easy and intermediate scenarios with the LSTM, its performance on the previous setup degrades. In contrast, if the agent relies heavily on the lateral connections, as might happen in the complex environment or in the models without LSTM, it is able to address the first-column task better. We prove that these hypotheses hold independently of the layers and lateral connections of the PNN. Additionally, we also showed that, as in [7], the agent trained with the PNN architecture without the LSTM layer has a worse performance than the one with the LSTM for all the adversarial scenarios.

In view of these conclusions, it is worth highlighting the following lines of future work in relation to the use of PNNs as a mechanism for transfer learning in sim-to-real problems:

- Regarding the relevance of the visual layers, address visual mismatches between both scenes by, for instance, preprocessing the real image to make it resemble the virtual setup more (e.g., to mitigate reflections and other visual artifacts) or applying DR or DA during the training phase of the teacher agent.

- Research on the different possibilities of joining the teacher and the student columns through the lateral connections, apart from the one proposed in this paper.
- Study if there is any mechanism, like the lateral connections drop-off, that might help the agent to balance old and new knowledge correctly and prevent partial forgetting.
- Extend the analysis to more complex visual discrepancies, such as variations in texture and lighting conditions. These multimodal perturbations would provide a deeper understanding of the generalization capabilities of PNNs under other domain shifts.
- Perform the sim-to-real experiments with the PNN architecture designed and tested in this paper, but with the student trained in the real setting.

Author Contributions: L.G.-L.: Conceptualization and design of this study, methodology, formal analysis and investigation, software, data curation, writing—original draft preparation, writing—review and editing. J.B.: conceptualization and design of this study, methodology, formal analysis and investigation, supervision, writing—review and editing. Á.J.L.-L.: conceptualization and design of this study, methodology, formal analysis and investigation, supervision, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: The authors would like to thank Andrei A. Rusu and his co-authors for their detailed response to the questions posed regarding the original implementation of the PNNs.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. PNN Hyperparameters

Table A1. MDP definition. The action set is configured as a discrete collection of actions. A negative or a positive reward per each step will be given to the agent based on the objective fulfillment [33].

Action Set [a]	Reward Distribution [b]	Success Distance
0 ±MPI ±MPI/10 ±MPI/100	70 if goal reached $-(2 \times \text{dist})^2$ if goal not reached	5 cm in training 10 cm in post-training evaluation

[a] MPI stands for Maximum Position Increment, which is calculated based on the joint range. [b] dist is the relative distance between the gripper and the target.

Table A2. PNN-like agent training hyperparameters for the student network.

Hyperparameter	
Seed	123
Training steps	30 million
Episode length	50 steps or target reached
Success distance	5 cm
Evaluation interval	50,000 steps
Evaluation episodes	40 episodes
Discount factor (γ)	0.99
Learning rate	1×10^{-4}
RMSprop decay	0.99
Entropy weight	0.01

The training hyperparameters for the BM (i.e., the teacher), which was also trained using an A3C, were the same except for the training steps, which were 70 million.

Table A3. PNN-like agent post-training evaluation hyperparameters.

Hyperparameter	
Seed	803
Success distance	10 cm
Episodes evaluated	1000 episodes

Appendix B. Environments Evaluated

Table A4. Environments tested with simple changes. The RGB code is provided to supplement the color name.

		Background		Gripper			
Environment	**Changes**	**Ground**	**Sky**	**Robot Link *n***	**Fingers**	**Connection Part**	**Target**
BM	–	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)
Env_{SC1}	Ground	Black (0.0, 0.0, 0.0)	–	–	–	–	–
Env_{SC2}	Ground	White (1.0, 1.0, 1.0)	–	–	–	–	–
Env_{SC3}	Sky	–	Black (0.0, 0.0, 0.0)	–	–	–	–
Env_{SC4}	Sky	–	White (1.0, 1.0, 1.0)	–	–	–	–
Env_{SC5}	Robot	–	–	$n \in [1,5]$ Black (0.0, 0.0, 0.0)	–	–	–
Env_{SC6}	Robot	–	–	$n \in [1,5]$ White (1.0, 1.0, 1.0)	–	–	–
Env_{SC7}	Robot	–	–	$n \in [1,6]$ Black (0.0, 0.0, 0.0)	–	–	–
Env_{SC8}	Robot	–	–	$n \in [1,6]$ White (1.0, 1.0, 1.0)	–	–	–
Env_{SC9}	Robot	–	–	$n \in [1,6]$ Red (1.0, 0.0, 0.0)	–	–	–
Env_{SC10}	Robot	–	–	$n \in [1,6]$ Blue (0.0, 0.0, 1.0)	–	–	–

Table A4. *Cont.*

Environment	Changes	Background		Robot Link n	Gripper		
		Ground	Sky		Fingers	Connection Part	Target
Env_{SC11}	Fingers changes	–	–	–	Black (0.0, 0.0, 0.0)	–	–
Env_{SC12}		–	–	–	Red (1.0, 0.0, 0.0)	–	–
Env_{SC13}	Target	–	–	–	–	–	Black (0.0, 0.0, 0.0)
Env_{SC14}		–	–	–	–	–	White (1.0, 1.0, 1.0)
Env_{SC15}		–	–	–	–	–	Blue (0.0, 0.0, 1.0)

Table A5. Environments tested with feature combination. The RGB code is provided to supplement the color name.

Environment	Changes	Background		Robot Link n	Gripper		
		Ground	Sky		Fingers	Connection Part	Target
BM	–	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)
Env_{MC1}	Gripper	–	–	–	Green (0.19, 0.30, 0.23)	Green (0.19, 0.30, 0.23)	–
Env_{MC2}		–	–	–	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC3}		–	–	–	White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–
Env_{MC4}	Background	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–	–	–	–
Env_{MC5}		White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–	–	–	–
Env_{MC6}		Blue (0.0, 0.0, 1.0)	Blue (0.0, 0.0, 1.0)	–	–	–	–
Env_{MC7}		Green (0.19, 0.30, 0.23)	Green (0.19, 0.30, 0.23)	–	–	–	–
Env_{MC8}		Gray (0.67, 0.71, 0.75)	Gray (0.67, 0.71, 0.75)	–	–	–	–
Env_{MC9}	Background & gripper	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC10}		Blue (0.0, 0.0, 1.0)	Blue (0.0, 0.0, 1.0)	–	Blue (0.0, 0.0, 1.0)	Blue (0.0, 0.0, 1.0)	–

Table A5. *Cont.*

Environment	Changes	Background		Robot Link n	Gripper		Target
		Ground	Sky		Fingers	Connection Part	
Env_{MC11}	Sky & gripper	–	Black (0.0, 0.0, 0.0)	–	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC12}		–	White (1.0, 1.0, 1.0)	–	White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–
Env_{MC13}	Robot	–	–	$n \in [3, 4]$ Black (0.0, 0.0, 0.0)	–	–	–
Env_{MC14}		–	–	$n \in [3, 4]$ White (1.0, 1.0, 1.0)	–	–	–
Env_{MC15}	Robot & Gripper	–	–	$n \in [4, 5]$ Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC16}		–	–	$n \in [4, 5]$ White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–

Appendix C. Adversarial Environments Results

Table A6. Outcomes from the adversarial environments evaluation. These scenarios are the ones that present an accuracy lower than 90%. The selected adversarial environments are highlighted in bold. Std. dev. stands for standard deviation.

Environment Change	Mean Return	Std. Dev. Return	Mean Episode Length	Std. Dev. Episode Length	Mean Failure Distance (cm)	Std. Dev. Failure Distance	Max. Failure Distance (cm)	Accuracy (%)
BM	55.65	0.79	25.73	0.64	0.00	0.00	0.00	**100.0**
White Ground	25.26	5.90	36.62	1.70	0.15	0.02	0.20	60.7
Black Sky	28.31	5.82	37.01	1.72	0.18	0.02	0.22	64.5
Black Robot	21.11	6.14	38.17	1.77	0.23	0.02	0.28	54.4
White Robot	19.59	5.24	38.91	1.52	0.23	0.02	0.27	54.0
Blue Robot	18.19	5.88	38.82	1.72	0.24	0.02	0.29	51.7
Black Target	35.21	4.76	34.95	1.57	0.18	0.02	0.22	72.9
White Target	39.20	4.78	32.85	1.30	0.20	0.03	0.26	**78.8**
Blue Target	17.43	6.18	39.67	1.63	0.20	0.01	0.22	**49.7**
Black Ground and Sky	37.76	4.87	32.82	1.71	0.16	0.02	0.19	76.2
White Ground and Sky	32.58	5.12	34.34	1.39	0.18	0.02	0.22	**69.8**
Blue Ground and Sky	27.54	5.74	35.82	1.63	0.27	0.04	0.34	65.1
Black Ground, Sky, Connection part and Fingers	37.18	5.63	33.15	1.71	0.16	0.03	0.27	75.8
Blue Ground, Sky, Connection part and Fingers	29.36	5.29	35.20	1.64	0.27	0.03	0.32	67.8

Table A6. *Cont.*

Environment Change	Mean Return	Std. Dev. Return	Mean Episode Length	Std. Dev. Episode Length	Mean Failure Distance (cm)	Std. Dev. Failure Distance	Max. Failure Distance (cm)	Accuracy (%)
Black Sky, Connection part and Fingers	24.80	5.04	38.08	1.39	0.19	0.01	0.22	60.0
Black Link 3 and Link 4	45.21	4.42	31.25	1.42	0.21	0.04	0.30	86.4
White Link 3 and Link 4	35.96	4.29	34.40	1.51	0.21	0.02	0.28	73.5
Black Link 4 and Link 5	38.17	3.97	33.87	1.53	0.19	0.02	0.25	75.6
Black Link 4, Link 5, Connection part and Fingers	39.34	4.59	33.16	1.37	0.20	0.04	0.27	78.2

References

1. François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M.G.; Pineau, J. An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.* **2018**, *11*, 219–354. [CrossRef]
2. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
3. Lecun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
4. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
5. Bengio, Y.; Courville, A.; Vincent, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828. [CrossRef]
6. Rusu, A.A.; Rabinowitz, N.C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; Hadsell, R. Progressive Neural Networks. *arXiv* **2016**, arXiv:1606.04671. [CrossRef]
7. Rusu, A.A.; Večerík, M.; Rothörl, T.; Heess, N.; Pascanu, R.; Hadsell, R. Sim-to-Real Robot Learning from Pixels with Progressive Nets. In Proceedings of the 1st Conference on Robot Learning (CoRL), Mountain View, CA, USA, 13–15 November 2017. [CrossRef]
8. Panzer, M.; Bender, B. Deep reinforcement learning in production systems: A systematic literature review. *Int. J. Prod. Res.* **2022**, *60*, 4316–4341. [CrossRef]
9. Rupprecht, T.; Wang, Y. A survey for deep reinforcement learning in markovian cyber–physical systems: Common problems and solutions. *Neural Netw.* **2022**, *153*, 13–36. [CrossRef]
10. Singh, B.; Kumar, R.; Singh, V.P. Reinforcement learning in robotic applications: A comprehensive survey. *Artif. Intell. Rev.* **2022**, *55*, 945–990. [CrossRef]
11. Zhao, W.; Queralta, J.P.; Westerlund, T. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020, Canberra, Australia, 1–4 December 2020; pp. 737–744. [CrossRef]
12. Qureshi, A.H.; Nakamura, Y.; Yoshikawa, Y.; Ishiguro, H. Intrinsically motivated reinforcement learning for human–robot interaction in the real-world. *Neural Netw.* **2018**, *107*, 23–33. [CrossRef]
13. Wang, M.; Deng, W. Deep Visual Domain Adaptation: A Survey. *Neurocomputing* **2018**, *312*, 135–153. [CrossRef]
14. Higgins, I.; Pal, A.; Rusu, A.; Matthey, L.; Burgess, C.; Pritzel, A.; Botvinick, M.; Blundell, C.; Lerchner, A. DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. In Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, Australia, 6–11 August 2017; Volume 70, pp. 1480–1490.
15. Jeong, R.; Aytar, Y.; Khosid, D.; Zhou, Y.; Kay, J.; Lampe, T.; Bousmalis, K.; Nori, F. Self-Supervised Sim-to-Real Adaptation for Visual Robotic Manipulation. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 2718–2724. [CrossRef]
16. Bousmalis, K.; Irpan, A.; Wohlhart, P.; Bai, Y.; Kelcey, M.; Kalakrishnan, M.; Downs, L.; Ibarz, J.; Pastor, P.; Konolige, K.; et al. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 4243–4250. [CrossRef]
17. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS), Lake Tahoe, NV, USA, 5–8 December 2013; Volume 27, pp. 2672–2680.

18. Zhang, J.; Tai, L.; Yun, P.; Xiong, Y.; Liu, M.; Boedecker, J.; Burgard, W. VR-Goggles for Robots: Real-to-Sim Domain Adaptation for Visual Control. *IEEE Robot. Autom. Lett.* **2019**, *4*, 1148–1155. [CrossRef]
19. Rao, K.; Harris, C.; Irpan, A.; Levine, S.; Ibarz, J.; Khansari, M. RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 11157–11166. [CrossRef]
20. Arndt, K.; Hazara, M.; Ghadirzadeh, A.; Kyrki, V. Meta Reinforcement Learning for Sim-to-Real Domain Adaptation. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 2725–2731. [CrossRef]
21. Wang, J.X.; Kurth-Nelson, Z.; Soyer, H.; Leibo, J.Z.; Tirumala, D.; Munos, R.; Blundell, C.; Kumaran, D.; Botvinick, M.M. Learning to reinforcement learn. *arXiv* **2017**, arXiv:1611.05763. [PubMed]
22. Ben-Iwhiwhu, E.; Dick, J.; Ketz, N.A.; Pilly, P.K.; Soltoggio, A. Context meta-reinforcement learning via neuromodulation. *Neural Netw.* **2022**, *152*, 70–79. [CrossRef] [PubMed]
23. Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 23–30. [CrossRef]
24. Vuong, Q.; Vikram, S.; Su, H.; Gao, S.; Christensen, H.I. How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies? *arXiv* **2019**, arXiv:1903.11774. [CrossRef]
25. Andrychowicz, M.; Baker, B.; Chociej, M.; Józefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **2020**, *39*, 3–20. [CrossRef]
26. Chen, X.; Hu, J.; Jin, C.; Li, L.; Wang, L. Understanding Domain Randomization for Sim-To-Real Transfer. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 25 April 2022.
27. Ramos, F.; Possas, R.C.; Fox, D. BayesSim: Adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv* **2019**, arXiv:1906.01728. [CrossRef]
28. Muratore, F.; Eilers, C.; Gienger, M.; Peters, J. Data-Efficient Domain Randomization with Bayesian Optimization. *IEEE Robot. Autom. Lett.* **2021**, *6*, 911–918. [CrossRef]
29. Mehta, B.; Mila, M.D.; Mila, F.G.; Mila, C.J.P.; Montréal, P.; Paull, C.L. Active Domain Randomization. In Proceedings of the 4th Conference on Robot Learning (CoRL), Virtual, 16–18 November 2020; pp. 1162–1176. . [CrossRef]
30. Prakash, A.; Boochoon, S.; Brophy, M.; Acuna, D.; Cameracci, E.; State, G.; Shapira, O.; Birchfield, S. Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 7249–7255. [CrossRef]
31. Yue, X.; Zhang, Y.; Zhao, S.; Sangiovanni-Vincentelli, A.; Keutzer, K.; Gong, B. Domain Randomization and Pyramid Consistency: Simulation-to-Real Generalization without Accessing Target Domain Data. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 2100–2110. [CrossRef]
32. Chebotar, Y.; Handa, A.; Makoviychuk, V.; MacKlin, M.; Issac, J.; Ratliff, N.; Fox, D. Closing the sim-to-real Loop: Adapting simulation randomization with real world experience. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8973–8979. [CrossRef]
33. Güitta-López, L.; Boal, J.; López-López, Á.J. Learning more with the same effort: How randomization improves the robustness of a robotic deep reinforcement learning agent. *Appl. Intell.* **2022**, *53*, 14903–14917. [CrossRef]
34. Hussein, A.; Gaber, M.M.; Elyan, E.; Jayne, C. Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 21. [CrossRef]
35. Nair, A.; McGrew, B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Overcoming Exploration in Reinforcement Learning with Demonstrations. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 6292–6299. [CrossRef]
36. Peng, X.B.; Abbeel, P.; Levine, S.; van de Panne, M. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *Assoc. Comput. Mach. Trans. Graph.* **2018**, *37*, 143. [CrossRef]
37. Ng, A.Y.; Russell, S. Algorithms for Inverse Reinforcement Learning. In Proceedings of the 17th International Conference on Machine Learning (ICML), Stanford, CA, USA, 29 June–2 July 2000; pp. 663–670.
38. Arora, S.; Doshi, P. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artif. Intell.* **2021**, *297*, 103500. [CrossRef]
39. Zhu, Y.; Wang, Z.; Merel, J.; Rusu, A.; Erez, T.; Cabi, S.; Tunyasuvunakool, S.; Kramár, J.; Hadsell, R.; de Freitas, N.; et al. Reinforcement and Imitation Learning for Diverse Visuomotor Skills. In Proceedings of the Robotics: Science and Systems, Pittsburgh, PA, USA, 26–30 June 2018. [CrossRef]
40. Ho, J.; Ermon, S. Generative Adversarial Imitation Learning. In Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS), Barcelona, Spain, 5–10 December 2016; pp. 4572–4580.

41. Tiboni, G.; Arndt, K.; Kyrki, V. DROPO: Sim-to-real transfer with offline domain randomization. *Robot. Auton. Syst.* **2023**, *166*, 104432. [CrossRef]
42. Rusu, A.A.; Colmenarejo, S.G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; Hadsell, R. Policy Distillation. In Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016. [CrossRef]
43. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. In Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS), Montreal, QC, Canada, 8–13 December 2014.
44. Traoré, R.; Caselles-Dupré, H.; Lesort, T.; Sun, T.; Díaz-Rodríguez, N.; Filliat, D. Continual Reinforcement Learning deployed in Real-life using Policy Distillation and Sim2Real Transfer. In Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019.
45. Kadokawa, Y.; Zhu, L.; Tsurumine, Y.; Matsubara, T. Cyclic policy distillation: Sample-efficient sim-to-real reinforcement learning with domain randomization. *Robot. Auton. Syst.* **2023**, *165*, 104425. [CrossRef]
46. Bellman, R. A Markovian Decision Process. *J. Math. Mech.* **1957**, *6*, 679–684. [CrossRef]
47. Parisi, G.I.; Kemker, R.; Part, J.L.; Kanan, C.; Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Netw.* **2019**, *113*, 54–71. [CrossRef]
48. Chen, Z.; Liu, B. *Lifelong Machine Learning*, 2nd ed.; Springer International Publishing: Cham, Switzerland, 2018; pp. 1–187. [CrossRef]
49. Hou, J.; Zhang, Y.; Liu, X. Advancing Continual Lifelong Learning in Neural Information Retrieval. *Inform. Sci.* **2025**, *687*, 121368. [CrossRef]
50. Yang, Q. Continual Learning: A Systematic Literature Review. *Neural Netw.* **2025**, *195*, 108226. [CrossRef]
51. Luo, Y.; Li, W.; Wang, P.; Duan, H.; Wei, W.; Sun, J. Progressive Transfer Learning for Dexterous In-Hand Manipulation with Multi-Fingered Anthropomorphic Hand. *IEEE Trans. Cogn. Dev. Syst.* **2024**, *16*, 2019–2031. [CrossRef]
52. Echchahed, A.; Castro, P.S. A Survey of State Representation Learning for Deep Reinforcement Learning. *Trans. Mach. Learn. Res.* **2025** . [CrossRef]
53. Yan, M.; Lyu, J.; Li, X. Enhancing visual reinforcement learning with State–Action Representation. *Knowl.-Based Syst.* **2024**, *304*, 112487. [CrossRef]
54. De Oliveira, B.L.M.; Martins, L.G.B.; Brandão, B.; Luz, M.L.D.; De Lima Soares, T.W.; Carvalho Melo, L. Sliding Puzzles Gym: A Scalable Benchmark for State Representation in Visual Reinforcement Learning. In Proceedings of the 42nd International Conference on Machine Learning, PMLR, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 12689–12717.
55. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016*; Balcan, M.F., Weinberger, K.Q., Eds.; PMLR: New York, NY, USA, 2016; Volume 48, pp. 1928–1937.
56. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 5026–5033. [CrossRef]
57. Hunter, J.D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **2007**, *9*, 90–95. [CrossRef]
58. Kadir, T.; Brady, M. Saliency, Scale and Image Description. *Int. J. Comput. Vis.* **2001**, *45*, 83–105. [CrossRef]
59. Rosynski, M.; Kirchner, F.; Valdenegro-Toro, M. Are Gradient-based Saliency Maps Useful in Deep Reinforcement Learning? In Proceedings of the Conference and Workshop on Neural Information Processing Systems (NeurIPS), virtually, 6–12 December 2020. [CrossRef]
60. Adebayo, J.; Gilmer, J.; Muelly, M.; Goodfellow, I.; Hardt, M.; Kim, B. Sanity Checks for Saliency Maps. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 2–8 December 2018; pp. 9525–9536.
61. Tieleman, T.; Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* **2012**, *4*, 26–31.
62. Stevens, E.; Antiga, L.; Viehmann, T. *Deep Learning with PyTorch*; Manning Publications: Shelter Island, NY, USA, 2020.