Image-Based Dataset Representations for Predicting Learning Performance in Offline RL

Enrique Mateos-Melero
Department of Computer Science and Engineering
Universidad Carlos III de Madrid
Leganés, Spain
enmateos@pa.uc3m.es

Miguel Iglesias Alcázar
Department of Computer Science and Engineering
Universidad Carlos III de Madrid
Leganés, Spain
migigles@pa.uc3m.es

Raquel Fuentetaja
Department of Computer Science and Engineering
Universidad Carlos III de Madrid
Leganés, Spain
rfuentet@inf.uc3m.es

Peter Stone
Department of Computer Science
University of Texas, Austin
Austin, Texas 78712-1188
pstone@cs.utexas.edu

Fernando Fernández
Department of Computer Science and Engineering
Universidad Carlos III de Madrid
Leganés, Spain
ffernand@inf.uc3m.es

Abstract

In this paper, we address the challenge of predicting learning performance in offline Reinforcement Learning (RL). It is a crucial task to ensure the learned policy performs reliably in the real world and to avoid unsafe or costly interactions. We introduce a new approach that utilizes Convolutional Neural Networks (CNNs) to analyze offline RL datasets, represented as images. Our model predicts the performance of policies learned from these datasets within a specific RL framework, including the selected algorithm and hyperparameters. We explore the model's transferability across different scenarios with alterations in state space size or transition functions. Furthermore, we demonstrate an application of our model in optimizing offline RL datasets. Leveraging genetic algorithms, we navigate through potential dataset subsets to identify a reduced version that enhances policy learning efficiency. This optimized dataset reduces training time while achieving comparable or superior performance to the complete dataset.

17th European Workshop on Reinforcement Learning (EWRL 2024).

1 Introduction

In offline reinforcement learning (RL), the quality of the dataset is crucial for learning an effective policy. The characteristics of the dataset and the method used to gather the data significantly impact the performance of the learned policy. While much of RL research focuses on optimizing algorithms to enhance policy learning, our approach shifts the focus to the dataset itself. Instead of improving the offline RL algorithm, we aim to efficiently evaluate a dataset to determine how "good" it is for learning.

Recent studies by Schweighofer et al. [18, 17] emphasize the importance of dataset characteristics such as Trajectory Quality (TQ), measured by the average dataset return, and State-Action Coverage (SACo), measured by the number of unique state-action pairs. Despite recognizing these characteristics as crucial for determining the quality of a learned policy, what constitutes a "good" dataset remains ambiguous.

In this paper, we develop a model to predict the performance of the policies that would be learned from offline RL datasets within a predefined RL framework. This framework includes a specific offline RL algorithm and its associated hyperparameters. To achieve this, we propose representing offline RL datasets as images and leveraging Convolutional Neural Networks (CNNs) [11] to forecast dataset performance.

Having a rapid estimator of dataset performance is highly useful. Such an estimator can quickly identify high-quality datasets, which is critical in applications like transfer learning [28, 21], curriculum learning [13], Sim2Real [27], and batch RL [10]. These applications often require evaluating multiple datasets or adapting to new environments efficiently, and our model facilitates these processes by providing swift and reliable performance predictions.

Furthermore, we assess the transferability of our predictive models by applying them to new scenarios with modified state space sizes and state transition functions. Our findings indicate promising potential for reusing models across similar tasks, opening avenues for applications in various RL contexts.

We also demonstrate an application of our model where we optimize an offline RL dataset to create a reduced version that maintains or improves policy performance. Using genetic algorithms, we explore the space of possible dataset subsets, treating a single episode as the minimum unit. Our predictive model computes the fitness function for the genetic algorithm, resulting in a streamlined dataset that enables faster training and improved policy learning compared to the complete dataset.

Our contribution is to introduce a novel approach to predict the performance of offline RL datasets by representing them as images and using CNNs. Moreover, we explore the use of genetic algorithms applying our predictive model to enhance dataset quality, providing a method to create efficient datasets for RL training.

The structure of this paper is as follows: Section 2 provides background on RL, Section 3 formalizes the Dataset Performance Prediction problem, its image-based solution and the results in three different environments. Section 4 frames the dataset reduction problem as an optimization task, and presents the experiments conducted and their results. Finally, Section 5 reviews related work, and Section 6 offers conclusions and directions for future work.

2 Preliminaries

Reinforcement Learning (RL) addresses the problem of learning to control a dynamic system defined by a Markov Decision Process (MDP) [19]. An MDP can be defined as a tuple $\mathcal{M} = \langle S, A, R, \mathcal{P}, \gamma, \mathcal{I}, \beta \rangle$, where S is a set of states; A is a set of actions; R is a reward function $R : A \times S \to Pr[\mathbb{R}]$, $\mathcal{P} : S \times A \times S \to Pr[S]$ is a transition distribution function; $\gamma \in [0, 1)$ is a discount factor; $\mathcal{I} : Pr[S]$ is an initial state distribution; and $\beta : S \to \{0, 1\}$ is an episode termination function. At time step t, the state of the environment is s_t and the agent, using its behavior policy $\pi : S \to Pr[A]$, selects an action a_t that, when executed in the environment, produces a reward $r_t \sim R(s, a)$ and leads to a new state $s_{t+1} \sim \mathcal{P}(s, a)$. The sequence (s_t, a_t, r_t, s_{t+1}) is called an *experience*. A sequence of experiences is an *episode*. When $\beta(s') = 1$ or a horizon T is reached the episode ends. The sequence of state-action transitions in an episode is a *trajectory*, $\tau = (s_0, a_0, s_1, a_1, \dots)$, which can be derived directly from the episodes.

Given a policy π, the *long-term discounted return* $\mathcal{J}(\pi)$, defined by Equation 1, is its expected cumulative discounted reward. An *optimal policy* π^* is a policy maximizing $\mathcal{J}(\pi)$: $\pi^* = \arg\max_\pi \mathcal{J}(\pi)$. The objective of an RL agent is to learn an optimal policy.

$$\mathcal{J}(\pi) = \mathbb{E}_{s_0 \sim \mathcal{I}, s_{t+1} \sim \mathcal{P}(s_t, \pi(s_t)), r_t \sim R(s_t, \pi(s_t))} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{1}$$

In online RL, the agent learns through trial and error by interacting with the environment [19]. However, in offline RL, which is a fully data-driven method, the agent is not allowed to interact with the environment during learning [10, 12]. Instead, it receives a dataset $\mathcal{D}_{env}$ consisting on set of M episodes sampled from the environment, $\mathcal{D}_{env} = \{e_i\}_{i=1,\dots,M}$. Since the training data is given, and it is usually finite, the agent can not expect to come up with an optimal policy. Then, the objective of the agent is to derive the *best* possible policy from the dataset for the given environment [10].

3 Image-based Dataset Performance Prediction

The long-term expected return (Equation 1) is a metric for a policy's performance. In offline RL, we assimilate the *quality (or performance) of a dataset* to the performance of the policy learned by an RL agent in a specific RL framework (algorithm, parameters, etc.), denoted as θ. Thus, a direct approach to estimate the performance of a dataset $\mathcal{D}$ is to perform the RL process to learn a policy π from $\mathcal{D}$, and then evaluate that policy through executions in the environment. Let E be a set of episodes obtained by executing π in the environment from initial states $s_0 \sim \mathcal{I}$. Then, we can define the estimate of the performance for $\mathcal{D}$ as the average discounted return obtained in those episodes:

$$\hat{\mathcal{J}}_{RL}(\mathcal{D} \mid \theta) = \frac{1}{|E|} \sum_{e \in E} \sum_{t=0}^{t_e} \gamma^t r_t^e \tag{2}$$

where t_e and r_t^e represent the last time step of episode $e \in E$ and the reward obtained at time step t in that episode, respectively.

However, the training and evaluation process to compute $\hat{\mathcal{J}}_{RL}(\mathcal{D} \mid \theta)$ can be time-consuming. Then, the described method is not useful when it is necessary to estimate the performance of a dataset quickly. It would be preferable to have a predictive model that allows obtaining a rapid estimate directly from the entire dataset at once. We define formally the problem of finding this model as the *Performance Prediction Problem (P3)*.

Definition 1 (Performance Prediction Problem (P3))**.** *Let $\mathcal{D}$ be a set of episodes and θ be a representation of an RL framework (algorithm, parameters, etc.). The P3 consists of finding a function $\hat{\mathcal{J}}_M : \mathcal{D} \to \mathbb{R}$ that approximates the return that would be obtained with the best policy, $\pi_\mathcal{D}$, that can be learned from $\mathcal{D}$ by an RL agent given θ: $\hat{\mathcal{J}}_M(\mathcal{D} \mid \theta) \approx \mathcal{J}(\pi_\mathcal{D} \mid \theta)$.*

Note that we always maintain the dependency on the RL framework, θ. It is clear that if this framework changes (e.g., the generalization method), the performance obtained from learning with the same dataset may vary.

To prepare datasets for input into predictive models, a suitable representation is essential. We adopt a novel approach by representing datasets as images, tailored to the specific characteristics of the domain from which the dataset originates.

The choice to represent datasets as images arises from the potential significance of spatial information in determining dataset quality. Furthermore, deep neural networks, particularly convolutional neural networks (CNNs), stand to benefit from this data format, leveraging their ability to extract features from spatial representations.

3.1 Representing Datasets as Images

Since the representation of datasets as images must be adapted to each domain, it is difficult to define a general method to define the images. Then, we propose different alternatives and study them in three domains: *Frozen Lake*, *Mountain Car* and *Acrobot*. These domains were selected to cover a range of RL environment characteristics, including state space (discrete or continuous), the number of state space dimensions, and the type of reward (delayed or instant).

Frozen Lake is a grid-like environment from the Toy Text section of OpenAI's Gymnasium[1]. The goal is to reach the treasure from the starting position. It is a discrete, bi-dimensional environment with delayed reward. **Mountain Car** is a classic control problem also from OpenAI's Gymnasium. The goal is to reach the top of the hill using momentum. It is a continuous, bi-dimensional environment with a delayed reward. Originally, Gymnasium's implementation featured an instant reward, but it was modified to match the characteristics of Frozen Lake, with a final goal to achieve. **Acrobot** is another classic control problem. It consists of two pendulums aiming to surpass a given height. This domain has a continuous, 6-dimensional state space with instant reward. Snapshots from these environments are shown in the top row of Figure 1.

Our image representation focuses solely on the states within the datasets. While actions could also be included, we opt to omit them from the images. This decision stems from the observation that actions can often be explicitly represented, and integrating them into images may not always be straightforward. Additionally, our experimental results demonstrate that utilizing only state information yields sufficiently accurate results. We propose two different ways of representing datasets as images: state (or state abstractions) scatter plots and rendered images.

State (Abstractions) Scatter Plots The idea is to use a straightforward approach by creating scatter plots of all states, providing a visualization of the spatial distribution. For bi-dimensional state spaces, this method captures the bi-dimensional structure of the environment and provides additional contextual information. Points represent the states, with the start position indicated by a red point and the goal position by a green point or area. In high-dimensional state spaces, generating those scatter plots is unfeasible. One solution is to employ dimensionality reduction techniques such as Principal Component Analysis (PCA) [16] and autoencoders [3] to transform states into two-dimensional spaces for visualization. We decided to use autoencoders and represent the bi-dimensional abstraction of the original states with a scatter plot. The middle row of Figure 1 shows examples of each environment using this representation.

Rendered Images The idea is to use rendered versions of the environments.[2] The first step is gathering the frames of each state represented in the dataset. The images are then converted to grayscale and thresholded to separate the background (black) from the environment objects (white). This step reduces the number of channels to feed the CNN, captures the motion in the environments, and ensures the background remains black in subsequent steps. Next, we sum all the pixel values from the dataset frames and normalize the result such that brighter areas are close to 255 and darker areas close to 0. Examples for each environment are shown in the bottom row of Figure 1.

3.2 Experimental Results

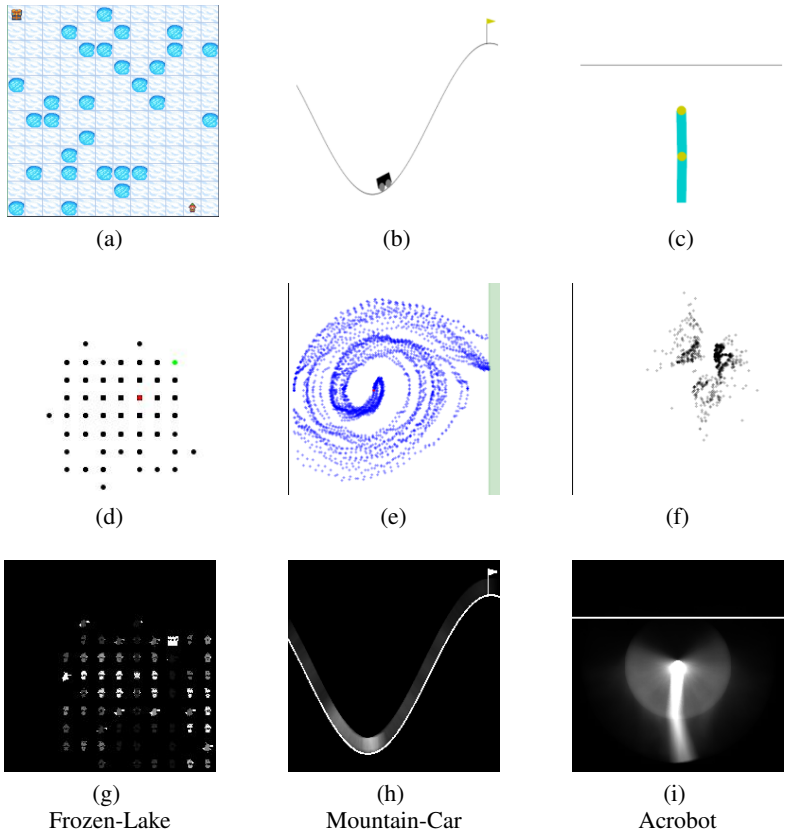
In this section, we evaluate the CNN models in each environment. Usually, in RL a change in the transition function means that a new policy must be learned. However, the spatial distribution of states will share characteristics in similar tasks. Hence, if we can train the CNN models for some specific transition functions, we would be able to transfer this knowledge to different transition functions in the same environment.

Given this assumption, we create various datasets with different configurations to modify these functions. Additionally, we modified the state space when possible (Frozen Lake) and the quality of the datasets (Mountain Car and Acrobot) to prove the robustness of our solution. In Frozen Lake, we create 23 maps, where 20 are of size 12x12 and the remaining (only in the test set) with increasing sizes 24x24, 48x48 and 96x96. Datasets are collected with sub-optimal policies. In Mountain Car, we create 4 configurations with different acceleration forces and gather datasets with varying policy quality. Finally, in Acrobot we create 4 setups with different torque and pendulum lengths and obtain datasets following policies with different qualities. In all cases, for each configuration mentioned, a new dataset is generated following a sub-optimal RL agent previously trained through an online approach. Further details of the generation process are provided in Appendix A. The image generation process can be followed by Figure 3 and Appendix B.

Given the characteristics of the reward of each environment, the output of the CNN differs from one to another. In the case of Frozen Lake and Mountain Car, where the result is a value of 0 or 1, the output

[1] https://gymnasium.farama.org/

[2] Note this is not possible in all environments.



(a) (b) (c)

(d) (e) (f)

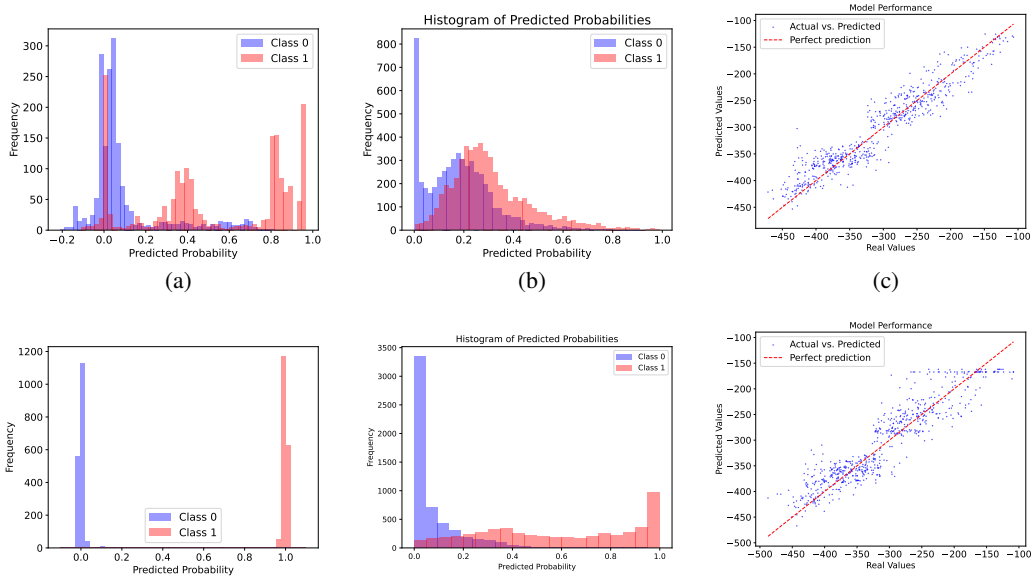
(g) Frozen-Lake (h) Mountain-Car (i) Acrobot

Figure 1: Representations of the Open AI's environments used in the paper. On the top row, a snapshot of each environment. On the middle row, the state representation/abstraction of the corresponding environments. On the bottom row, the rendered representation of the environments. Each column represent one domain: Frozen Lake, Mountain Car and Acrobot respectively.

is a value between 0 and 1, considered as the confidence level of being a good subset. In the case of Acrobot, the output of the CNN is a value between -500 and 0. The specific CNN architecture for each case is detailed in Appendix C. These models are trained by receiving as input examples of images from subsets of the original datasets that are previously labeled. We use some datasets and configurations during the training processes and different ones for test purposes. Figure 2 shows the results from the CNN models using rendered (top row) and state (bottom row) representations across different domains, which are summarized as follows.

Frozen Lake In this case, both image representations of the environment use the same CNN model architecture (see Appendix C). After training the models, the accuracy obtained for the scatter plotted representation is 99% and for the rendered images 92%. As illustrated in Figures 2a and 2d, the representation using a scatter plot of the states can discriminate between "good" and "bad" datasets. However, with the rendered representation it is more difficult to make this differentiation. After analyzing the results, we found that the red pick around 0.0 confidence in Figure 2a corresponds to all the results for a specific map. This map has the goal close to the start point and it may be more difficult for the model to distinguish the existence of a path between the red and green points.

Mountain Car Upon completion of training, the predictor trained with the images representing the states as points in two dimensions achieved an accuracy of approximately 79% on the test set. Using the rendered images the model barely surpasses the 50% accuracy. Given the results provided in Figure 2e, the state representation can accurately differentiate between cases. Additionally, "bad" cases are better discriminated than "good" ones. As expected from the low accuracy of the model, the rendered images (Figure 2b) are not suitable to represent "good" and "bad" cases in this environment.



(a) (b) (c)

(d) Frozen Lake (e) Mountain Car (f) Acrobot

Figure 2: Results from the CNN models using rendered (top row) and state (bottom row) representations across different domains. Each column corresponds to a different domain: Frozen Lake, Mountain Car, and Acrobot. The plots for Frozen Lake and Mountain Car show the distribution of confidence provided by the model for "good" (red) and "bad" (blue) datasets. The plots for Acrobot compare the true labels of the test set instances with the predicted values, with the dashed red line representing the ideal scenario where predicted values perfectly match the real labels.

Acrobot Following training, an L1 error, which measures the average absolute differences between predicted and actual values, of approximately 22 was attained when utilizing rendered images. Conversely, an error of 25 was recorded when employing images of an autoencoder-based dimensionality reduction. In Figures 2c and 2f we provide the predicted labels against real labels scatter plots. Both representations distribute the results along the dashed diagonal line that assumes the perfect model. However, when using the autoencoder representation, the model demonstrates underperformance in higher value ranges. The rendered image model is capable of such distinction.

Already in Definition 1 we considered the hyperparameters and type of algorithm in the RL process. Given the results of the predictors, we can observe that the representation used is also a key factor for each environment. In our results, the scatter plots representation is better for the bi-dimensional environments. Conversely, the rendered images are more suitable for Acrobot.

4 Case Study: Dataset Reduction

Once we prove the possibility of distinguishing between "good" and "bad" examples of datasets we present one of its possible applications. In offline RL, the input dataset, denoted as $\mathcal{D}_{env}$, can often be overwhelmingly large, making the execution time of the learning algorithm impractical. Additionally, the dataset's source may be a mixture of "good" and "bad" policies. Therefore, a fundamental question arises: Can we significantly reduce the size of $\mathcal{D}_{env}$ while still enabling the learning algorithm to produce a high-quality policy? This question essentially frames the problem as a multi-objective optimization task. On the one hand, we aim to minimize the size of the resulting dataset, and on the other hand, we seek to maximize the quality of the policy learned from it. However, it is possible to simplify this into a single-objective optimization problem by combining both objectives into a single metric. Let us define the Episode Selection Problem (ESP) metric, denoted as ψ, as follows:

Definition 2 (ESP metric ψ)**.** *Let $\mathcal{D}$ be a set of episodes, θ be a representation of the RL framework (algorithms, parameters, etc.), and $\hat{\mathcal{J}}(\mathcal{D} \mid \theta)$ be an estimate of the return that would be obtained with*

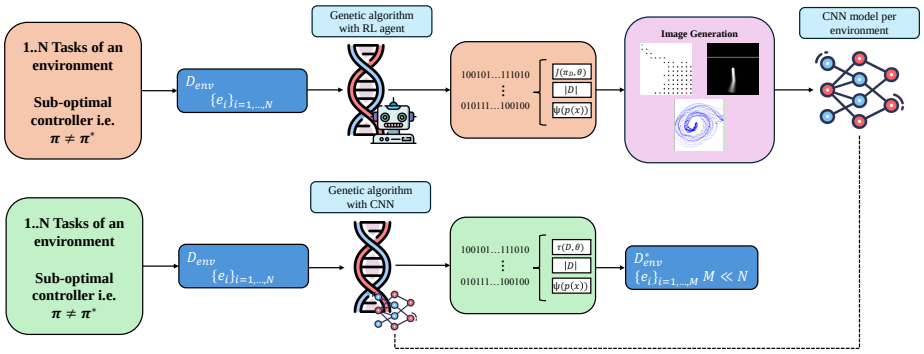


Figure 3: Scheme used to reduce the datasets. In orange, we consider the environment configurations used to train the CNN models. Labels for each image generated (square in pink) are obtained from Equation 1 given an RL training and evaluation process for each individual of the genetic algorithm. In green, we consider the environment configurations and datasets used to test our solution. The fitness function uses the trained CNN models. The output is a reduced dataset that maintains or improves the learned policy.

the best policy by an RL agent given θ (Definition 1). Then, an ESP metric $\psi : \mathcal{D} \to \mathbb{R}$ is a function of the return estimate and the number of episodes in $\mathcal{D}$: $\psi(\mathcal{D}) = f(\hat{\mathcal{J}}(\mathcal{D} \mid \theta), |\mathcal{D}|)$

Therefore, with this metric in place, we can formally define the Episode Selection Problem (ESP) as follows:

Definition 3 (Episode Selection Problem, ESP$(\mathcal{D}, \psi)$)**.** *Given a set of episodes $\mathcal{D}$ and an ESP metric ψ, find a reduced subset $\mathcal{D}^* \subseteq \mathcal{D}$ that maximizes ψ:* $\mathcal{D}^* = \arg\max_{\mathcal{D}' \subseteq \mathcal{D}} \psi(\mathcal{D}')$

An ESP optimally could theoretically be solved by a brute-force approach, by generating all possible subsets of the input dataset and evaluating them using the metric ψ under θ. However, given the potentially enormous size of $\mathcal{D}_{env}$ and the exponential number of subsets ($2^{|\mathcal{D}_{env}|}$), such a brute-force approach is not feasible. Instead, we propose a method to find a suboptimal solution using stochastic local search. Specifically, we employ a genetic algorithm [7, 14], where each individual in the population represents a different subset of the training set, and the fitness function is defined in terms of the ESP metric, ψ.

4.1 Genetic Algorithm for Solving the Episode Selection Problem

To solve an episode selection problem, ESP$(\mathcal{D}, \psi)$, with a genetic algorithm (GA) we need to represent the reduced datasets as individuals of the population and define the fitness function.

Each individual chromosome is encoded as a string of binary values representing which episodes in the input dataset $\mathcal{D}$ are selected. Each individual thus represents a reduced dataset $\mathcal{D}' \subseteq \mathcal{D}$ containing the selected episodes. The length of the string is the number of episodes in $\mathcal{D}$, $|\mathcal{D}|$. Each gene of the chromosome represents the presence of the corresponding episode in $\mathcal{D}'$. Formally, let $\mathcal{D} = \{e_1, e_2, \ldots, e_M\}$ be the input dataset, where e_i is the i-th episode in $\mathcal{D}$ and $|\mathcal{D}| = M$. Let X be the space of all possible individuals. Then, every $x \in X$ is defined as $x = x_1, x_2, \ldots, x_M$, where x_i is the i-th gene of the individual and $x_i \in \{0, 1\}$. The mapping function from the genotype to the phenotype is $\rho : X \to 2^{\mathcal{D}}$, so that given an individual $x \in X$, the corresponding reduced dataset is $\rho(x) = \{e_i \in \mathcal{D} \mid x_i = 1\}$.

The fitness of each individual $x \in X$ is computed using the ESP metric ψ (Definition 2) by applying ψ to $\rho(x)$. Thus, the fitness function, $\phi : X \to \mathbb{R}$, is defined as $\phi(x) = \psi(\rho(x))$. The performance of each individual, $\hat{\mathcal{J}}(\mathcal{D} \mid \theta)$, used to compute ψ, is obtained by transforming the individual into an image representation and using a trained model (CNN) as the evaluator. Figure 3 shows an scheme illustrating our solution.

Table 1: Results of generalization of the method for the first and second group of experiments, separated by a double line. Column *Original* denotes the complete dataset. Columns *States* and *Rendered* refers to the type of images used for the CNNs (state scatter plots and rendered environment images, respectively). Domains are represented by one letter followed by the type of modification performed. Frozen Lake (F), Mountain Car (M), Acrobot (A). Modifications to transition function ($\mathcal{P}$), state space (XS where X is multiplying factor to the base 12x12), policy expertise (π') and epsilon (ϵ).

	Obtained policy			**Number of tuples**		
Task	Original	States	Rendered	Original	States	Rendered
F$\mathcal{P}$	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1111.69 ± 605.11	91.34 ± 52.39	130.31 ± 56.52
F2S	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	1006	105.83 ± 6.77	344.67 ± 23.12
F3S	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	1587	154.08 ± 34.06	307.92 ± 15.31
F4S	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	2534	370.34 ± 26.34	960.58 ± 72.41
M$\mathcal{P}$	0.67 ± 0.5	0.8 ± 0.42	0.66 ± 0.5	15804.33 ± 3235	4748.8 ± 2396.55	6098 ± 2376
A$\mathcal{P}$	-341.92 $\pm$ 86.87	-157.729 $\pm$ 70.56	-183.99 $\pm$ 84.97	22623.75 ± 5644.55	1504.29 ± 646.77	1528.92 ± 573.60
Mπ'	0.5 ± 0.55	0.83 ± 0.41	0.83 ± 0.41	16978.5 ± 1860.61	5656.5 ± 636.36	7401.33 ± 1244.02
Mϵ'	1.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	19443.0 ± 985.32	6097.67 ± 110.98	8340.0 ± 2099.02
Aπ'	-397.16 $\pm$ 85.58	-188.5 $\pm$ 38.94	-158.97 $\pm$ 59.65	27279 ± 7760.13	2465.00 ± 335.23	964.08 ± 210.45
Aϵ	-439.87 $\pm$ 50.25	-381.59 $\pm$ 59.22	-163.82 $\pm$ 93.69	23627 ± 12172.59	2165.00 ± 455.64	721.00 ± 150.40

4.2 Experimental Results

In this section, we evaluate the performance of the best individual (best-reduced dataset) obtained when the ESP is solved by the GA using the CNN predictor, in comparison to the real value of the performance of the original dataset, computed by Equation 2 and using the policy learned by the offline RL algorithm. We use the same domains as before. Experiments are divided into two groups, described below.

In the first group, we evaluate the generalization ability of the method when datasets gather data from configurations with different transition functions and, additionally, in Frozen Lake, we consider datasets with modified state space. Specifically, in Frozen Lake, the 12x12 maps are joined as the modification of the transition function, and bigger maps are presented separately. In Mountain Car, we consider the different forces as the modification of the transition function. In Acrobot, we consider different lengths of the lower pendulum and the torque force applied to modify the transition function. Frozen Lake is the only environment with state space modifications because it is the only one with this possibility (changing the map size).

In the second group, we evaluate the generalization ability of the method when there are changes intrinsic to the RL training process by using different policy qualities to gather the data. Specifically, we include 3 policy values and a single epsilon value for both Mountain Car and Acrobot. Frozen Lake is not included in this group due to its learning simplicity and thus difficulty in obtaining different policy expertise.

All experiments use $\psi(\mathcal{D}) = 0.8 \times \hat{\mathcal{J}}(\mathcal{D} \mid \theta) + 0.2 \times |\mathcal{D}|$ as ESP metric, where $\hat{\mathcal{J}}(\mathcal{D} \mid \theta)$ is obtained by the trained CNN models used in Section 3.2. The initial population for the GA is generated randomly with 50% chance for each episode to appear in the given subset.

All configurations were executed 5 times. Table 1 shows the results for the first and second groups of experiments in terms of the average and standard deviation of the performance metric and the number of tuples used for both the complete dataset and the resulting subset from the GA process, depending on the type of images used for the CNNs. The results show that our method generally maintains (Frozen Lake) or improves (Mountain Car and Acrobot) the learned policy. This is not fulfilled in the rendered version of bigger maps in Frozen Lake. Moreover, we can observe between 70% to even 95% dataset size reduction in most cases.

In Figure 4, we present a comparative analysis of the performance of datasets generated through the genetic algorithm in contrast to random datasets of varying sizes. Note that we also include the complete dataset in this comparison. Each figure in the plot series corresponds to the training progression of the respective reinforcement learning (RL) algorithm. While training, the policy obtained at each interval was evaluated, with the x-axis denoting the training time and the y-axis representing the reward achieved. We executed each case 100 times and computed the mean value and standard deviation. In the case of Mountain Car and Acrobot, we use a moving average for visualization purposes. Comprehensive details of the training procedures employed to generate these

plots can be found in Appendix D. Results show that our solution outperforms the random selection and the complete dataset in the number of updates needed for the training to converge, rewards obtained, or both.

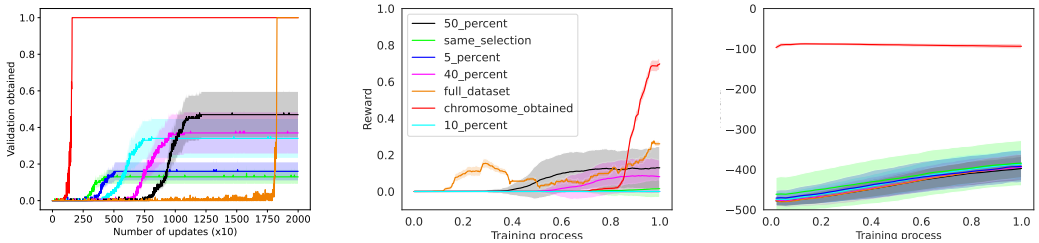


Figure 4: Comparison of performance of the genetic algorithm against random and full datasets for Frozen Lake, Mountain Car and Acrobot respectively.

5 Related work

In offline RL, data quality is an important factor. Data quality is not considered in many of the offline RL algorithms available in the literature diminishing their performance. We will divide this section between approaches that estimate dataset performance and dataset reduction.

5.1 Dataset performance estimation

There are different approaches to estimating the performance of datasets in Offline RL. Some metrics have been developed to quantify the data quality like Estimated Relative Return Improvement (ERI) [20] or TQ and SACo [17]. However, these factors do not actually reflect the possible performance after a training process with the dataset. The metrics cannot determine the quality of a dataset since they are based on averaging the expected returns from the episodes that constitute the dataset. Additionally, they do not take into consideration the existence of datasets created from a mixture of policies and still rely on the expectation formula (Equation 1).

Such considerations are reflected in algorithms like LBRAC-v [26]. The algorithm assumes the datasets are generated from different behavior policies. With this assumption, it tries to learn a latent variable model that is capable of grouping all the trajectories in the dataset into their corresponding behavior policy. This latent variable is used to solve the degeneration problem from the BRAC-v [22] algorithm. However, it still does not estimate the performance of a dataset as a whole.

An example of dataset estimation is DVORL [1]. This algorithm has a sub-module that performs data valuation which consists of estimating datum quality in overall performance [4, 25]. This is performed on a subset of the dataset to make the best possible selection for the target task. However, this algorithm assumes the existence of a target dataset, which may not exist or be unknown. The algorithm is unusable without the target dataset since it is used for a KL divergence metric [8] during the learning process. We can estimate without the need for an auxiliary target dataset.

5.2 Dataset reduction

Offline Imitation Learning algorithms are useful to learn from demonstrations. Most of them are based on Behavioural Cloning (BC) [2] which uses supervised learning techniques to learn. However, the main problem with this type of algorithm is that they imitate equally data with expert and sub-optimal behavior. This leads to bad performances. Hence there are various approaches to reduce datasets to keep only the useful data for the training process.

The approaches to solving this issue are based on using discriminators to distinguish between expert data and non-expert data. In *Discriminator-Weighted Offline Imitation Learning from Sub-optimal Demonstrations* [24] the idea is to build a discriminator on top of a BC algorithm to weight the input data depending on its expertise. They use a cycle to learn both the model and discriminator. COIL [13] assumes that each trajectory has been collected by an independent policy. It measures the KL

divergence of each trajectory to determine the similarity with the learned policy and filter the dataset. Other examples like 2IWIL and IC-GAIL [23] based their discrimination strategies on importance weighting or Generative Adversarial Networks (GANs) [5] using GAIL [6] as their base algorithm. Our approach creates a new "cleaned" dataset instead of using the original one.

6 Conclusions

In this paper, we propose an efficient method for estimating the performance of a dataset in Offline RL without a conventional training process. We tackle different RL problems such as continuous and high dimensional state spaces and delayed and immediate reward functions. Hence, we have represented datasets as images and used a convolutional neural network (CNN) as our approximation function. Additionally, we present one of the possible applications of this method. We managed to reduce dataset size in offline reinforcement learning while maintaining or improving the resulting policy quality. We use a genetic algorithm to search on the enormous collection of subsets.

We evaluated our approach in three well-known RL domains, Frozen Lake, Mountain Car, and Acrobot, achieving significant estimation models capable of differentiating "good" and "bad" datasets. In our application, we reduce the dataset size while keeping or enhancing policy quality. We also acknowledge that our estimators can transfer the learned knowledge to similar tasks.

The positive outcomes motivate future research directions. We plan to test our method in more complex environments and with continuous action spaces. Adapting our method to other representation forms is another possibility. Additionally, exploring new RL applications could lead to innovative dataset-reduction techniques. This research lays the foundation for refining and expanding efficient dataset reduction methods in offline and batch RL.

Acknowledgments and Disclosure of Funding

This work was partially funded by grant PID2021-127647NB-C21 from MCIN/AEI/10.13039/501100011033, by the ERDF “A way of making Europe”, and by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation). The work was also funded by Repsol S.A. under grant with UC3M. Fernando Fernández was also partially funded by a grant from Spanish *Ministerio de Universidades* though the mobility program "Salvador de Madariaga y José Castillejo".

References

[1] A. Abolfazli, G. Palmer, and D. Kudenko. Data valuation for offline reinforcement learning. *arXiv*, 2205.09550, 2022.

[2] M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence 15, Intelligent Agents*, pages 103–129. Oxford University, 1999.

[3] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. In *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, pages 353–374. Springer International Publishing, 2023.

[4] A. Ghorbani and J. Zou. Data shapley: Equitable valuation of data for machine learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2242–2251. PMLR, 2019.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems. Proceedings of the twenty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2014)*, volume 27, pages 2672–2680. Curran Associates, 2014.

[6] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems. Proceedings of the thirtieth Annual Conference on Neural Information Processing Systems (NeurIPS 2016)*, volume 29, pages 4565–4573. Curran Associates, 2016.

[7] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[8] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[9] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems. Proceedings of the thirty-fourth Annual Conference on Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 1179–1191. Curran Associates, 2020.

[10] S. Lange, T. Gabel, and M. Riedmiller. *Batch reinforcement learning*. Springer, 2012.

[11] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[12] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, 2005.01643, 2020.

[13] M. Liu, H. Zhao, Z. Yang, J. Shen, W. Zhang, L. Zhao, and T.-Y. Liu. Curriculum offline imitating learning. In *Advances in Neural Information Processing Systems. Proceedings of the thirty-fifth Annual Conference on Neural Information Processing Systems (NeurIPS 2021)*, pages 1–12. Curran Associates, 2021.

[14] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3nd, extended ed.)*. Springer, New York, NY, USA, 1996.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv*, 1312.5602, 2013.

[16] K. Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[17] K. Schweighofer, M. Hofmarcher, M.-C. Dinu, P. Renz, A. Bitto-Nemling, V. P. Patil, and S. Hochreiter. Understanding the effects of dataset characteristics on offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021.

[18] K. Schweighofer, A. Radler, M.-C. Dinu, M. Hofmarcher, V. Patil, A. Bitto-Nemling, H. Eghbal-zadeh, and S. Hochreiter. A dataset perspective on offline reinforcement learning. In *Proceedings of the 1st Conference on Lifelong Learning Agents*, volume 199, pages 470–517. PMLR, 2022.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[20] P. Swazinna, S. Udluft, and T. Runkler. Measuring data quality for dataset selection in offline reinforcement learning. In *Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2021.

[21] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(56):1633–1685, 2009.

[22] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv*, 1911.11361, 2019.

[23] Y.-H. Wu, N. Charoenphakdee, H. Bao, V. Tangkaratt, and M. Sugiyama. Imitation learning from imperfect demonstration. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 6818–6827. PMLR, 2019.

[24] H. Xu, X. Zhan, H. Yin, and H. Qin. Discriminator-weighted offline imitation learning from suboptimal demonstrations. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 24725–24742. PMLR, 2022.

[25] J. Yoon, S. Arik, and T. Pfister. Data valuation using reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 10842–10851. PMLR, 2020.

[26] G. Zhang and H. Kashima. Behavior estimation from multi-source data for offline reinforcement learning. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 11201–11209. AAAI Press, 2023.

[27] W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[28] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45:13344–13362, 2020.

A Details on Dataset Generation

The initial phase involves gathering datasets for training and evaluating the transferability of a performance predictor across three environments: Frozen Lake, Mountain Car, and Acrobot. We employed different RL algorithms and data collection strategies tailored to each environment to generate diverse datasets. In general, the process consists of generating a policy training an online RL algorithm and then sample datasets from that policy or policies. The details for each environment are as follows:

- **Frozen Lake:** This environment consists of 23 maps with varying sizes and obstacle placements. We used tabular Q-learning to generate policies online, which were then used to create the datasets. The parameters used for tabular Q-learning are $\gamma = 0.99$ and varying α values depending on the specific configuration. The datasets were divided into training, validation, and testing sets with the following splits:
  - **Training Set:** 10 maps with different sizes and obstacle configurations. These maps are of size 12x12, the start and goal positions are randomly placed along the map and 20% of the tiles are chosen to fill them with obstacles.
  - **Validation Set:** 4 maps were used to tune the predictor. These maps use the same creating procedure as the training ones.
  - **Testing Set:** The remaining 9 maps to evaluate transfer capabilities across different settings. In this case, 6 maps still have size 12x12 and follow the same creation procedure. The other three maps are of size 24x24, 48x48, and 96x96 with the same creation procedure. For these last three maps, datasets are generated by solving the path with the A* algorithm (due to complexity) and gathering the data following this path 80% of the time and 20% randomly.

  The variation in map configurations tests the predictor's ability to generalize across different scenarios within the Frozen Lake environment. During the image generation process, we apply data augmentation to these images by rotating 90º, 180º, and 270º and performing mirrors along the vertical axis, the horizontal axis, and both. The result is 42755 images to train, 2213 images to validate, and 3584 images to test. Images are of size 204x204.
- **Mountain Car:** The state space in this environment was discretized uniformly into $40X40$ states. Datasets were generated with variations in three key parameters:
  - **Applied Force:** The magnitude of force applied to the car, specifically using the values: 0.001, 0.0015, 0.002, and 0.0025.
  - **Policy Expertise:** Datasets were generated using policies with varying expertise levels, from near-random (0) to highly optimized (100). These policies were obtained by sampling 100 stages from the online training process of the final policy.
  - **Random Action Probability:** This parameter reflects the likelihood of the policy taking a random action, with probabilities set to 30%, 50%, or 70%.

  The Q-learning parameters used for online policy training were: $\alpha = 0.2$, $\gamma = 0.99$, and we also used a linear epsilon decay exploration method with a starting epsilon of 0.9 and a final epsilon of 0.1. The policy was trained over 20000 episodes. The datasets were split into training and testing sets to assess the performance predictor's robustness across policy and environment variations. Specifically, the training set included datasets generated with applied forces of 0.0015 and 0.002, policy expertise of 50, and random action probabilities of 30% and 50%. The remaining datasets were used for testing. We used 86139 images to train, 10765 to validate, and 10770 to test. Size of images is 220x220.
- **Acrobot:** For Acrobot, we used a Deep Q-Network (DQN) algorithm [15] to obtain the policies to generate datasets. The original environment settings were retained, but the data collection policies varied:
  - **Behavior Policies:** Datasets were generated using different behavior policies to capture a range of performance levels. These levels are represented by using different policies obtained during the online training process.

  The parameters for the DQN algorithm were:
  - **Batch Size:** 128

- **Target Update Interval:** 250
- **Gamma:** 0.99
- **Learning Rate:** 6.3×10^{-4}
- **Training Steps:** 1000
- **Buffer size:** 100000
- **Explorer:** `Linear decay Epsilon Greedy`

The predictor was trained on datasets generated from certain behavior policies and tested on others to evaluate its ability to generalize across different policy behaviors. Specifically, policies obtained at 10%, 20%, and 53% of the online training were used for the training set, and policies at 15%, 24%, 44%, and 57% were used for the testing datasets not used to train the CNN. We used 2887 images to train, 618 to validate, and 620 to test. Images size is 220x220.

B Details on image generation

We employed genetic algorithms to generate the images. During the execution process, each chromosome (representing a subset of the dataset) visited was stored along with its corresponding label. This label is computed as part of the fitness function, which evaluates the policy obtained after training an offline RL algorithm on the selected subset of the dataset. After this process, the images were generated from the chromosomes, with data balancing to ensure a uniform label distribution.

B.1 Genetic Algorithm Parameters

The genetic algorithm was configured with the following parameters:

- Initial population: Randomly sampled individuals containing approximately 50% of the episodes.
- Mutation probability: 0.1.
- Selection method: Tournament selection with a tournament size of 10.
- Crossover method: Bit crossover performed by all individuals.
- Population size: 50.
- Number of generations: 100.

B.2 Offline RL Algorithms and evaluation

For the evaluation of policies and labeling of the images, we employed the following offline RL algorithms tailored to each environment:

Frozen Lake Environment: We use tabular Q-learning with the following parameters:

- Learning rate (α): 0.1.
- Discount factor (γ): 0.99.
- Number of episodes: 20.

Mountain Car Environment: We used tabular Q-learning by discretizing the continuous environment. We used a uniform discretization of 40 bins for both position and velocity. Algorithm parameters are:

- Learning rate (α): 0.5.
- Discount factor (γ): 0.99.
- Number of episodes: 10.

In order to evaluate a learned policy in Mountain Car and Frozen Lake, we run 10 episodes following that policy and compute the mean of the undiscounted rewards obtained.

Acrobot Environment: We employed the discreteCQL algorithm [9] from D3rlpy library with the following hyperparameters:

- Batch size: 128.
- Target update interval: 10^{20}.
- Discount factor (γ): 0.99.
- Learning rate: 6.3×10^{-4}.
- Number of critics: 5.
- Maximum number of steps: 1000.

To assess the policy acquired through the offline RL algorithm, we adopted a strategy of storing 100 intermediate policies obtained during the training process, with each policy stored at 1% intervals during the training. Subsequently, we evaluated each policy by executing one episode sampled from it. The resulting performance of each episode execution was averaged to derive the validation metric for the dataset. This approach essentially computes the average performance across 10% of the training process. We opted for this method due to encountered convergence issues.

Our decision stems from the observation that relying solely on the final policy learned and evaluating it multiple times could yield markedly different results in each evaluation, despite the training process exhibiting a consistent trend. Thus, by averaging the performance over multiple intermediate policies, we obtain a more stable representation of the training progress and the effectiveness of the learned policy.

C CNN architectures and training

C.1 Frozen Lake:

Feature Extraction Layers:

```
self.features = nn.Sequential(
    nn.BatchNorm2d(3),
    nn.Conv2d(3, 32, kernel_size=5, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=3),
    nn.Dropout(p=0.25),
    nn.Conv2d(32, 64, kernel_size=5, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=3),
    nn.Dropout(p=0.25),
    nn.Conv2d(64, 128, kernel_size=5, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=3),
    nn.Dropout(p=0.25)
)
```

Fully Connected Layers:

```
self.fc = nn.Sequential(
    nn.Linear(1152, 256),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.25),
    nn.Linear(256, 1)
)
```

Training Details:

- **Batch Size:** 64
- **Loss Function:** MSE
- **Optimizer:** AdamW with learning rate 0.001
- **Epochs:** 50
- **Early Stopping:** patience of 10

C.2 Mountain Car (states):

Feature Extraction Layers:

```
self.features = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=11, padding=0, bias=True),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(32, 64, kernel_size=7, padding=0, bias=True),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(64, 128, kernel_size=5, padding=0, bias=True),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(128, 256, kernel_size=5, padding=0, bias=True),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2)
)
```

Fully Connected Layers:

```
self.fc = nn.Sequential(
    nn.Linear(3200, 256),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.2),
    nn.Linear(256, 1)
)
```

Training Details:

- **Batch Size:** 256
- **Loss Function:** BCE with logit loss (weight: 0.25)
- **Optimizer:** AdamW with learning rate 0.0025
- **Epochs:** 200

C.3 Acrobot:

Feature Extraction Layers:

```
self.features = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=11, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(32, 64, kernel_size=7, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(64, 128, kernel_size=5, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(128, 256, kernel_size=5, padding=0, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2)
)
```

Fully Connected Layers:

```
self.fc = nn.Sequential(
    nn.Linear(16384, 256),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.2),
    nn.Linear(256, 64),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.2),
    nn.Linear(64, 1)
)
```

Training Details:

- **Batch Size:** 64
- **Loss Function:** L1 loss
- **Optimizer:** AdamW with learning rate 0.0025
- **Epochs:** 500

We used the same architecture for the rendered images of Mountain Car as the one used for Acrobot.

D Details on solution comparison

D.1 Frozen Lake:

We decide to contrast our results from the genetic algorithm with randomly selected subsets. Hence, we created 100 chromosomes for each different number of episodes: 20, 30, 40, 50, and the same size as our solution result. We train all chromosomes and compare the results with our solution and the complete dataset. We evaluate the performance of all the training processes every 10 updates of the Q-table over 10 episodes. The results use the average and standard deviation of every group.

D.2 Mountain Car:

To analyze the training performance across datasets of various sizes in the Mountain Car environment, we initiated the process by randomly selecting 100 datasets for each of different sizes. We used different numbers of episodes: 5, 10, 40, 50, a size identical to the best subset obtained through our genetic algorithm method, and 100 episodes. Subsequently, we commenced the training process for each dataset, aiming to assess its efficacy. Additionally, we included the training process for the best subset obtained through our genetic algorithm method, considering that only one chromosome is selected by this method, resulting in a training process with a standard deviation of 0.

However, due to the deterministic nature of the Mountain Car environment, with rewards being either 1 or 0 only, rendering the training curves directly posed challenges. To enhance visualization, we applied a moving average with a window size of 50 to the plotted lines. This modification allowed us to better observe the trend of the training, which is the primary focus of our analysis.

Throughout the training process with each dataset, we evaluated the performance at regular intervals, specifically every 0.001% of the total training updates. This resulted in a total of 1000 evaluation points per dataset approximately. To facilitate comparison across different datasets and training processes, we normalized the X-axis to represent the percentage of the training process executed.

D.3 Acrobot:

Initially, we randomly selected 100 episodes for each of the different dataset sizes, including those with the same size as the best chromosome obtained, as well as sizes of 10, 20, 30, 50, and 100 episodes.

During the training process, we adopted a strategy of evaluating the policy at regular intervals. Specifically, we stored 100 intermediate policies, with each policy saved at 1% intervals of the training. Subsequently, we evaluated each of these policies by executing a single episode sampled from it.

To ensure stability in our evaluation, we applied a moving average technique with a window size of 50 to the obtained performance metrics. This allowed us to better observe the trend of the training process and the effectiveness of the learned policies over time.

E Computational Resources

This appendix provides detailed information about the computational resources used for the experiments conducted in this project. The specifications include both CPU and GPU details for both machines used.

Attribute	**Specification**
Architecture	x86_64
CPU Operation Modes	32-bit, 64-bit
CPU(s)	16
Model Name	Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz
Thread(s) per Core	2
Core(s) per Socket	8
CPU Max MHz	4800.0000
CPU Min MHz	800.0000
L1 Cache	256 KiB
L2 Cache	2 MiB
L3 Cache	16 MiB

Table 2: CPU Specifications machine 1

Attribute	**Specification**
GPU Name	NVIDIA GeForce 2060
Memory Usage	7974MiB

Table 3: GPU Specifications machine 1

Attribute	**Specification**
Architecture	x86_64
CPU Operation Modes	32-bit, 64-bit
CPU(s)	20
Model Name	12th Gen Intel(R) Core(TM) i7-12700F
Thread(s) per Core	2
Core(s) per Socket	12
CPU Max MHz	4900.0000
CPU Min MHz	800.0000
L1 Cache	512 KiB (12 instances)
L2 Cache	12 MiB (9 instances)
L3 Cache	25 MiB (1 instance)

Table 4: CPU Specifications machine 2

Attribute	**Specification**
GPU Name	NVIDIA GeForce 3060
Memory	12045MiB

Table 5: GPU Specifications machine 2