Learning to Recognize Reachable States from Visual Domains

Ella Morgan†*, Christian Muise†
† Queen’s University, Kingston, ON, Canada

Abstract
While planning models are symbolic and precise, the real world is noisy and unstructured. This work aims to bridge the gap between noise and structure by aligning visualizations of planning states to the underlying state space structure. Further, we do so in the presence of noise and augmentations that simulate a commonly overlooked property of real environments: several variations of semantically equivalent states. First, we create a dataset that visualizes states for several common planning domains; each state is generated in a way that introduces variability or noise. E.g., objects changing in location or appearance in a manner that preserves semantic meaning. First we train a contrastive learning model to predict the underlying states from the images. We then evaluate how we can align the predictions of a given sequence of visualized states with the problem’s reachable state space, taking advantage of the known structure to improve predictions. We compare two methods for doing so: a greedy approach and Viterbi’s algorithm, a well-established algorithm for observation decoding given a hidden Markov model. The results demonstrate that these alignment methods can correct errors in the model and significantly improve predictive accuracy.

Keywords: representation learning, contrastive learning, planning state visualizations, planning state representation learning

1. Introduction

There’s a significant gap between the reasoning capabilities of humans and computers, especially when it comes to making sense of real-world processes. While automated planning methods can form complex plans to achieve goals, they rely on symbolic models that are difficult to align with the noisy and unstructured nature of the real world. Deep learning excels at learning from large amounts of data, but often struggles with logical reasoning, interpretability, and long-term planning. This has led to the development of neurosymbolic methods, which aim to combine the power of learning from data with the logical reasoning capabilities of symbolic methods.

In this work, we propose a neurosymbolic approach for reasoning about predictions made by a deep learning model. As input, we take in visual representations of planning states with variations that make images representing the same state appear different visually. After training a deep learning model that predicts the state from these images, we then take advantage of prior knowledge about the state space structure to validate and improve on these predictions. Specifically, given a sequence of images \( O = (o_1, o_2, \ldots, o_T) \) we make the assumption that there is a single action applied between each sequential pair of images. Given prior knowledge of which transitions are possible in the given domain, we can then maximize the probability of the sequence of observations aligning with the state space. We show how this problem can be modeled as a hidden Markov model (HMM) and apply two different algorithms to align the observations with the given state space.

Classification models are often evaluated on both the top-1 and top-n accuracy. The former measures how frequently the method’s top prediction is correct, and the latter measures how frequently the correct value was within the top n values the model ranked as most likely. Even in problems with thousands of states, we find that even if the top-1 accuracy is quite poor the correct prediction is generally ranked quite high and can be found searching...
Figure 1. The top of the figure displays four images of visualized states from the grid domain. Possible actions include moving the robot between adjacent rooms, and picking up or placing an object in the room the robot is located in. In the first state, the robot is holding a t-shirt, which is represented by an icon in the bottom left corner. In the next states, the item is placed in the current room, and then the robot moves down and then right. The state space is shown in the graphs below the images, where nodes represent states and edges represent valid actions between states, with the current and previous states highlighted. First, a model is trained to predict the states from the images. Afterwards, we use the structure of the state space to align the sequence of the predictions to a path in the state space graph, maximizing the probability of the sequence of state predictions aligning with the state space.

within a relatively small fraction of the most-likely options. This motivates our addition of a knowledge base to validate which predictions are correct and take advantage of a model with low top-1 and high top-n accuracy that otherwise cannot determine the single best prediction.

In order to align real-world observations with an underlying structure, we first generate a dataset of visualizations. Each visualization represents a state in the planning domain, with all information necessary for determining the state present. We treat these states as blackboxes, with the goal being only to identify which state is represented. To generate the visualizations, we generate variations that result in visual differences, such as changing the appearances or locations of objects that don’t affect the underlying state. This requires a model trained on the images to identify which variations are meaningful and do affect the underlying state, and which are meaningless and should be ignored.

Our contributions are two-fold. First, we create datasets of four planning domains that allow for visualizations of the same state to appear visually different while retaining the same semantic meaning. To predict the underlying states, we train a contrastive learning model that learns to disregard the variations and focus only on semantically meaningful differences. Second, we propose aligning visualized states with the problem’s state space structure to boost prediction accuracy. We compare two methods for doing so: a greedy algorithm that only considers the top-n best predictions for each image, and Viterbi’s algorithm, a well-established algorithm for finding the most
likely sequence of ‘hidden’ states that resulted in a known sequence of observed ‘emissions’. The idea of aligning images with the state space is demonstrated in Figure 1. Our results show that these alignment methods can be used to correct errors in prediction models, significantly improving prediction accuracy.

2. Background

2.1. Planning State Spaces

Planning problems involve finding a sequence of actions that can achieve a specific goal when executed from an initial state. These problems are commonly described using the Planning Domain Definition Language (PDDL) [1]. There are two types of files within PDDL - one which specifies the domain, which provides action schema and object types, and one that specifies the problem, which provides the initial state and goal. States are represented as sets of fluents, which are binary facts that are considered true if the fluent is in the set representing the state, and false otherwise.

In this work we are interested in representing a planning domain and problem through its underlying state space. We formalize a planning state space as a tuple $P = (S, A, f)$ where $S$ is a finite and non-empty set of states, $A$ is the set of actions, $A(s) \subseteq A$ denotes the actions applicable in each state $s \in S$, and $f(a, s) \in S$ denotes a state transition function for all $s \in S$ and $a \in A(s)$. While the initial state and set of goal fluents is commonly included in this formalism, we exclude them here as they are not relevant in our setting. We consider traces to be a sequence of states $t = (s_1, s_2, \ldots, s_T)$ where each $s_i \in S$, and each successor state is the result of applying a single action in the previous state. I.e., $s_{i+1} = f(a, s_i)$ for some $a \in A(s_i)$. This provides us with a vital assumption that we can use to improve our predictions: for a sequential pair of states $(s_i, s_{i+1})$, there should be a valid action $a \in A(s_i)$ which when applied has a nonzero probability of transitioning to state $s_{i+1}$ from state $s_i$.

In this setting, it is not important to know which action takes place in between states, only that there is only a single action for the purpose of aligning a sequence of observations to a path in the graph.

2.2. Hidden Markov Models

A HMM is a statistical model that ‘generates’ an observable sequence $O = (o_1, o_2, \ldots, o_T)$ from an observation space $o_i \in Y$, where observations are emitted from a sequence of ‘hidden’ states $X = (x_1, x_2, \ldots, x_T)$ from a state space $x_j \in S$. It satisfies the Markov property, thus transitions are dependent only on the current state. State transitions occur according to state-transition probabilities, and each state ‘emits’ an observation according to its emission probability distribution. Only the emissions are observable, the sequence of hidden states $X$ are unknown. Through this model, we can reason about the sequence of hidden states $X$ given the sequence of emissions $O$. Viterbi’s algorithm is one such method for finding $X$ from $O$, through maximizing the probability of the states given the emissions, with prior knowledge on the state transition and emission dynamics.

2.3. Contrastive Learning

Contrastive learning is a framework that aims to represent similar instances in a similar way while keeping the representations different from those of instances that are not similar. In an image classification setting, each image is passed through a neural network to obtain a vector representation, which may be referred to as a latent vector or embedding. The objective of contrastive learning is to have embeddings of images which belong to the same category be similar, while being distinguishable from embeddings of images that are from different categories.
Figure 2. Contrastive learning aims to map images to vector representations, while ensuring that images that depict the same state are mapped to similar representations that are different from the representations of other states. Here the green, orange, and blue circles represent clusters of latent space that images of the same state should be mapped to, the white dotted arrows representing the points we wish to draw closer together. The aim is to reduce the distance between these points while maintaining the separation of clusters for each class, so that the representations are easily distinguishable from those of other classes.

To achieve this, SimCLR [2] uses a contrastive loss that pulls together embeddings that should be the same while pushing away embeddings that should be different. In the standard setting, the model includes a data augmentation module that creates multiple views of the same image, each with different transformations. Contrastive learning is typically used as a self-supervised method, where images do not have class labels. This means the model must create its own examples from the provided data through creating different views. Each image is treated as its own class, and combinations of crops, colour distortions, and Gaussian blur is applied to create the different views. This results in multiple versions of the same image that appear visually different, and the model must learn to represent these different views in similar manners. In our work, we do not augment the images we generate. Instead we consider the images to already be augmented through the randomized generation process, which we discuss further in Section 4. Thus, instead of treating different ‘views’ of the same image as a class, we treat different visualizations of the same state to be the same class.

3. Approach

3.1. Problem Formalization

We formalize this problem as a hidden Markov model. Given a sequence of observations $O = (o_1, o_2, \ldots, o_T)$ we want to find the sequence of states $X = (x_1, x_2, \ldots, x_T)$ with the highest probability of having generated the observations $O$. The problem formulation is shown in Figure 3. These observations are the images which we want to identify the underlying states of. The state transition probabilities are considered to all be uniform with respect to the other outgoing edges for a given state. That is, from each state we consider there to be an equal probability of transitioning to any other state reachable through a single action. The probability of a state $s_i$ emitting an observation $o_j$ is given by the deep learning model, which for an observation $o_j$ outputs a predicted probability distribution over all states $S$, where $p_{i,j}$ is the probability that observation $o_i$ was emitted by state $s_j$, and thus the probability that $x_i = s_j$. 
Figure 3. An interpretation of a hidden Markov model to represent planning domains. The sequence \((x_1, x_2, x_3, x_4, x_5)\) represents the transitions undergone between states, where each sequential state \(x_i\) is the result of taking an action \(a_{i-1}\) in the previous state \(x_{i-1}\). These underlying states are hidden as we must reason about them from the sequence of emitted observations \((o_1, o_2, o_3, o_4, o_5)\), which represent the images we observe representing the underlying state. We use a trained state prediction model to obtain the emission probabilities of the observations. This model outputs a probability distribution over all states for a given observation, giving the probability \(p_{i,j}\) of observation \(o_i\) being emitted from state \(x_j\) for all \(x_j \in S\).

### 3.2. State Representation Learning

First, we train a model to predict the state of a given image. Contrastive learning is a suitable technique because it allows us to provide examples of images that all represent the same state and contrast these with sets of images that represent different states. We use the implementation of SimCLR provided in solo-learn [3], with the default set parameters. The model is a ResNet-18 architecture with a single linear layer after it, where the final output has a neuron for every state. During training, the contrastive loss is computed based on the output after the ResNet and a ‘projector head’, which is two linear layers. Here, representations of similar states are pulled together while distancing from representations of different states. A linear classifier layer is also trained after the ResNet (before the projector head), with an output for every state. It is standard practice in contrastive learning to not use the full network for downstream tasks. The classification layer is trained through cross-entropy loss to activate for the state the image represents and deactivate for all others, and the weights of the ResNet are frozen while training the classifier. During the inference phase, we pass a single image through the model and obtain a scalar value for each state, indicating how strongly the network ‘fires’ for that state, the higher the value the stronger the network believes this is the underlying state. We acquire the probability distribution by taking this output after the linear layer, and normalizing all values to be non-negative and sum to 1.

### 3.3. Aligning Algorithms

We compare two alignment algorithms. The first is a greedy approach, presented in Section 3.3.1. The other is the Viterbi algorithm, a well-established algorithm for the observation decoding problem for HMMs, which we present in Section 3.3.2.

#### 3.3.1. Greedy Align

The greedy method constructs the full trace by finding edges in the graph one at a time, starting from the beginning of the trace. It begins by finding the best edge between the first two observations, taking into account the probabilities of both states. The general optimization problem for finding the best edge between two states is

\[
x_{i}, x_{i+1} = \arg \max_{j,k} p_{i,j} \times p_{i+1,k}.
\]
Here, $p_{i,j}$ is the probability of observation $o_i$ being emitted by state $s_j$, and we only consider the top $n$ most likely predictions for each observation. This is shown starting in line 4 of Algorithm 1, this procedure is called if the current state is unknown. A state is unknown if it is either the first state or an edge failed to be found in the previous step. Once we find an edge, we try and continue the path by selecting the next most likely state which continues the connected path

$$x_{i+1} = \arg \max_j p_{i,x_i} \times p_{i+1,j}.$$  

Here, $p_{i,x_i}$ is based on the previous selection, and again, only the top $n$ most likely predictions are considered for the next state. This is shown starting in line 18 of Algorithm 1. If this next step fails, we return to the “optimize between two states” procedure to find the most likely state between the next two. This breaks the chain but allows the method to recover from coming off-route.

---

**Algorithm 1: Greedy Trace Aligning Algorithm**

**Input:** Observations $O = (o_1, o_2, \ldots, o_T)$ planning problem $P = (S, A, f)$, for every observation $o_i \in O$, the top $n$ most likely states $P_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,n})$ where $p_{i,j} \in S$, and their probabilities $\Pi_i = (\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,n})$.

**Output:** Prediction sequence $X = (x_1, x_2, \ldots, x_T)$.

1. Mark $x_1, x_2, \ldots, x_T$ as unknown
2. for $i$ in { 1, 2, ..., T - 1 } do
3.   $p_{\text{max}} \leftarrow 0$
4.   if $x_i$ is marked as unknown then
5.     for $j$ in { 1, 2, ..., n } do
6.       for $k$ in { 1, 2, ..., n } do
7.         if $\exists a \in A$ such that $p_{i+1,k} = f(a, p_{i,j})$ and $\pi_{i,j} \cdot \pi_{i+1,k} > p_{\text{max}}$ then
8.           $x_i, x_{i+1} \leftarrow p_{i,j}, p_{i+1,k}$
9.           $p_{\text{max}} \leftarrow \pi_{i,j} \cdot \pi_{i+1,k}$
10.          Mark $x_i, x_{i+1}$ as known
11.         end if
12.       end for
13.     end for
14.     end if
15.     $x_i \leftarrow p_{i,1}$
16.     Mark $x_i$ as known
17. end if
18. else
19.   $p_{\text{max}} \leftarrow 0$
20.   for $j$ in { 1, 2, ..., n } do
21.     if $\exists a \in A$ such that $p_{i+1,j} = f(a, x_i)$ and $\pi_{i,x_i} \cdot \pi_{i+1,j} > p_{\text{max}}$ then
22.       $x_{i+1} \leftarrow \arg \max_j \pi_{i,x_i} \cdot \pi_{i+1,j}$
23.       $p_{\text{max}} \leftarrow \pi_{i,x_i} \cdot \pi_{i+1,j}$
24.       Mark $x_{i+1}$ as known
25.     end if
26.   end for
27. end if
28. end for
3.3.2. Viterbi Align

Viterbi's algorithm finds the most probable state sequence by finding the path that maximizes the joint probability of the observed sequence and the sequence of states. It constructs a table $T_1$ of size $K \times T$, where $T$ is the length of the trace and $K$ is the number of states. The entry $p_{i,j}$ of $T_1$ represents the joint probability of the most likely path up to predicting state $s_i$ for observation $o_j$. Thus, for each observation $o_j$, it keeps track of the most likely path up to every potential state we could predict for $o_j$. A second table $T_2$ the same size as $T_1$ tracks the states of the path selected up to each state for each observation, to allow backtracking to find the solution. The first column is filled with the probabilities for each state for the first observation, as returned by the model. For the next column, it finds the most probable edge for each state with a state from the previous column. This process is repeated until the last column, and the highest value in the last column of $T_1$ is then chosen as the best solution.

Algorithm 2: Viterbi Aligning Algorithm

Input: Observations $O = (o_1, o_2, \ldots, o_T)$ where $o_n \in Y$, planning problem $P = (S, A, f)$, and for every observation $o_i \in O$, the emission probabilities $\Pi_i = (\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,K})$ where $K$ is the number of states.

Output: Prediction sequence $X = (x_1, x_2, \ldots, x_T)$ where $x_n \in S$.

1: for $i$ in $\{1, 2, \ldots, K\}$ do
2: \hspace{1em} $p_{1,i} \leftarrow \pi_{1,i}$
3: \hspace{1em} $\hat{x}_{1,i} \leftarrow s_i$
4: end for
5: for $i$ in $\{2, 3, \ldots, T\}$ do
6: \hspace{1em} for $j$ in $\{1, 2, \ldots, K\}$ do
7: \hspace{2em} $p_{i,j} \leftarrow \max_k p_{i-1,k} \cdot \pi_{i,j}$
8: \hspace{2em} $\hat{x}_{i,j} \leftarrow s_k$
9: \hspace{1em} end for
10: end for
11: Let $x_T \leftarrow s_i$ where $i \leftarrow \arg \max_i p_{T,i}$ to obtain the last state, then obtain the solution by backtracking through $x_{k-1} \leftarrow \hat{x}_{k,x_k}$.

4. Dataset Creation

To achieve our goal of aligning visualizations of planning states to the underlying state space structure, we need a dataset that includes multiple examples of each state, which may appear visually different despite representing the same underlying semantic meaning. Although there are existing frameworks for visualizing planning states or plans [4–6], they generally do not include the variations in state representations that we desire. Our motivation is to learn from real-world observations, where the locations and appearances of objects can vary. As humans, we can easily identify that these differences may not make a difference semantically, but for computer vision approaches, this can be more challenging. The model must learn which variations do change the underlying state and which differences should be ignored. To simulate these visual variations in our state visualizations, we sample objects from the fashion-MNIST dataset [7]. This dataset consists of images of articles of clothing across 10 categories of clothing types. The dataset was proposed as a more challenging replacement for the MNIST handwritten digit dataset due to the increased complexity, better simulating modern computer vision tasks. Images of the same clothing type can vary greatly, some looking quite different from the rest. For example, a full-body picture of a
Figure 4. Examples of transitions between pairs of states, where each image on the bottom row is the result of applying an action in the state shown in the top row. From left to right the domains are Blocks World, Elevator, Grid, and Towers of Hanoi, and the actions applied respectively are: picking up a block, moving the elevator down one floor, moving the robot to the room to the left, and moving a disk (represented as an article of clothing to increase the difficulty of the domain) from one peg to another. The domains are described in further detail in Section 4.1. Objects are samples from fashion-MNIST, and appear in random locations in the rooms in the Elevator and Grid domains to increase variation in the generated images.

4.1. Domains

We have developed visualization methods for four different PDDL domains: Blocks World, Elevator, Grid, and Towers of Hanoi. For each domain, we introduce random variations that simulate the noisy and unstructured nature of the real world. We describe these domains and the sources of variation below.

**Blocks World.** In this domain a number of labeled blocks are stacked in columns. Blocks can be moved by first picking one up off the top of a column, and placing on top of an existing column of blocks, or on the table. The goal is to reach a specified arrangement of the stacked blocks. To add random variation, we sample the block’s appearance, each block is assigned a unique ‘type’ that is represented as a type of clothing and sampled from fashion MNIST. We test on a problem with 5 blocks and a problem with 6 blocks, resulting in state spaces with 866 states and 7,057 states, respectively.

**Elevator.** In this domain, passengers with origin and destinations floors are transported between floors by an elevator. The elevator moves between floors, and passengers can get on or off the elevator at the floor the elevator is presently located at. To simulate noise, we represent the passengers as distinct clothing types sampled from fashion-MNIST to add challenge to identifying each passenger. A passenger’s location on each floor or on the
Grid Hanoi Blocks Elevator

<table>
<thead>
<tr>
<th>Number of states</th>
<th>Grid</th>
<th>Hanoi</th>
<th>Blocks</th>
<th>Elevator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 accuracy</td>
<td>59.20%</td>
<td>52.78%</td>
<td>80.37%</td>
<td>76.21%</td>
</tr>
<tr>
<td>Top-5 accuracy</td>
<td>100.0%</td>
<td>100.0%</td>
<td>99.95%</td>
<td>99.55%</td>
</tr>
<tr>
<td>Greedy accuracy</td>
<td>94.04%</td>
<td>78.74%</td>
<td>100%</td>
<td>99.99%</td>
</tr>
<tr>
<td>Viterbi accuracy</td>
<td>99.01%</td>
<td>85.52%</td>
<td>100%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Greedy time</td>
<td>0.19s</td>
<td>0.44s</td>
<td>0.21s</td>
<td>0.64s</td>
</tr>
<tr>
<td>Viterbi time</td>
<td>169.81s</td>
<td>895.02s</td>
<td>390.57s</td>
<td>4810.96s</td>
</tr>
</tbody>
</table>

Table 1. Top-5 accuracy reports on how frequently the correct prediction was within the top-5 most likely predictions. For the Blocks, Hanoi, and Grid domains we report the accuracy obtained at a cutoff value of 10% of the state space size as the top-n.

5. Results

Results can be found in Table 1. We report top-1 and top-5 accuracy on the testing data, as well as the accuracy after being aligned through the greedy and Viterbi algorithms, along with the times of each algorithm. Note that the top-5 accuracy is not directly comparable with other accuracy measures as it only represents whether the correct prediction was within the set of the top-5 highest ranked predictions, while other measures report on the accuracy of a single selected value. The top-5 accuracy is valuable for understanding how close the correct prediction was to being chosen, even if it might not have actually been selected.
While the Viterbi algorithm performed better than the greedy algorithm in general, in many cases their performances were quite close. For example, on the Blocks domain with 7,057 states greedy had an accuracy of 99.94% while Viterbi had an accuracy of 99.96%, but with extremely significant differences in time. The greedy approach ran in under 2 seconds, while Viterbi’s algorithm took over 2.5 hours to run.

![Greedy Align Accuracy at different Top-n](image)

*Figure 5. This figure compares the accuracy of the greedy algorithm for different top-n values. Only the top-n predictions are considered for each observations when optimizing for the most likely valid transition. The first point shows the original top-1 accuracy before alignment, followed by the accuracy after the greedy algorithm for cutoff values of 5, 10, 10% of the size of the state space, 50% of the size, and with no cutoff.*

We also provide an analysis of which selection of top-n was best for the greedy align, shown in Figure 5. In this experiment we generated one set of traces. First we report the top-1 accuracy of the model simply predicting the states without any alignment applied. Then we apply the greedy algorithm at varying levels of top-n, only considering n points for each observation. To obtain these values, we rerun the algorithm over the same data at the various cutoff points. Interestingly, in some domains considering too many options could at times hurt performance. We hypothesize that since the greedy method is incentivized to choose one path and continue with it until no longer feasible, it may be prone to falling ‘off track’ and then not being able to recover. Within the state space their are various isomorphic structures. Consider the Grid domain, with an object in one room, and a sequence of states where the robot is moving between rooms. In the state space graph there would exist a similar structure for the object being placed in a different room, but having the agent still move between rooms in the same manner. These two structures would be isomorphic, the only difference would require correctly predicting which room the object is in. In the greedy approach, if an incorrect step occurs that places an object in an incorrect room, then it may reinforce this mistake by traversing the path that moves the agent in the correct manner, without correcting for the mistake of the incorrect item placement as it is more biased towards continuing the path over correcting for mistakes.

Furthermore, across many of the domains we see a high top-5 accuracy, indicating that the correct prediction can be generally found within the top-5 predictions. We believe allowing the greedy method to consider more possibilities for each observation can hurt as it can potentially encourage the greedy approach to continue the chain of incorrect predictions. When restricted to only considering the top-n predictions for each observation, if an observation that aligns with the graph isn’t found at a given step then it will ‘break the chain’, trying to optimize between the next two observations instead. For higher values of
top-\(n\), there are more ways to ‘continue the chain’ but these predictions may be ranked as less likely by the model. Since we can see that generally the correct predictions are ranked quite high, looking down to these lower predictions can result in a less accurate alignment.

The Elevator domain was one case where we did not observe a high top-5 accuracy, this was quite low at 15.04\%. From Figure 5 we observe that the correct predictions tended to be within the top-10\% of predictions, where accuracy substantially increased from 2.64\% using the top-10 predictions to 90.23\% with the top-10\%. In this domain, the top-10\% of predictions corresponds to the top-136 predictions. On this domain it does appear to increase in performance when considering more predictions, thus assuming that the correct prediction is within a certain threshold may not be a useful assumption in this case. Thus, the top-\(n\) parameter may benefit from being specifically tuned to each domain.

6. Related Work

Previous work has approached the problem of observation decoding for plan and goal recognition. In [9] the problem is formalized through classical planning and solved using a classical planner, rather than modelling it as an HMM as we do in out work. While HMMs can be limited by missing observations, the formalism in this work can handle this aspect. Our work differs in that we are focusing on empirically demonstrating how a system that connects images of real-world environments and a symbolic model of the world can be linked. This formalism is currently beyond the scope of our work, but it would be an interesting avenue to explore in future extensions.

In LatPlan [10], classical planning models are learned from image pairs demonstrating a valid transition in the domain. From these image pairs, the model learns symbolic representations of the states and action theories, as well as preconditions for actions and how to apply a given action to a state. PDDL planning models are then generated from these learned representations. This is in a different direction than our work, as we are not learning a model, but instead taking advantage of a provided model to correct and improve the predictions of a state classification model. The image sources in LatPlan are also less complex, as they only involve adding Gaussian noise to the images, while we consider more complex forms of variations between images representing the same state.

Bonet and Geffner [11] took an orthogonal approach where PDDL planning models are learned purely from the structure of the state space. This is done combinatorially through the use of a SAT solver to search for satisfying models. This approach raises interesting directions where one could attempt to learn the state space structure from the data we generated and then extract a PDDL model.

7. Conclusion and Future Work

In this work, we have presented a methodology to bridge the gap between noisy observations and symbolic planning models by creating a dataset which visualizes planning states with random variation, training a deep learning model to predict the states of these visualizations, and evaluating two algorithms for aligning the predictions with a provided state space structure. Our approach can greatly improve the prediction accuracy and compensate for errors in the deep learning model. However, we made the assumption that every sequential pair of images in a trace is the result of applying exactly one valid action. For future work, it would be interesting to extend this to sequences of data where sequential frames may either represent the same state or skip over intermediate transition states. The challenge would be to identify when the underlying state transitions from one to another, as well as account for missing observations. Overall, we believe our work lays a foundation for advancing planning models to be more robust and scalable in real-world applications.
Acknowledgements

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Vector Scholarship in Artificial Intelligence, provided through the Vector Institute.

References