Self-Supervised Representation Learning from Demonstration

by

Ercan Alp Serteli

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

July 13, 2021

Self-Supervised Representation Learning from Demonstration

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Ercan Alp Serteli

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by the final

examining committee have been made.

Committee Members:

Assist. Prof. Barış Akgün (Advisor)

Assist. Prof. Emre Uğur

Assoc. Prof. Aykut Erdem

Assoc. Prof. Erkut Erdem

Assist. Prof. Fatma Güney

Date:

ABSTRACT

Self-Supervised Representation Learning from Demonstration
Ercan Alp Serteli
Master of Science in Computer Science and Engineering
July 13, 2021

Given the growing demand for the application of robotics in an increasingly wide range of tasks and environments, robot learning as opposed to programming is getting more relevant by the day, since programming robots is tedious and usually only feasible in controlled domains. However, gathering data with robots is an expensive and time-consuming task, so the learning methods must cope with the limited amount of data available. When working with high dimensional perceptual data, learning low-dimensional, useful representations is a key aspect in dealing with the low-data setting. In this thesis, we develop a neural network architecture to learn perceptual representations from few human skill demonstrations in a self-supervised manner. The developed models take the sequentiality of the data and the low-data nature of the problem into account. These representations are used to learn perceptual goal models of the demonstrated skills. These models can monitor the learned skill executions and be used in reinforcement learning to generate reward signals, without explicit reward engineering. Our simulated and real robot evaluations with object manipulation skills show that the learned representations result in better goal models in terms of monitoring and reinforcement learning performance compared to generic dimensionality reduction methods. We further introduce transfer learning approaches in the context of learning from demonstration and show positive transfer between different objects for the same skill, between the same object for different skills under certain conditions, and also between different perceptual domains. Transferring knowledge as enabled by our modular neural architecture allows us to leverage existing data for continual learning into the future. Overall, we show that our proposed self-supervised representation learning architecture has the potential to improve learning from demonstration approaches with a perceptual component.

ÖZETÇE

Gösterimden Kendinden Denetimli Temsil Öğrenme
Ercan Alp Serteli
Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans
13 Temmuz 2021

Robotiğin giderek daha geniş bir görev ve ortam yelpazesinde uygulanmasına yönelik artan talep göz önüne alındığında, robot programlamanın aksine robot öğrenmesi gün geçtikçe daha önemli hale gelmektedir. Ancak robotlarla veri toplamak pahalı ve zaman alıcı bir iştir, bu nedenle öğrenme yöntemleri sınırlı miktarda veri ile çalışmak durumundadır. Yüksek boyutlu algısal verilerle çalışırken, düşük boyutlu, kullanışlı temsiller öğrenmek, düşük veri ortamının çıkardığı zorluklarla başa çıkmada önemli bir husustur. Bu tezde, kendi kendini denetleyen bir yapıda az sayıda insan beceri gösteriminden algısal temsiller öğrenen bir sinir ağı mimarisi geliştiriyoruz. Geliştirilen modeller, verilerin sıralılığını ve problemin düşük veri yapısını dikkate almaktadır. Öğrenilen temsiller, gösterilen becerilerin algısal hedef modellerini öğrenmek için kullanılır. Bu modeller, öğrenilen beceri yürütmelerini izleyebilir (başarılı/başarısız şeklinde değerlendirebilir) ve ödül mühendisliği olmadan ödül sinyali oluşturmak için pekiştirmeli öğrenmede kullanılabilir. Nesne tabanlı manipülasyon becerileri üzerine yaptığımız simüle edilmiş ve gerçek robot değerlendirmeleri, öğrenilen temsillerin, genel boyutluluk azaltma yöntemlerine kıyasla izleme performansı ve pekiştirmeli öğrenme performansı açısından daha iyi hedef modelleri ile sonuçlandığını göstermektedir. Bu çalışmada ek olarak, gösterimlerden öğrenme bağlamında aktarımlı öğrenme yaklaşımları tanıtıyoruz ve aynı beceri için farklı nesneler arasında, belirli koşullar altında aynı nesne için farklı beceriler arasında ve ayrıca farklı görsel girdi uzayları arasında pozitif aktarım gözlemlendiğini gösteriyoruz. Modüler sinirsel mimarimizin mümkün kıldığı bilgi aktarımı teknikleri, geleceğe yönelik öğrenme için mevcut verilerden yararlanarak veri gereksinimini daha da azaltmamızı sağlamaktadır.

ACKNOWLEDGMENTS

This thesis would not be possible without the help and support of many individuals.

This work is funded by the Scientific and Technological Research Council of Turkey (TUBITAK) with the project number 117E949. I gratefully acknowledge the funding received from TUBITAK, and the funding received from KUIS AI Lab towards my master's degree.

I would like to thank and acknowledge my advisor Barış Akgün for his endless support and encouragement, and his sharing of insightful knowledge throughout my master's education.

I would also like to thank my labmates Farzin Negahbani and Onur Berk Töre for offering me a helping hand whenever I needed it.

I thank my mother for continuing to supply me with emotional and edible support, despite great distances.

Finally, I thank my cat Ponpon for always being there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

LfD	Learning from Demonstrations
RL	Reinforcement Learning
PCA	Principal Component Analysis
UMAP	Universal Manifold Approximation and Projection
HMM	Hidden Markov Model
PCAE	Point Cloud Auto Encoder
DMP	Dynamic Movement Primitives
CNN	Convolutional Neural Network
MLP	Multilayer Perceptron
RNN	Recurrent Neural Network
NSP	Next State Predictor
LSTM	Long-Short Term Memory
GRU	Gated Recurrent Unit
VAE	Variational Autoencoder
MSE	Mean Squared Error
VFH	Viewpoint Feature Histogram
GASD	Globally Aligned Spatial Distribution

Chapter 1

INTRODUCTION

With the advancements in hardware, electronics, computation power and artificial intelligence along with affordability, the adoption of robots is becoming increasingly widespread across a large range of environments. There is an increasing demand for robots to perform a plethora of tasks and interact with a broad spectrum of people, encompassing expert and non-expert users alike. Programming robots is difficult and only feasible in controlled environments or for a narrow range of tasks. It requires significant expertise, time, and effort. Therefore, we cannot expect robots to be useful in a large variety of real world settings with uncertain and dynamic environments, and everyday end-users without adaptation and learning. Learning from demonstration (LfD), which involves teaching the robot a skill by performing it as a human teacher, evolved from this motivation. An example LfD interaction can be seen in Figure 1.1. In addition to learning from human teachers, LfD also provides a good starting point for reinforcement learning (RL), where the robot improves the task by itself, especially useful under uncertainty.

Having access to a limited amount of data is a common problem in many real-world robotics learning settings. Gathering data from robots is a time-consuming and expensive task. Therefore, robot learning methods either need to utilize simulators or be designed to deal with limited data. The former is not straightforward for LfD settings. The latter is especially salient for LfD since the patience of an average user is an important limiting factor, considering the variety of tasks expected from robots. In this work, we concentrate on making the most out of limited user demonstrations.

We build our approach on an existing LfD framework which learns action models

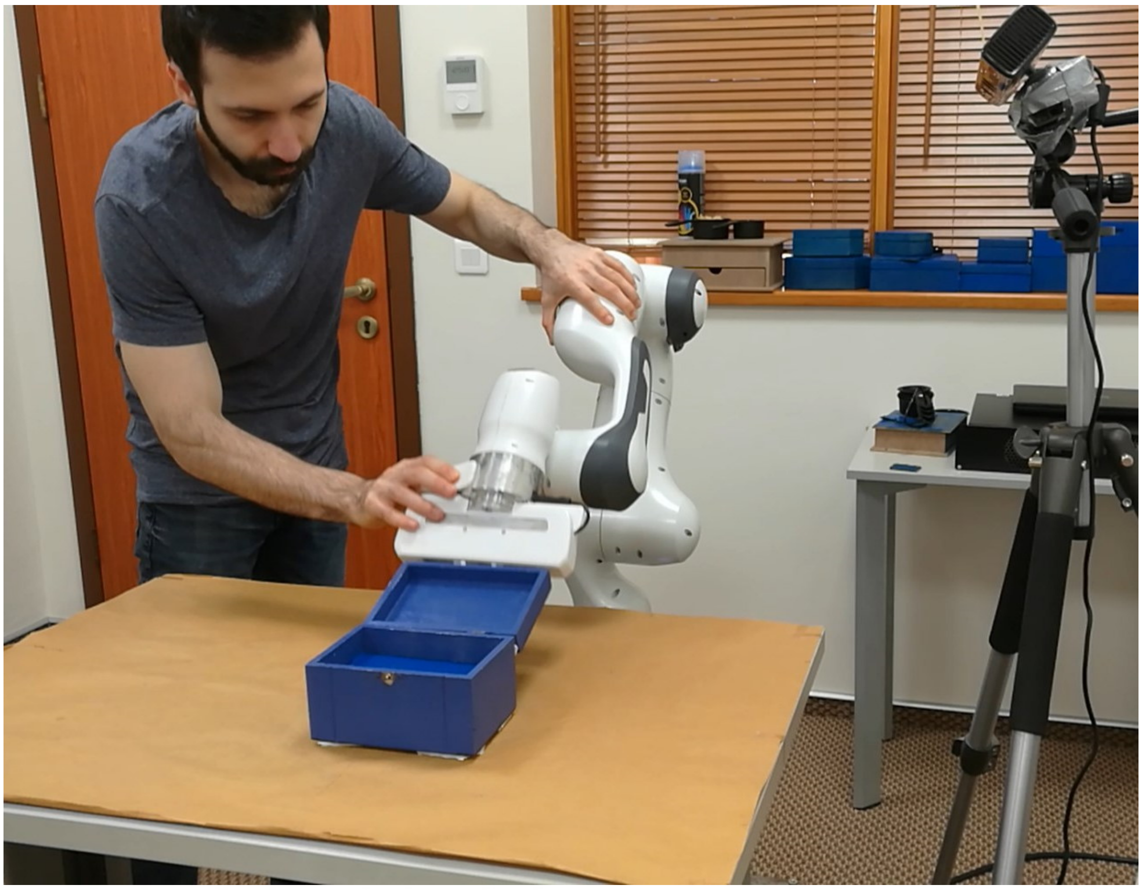


Figure 1.1: A teacher kinesthetically demonstrating the closing skill on a box.

and goal models from demonstrations. The action model is learned from robot kinematic data and is used to execute the learned skills. The goal model is learned from object related perceptual data, and is used to monitor the success/failure of robot executions (deciding whether the robot succeeded in reaching the goal of the skill or not) [Akgun and Thomaz, 2016]. The goal models can be used to get reward signals for reinforcement learning as well, without explicit reward function programming and can be used to improve initially unsuccessful action models [Akgun and Thomaz, 2015] [Eteke et al., 2020].

Monitoring and reward learning requires effective goal models but the high-dimensional perceptual data consisting of only a few user demonstrations makes it a challenging endeavour. Developing models with appropriate inductive bias and using low-dimensional data are common approaches to problems involving low amount of data. In this work, we develop a neural network architecture for self-supervised learning of low-dimensional representations that will lead to effective goal models learned from a small number of sequential data (i.e. demonstrations). In order to work with a low amount of data, pieces of the proposed architecture are kept as simple as possible while retaining effectiveness.

Transfer learning is another approach to deal with low amount of data. Transferring information between skills and/or objects is an interesting challenge in LfD. Representation learning with a neural network architecture provides potential for such transfer learning. Thus, we introduce suitable transfer learning approaches for our architecture within the context of LfD.

We evaluate our proposed model on a LfD setting involving real-world object manipulation skills. We specifically look at execution monitoring and reinforcement learning performances. Our results suggest that our tailor-made approach performs better than principal component analysis (PCA) and uniform manifold approximation and projection (UMAP). Qualitative analysis also shows that our learned representations retain sequentiality information. We further show positive transfer under certain conditions with very few target data.

To summarize, we develop a neural network architecture for perceptual representation learning from few demonstrations. This is used to learn goal models which have higher monitoring and reinforcement learning performances and also used to facilitate transfer between demonstrated skills. The small parts of our architecture are relatively simple, however, their combination, application and obtained results are novel to the best of authors' knowledge.

The contributions of this work can be summarized as:

1. A neural-network architecture for learning low-dimensional representations from demonstration data

2. Application of the proposed architecture in learning a goal model

   (a) Achieving an improved monitoring performance with the goal model with respect to off the shelf dimensionality reduction methods

   (b) Achieving an improved RL performance with the learnt rewards from the goal model with respect to off the shelf dimensionality reduction methods

3. Leveraging the proposed architecture to facilitate positive transfer learning (especially under low data conditions)

   (a) Methods for transfer learning between skills and objects

   (b) A method for transfer learning between perceptual domains

All of the contributions are evaluated in a real-world robotics setup.

1.1 Thesis Statement

A purpose designed neural network architecture for self-supervised representation learning from few demonstrations leads to better perceptual goal models, compared to off-the-shelf dimensionality reduction methods, resulting in better monitoring performance and reinforcement learning performance. The neural network based model further facilitates positive transfer learning for multiple scenarios under appropriate circumstances.

1.2 Organization of the Thesis

In the next chapter, we give an overview of related literature and how this work fits into the picture. Then in Chapter 4, we present a detailed description of the proposed model including its various modules and loss function components. After that, in Chapter 5 we describe the setup in which the experiments are conducted, and the demonstration dataset used in those experiments. The next three chapters consist of our experimental evaluations and discussions of their results. Chapter 6 includes the monitoring experiment and further analyses involving monitoring performance such as different input features, an ablation study and a qualitative analysis of the latent representations. Chapter 7 contains the details and results of the reinforcement learning experiment. Chapter 8 introduces procedures and evaluates the results for three separate transfer learning settings, using the proposed model. Finally in Chapter 9, we conclude by summarizing our work, discussing the results, and showing some pointers for potential future work.

Chapter 2

LITERATURE REVIEW

2.1 Unsupervised Representation Learning

The main problem tackled in this work is to learn useful low dimensional representations of the perceptual features of an object being manipulated. Since there are no labels available, this problem can be categorized as unsupervised representation learning which has recently shown significant growth in various problem domains.

Most unsupervised techniques today fall under the category of self-supervised learning. Self-supervised learning refers to approaches where a supervision signal is automatically generated from the input, as opposed to supervised learning where data needs to be labeled manually. This applies to our proposed model as well. On the other hand, various clustering, embedding and most dimensionality reduction methods can be thought of as unsupervised approaches that do not make use of self-supervision.

The skill-specific feature extraction step in [Eteke et al., 2020] is done using Principal Component Analysis (PCA). PCA is a very commonly used unsupervised method of dimensionality reduction where the data is projected onto a small number of basis vectors while keeping as much of the variance explained as possible. We propose a learning a neural network model as the skill-specific feature extractor in place of PCA. When evaluating our proposed model, we compare it with not only PCA, but also Uniform Manifold Approximation and Projection (UMAP) [McInnes et al., 2018]. While PCA applies a linear transformation to the data, UMAP is a non-linear embedding method. Since our proposed network model contains non-linear activation between layers, it makes sense to include a non-linear method in the comparisons.

2.2 Self-Supervised Representation Learning

Self-supervised learning techniques work by defining a problem to be solved, making use of only the unlabeled input. The assumption is that solving the self-imposed problem requires the model to learn useful representations of the data.

2.2.1 Autoencoders

Perhaps the most direct problem to impose is to make the model learn to copy its input to its output, resulting in what is called an autoencoder. Autoencoders can be forced to capture salient information by keeping the code dimension smaller than the input dimension [Goodfellow et al., 2016]. Different types of autoencoders include sparse autoencoders which employ a penalty for keeping the code sparse [Ranzato et al., 2007b] [Ranzato et al., 2007a], denoising autoencoders that try to reconstruct the original data from a corrupted input [Vincent et al., 2008], among many others.

In our framework, as explained in Section 3, there is a two step procedure for obtaining representations. The first step of skill-agnostic feature extraction is due to not having enough data to learn directly from high-dimensional sensor input. In order to extract features from point cloud data, we use a point cloud autoencoder (PCAE) [Achlioptas et al., 2018] trained on the ShapeNet dataset [Chang et al., 2015]. Since points in a point cloud are unstructured and permutation invariant, standard convolutional neural networks (CNNs) or multilayer perceptrons (MLPs) are not suitable in this task. The authors of [Achlioptas et al., 2018] use 1D convolutional layers with kernel size 1 to encode each point independently in their encoder structure. There are other approaches to learning point cloud representations with autoencoders, such as FoldingNet [Yang et al., 2018] and Capsule Networks [Zhao et al., 2019], which could also be used for the skill-agnostic step in our framework.

2.2.2 Self-Supervised Representation Learning in Images

Many self-supervised representation learning methods make use of various predictive, contrastive, adversarial and generative techniques, and data augmentation methods to learn useful features. Some methods use spatial context in images to predict which direction an image patch is cut from [Doersch et al., 2015] or to solve a jigsaw puzzle made from the image [Noroozi and Favaro, 2016]. Other approaches for image data include predicting image rotations [Gidaris et al., 2018], filling holes in images [Pathak et al., 2016], colorizing gray scale images [Larsson et al., 2017], or a combination of such tasks [Doersch and Zisserman, 2017]. On the other hand, [Makhzani et al., 2015] and [Donahue et al., 2016] utilize generative adversarial networks for representation learning. Another method that makes use of adversarial learning is [Hjelm et al., 2018], where they maximize mutual information between the input and its representations. Another family of frequently used methods is contrastive learning. The idea with contrastive learning is to create negative samples in addition to positive samples, in order to push apart negative samples while pulling together positive samples in the representation space. In the context of self-supervised learning, this is usually achieved by employing data augmentation techniques to create altered versions of a data point, which constitute the positive samples, while pairing differing data points together to create the negative samples. MoCo [He et al., 2020] and SimCLR [Chen et al., 2020] are popular methods in the image domain while CPC [Oord et al., 2018] presents a more universally applicable method.

2.2.3 Self-Supervised Representation Learning with Sequential Data

The data we are dealing with in this work are demonstrations. Demonstrations are sequences of goal (perceptual features) and action (manipulator poses) data. Our focus is on learning useful and low dimensional representations of the perceptual data with the small amount of available data, such that a good goal model can be fit on those representations. Since the data is made of sequences, it makes sense to look into sequence modeling. Recurrent Neural Networks (RNNs) are the most well-known family of neural architectures for modeling sequences. RNNs make use of recurrent connections to share parameters across time steps and process sequences of arbitrary length. We utilize a simple RNN structure named Elman RNN as part of our proposed model as well. Other well known RNN structures include long-short term memory (LSTM) and gated recurrent unit (GRU). We tested and failed to see an empirical advantage in using such gated RNNs in our particular task, and decided to proceed with the simpler option.

Various sequential domains of interest include text, audio, and many kinds of time series. Most of these domains differ substantially from ours in terms of dimensionality, sequence length, periodicity, relationship with time and the amount of data, therefore methods designed for them do not apply well to our problems.

When it comes to representation learning in sequential data, one task is to produce a single feature vector that represents the entire sequence, whereas a different task is to produce a feature vector for each element of the sequence. Examples of the former task in settings similar to ours can be seen in methods involving skill embeddings, where an entire sequence of experience or demonstration data is mapped to a point in a learnt latent space [Hausman et al., 2018] [Sharma et al., 2019] [Lynch et al., 2020]. However in the scope of this work, we are interested in the latter task due to the framework that we decide to work in. In this way, it is possible ignore the fact that we are dealing with sequences entirely and model each frame of a demonstration as if it was independent of the rest. In fact, that is what we do when using PCA for this purpose. However, this may result in losing on the information in the inherent dependencies between frames of the same demonstration. In our model, we make use of this information by employing a prediction task. Predicting the future is a natural task when dealing with sequences.

2.2.4 Predictive Approaches

One field that engages in prediction with sequences is video prediction, where the aim is to predict the future frames in a video given the past. A detailed review can be found in [Oprea et al., 2020]. Using video frame prediction to achieve a better understanding of objects in the scene is inspired by the ”predictive coding” concept from the neuroscience literature [Lotter et al., 2016]. There is large number of work done on modeling objects and dynamics through video prediction [Finn et al., 2016] [Minderer et al., 2019]. Other approaches for doing self-supervised learning from video include motion based segmentation [Pathak et al., 2017], sequential verification of frames [Misra et al., 2016], sorting shuffled frames [Lee et al., 2017], and many more. The difference of videos from our domain is that video data is very dense, high dimensional and in a grid structure. Learning in such high dimensionality requires a large amount of data, and as such video datasets often consist of millions of frames. Video models make use of CNNs to efficiently work with the structured frames. On the other hand, we need to deal with human demonstrations on specific objects, which is completely infeasible to collect in large numbers. In the case of keyframe demonstrations, a 30 second demonstration can consist of 3 to 8 keyframes, which is much sparser than a video with 30 frames per second. We also normally use point cloud data for the geometric information otherwise unavailable, to make the the most out of the available data. For these reasons, rather than working on the data directly, we use the lower dimensional encoded representations from a PCAE model.

2.2.5 Stochastic Approaches

While some predictive methods use a deterministic architecture, others utilize stochastic approaches to better deal with the inherent uncertainty of predicting the future, taking a more generative approach. These methods make use of stochastic variational inference [Hoffman et al., 2013] in a way similar to variational autoencoders (VAEs). VAEs are a kind of likelihood based generative models with stochastic latent variables [Kingma and Welling, 2014]. [Bayer and Osendorfer, 2014] and [Chung et al., 2015] extend VAEs to recurrent networks working in sequences. [Krishnan et al., 2015] utilizes these kinds of models to do counterfactual inference. [Watter et al., 2015] shows their application in learning dynamics models for complex control problems. [Fraccaro et al., 2016] improves on previous work by separating the deterministic and stochastic pieces to make posterior inference more efficient while [Goyal et al., 2017] adds a backward RNN and an auxiliary cost to improve the usefulness of latent variables, and [Serban et al., 2017] presents a hierarchical architecture that reuses the sampled latent variable at consecutive time steps. While these methods are often applied in low-dimensional sequence domains, stochastic approaches are prevalent in video prediction as well [Babaeizadeh et al., 2017] [Lee et al., 2018] [Franceschi et al., 2020] [Kaiser et al., 2019]. [Hafner et al., 2019] solves control tasks using planning in latent space by learning a dynamics model from pixels. Similarly, [Ha and Schmidhuber, 2018] learns a generative recurrent world model and uses it for not just planning, but also for training. More work on generative world models include [Buesing et al., 2018] and [Gregor et al., 2019], both of which along with [Hafner et al., 2019] utilize temporal abstraction, which is the idea of predicting states that are an arbitrary amount of time into the future instead of doing sequential rollouts. Temporal abstraction is introduced in [Neitz et al., 2018] and included as part of [Gregor and Besse, 2019] as well. In another direction, [van den Oord et al., 2017] learns discrete latent variables, as opposed to continuous, to circumvent issues of posterior collapse with autoregressive decoders, in addition to other work such as [Fortuin et al., 2019] [Kaiser and Bengio, 2018] [Chung et al., 2020]. [Kaiser et al., 2019] uses a video prediction based world model to do model-based reinforcement learning in Atari games. They find that a stochastic model with discrete latents performs better than only a stochastic model or a deterministic model with discrete latents.

While there is a vast collection of work done on stochastic latents and generative environment models, in our work we decided to employ a deterministic model. Variational models can generate diverse, plausible looking futures, but we do not perform planning or do model-based RL in the latent space, so we do not necessarily need that. Next state prediction in our case is mostly a task imposed to assist learning useful representations. Also, none of the aforementioned methods are applied in a real world setting, since virtual settings are much more convenient for generating the amount of data required to feed such models. We stick with the simpler architecture as it makes more sense in our data constrained setting.

2.2.6 Contrastive Approaches

Contrastive approaches show promise in learning world state representations as well [Anand et al., 2019] [Srinivas et al., 2020]. The authors of [Kipf et al., 2020] use a contrastive approach to learn a structured world model as a graph neural network, eliminating the need for prediction. They state that prediction at the perceptual level can hinder the model by forcing it to waste capacity learning irrelevant details and miss small but crucial information. We employ the same kind of negative hinge loss that they use in our proposed model. This will be explained more clearly in the later chapters. Another work that focuses on representation learning without reconstruction is [Ghosh et al., 2019], where they use a goal-conditioned policy to explore and train to learn actionable representations, those that are useful for decision making. Many works such as this make use of RL as part of learning representations [Kaiser et al., 2019] [Zhang et al., 2018] [François-Lavet et al., 2019].

2.2.7 Attention and Other Approaches

One prevalent approach for sequence learning is the use of attention mechanisms. As opposed to predicting the next (or the previous) step from the current step in a sequence, attention allows the model to access any possibly relevant step in the sequence for contextual information. Transformers [Vaswani et al., 2017] are a type of model utilizing self-attention mechanisms that became very successful in the field of language modeling. BERT [Devlin et al., 2018] introduces a method of pre-training transformer models using a masked reconstruction task. Masked reconstruction is proposed on domains other than text as well, such as speech [Wang et al., 2020] or human activity [Haresamudram et al., 2020]. [Bai et al., 2020] utilizes it as well, but in addition maximizes a mutual information estimate specialized for sequence data between the input and latent representations.

While there are many approaches to self-supervised sequential representation learning, we found a sequence prediction task with an additional contrastive component to be the most appropriate for the domain and problem that we are tackling in this work. There may be other promising techniques in the literature, but we leave it to future work for a more extensive search on such approaches.

2.3 Transfer Learning

One important aspect we focus in this work is transfer learning. Refer to [Zhuang et al., 2021] for a detailed survey on transfer learning techniques. [Yan et al., 2019] demonstrates that disentangling the representation and transition models can allow transfer learning between environments. We use a similar disentangled structure in our proposed model. While they utilize a cycle consistency loss to adapt between different looking environments in a game, we use other kinds of methods for transfer learning between skills, objects and sensor modalities. Similarly, [François-Lavet et al., 2019] shows that a modular structure allows transfer learning by retraining the encoder component to fit the same representations for target observations via a mean-square error (MSE) loss, which is a technique we make use of as well (see Section 8.3). In a similar way, [Zhang et al., 2018] presents a decoupled structure that enables transfer learning. They train a dynamics model with loss components resembling ours, using data gathered with a random policy in a similar way to how we use demonstration data. They describe a way of transfer by reusing encoder-decoder pairs in a way akin to one of our methods (see Section 8.1). [Devin et al., 2017] use modular policy networks with a task-specific and a robot-specific module in order to transfer to new robot-task combinations by mixing and matching previously learned modules. [Fitzgerald et al., 2021] details a taxonomy of transfer learning problems at different levels of abstraction in a robotic object manipulation setting similar to ours. The easy problem of object displacement as described in that paper is solved by using actions with respect to the object in our case, while object replacement corresponds to the transfer problem we tackle in Section 8.2. On a different note, [Pertsch et al., 2020] proposes a method of efficiently transferring knowledge to new tasks by extracting skills and simultaneously learning skill embeddings and priors from an unstructured dataset of past experience. However, their focus is on making efficient use of a large amount of unlabeled experience data whereas we focus on learning from a small set of demonstration data on predefined skills. The details related to our methods will become clear in the later chapters.

2.4 Conclusion

In this chapter, we discussed the past and ongoing work in the literature relevant to our work, including various unsupervised/self-supervised representation learning approaches and transfer learning. We illustrated the relationships between different branches of the literature, and how our work fits into the overall picture and connects with related work.

Chapter 3

THE LEARNING FROM DEMONSTRATIONS FRAMEWORK

Our goal in this work is to improve the general performance of LfD systems by introducing a self-supervised representation learning approach along with a transfer learning component. The idea is to learn goal models to achieve better execution monitoring performance, get better reward models only using the demonstrations (vs hand crafting) and to facilitate effective transfer learning in various settings. In this section, we are going to describe the LfD setting and the LfD framework we build our work atop and as such we do not claim any novelty about the information we present in this section. We refer the reader to [Akgun and Thomaz, 2016, Eteke et al., 2020] for further details.

3.1 Summary of the Framework

The LfD framework we use learns an action and a goal model from the demonstrations of a skill [Akgun and Thomaz, 2016]. An action model learns the path that the end-effector follows to perform the skill. A goal model learns how the object of interest changes throughout the performed skill. Action model is used for executing the skill while goal model is used for monitoring the success of the execution. Uses of monitoring include error recovery or asking for help in case of failure during execution, or moving on to the next task in case of success [Akgun and Thomaz, 2016]. It has been observed that non-expert teachers act in a goal oriented manner during teaching, and as such manage to teach successful goal models, but not very good action models [Akgun et al., 2012, Akgun and Thomaz, 2016]. The successfully learned goal model can be used to improve the action model [Akgun and Thomaz, 2015]. This can be done with the help of the binary success signal from monitoring, but an even more effective method is to learn dense rewards from the goal model and do reinforcement learning (RL) via policy search [Eteke et al., 2020]. Our purpose in this work is to learn a better goal model in order to improve its monitoring and reinforcement learning performances.

We focus on learning table-top manipulation skills with a single object of interest, such as closing a box depicted in Figure 1.1. Even though this excludes dynamics skills such as pole balancing, we argue that it encompasses a large set of everyday tasks where the main idea is to change the state of an object.

3.2 Demonstrations and Collected Data

In LfD, skills are performed by human teachers. This can be done in various ways, such as in a kinesthetic manner by guiding the robot physically through the movements (Figure 1.1), perceptually recording the human perform the task and corresponding human joints, through operating the robot with a controller or an exoskeleton etc. While different techniques have their advantages and disadvantages, in this work we utilize kinesthetic teaching. We collect both kinematic data and raw perceptual data during demonstrations.

3.2.1 Keyframe and Continuous Demonstrations

One way to collect data during demonstrations is to record the entire kinematic and perceptual data as a continuous stream with a constant frame-rate, called continuous or trajectory demonstrations. Another way is to manually set points considered salient in the skill during teaching and record those keyframes, called keyframe demonstrations [Akgun et al., 2012]. Some skills can be more suitable for continuous demonstrations, especially those with gradual movement and continuous changes in the affected object. On the other hand, giving keyframe demonstrations can be easier as a result of not having to perform the task in one complete motion, but being able to divide the task in pieces and move at the teacher's desired pace. Since either method has its own advantages, we focus on being capable of working with both options.

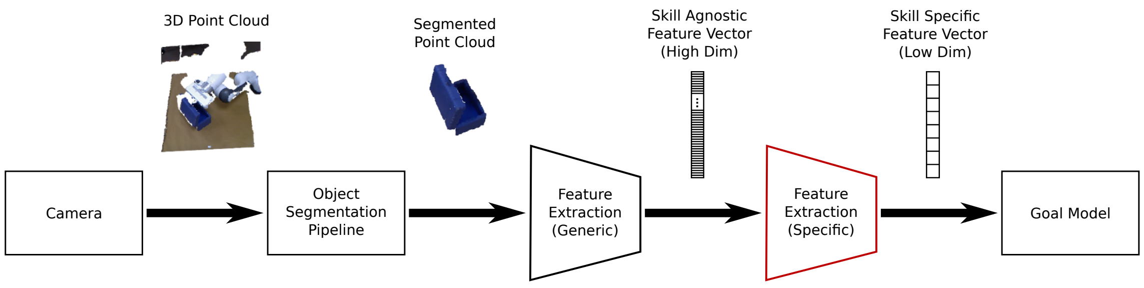

3D Point Cloud
Segmented Point Cloud
Skill Agnostic Feature Vector (High Dim)
Skill Specific Feature Vector (Low Dim)
Camera
Object Segmentation Pipeline
Feature Extraction (Generic)
Feature Extraction (Specific)
Goal Model

Figure 3.1: The workflow from sensor to goal model. The generic feature extraction can be done by any point cloud feature extraction method. In this work, we utilize a pretrained point cloud autoencoder for that purpose. For the skill-specific feature extraction stage, we use our proposed model and compare it with PCA and UMAP in the evaluations.

3.2.2 Collected Data

The kinematic data is the joint angles. The raw perceptual data can be entire colored point clouds (RGBD data), segmented point clouds or extracted generic features (see Section 3.3). The multiplicity of options for perceptual data stems from the fact that it is not feasible to record high-dimensional data with a high frequency during trajectory demonstrations. Both of these data types are processed further before being used to learn action and goal models as described next.

3.3 Goal Model

We use Hidden Markov Models (HMMs) with Gaussian emission distributions as our goal models. The generative and probabilistic nature of HMMs allows us to use the goal models for monitoring skill executions and to create reward signals.

The main part of the goal model for the purposes of this work is the emission space which we aim to learn representations for. The goal models are learned from perceptual data obtained from demonstration point clouds[1]. The raw perceptual data is high-dimensional which makes it difficult to learn from.

[1]Another vision sensor would also work but the described LfD framework implementation relies on RGBD cameras.

3.3.1 Skill-Agnostic and Skill-Specific Feature Extraction

In order to make the learning problem more manageable, some form of dimension reduction is required. In both supervised and unsupervised settings, implicitly learned features from neural architectures generally outperform classical feature extraction methods. While the authors of [Akgun and Thomaz, 2016] and [Akgun and Thomaz, 2015] use hand crafted feature extraction techniques, [Eteke et al., 2020] employs a two phase approach consisting of a skill-agnostic step and a skill-specific step to improve the extracted features. In the skill-agnostic phase, they train a deep neural point cloud auto encoder (PCAE) model [Achlioptas et al., 2018] on a pre-existing object point cloud dataset [Chang et al., 2015] and use its encoder to get a 128 dimensional feature representation. Then, in the skill-specific phase, they use Principal Component Analysis (PCA) to further reduce the general perceptual features obtained from the encoder down to 8 dimensions. A different PCA is fitted for each skill, using only the demonstrations for that skill. The abstraction of this workflow is as depicted in Figure 3.1.

In this work, we also use the PCAE model with 128 dimensional output as our skill-agnostic feature extraction step. However, our method can use any relevant input space (Generic Feature Extraction step in Figure 3.1), and as such we experiment on 3D point cloud feature extraction methods such as VFH [Rusu et al., 2010] and GASD [Lima and Teichrieb, 2016], and a pretrained 2D deep CNN architecture ResNet [He et al., 2015]. In the case of ResNet, we use the 2048 dimensional output right before the final classification layer of a 101-layer ResNet model pretrained on ImageNet [Deng et al., 2009]. For ResNet's input, we project unsegmented colored point clouds down to 2D RGB images. On top of this skill agnostic feature extraction, we design a neural network architecture that can learn better skill-specific features (Specific Feature Extraction step in Figure 3.1), to be used instead of PCA generated features, in order to achieve better goal models and facilitate transfer.

The HMMs are learned from the skill-specific features, wherever they may come from. The emission covariance matrices are taken to be diagonal. The number of hidden states is determined using the Akaike information criterion. The HMMs are trained using the Baum-Welch Algorithm with the training demonstrations.

3.3.2 Monitoring Method

For monitoring, we extract the most likely hidden state sequence and calculate the log likelihood of execution observations. These observations are actually the skill specific features we extract from the point cloud data. If an execution's final hidden state has not been seen as one of the terminal states during training, it is deemed to have failed. However, an execution may still be a failure with the correct final state. For this, a threshold is calculated that can separate the log likelihoods of training sequences and their order-reversed versions. This threshold is used as the separation point in log likelihoods between success and failure. The idea with reversing the order of a recorded demonstration is that it will be a negative example for that skill, which holds true given that the skill is not symmetric. A negative example is expected to have a low likelihood of happening according to the goal model. The threshold is chosen as the middle point of two log likelihoods that when separated result in the highest accuracy. We use the Viterbi algorithm to find the most likely hidden state sequence and the Forward algorithm to calculate the log likelihood.

3.3.3 Reward Learning from Goal Model

The reward learning idea is to convert the goal HMM into a Markov Reward Process (MRP) by giving a positive reward to the terminal hidden states and using a Monte Carlo Approach to calculate the values of the remaining hidden states, utilizing the transition probabilities. The MRP and its values, and the HMM emissions are then used to define a Partially Observable MRP (POMRP) which is used to calculate the episode returns for reinforcement learning.

3.4 Action model

Action model is responsible for executing the skill and is learned from the the robot's end-effector (eef) pose with respect to the object. This action data is represented as a 3D vector for translation and a 4D unit quaternion for rotation. The joint angles recorded during the demonstrations are converted to eef poses with forward kinematics. The object 2D pose (2D translation and 1D rotation wrt table normal) with respect to the camera frame is calculated by the segmentation pipeline. Then the calibration between the robot base and the camera is used to convert eef poses to the object frame.

3.4.1 Continuous vs Keyframe Settings

In the continuous setting, Dynamic Movement Primitives (DMPs) [Ijspeert et al., 2002] are used to model the actions, as in [Eteke et al., 2020]. In the keyframe setting, we use HMMs for action modeling instead, as in [Akgun and Thomaz, 2016]. We get eef trajectories by either random sampling or taking the most likely sequence and use splines between the sampled points. The obtained trajectories, either with the DMP or the HMM, are transformed to the robot base frame, and the joint angles are attained via inverse kinematics which are then used to execute the skill.

Both the DMP parameters and HMM emissions can be updated with reinforcement learning using the learned reward model described in Section 3.3 using the method described in [Eteke et al., 2020]. One of our goals in this work is to attain a better reward model by learning a better goal model.

3.5 Conclusion

In this chapter, we described the relevant background information regarding the LfD framework we are working in. We gave details about the data collection, and explained the action and goal models that make up the LfD framework in sufficient detail. While this work is motivated by LfD and this framework is where we chose to perform our evaluations in, the methods introduced in this work are ultimately not limited to a specific framework or LfD in general.

Chapter 4

METHOD

Our purpose is to obtain a low-dimensional representation space to learn better goal models with improved monitoring performance, leading to better rewards for improved reinforcement learning performance, and to leverage the learned representations for transfer learning in various settings.

4.1 Architecture

The proposed model learns to represent the input domain in a low dimensional latent space, and to model the dynamics within that latent space, in a decoupled structure. The model takes demonstrations[1] as input. Each demonstration is a sequence of frames consisting of perceptual features and kinematic action data. The frames may be sparse in time, corresponding to the key points in the skill according to the demonstrator in the case of keyframe demonstrations, or densely recorded throughout the skill in the case of continuous demonstrations.

Figure 4.1 shows the structure and loss components that make up our model. The model makes use of the sequentiality of consecutive frames by learning to predict the state representation of the next time step, given the current state and the next action. This self-supervised learning task is to make sure that the learned state representation contains relevant dynamics information since the data is sequential. Prediction within the latent space is done by the Next State Predictor (NSP) module, whereas the Encoder and Decoder modules learn mappings between the input space and the latent space. Purpose of the reconstruction task performed by the Decoder module is to ground the latent space with a constraint that forces it to avoid trivial solutions such as converging to a constant function.

[1]We assume that all demonstrations are successfully performed

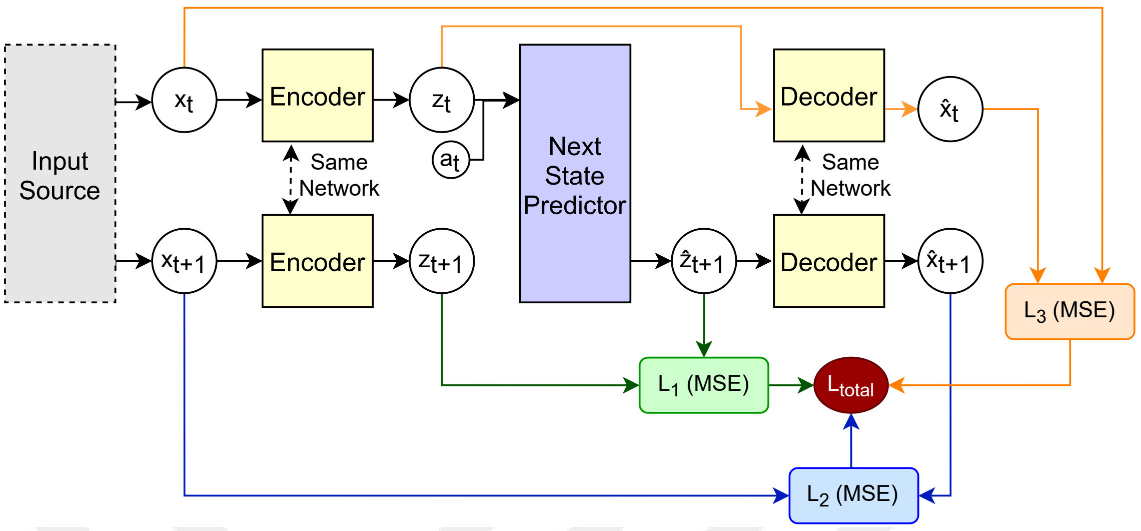


Figure 4.1: A diagram of our model and its loss function components. The Encoder and Decoder modules are feed-forward multilayer perceptrons, while the Next State Predictor module is a recurrent neural network.

4.1.1 Actions

The action information is used only in NSP, and it is included by directly concatenating the continuous action vector with the latent vector from the Encoder. In our application, the action vector consists of the end-effector poses of the robot with respect to the object at times t and $t + 1$. While this is easier for notation, ideally we think of the end-effector pose and the latent vector at time t combined as representing the state and the pose at time $t + 1$ representing the action.

4.1.2 Modules of the Architecture

Due to the low-data regime, we aim for a simple architecture that can produce good results without being data-hungry. While more advanced architectures could be used, they often require more data and tweaking to train, which is not very suitable for the problem we are tackling.

The model is composed of neural network modules whose parameters are learned using back-propagation. The Encoder and Decoder modules are 2-layer multi-layer perceptrons with ReLU non-linearity. The Next State Predictor is a single layer Elman RNN with tanh non-linearity, plus a linear output layer. We opted for an

Elman layer for simplicity. Keyframe demonstrations are not very long and we did not see any adverse behavior in continuous demonstrations. However, LSTMs or GRUs can also be used here.

4.1.3 Notation

The following is the list of notations used in defining our model and its loss components (described in Section 4.2):

$$\begin{aligned}
X &: \text{Perceptual feature (input) space} \\
Z &: \text{Latent representation space} \\
A &: \text{Action space} \\
x_t^i &: \text{Perception frame of demo } i \text{ at step } t \\
a_t^i &: \text{Action frame of demo } i \text{ at step } t \\
M_E : X \rightarrow Z &: \text{Encoder} \\
M_D : Z \rightarrow X &: \text{Decoder} \\
M_N : Z \times A \rightarrow Z &: \text{Next State Predictor} \\
z_t^i = M_E(x_t^i) &: \text{Latent representation from encoder} \\
\hat{z}_{t+1}^i = M_N(z_t^i, a_t^i) &: \text{Predicted next latent representation} \\
\hat{x}_t^i = M_D(\hat{z}_t^i) &: \text{Decoded output of demo } i \text{ at step } t \\
MSE(a, b) &: \text{Mean square error between a and b}
\end{aligned}$$

4.2 Loss Function

For each batch of demonstrations, B, that contains n demonstrations with k frames[2] each, the loss function L_{total} and its components are defined as in Equation 4.1.

[2] n can be different for each batch and k can be different for each demonstration.

$$
\begin{aligned}
L_1 &= \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{k-1} MSE(z^i_{t+1}, \hat{z}^i_{t+1})) \\
L_2 &= \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{k-1} MSE(x_{t+1}, \hat{x}^i_{t+1}) \\
L_3 &= \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{k-1} MSE(x_t, M_D(z^i_t)) \\
L_{total} &= \alpha L_1 + \beta L_2 + \gamma L_3
\end{aligned}
\tag{4.1}
$$

L_1 can be thought of as the dynamics modeling error in the latent space. It is mainly for the training of NSP and is unaffected by the Decoder. L_2 is the grounding loss component. Its gradients flow through all modules. It acts as a regularizing component for the information quality of the latent space. L_3 is the "auto-encoder" loss component. It bypasses NSP, being affected only by Encoder and Decoder. The purpose of L_3 is to make sure that only the Decoder is responsible for the reconstruction task, instead of NSP and Decoder getting entangled in their tasks. α, β and γ are hyper-parameters for weighting the effect of different components in the total loss. The values of these hyper-parameters are determined empirically. β and γ are set to 1 in all cases, while α is set to 4 in the keyframe setting and 1 in the continuous setting.

4.2.1 Contrastive Loss Component

We additionally use a contrastive loss component, L_C, computed as the L_{total} for the batch where the order of the frames in each demonstration are randomly shuffled, named $B_{shuffled}$. A negative hinge loss is computed from this which is added to get the final loss as shown in Equation 4.2. The idea with L_C is that the shuffled demonstrations are expected to constitute a negative example, and as such can be thought of as a data augmentation technique to generate and make use of negative samples. The contrastive loss gets disabled when dealing with continuous demonstrations, as it does not improve the model's performance. This is because the continuous demonstrations contain many similar frames, especially at the beginnings and ends, and as such shuffling the order of frames has a high chance of producing positive looking samples, defeating the purpose.

During the ablation study (see Section 6.3), the contrastive loss by itself was able to fulfill the purpose of regularizing the latent representations, preventing trivial solutions and possibly removing the need for decoding.

$$\begin{aligned} L_C(B) &= MAX(0, 0.01 - L_{total}(B)) \\ L_{final}(B) &= L_{total}(B) + L_C(B_{shuffled}) \end{aligned} \tag{4.2}$$

4.3 Conclusion

In this chapter, we explain in detail the proposed neural network architecture. We describe the modules that make up the model, and formulate the loss function in mathematical notation while giving the reasoning behind its individual components.

Chapter 5

EXPERIMENTAL SETUP

This section contains the details regarding the experiment setup, training of the proposed model, and the dataset of demonstrations we use in evaluations. Our experiments and data collection are done using Franka Emika Panda, a 7-DoF robotic arm, along with a Microsoft Kinect V1 RGB-D camera as the external sensor. The setup, shown in Figure 1.1 consists of the robot in front of a table, on top of which objects to be manipulated are placed. The camera is located at the side of the table, perpendicular to the robot, with an overhead view of the environment.

In order to test our model in a rigorous manner, we gathered a demonstration dataset of skills performed on a number of objects, where the demonstrations are given by multiple people. A description of this dataset can be found in Section 5.2.

First, we compare the monitoring performance of our representation learning method with the alternatives by measuring the accuracy of separating positives from near-misses. Section 6 describes the details of this experiment.

Next, we evaluate the reinforcement learning performance of our method in both simulated and real-world environments which is presented in Section 7.

Finally, in Section 8, we explore different transfer learning methods and measure their efficacy with varying target training data amounts.

5.1 Training Details

Training is done using the Adam optimizer with 1e-5 weight decay and 1e-3 learning rate. Dropout with a 50% ratio is used in the linear output layer of NSP for regularization. The hyper-parameters are selected with some grid search and a tendency to err on the side of simplicity. In the monitoring experiments, data is split as 50%, 25%, 25% into training, validation and test sets respectively. The model is trained

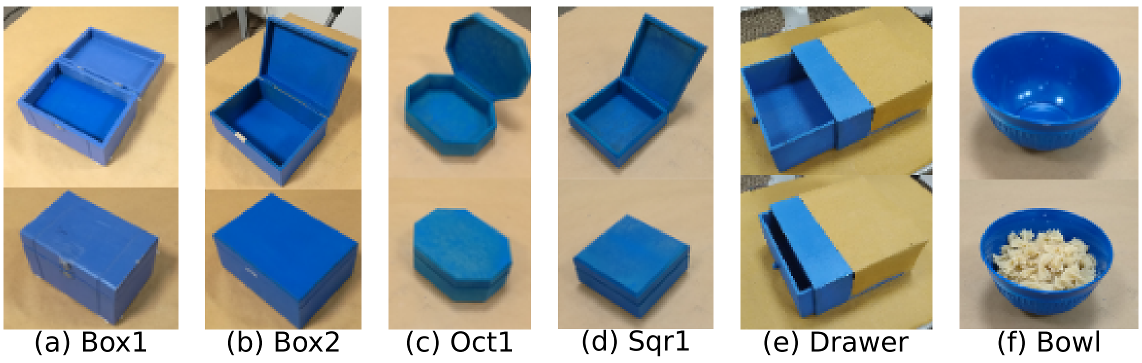


Figure 5.1: Objects used in gathering the dataset. The top row shows their open (empty for Bowl) state while the bottom row shows their closed (full for Bowl) state.

for 400 epochs and the parameters at the epoch with the best validation accuracy is kept as the final version. In all monitoring experiments, including the transfer learning evaluations, every experiment is run 20 times with changing random seed, and as such the train-test splits are also randomized at every run.

5.2 Demonstration Dataset

We gathered a demonstration dataset consisting of multiple objects and multiple skills performed by 2 to 4 people each. There are two sets of data, one with keyframe and one with continuous demonstrations. The keyframe dataset is larger, consisting of 6 objects while the continuous dataset contains 2 objects. Table 5.1 shows how many demonstrations for each object-skill pair there are in the dataset. Images of the objects used for demonstrations can be seen in Figure 5.1. The skills Close and Open refer to manipulating the lid in the case of hinged boxes, and drawing or pushing in the case of Drawer. The Pour skill refers to pouring pasta into a container from a ladle held by the end effector of the robot.

While the number of demonstrations and people giving demonstrations varies from object to object, this does not cause an imbalance problem in our evaluations as the sets are independent in the monitoring experiments, and the numbers are almost always equal within groups in the transferring experiments. One exception to this is the transfer between object experiment (Section 8.2) with Box2, Oct1 and

	Object ID	**Close**	**Open**	**Pour**	**# of People**
Keyframe	Box1	40+40	40+40	-	4
	Box2	40+40	40+40	-	3
	Oct1	30+30	30+30	30+30	3
	Sqr1	30+30	30+30	-	3
	Drawer	20+20	20+20	-	2
	Bowl	-	-	30+30	2
Continuous	Box1	50+50	50+50	-	3
	Box2	30+30	30+30	-	2

Table 5.1: Table showing the number and properties of the data available. Plus sign is used to separate between positive and near-miss samples, e.g. 40+40 means there are 40 positive and 40 near-miss demonstrations.

Sqr1, where the number of people are equal but the number of demonstrations are 40 in Box2 and 30 in the other two. This did not cause a problem in the results, and therefore we do not investigate this further.

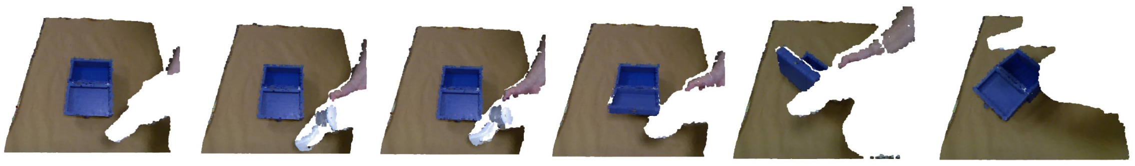
For all object-skill pairs, there are as many near-miss demonstrations as there are positive demonstrations. Near-misses are almost successful demonstrations but they fail to achieve the goal of the demonstrated skill. These are not used during training but are used when measuring the monitoring performance. We use near-misses instead of complete failures or other skill demonstrations to make our evaluations more realistic. While we do not give a formal definition for near-misses, Figure 5.2 shows one example of near-miss demonstrations for each skill in the dataset to give a more clear idea. Note that there are various ways to fail a skill, and the given near-miss demonstrations show this variety as well.

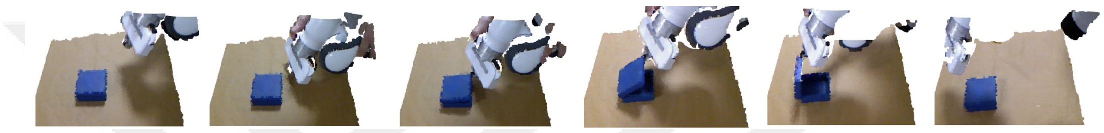
The demonstrations are given by kinesthetically guiding the end effector of the robot. In the keyframe setting, the teacher pauses at the points they deem important to mark as keyframes and save data. In the continuous setting, data is saved at a regular frame-rate and the demonstration is performed from start to finish without pause.

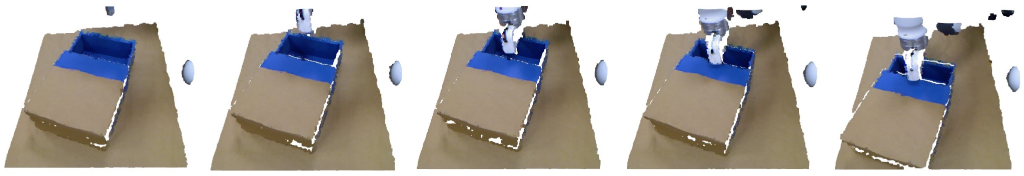
The collected data contains point cloud snapshots of the scene, and the pose of the end-effector with respect to the robot's base. From the segmentation pipeline, a segmented object point cloud is produced and the pose of the object is inferred by fitting a bounding box from the convex hull of the segmented points. Using this information, the end effector-poses are transformed to the object's frame of reference, which are saved as the action data. The segmented point cloud is fed into a pre-trained point cloud auto-encoder network and the encoded result is saved as the perception data.

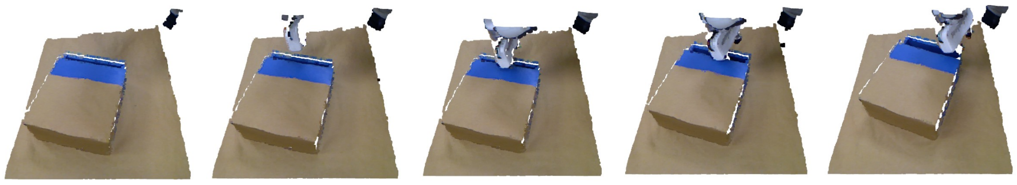
5.3 Conclusion

In this chapter, we described the setup in which our experiments are conducted, the structure and contents of the dataset used in those experiments, and details regarding the training of the proposed model as performed in the experiments.


(a) Close skill on Box1


(b) Open skill on Sqr1


(c) Close skill on Drawer


(d) Open skill on Drawer

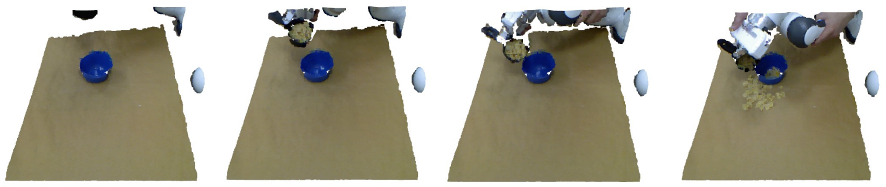

(e) Pour skill on Bowl

Figure 5.2: Examples of near-miss demonstrations for each skill.

Chapter 6

MONITORING

In order to compare the effect of different methods on monitoring performance, we conduct an experiment on the dataset mentioned in Section 5. We compare our method with Principal Component Analysis (PCA) as used in [Eteke et al., 2020] and also Uniform Manifold Approximation and Projection (UMAP), a non-linear dimension reduction technique.

6.1 Monitoring Experiment

The experiment is performed as follows:

- Positive data is split into train, validation and test sets as 50%, 25% and 25% respectively.
- Model is trained using the training data.
- An HMM goal model is trained on the encoded training data from each method.
- The goal model predicts success or failure for the set of positive test data and an equal amount of near-miss data, based on the likelihood threshold and terminal states determined during training.
- Accuracy is calculated as the percentage of correct classifications
- This is repeated 20 times with a randomly sampled train/test split, and the mean and standard deviation of the accuracy are reported as the final metrics

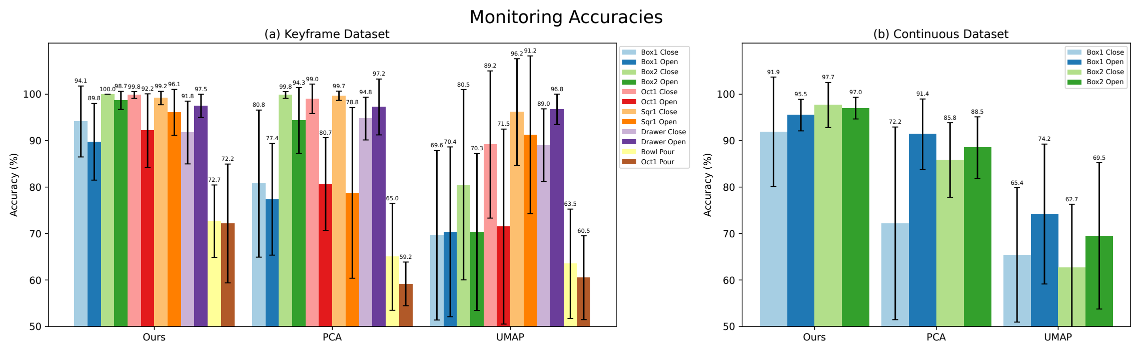


Figure 6.1: Monitoring accuracy results of our model, PCA and UMAP for each task in the Keyframe (left) and Continuous (right) datasets. Each experiment is run 20 times. The averages across 20 runs are plotted in a bar chart format, while the standard deviances are displayed as error bars.

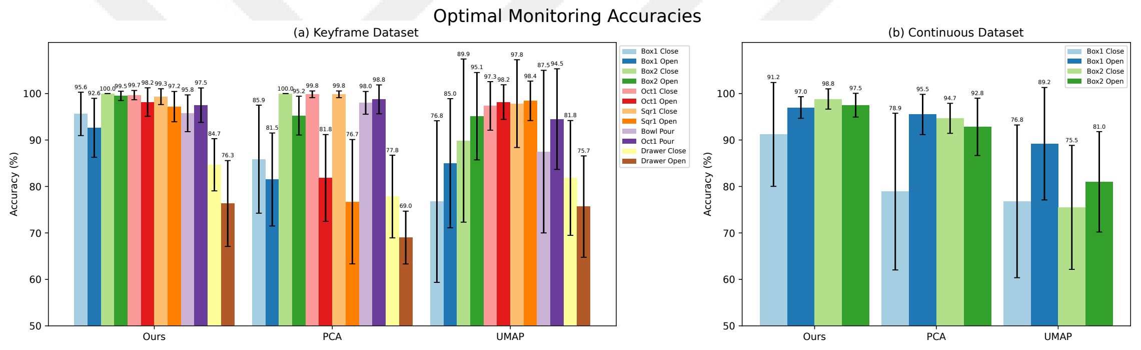


Figure 6.2: Monitoring accuracy results as in Figure 6.1, but with optimal threshold calculation.

The goal model and its method of determining the threshold is explained in Subsection 3.3. However, in order to make sure that the threshold selection method is not a cause of unfair error, we also calculate optimal threshold accuracies. Optimal threshold is the one resulting in the highest possible accuracy from a given test set. Determining it requires knowledge of the test set in advance, so it is not possible in practice but it is useful for showing an upper limit, isolated from the variance of threshold selection.

6.1.1 Results

The averaged accuracies across all tasks are shown in Table 6.1 whereas the results for each task separately are plotted in Figures 6.1 and 6.2 in detail. It is shown

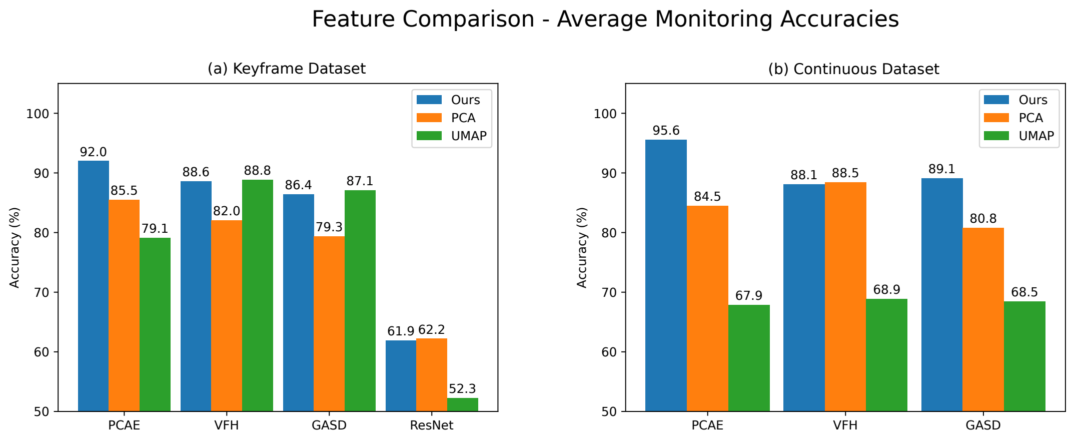


Figure 6.3: Averaged keyframe and continuous monitoring accuracy results comparing different perceptual features as input. Every experiment is repeated 20 times with randomly changing train/test splits.

	Normal Threshold		**Optimal Threshold**	
Method	**Keyframe**	**Continuous**	**Keyframe**	**Continuous**
Ours	**92.0**	**95.6**	**94.7**	**96.1**
PCA	85.5	84.5	88.7	90.5
UMAP	79.1	67.9	89.8	80.6

Table 6.1: Average accuracies across all of the Keyframe and Continuous tasks in the normal and optimal threshold calculation settings.

that our proposed model outperforms PCA and UMAP in average accuracy by a considerable margin. This is true for not only the default threshold calculation method, but also for optimally selected thresholds as well, showing that this result is not dependent on the method used for threshold selection.

In terms of different tasks, we can see from Figure 6.1 that ours is the best in 10 out of 12 Keyframe tasks and 4 out of 4 Continuous tasks. In the optimal threshold case, Figure 6.2 shows that ours is the best in 5 out of 12 Keyframe tasks with PCA being the best in 4 and UMAP in 1 out of 12 and two ties between ours and PCA/UMAP. In two cases where PCA is the best, there is less than a percentage of difference from ours. In Continuous tasks, ours is again the best in 4 out of 4. Ours is also never the worst out of the three in any case. Thus we can conclude that, while our method is not necessarily the best performing in every single case, it is most consistently the best and never the worst, in both Keyframe and Continuous tasks.

6.2 Different Skill Agnostic Features

We use a PCAE to extract generic perceptual features and train our model on those features, as explained previously. However, our model can be used with any other input domain in the same way. In this section, we compare the monitoring performance of our model, PCA and UMAP with three different features in addition to PCAE. These are VFH (Viewpoint Feature Histogram) [Rusu et al., 2010] and GASD (Globally Aligned Spatial Distribution) [Lima and Teichrieb, 2016] extracted using the segmented point clouds in the same way as PCAE, and ResNet [He et al., 2015] extracted from the colored 2D projection of the unsegmented point clouds. We remove the final classification layer from an ImageNet pretrained ResNet model and use its penultimate layer for the features.

Figure 6.3 shows the average monitoring accuracies for the keyframe and continuous data. ResNet results with continuous data are not presented, because the unsegmented point clouds required for ResNet were not saved due to storage space constraints. It can be seen from the results that the highest accuracy is seen when using the PCAE features with our model. A point to note is that UMAP performs better with VFH and GASD features compared to PCAE features in the keyframe setting, although it is still lackluster in the continuous setting. This shows that there is value in exploring different generic features for different problems.

The ResNet results are quite low compared to the rest. This can be explained by the geometric information lost in projecting depth images to 2D, and the fact that the ImageNet pretrained ResNet is not necessarily a suitable model for distinguishing between changing states of the same object. However, later in Section 8.3 we present a transfer learning technique that allows us to utilize ResNet features in a better way.

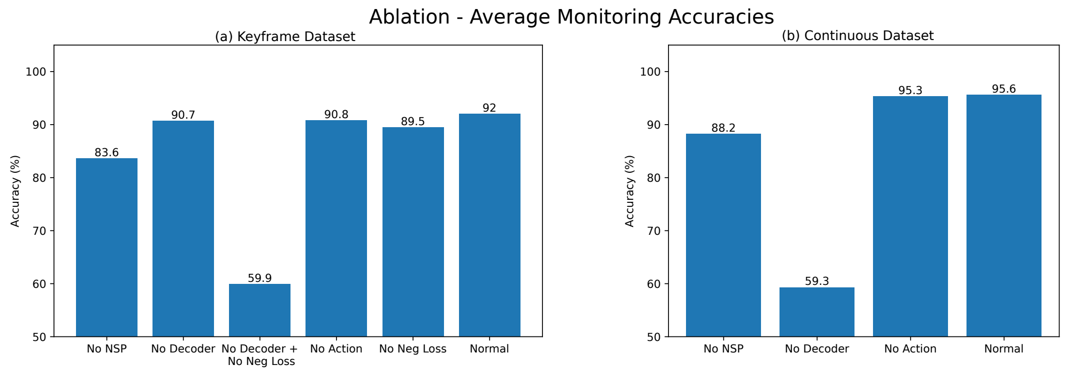


Figure 6.4: Average accuracies for the ablation study in keyframe and continuous datasets. Negative loss component is not activated in any of the continuous cases, so the "No Decoder" in the continuous plot is analogous to the "No Decoder + No Neg Loss" in the keyframe plot.

6.3 Ablation Study

We study how the monitoring accuracy changes when individual components of our model are disabled. Using this information, we can see an estimate of how important each component is in practice. We test the following cases:

- No NSP: This is effectively an auto-encoder model.
- No Decoder: Decoder module and the loss components associated with it are disabled.
- No Decoder + No Neg Loss: A version of "No Decoder" with the contrastive negative loss component disabled. This is only tested in keyframes, because we already do not use the contrastive loss in the continuous case. The "No Decoder" case in the continuous case is equivalent to this.
- No Action: The action input to the NSP is disabled.
- No Neg Loss: Contrastive negative loss component disabled. This is only tested in keyframes, because we already do not use the negative loss in the continuous case.

- Normal: This is the default model.

The average accuracies for each case in the keyframe and continuous data are shown in Figure 6.4.

6.3.1 No Decoder

It can be seen that the biggest fall in accuracy is seen in the ”No Decoder” case in continuous and its counterpart ”No Decoder + No Neg Loss” in keyframe. Without a decoder, the only MSE loss getting optimized is L1 in Equation 4.1. Without anything to stop it, the model can optimize to a trivial solution where any input is mapped to a constant in the latent space. A surprising result is that ”No Decoder” with negative loss enabled does not suffer from this problem, meaning that the Decoder may not be necessary when there is a contrastive loss, which is enough by itself to stop the model from converging to a trivial solution. However, there is still a small difference in accuracy between having a decoder or not even in that case.

6.3.2 No NSP

The next biggest fall is seen in the ”No NSP” case. The model in this case does not do any prediction of the next state. Instead, the encoder and decoder are directly connected, effectively creating an auto-encoder model. This model has no incentive to model dynamics in the latent space, and as such suffers a significant loss of monitoring performance in both keyframe and continuous data. One reason for drop in the performance is that this case does not take the sequentiality of the data into account.

6.3.3 No Contrastive Loss

The ”No Neg Loss” case suffers from a 2.5% loss of accuracy, which while not very large, is still noticeable. The negative loss is already disabled by default on continuous as it is not helpful. The reason for negative loss not working well in the continuous case is because there are many frames that are similar to each other in the continuous demonstrations. Since the negative loss is computed from creating a negative sample by randomly shuffling the order of frames in the demonstrations, there is a high chance that similar frames end up in consecutive orders, therefore creating bad negative samples that confuse the model.

6.3.4 No Action

The "No Action" case does not show any significant difference in the continuous case. We concur this to be because the deltas in the end effector pose between two consecutive frames are too tiny to provide any useful signal to the model. In the keyframe case, there is a small difference of slightly more than 1% accuracy. This means we can likely omit the action information all together without having much of an effect on the monitoring performance.

6.4 Qualitative Analysis of Learned Representations

We train our proposed model, PCA and UMAP with a 2 dimensional latent space and plot the 2D frame representations of our demonstrations directly as displayed in Figure 6.5. Each dot represents a keyframe, where the color of the dot is determined by its order in the demonstration, e.g. purple dots are the beginning frames while yellow dots are the final frames. Note that yellow dots are mixed with light green dots and purple dots are mixed with dark blue dots, because in most demonstrations, the object looks similar in the first and last 2 frames as a result of moving the robot arm into and out of position without interacting with the object during those frames.

6.4.1 Discussion

Since PCA works by applying a linear transformation to the data, we expect its output to resemble the original shape of the data with certain rotations. On the other hand, UMAP seems to be good at separating different states of the object into different locations in the space, but it often splits what should be a single state into multiple clusters, and understandably does not usually show the sequentiality of frames. Our model seems to manage to bring all frames of the same state together, at least in the beginning and end states, unlike UMAP. One thing to note is the existence of scattered points and outliers, which UMAP seems to deal with more effectively. A nice property of our model's features is that they carry the sequentiality in the representation space. This is most clearly seen in Box1, Sqr1 and Drawer, but is visible in all objects. The more uniform clustering and preservation of sequentiality seen in our model may be an explaining factor in the superior performance.

6.5 Conclusion

In this chapter, we gave details regarding the monitoring experiment and present its results. We discussed the results in both the realistic threshold selection case, and the idealistic optimal threshold case, showing that our model brings a definitive advantage over PCA and UMAP. In addition, we presented experimental results regarding the use of skill agnostic perceptual features other than PCAE. After that, we showed an ablation study where we disable various features employed in our model in isolation in order to measure their contributions to the monitoring performance. One result of the ablation study displayed the importance of grounding the latent representations with a decoder or a contrastive loss, but that we could possibly do without one or the other. Finally, we performed a qualitative analysis of the learned representations of our model, PCA and UMAP, noting the unified clustering and sequentiality preservation seen in our representations.

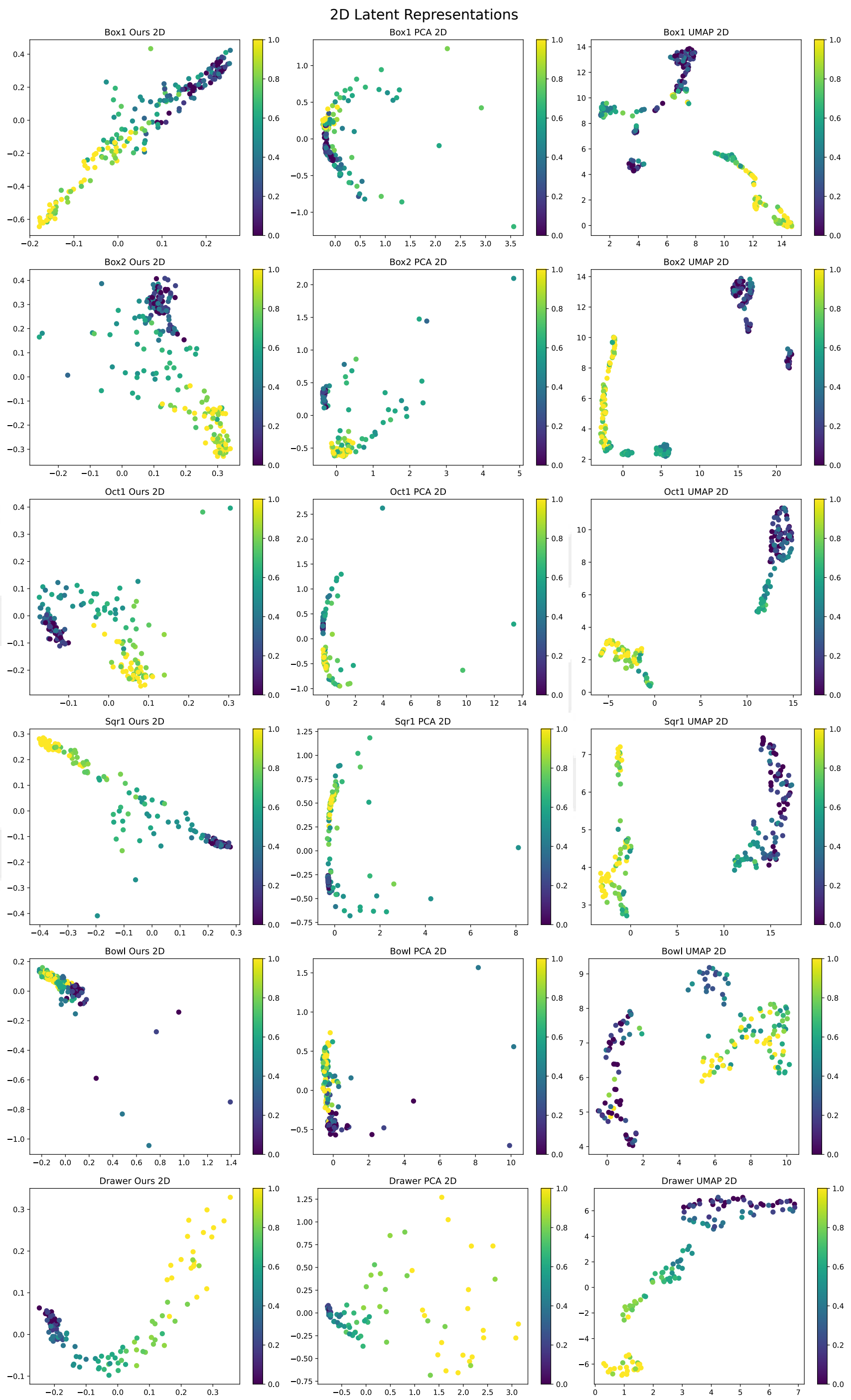


Figure 6.5: Plot of keyframe demonstration frames from the Open skill, in 2D latent space of Ours, PCA and UMAP. Colors assigned from 0 to 1 represent the position of the individual frame in the demonstration's progress, the first frame being 0 and the last frame being 1.

Chapter 7

REINFORCEMENT LEARNING

One of the most important uses of the goal models is in reinforcement learning (RL). While it is possible to learn a skill successfully only by imitation, it is often not the case, especially if the demonstrations are not given by an expert and/or there is uncertainty. It is easier to learn a successful goal model than it is to learn a successful action model [Akgun and Thomaz, 2015]. Thus, goal models can be used to turn unsuccessful action models successful via further reinforcement learning.

Authors of [Eteke et al., 2020] describe a way of modeling rewards from a goal model. While reinforcement learning can be done using the sparse rewards directly from the goal model's binary monitoring decision, they show that the dense rewards learned from the goal model lead to better results. They use PCA as their skill-specific feature extraction step. Our aim in this section is to show that the goal model learned with our model's latent space leads to a better reward model as evidenced by improved RL performance.

7.1 Setup

While the authors of [Eteke et al., 2020] consider only continuous demonstrations, we also consider keyframe demonstrations. We use PI^2-ES-Cov as proposed in [Eteke et al., 2020] as our policy search method. In the continuous setting, we use DMPs (Dynamic Movement Primitives) [Ijspeert et al., 2002] to model the actions as described in [Eteke et al., 2020], but in the keyframe setting we use HMMs (Hidden Markov Models) instead. We compare our proposed model against PCA and UMAP.

The keyframe and continuous demonstrations used for learning initial policies are explained in Section 5.2. Experiments are done in a simulation environment and in the real world. In either case, the setup is the same with Franka Emika Panda,

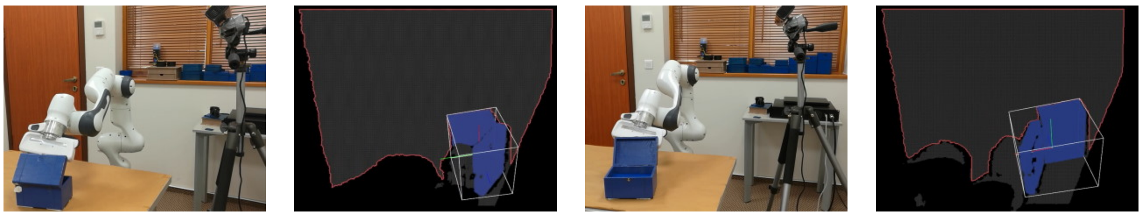


Figure 7.1: RL environment in the real robot setting. (a) shows the robot executing the open skill, while (b) shows the same scene from the robot's perception view. (c) and (d) show the same views for an execution of the close skill. The perception view shows the segmented object and its calculated bounding box.

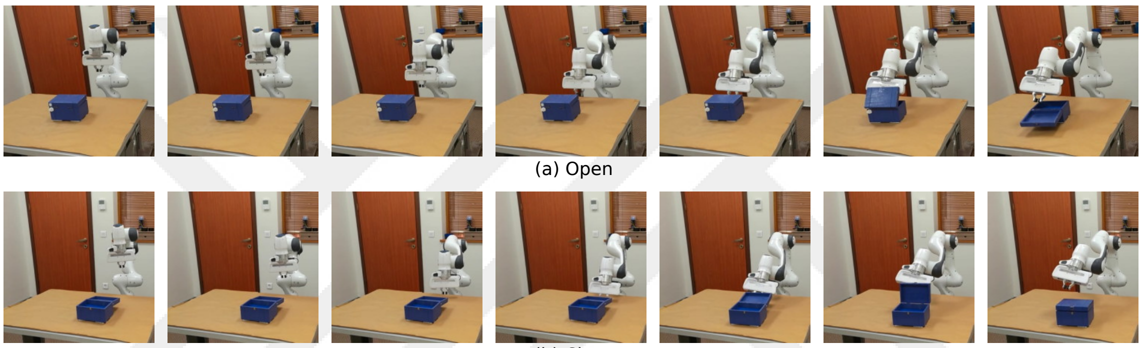


Figure 7.2: Successfully learnt executions of Open (a) and Close (b) skills in the real world setting.

Microsoft Kinect V1 and objects on a table as described in Section 5. The simulator used is Gazebo. Figure 7.1 shows the environment setup in the real setting, including the robot's view of the environment.

In all settings, the experiments are run for 5 episodes with 6 rollouts each. Greedy/maximum likelihood trajectories are executed before the first episode, and after each episode. The reported success ratio results are gathered from these greedy runs. The latent representations are set to be 8 dimensional for all methods to keep consistent with [Eteke et al., 2020].

7.2 Continuous Setting

In this setting, the demonstrations consist of action and goal frames taken at regular intervals with a relatively high frame rate. DMPs are learned as the action models

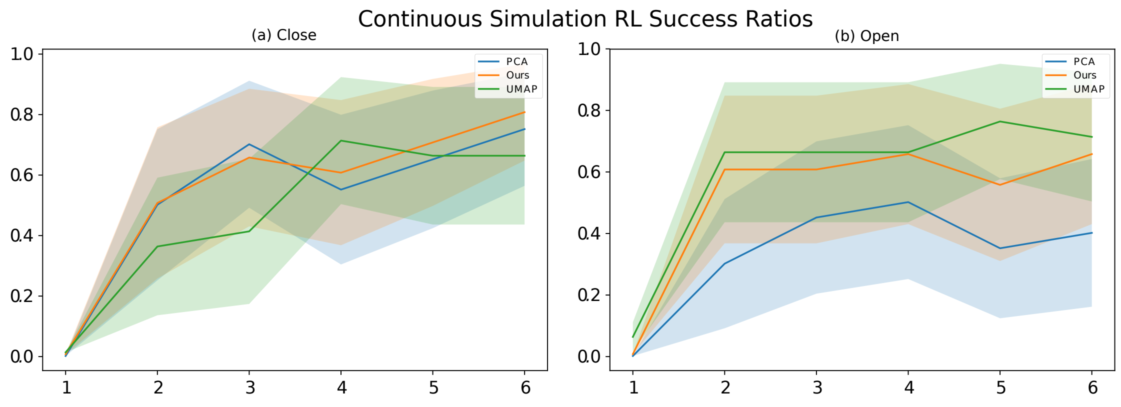


Figure 7.3: Results from continuous RL in simulation. The plotted lines are the average success ratios of each method in greedy executions between the episodes. Shaded regions represent the variance. Each experiment is run 20 times.

and HMMs are learned as the goal models.

We use one box in the simulation and one box in the real world experiment with Close and Open skills. Example executions of these skills in the real world setting can be seen in Figure 7.2.

In both simulation and real robot experiments, we use 20 DMP bases for Open. For Close, we use 10 bases in simulation and 13 in real robot, chosen empirically.

7.2.1 Simulation Results

Each experiment is run 20 times and the success ratios for each episode are shown in Figure 7.3.

In the Close skill, there is not a considerable difference between the methods. While our method reaches the highest success ratio, PCA is close behind and the variances are high enough for the difference to not be significant. UMAP is a bit slower to learn than others, but not by a large margin. In the Open skill, UMAP performs the best while ours is very close but PCA performs considerably worse than the others. Overall, our method does not result in significantly better RL performance but it works consistently well for both skills. We cannot say the same for PCA and UMAP.

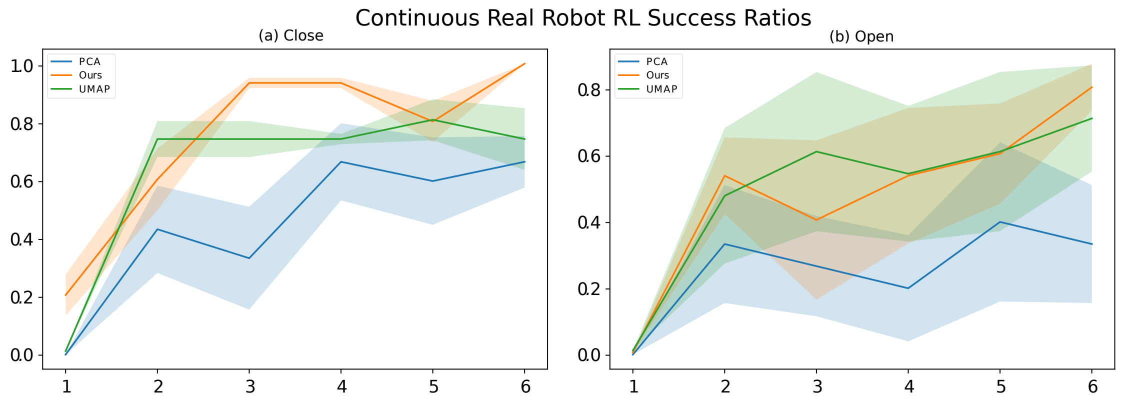


Figure 7.4: Results from continuous RL on real robot. The plotted lines are the average success ratios of each method in greedy executions between the episodes. Shaded regions represent the variance. Each experiment is run 5 times, with greedy executions repeated 3 times.

7.2.2 Real Robot Results

Each experiment is run 5 times and the success ratios are shown in Figure 7.4. Greedy executions are repeated 3 times, so the results are averaged over 15 runs at each episode.

In the Close skill, ours is overall the best performing method, although UMAP is close. ours manages to reach 100% success rate at the final episode. In the Open skill, ours performs around the same as UMAP. In either skill, PCA falls behind the others. Also there is more variance between runs in Open than in Close, which may be because it is a more difficult skill to perform.

Overall, the results are mostly similar to the simulation experiments, except ours has an edge in the Close skill in this experiment. We can again say that our method works consistently well in either skill.

7.3 Keyframe Setting

In this setting, the demonstrations consist of action and goal keyframes taken at points that the teacher dictates. Two HMMs are learned, one for the action model and one for the goal model, from the keyframes.

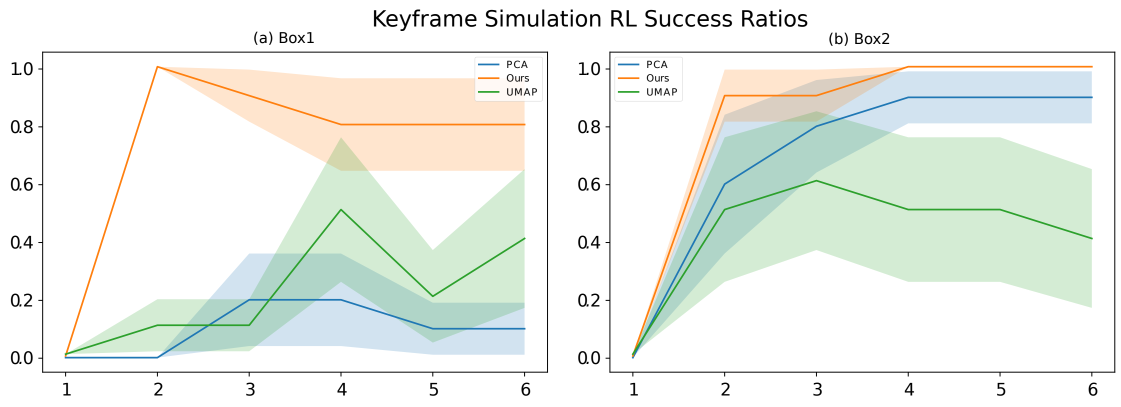


Figure 7.5: Keyframe setting RL results. The plotted lines are the average success ratios of each method in greedy executions between the episodes. Shaded regions represent the variance. Each experiment is run 10 times.

7.3.1 Simulation Results

We use two differently shaped boxes, called Box1 and Box2, with the Close skill. Each experiment is run 10 times and the success ratios for each episode are shown in Figure 7.5.

It can be seen that in Box1, ours learns faster and performs better than both PCA and UMAP. PCA has an especially low success ratio. In the case of Box2, ours again performs considerably well in comparison, while PCA is not too far off. From these results, we conclude that our method does well in the Keyframe setting and does not suffer from much variance between tasks, unlike the other methods.

7.4 Monitoring Accuracy During RL

The ground truth success results of the greedy runs in RL experiments are taken manually from recorded frames. Figure 7.6 shows the accuracy of the monitoring decisions made by the goal model in greedy runs. Overall, our model has the highest accuracy while UMAP has the lowest. It is notable that UMAP shows a decent RL performance despite its poor monitoring results. This indicates that monitoring performance may not directly correlate with RL performance, possibly due to the denseness of the rewards.

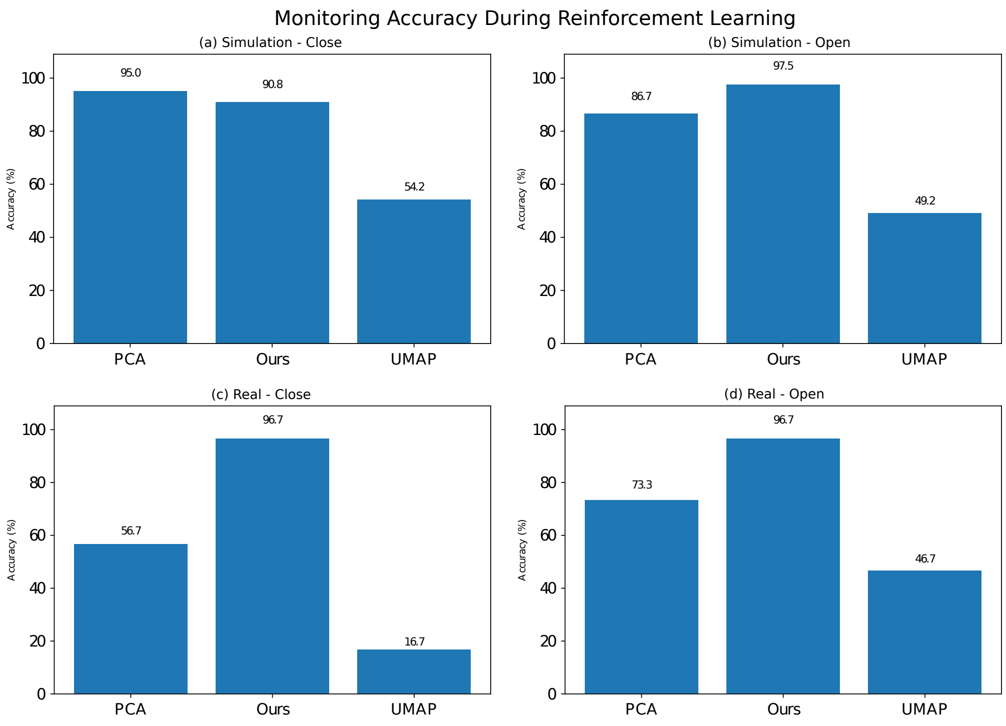


Figure 7.6: The accuracy of monitoring performed by the goal model during the continuous RL experiments.

7.5 Conclusion

In this section, we presented RL experiments in the continuous and keyframe settings with the simulation and the real robot. We explain the setup for these experiments, and then display and discuss the results. In addition, we show an analysis of monitoring accuracy during the RL experiment.

Chapter 8

TRANSFER LEARNING

Transfer learning is the idea of using knowledge attained from a source task to improve the performance of a target task. This is an important concept to consider when dealing with low-data settings like ours. In this section, we present three different use cases for transferring knowledge by making use of our proposed model, as follows: (1) learning a new skill with a previously encountered object, (2) learning a previously encountered skill with a new object, and (3) learning with a different perception sensor using the representations learned by an existing one.

8.1 Same Object - Different Skill

8.1.1 Method

In the case where different skills are performed on the same object, it is possible to transfer knowledge gained from one skill's data to another one. Since the object being acted on is the same, it is expected that there exists some commonality in the observations gathered from the object. The motivation for doing transfer learning in this setting is to reduce the data requirement when teaching a new skill on an old object, by reusing the data already gathered on that object.

When transferring between the Close and Open skills on a box, the perceptually observed object states are expected to be very similar, since both will consist of the object states that range from fully closed to fully open. We expect a good amount of positive transfer to be possible in such a case. On the other hand, a skill pair such as Open and Pour do not have as much in common in terms of observed object states, since they affect the object in different ways. In this case, we expect less of an advantage from transfer learning.

For transfer in the same object - different skill setting, we make use of the

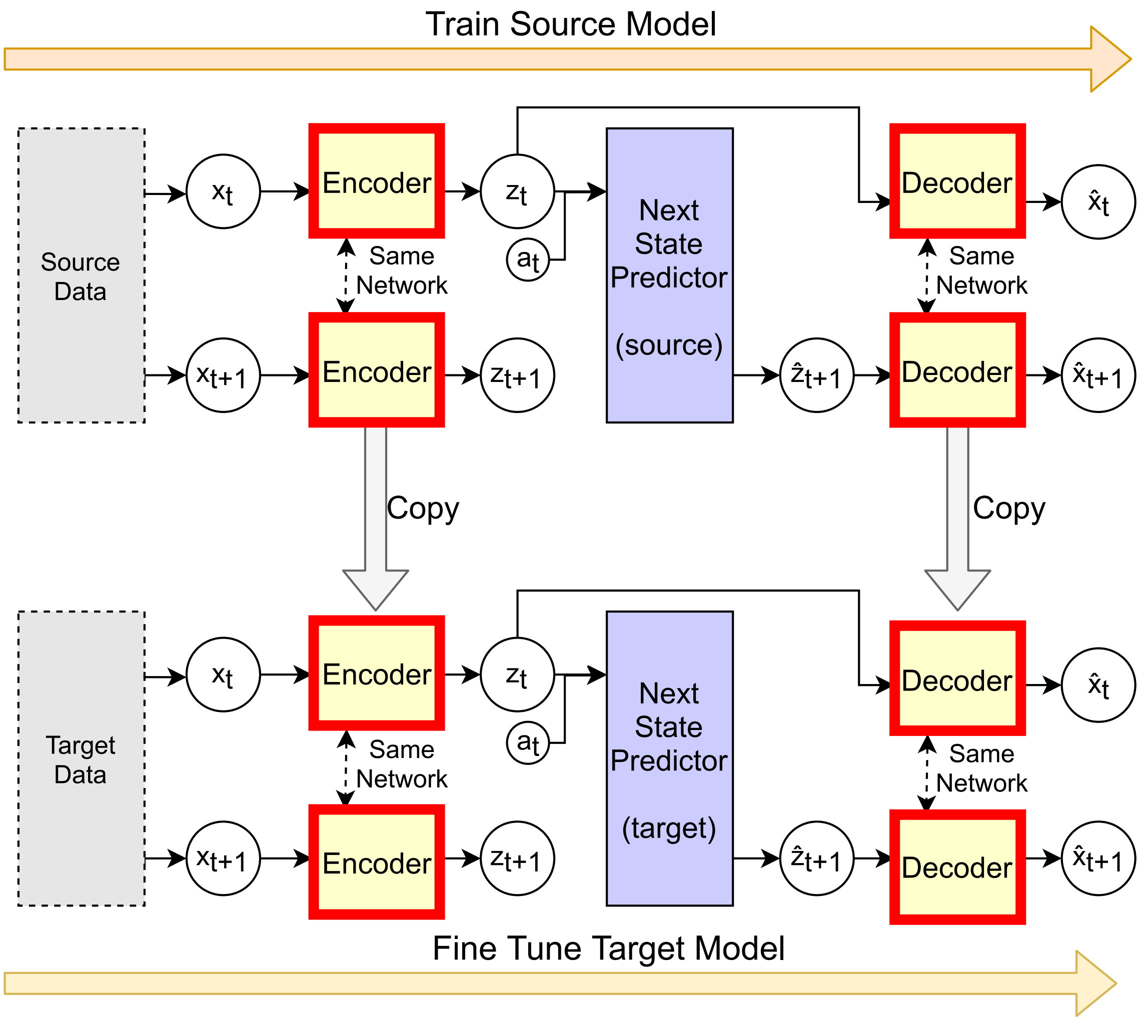


Figure 8.1: The proposed method for transfer learning between different skills on the same object.

architecture of our model and propose the following procedure, depicted visually in Figure 8.1:

- Pre-train the model on source skill demonstrations
- Copy the trained Encoder and Decoder modules to a new model
- Fine tune the new model on (potentially fewer) target skill demonstrations, with a lowered learning rate

The idea is that the Encoder and Decoder modules learn the mapping of the

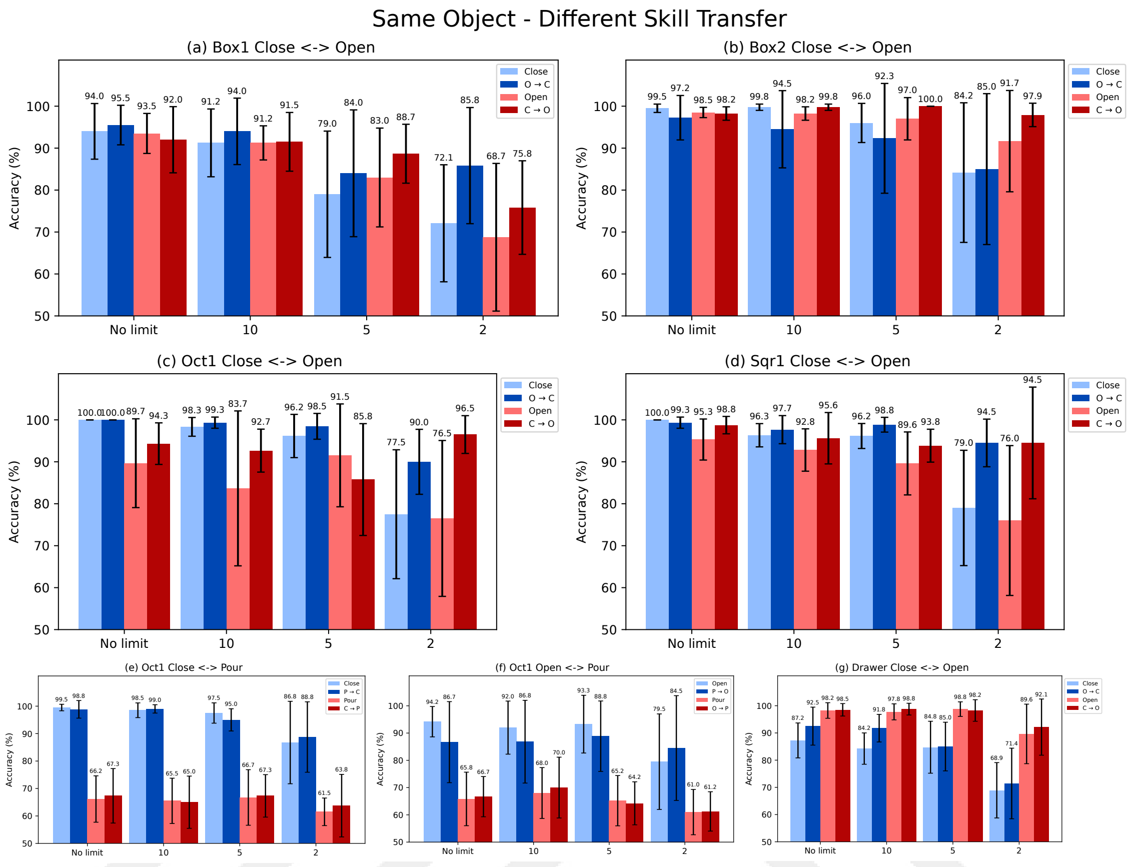


Figure 8.2: Same object - different skill transfer learning in keyframe setting accuracy results at different target data limitations. Each plot displays the results for two skills on one object as indicated on the titles. At the x-axis are the target data limitation settings. Amount of target data used in fine-tuning gets lower as we go right. Amount of data at ”No limit” is either 15 or 20 depending on the object. Light blue and red bars show the accuracy got with no transfer applied whereas dark blue and red bars show the accuracies with transfer learning. Difference between the dark and light bars of the same color show the advantage gained by applying transfer.

object states between the perceptual and latent spaces. Therefore, by copying the learnt parameters of these modules, we are transferring the knowledge of this mapping, which can be reused as long as there is an intersection between the object states seen in the source and target skills. By using this as the initial point and doing fine tuning, the model can specialize in the desired target task more easily than it would from a random starting point. We do not transfer the NSP module, since it learns a dynamics model specific to the source skill which does not help the

Data limit	**-**	**10**	**5**	**2**
Positive Transfer	3	2	2	8
Negative Transfer	1	1	1	0
No significant effect	10	11	11	6

Table 8.1: Number of positively or negatively effected cases at different target data amounts for the same object - different skill transfer.

target skill. In principle, it is possible to learn a universal NSP that models all skills for an object simultaneously, but that would requires significantly more data than is available in this setting, and a higher capacity model.

We empirically determined the fine tune learning rate to be set to 0.2x the original rate to give the best results.

8.1.2 Results

We test the proposed method on 5 objects and 8 skill pairs in total. In order to see the effectiveness of this method in alleviating low data problems, we experiment with 4 settings of limited target data amounts. These settings are ”No limit”, 10, 5 and 2 demonstrations in descending order.

The overall effect of this transfer method at different target data limits is summarized in Table 8.1. Detailed results can be found in Figure 8.2. We conclude from the results that this method shows its advantage the best at low target data settings. A positive transfer effect can be seen in the majority of cases at 2 target demonstrations, while there is not a significant effect in the majority of cases at settings when more target demonstrations are allowed.

The accuracy advantage gained from transfer can be as high as 20%, as seen in the Oct1 Close - Open case at 2 target demonstrations. We see much higher advantage in Close - Open pairs than in Close - Pour or Open - Pour as expected. We deem this to be due to the different object states encountered in these skill pairs, as mentioned earlier. One unexpected result is that the Drawer object did not show a significant effect from transfer between Close and Open skills. It is unclear why

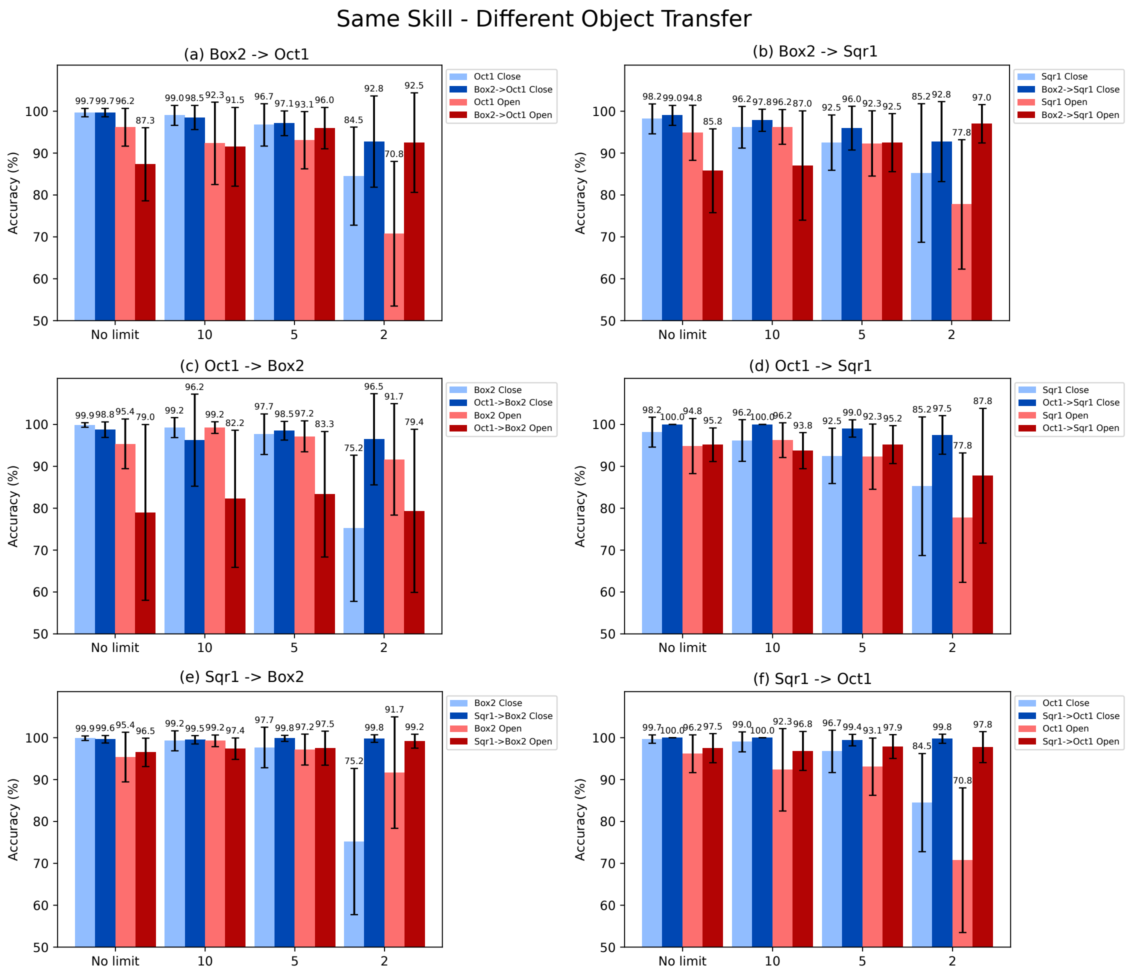


Figure 8.3: Same skill - different object transfer learning in keyframe setting accuracy results at different target data limitations. Plot format is the same as the skill transfer plot in Figure 8.2. Light colored bars represent models trained only on the target data whereas their dark colored counterparts represent the models trained on a combined set of source and (limited) target data. Difference between the dark and light bars of the same color show the advantage gained by applying transfer.

that is the case, but the effect is still in the positive direction even if very small.

8.2 Same Skill - Different Object

8.2.1 Method

In order to attain positive transfer in the same skill - different object case, we expect that there is sufficient similarity between the visual states and state transitions that the objects undergo during the skill. The procedure we propose for this transfer

Data limit	**-**	**10**	**5**	**2**
Positive Transfer	0	0	1	11
Negative Transfer	3	2	1	1
No significant effect	9	10	10	0

Table 8.2: Number of positively or negatively effected cases at different target data amounts for the same skill - different object transfer.

task is simply:

- Combine the data from source and target objects.
- Train the model on the combined data.

Since the source and target problems are solved simultaneously, this procedure more formally falls under the category of multi task learning than transfer learning. However, we place it under the concept of transfer for practical purposes.

8.2.2 Results

We test this method between three objects in the keyframe dataset on both open and close skills. While these objects vary considerably in shape and size, the way they open and close has enough similarity to be potentially exploited by transfer learning.

The overall effect of this transfer method at different target data limits is summarized in Table 8.2. Figure 8.3 shows the monitoring accuracies of each case in transfer and no transfer scenarios for comparison. It can be seen that the effect is overwhelmingly positive at the lowest target data setting, and not so much otherwise. At higher target data settings, it seems that there is simply no need for transfer as the monitoring performance does not show any significant fall compared to the full data setting. This result is similar to the result of same object different skill transfer scenario.

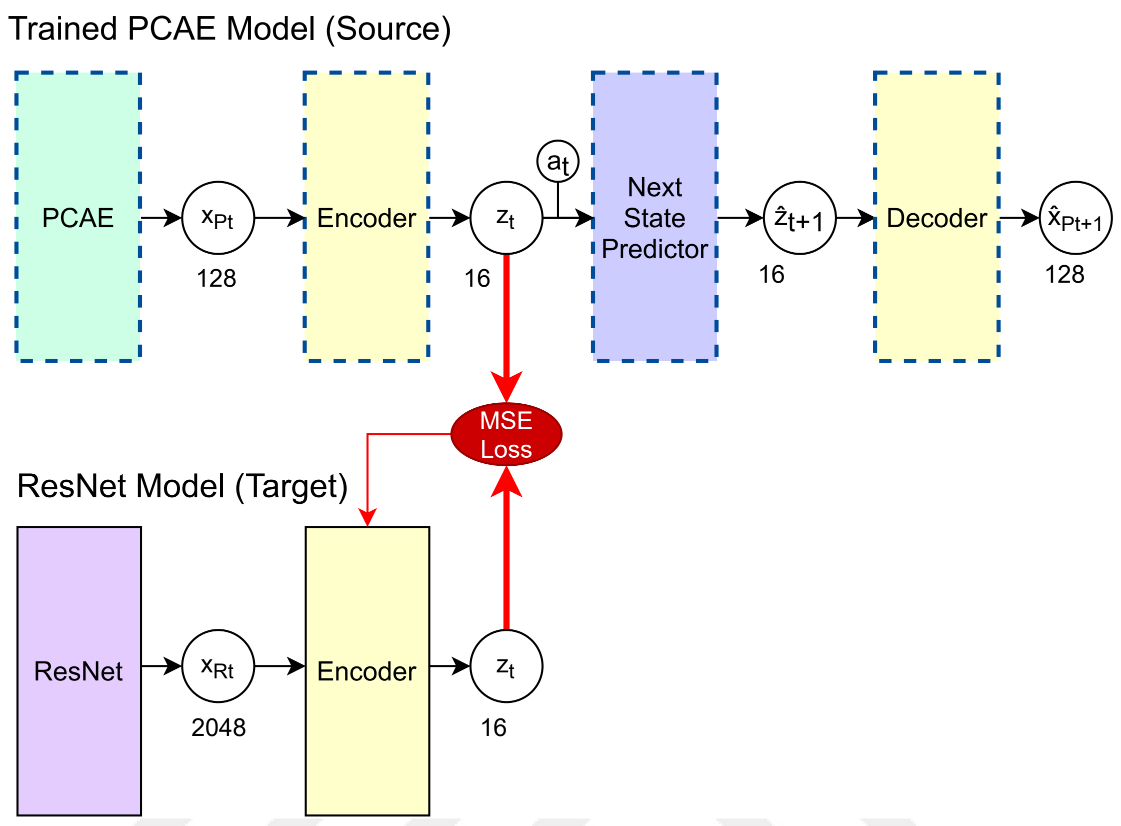


Figure 8.4: The proposed method for domain adaptation between PCAE and ResNet features.

8.3 Perceptual Domain Adaptation

8.3.1 Method

Transfer learning can be done between different perceptual domains. The perceptual domain is determined by the sensor and the used skill-agnostic features. Different sensor types come with different pros and cons, hence the motivation for transferring between them. We describe a procedure for this kind of domain adaptation and evaluate the results of adapting from PCAE features to ResNet features.

Getting PCAE features requires taking point clouds from a depth camera and segmenting the object before feeding it into the encoder. On the other hand, getting ResNet features requires an RGB camera and no segmentation is needed. While depth cameras are becoming more available, 2D RGB cameras are still more common and affordable in comparison. However, the monitoring performance is quite low when ResNet features are used directly. By transferring knowledge from point cloud features to RGB features, one can get much better monitoring performance with a 2D camera than is possible by training only with 2D data. Then the goal model can be used to do monitoring and RL without having access to the sensor used during demonstration recording. The procedure for domain adaptation depicted in Figure 8.4 can be described as follows:

- Prepare a set of demonstrations in both the source and target domains. In the case of PCAE and ResNet, we project the colored point clouds in order to get RGB images that can be fed into ResNet.
- Train a model on the source domain data
- Randomly initialize a new Encoder module with the same output dimensionality but with the input dimensionality of the target domain.
- Train the target Encoder to mimic the output of the source Encoder for the corresponding data via a mean square error loss.

A similar procedure for transfer learning is noted in [François-Lavet et al., 2019]. Just like our method, they propose a modular architecture, enabling these kinds of transfer techniques.

8.3.2 Results

The monitoring accuracy results from applying domain adaptation from PCAE features to ResNet features can be seen in Figure 8.5. Experiments are done in different target training data limitation settings in the same way as the previous two transfer learning experiments.

It can be seen that simply training in the ResNet feature domain results in much lower monitoring accuracy than training in the PCAE feature domain. Applying domain adaptation pulls the monitoring performance up to levels close to training in the PCAE domain. We see that this method shows an advantage in every case and data limit setting tested, and that the advantage is often quite significant.

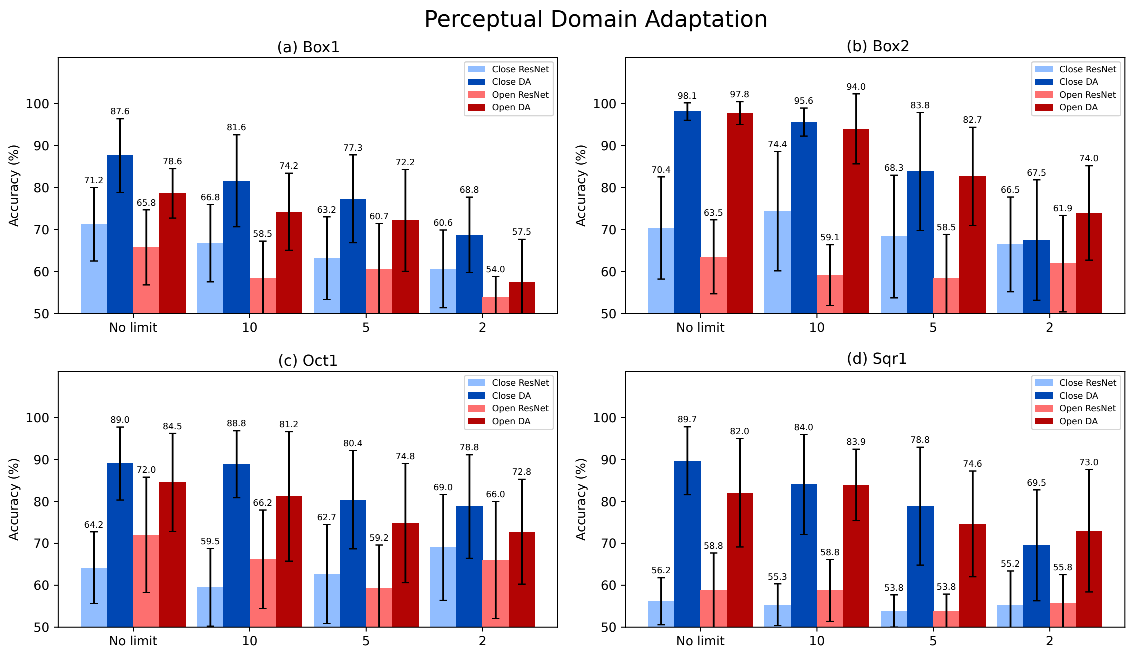


Figure 8.5: Domain adaptation from PCAE to ResNet accuracy results at different target data limitations. Plot format is the same as the skill and object transfer plots in Figures 8.2 and 8.3. Light colored bars represent models directly trained on the ResNet features whereas their dark colored counterparts represent the domain adapted models from PCAE features to ResNet features. Difference between the dark and light bars of the same color show the advantage gained by applying domain adaptation.

8.3.3 Conclusion

In this chapter, we presented three methods to achieve transfer learning by making use of our proposed model in different ways. The three settings are; between same object - different skills, between same skill - different objects, and between different perceptual domains. We showed that all three methods demonstrate positive transfer when there is a low amount of target training data, emphasizing the effect of transfer learning for alleviating low-data problems. In addition, the perceptual domain adaptation experiment showed positive transfer in all target data amount settings.

Chapter 9

CONCLUSION

We propose a neural network architecture to be used for skill-specific representation learning from few demonstrations in order to learn a better goal model. The motivation is the fact that a better goal model has an increased skill monitoring performance, which will in turn be useful to the robot after learning. Additionally, we aim for an improved RL performance by way of reward learning from the goal model, which would lead to better skill execution policies. The proposed architecture learns in a self-supervised manner using predictive and contrastive approaches. It is made up of simple pieces, conductive to effective learning in a low-data regime. In addition, the modularity of the model enables techniques for various transfer learning scenarios.

We compare our method with the currently used PCA, and a different candidate method UMAP. In order to test monitoring performance in a thorough fashion, we create a dataset of demonstrations gathered from multiple objects, skills, and teachers. On this dataset, the proposed model exhibits superior monitoring performance in both keyframe and continuous trajectory settings.Next, we conduct RL experiments in continuous and keyframe settings, and conclude that while our model works well in both, it shows a definitive advantage in the keyframe setting. Finally, we present procedures for three distinct transfer learning settings between skills, objects and visual sensors. Experimental results show that there is a huge benefit to be gained from transfer in each setting, especially when the available target data is very small.

9.1 Future work

One possible direction is to incorporate monitoring and reward modeling as part of our model, removing the need for separate goal and reward models to be learned on top. This can open up avenues by adding more flexibility and homogeneity to the overall framework.

One can also attempt model-based reinforcement learning strategies in the latent space of our model, using NSP as the environment model. It may require a more complete modeling of the environment than is needed in our current task and as a result need more data to be trained, but it is a direction that can lead to interesting outcomes.

While in this work we use only positive samples for training, one may try to use near-miss samples as well. Near-miss samples may be used for an arguably better threshold selection method than the current one, at the cost of user having to give one or a few near-miss demonstrations.

In the current framework, action modeling is done by generating trajectories at the start, and perceptual input is not used. Incorporating perception as part of action modeling is a possible direction of future work.

The proposed domain adaptation procedure in Section 8.3 requires the demonstrations to be recorded in multiple modalities at the same time, which may limit its use cases. A future direction may be to loosen this constraint through alignment or another approach. This can open up the procedure's use-cases to a wider set of problems including transfer between different environments, and transfer between different objects (as an alternative to the current object transfer procedure).

While in this work we utilize the model in a learning from demonstrations setting, it is a generic structure that can be employed in a variety of sequential domains. Applications of this model in different problem domains is a possible area of future investigation.

BIBLIOGRAPHY

[Achlioptas et al., 2018] Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. (2018). Learning representations and generative models for 3D point clouds. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 40–49. PMLR.

[Akgun et al., 2012] Akgun, B., Cakmak, M., Yoo, J. W., and Thomaz, A. L. (2012). Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398.

[Akgun and Thomaz, 2016] Akgun, B. and Thomaz, A. (2016). Simultaneously learning actions and goals from demonstration. *Autonomous Robots*, 40(2):211–227.

[Akgun and Thomaz, 2015] Akgun, B. and Thomaz, A. L. (2015). Self-improvement of learned action models with learned goal models. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5259–5264. IEEE.

[Anand et al., 2019] Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M.-A., and Hjelm, R. D. (2019). Unsupervised state representation learning in atari. *ArXiv*, abs/1906.08226.

[Babaeizadeh et al., 2017] Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. (2017). Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*.

[Bai et al., 2020] Bai, J., Wang, W., Zhou, Y., and Xiong, C. (2020). Representation learning for sequence data with deep autoencoding predictive components. *arXiv preprint arXiv:2010.03135*.

[Bayer and Osendorfer, 2014] Bayer, J. and Osendorfer, C. (2014). Learning stochastic recurrent networks. *ArXiv*, abs/1411.7610.

[Buesing et al., 2018] Buesing, L., Weber, T., Racanière, S., Eslami, S., Rezende, D. J., Reichert, D. P., Viola, F., Besse, F., Gregor, K., Hassabis, D., and Wierstra, D. (2018). Learning and querying fast generative models for reinforcement learning. *ArXiv*, abs/1802.03006.

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

[Chen et al., 2020] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. E. (2020). A simple framework for contrastive learning of visual representations. *ArXiv*, abs/2002.05709.

[Chung et al., 2015] Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. (2015). A recurrent latent variable model for sequential data. In *NIPS*.

[Chung et al., 2020] Chung, Y.-A., Tang, H., and Glass, J. (2020). Vector-quantized autoregressive predictive coding. In *INTERSPEECH*.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[Devin et al., 2017] Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. (2017). Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2169–2176.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Doersch et al., 2015] Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430.

[Doersch and Zisserman, 2017] Doersch, C. and Zisserman, A. (2017). Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060.

[Donahue et al., 2016] Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

[Eteke et al., 2020] Eteke, C., Kebüde, D., and Akgün, B. (2020). Reward learning from very few demonstrations. *IEEE Transactions on Robotics*.

[Finn et al., 2016] Finn, C., Goodfellow, I., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. *Advances in neural information processing systems*, 29:64–72.

[Fitzgerald et al., 2021] Fitzgerald, T., Goel, A., and Thomaz, A. (2021). Abstraction in data-sparse task transfer. *Artificial Intelligence*, page 103551.

[Fortuin et al., 2019] Fortuin, V., Hüser, M., Locatello, F., Strathmann, H., and Rätsch, G. (2019). Som-vae: Interpretable discrete representation learning on time series. *ArXiv*, abs/1806.02199.

[Fraccaro et al., 2016] Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. (2016). Sequential neural models with stochastic layers. In *NIPS*.

[Franceschi et al., 2020] Franceschi, J.-Y., Delasalles, E., Chen, M., Lamprier, S., and Gallinari, P. (2020). Stochastic latent residual video prediction. In *International Conference on Machine Learning*, pages 3233–3246. PMLR.

[François-Lavet et al., 2019] François-Lavet, V., Bengio, Y., Precup, D., and Pineau, J. (2019). Combined reinforcement learning via abstract representations. In *AAAI*.

[Ghosh et al., 2019] Ghosh, D., Gupta, A., and Levine, S. (2019). Learning actionable representations with goal-conditioned policies. *ArXiv*, abs/1811.07819.

[Gidaris et al., 2018] Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Autoencoders. In *Deep Learning*, chapter 14, pages 499–523. MIT press.

[Goyal et al., 2017] Goyal, A., Sordoni, A., Côté, M.-A., Ke, N., and Bengio, Y. (2017). Z-forcing: Training stochastic recurrent networks. In *NIPS*.

[Gregor and Besse, 2019] Gregor, K. and Besse, F. (2019). Temporal difference variational auto-encoder. *ArXiv*, abs/1806.03107.

[Gregor et al., 2019] Gregor, K., Rezende, D. J., Besse, F., Wu, Y., Merzic, H., and van den Oord, A. (2019). Shaping belief states with generative environment models for rl. *ArXiv*, abs/1906.09237.

[Ha and Schmidhuber, 2018] Ha, D. R. and Schmidhuber, J. (2018). World models. *ArXiv*, abs/1803.10122.

[Hafner et al., 2019] Hafner, D., Lillicrap, T., Fischer, I. S., Villegas, R., Ha, D. R., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. *ArXiv*, abs/1811.04551.

[Haresamudram et al., 2020] Haresamudram, H., Beedu, A., Agrawal, V., Grady, P., Essa, I., Hoffman, J., and Plötz, T. (2020). Masked reconstruction based self-supervision for human activity recognition. *Proceedings of the 2020 International Symposium on Wearable Computers.*

[Hausman et al., 2018] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations.*

[He et al., 2020] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. B. (2020). Momentum contrast for unsupervised visual representation learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Hjelm et al., 2018] Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. (2018). Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670.*

[Hoffman et al., 2013] Hoffman, M., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *ArXiv*, abs/1206.7051.

[Ijspeert et al., 2002] Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Learning attractor landscapes for learning motor primitives. Technical report.

[Kaiser et al., 2019] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374.*

[Kaiser and Bengio, 2018] Kaiser, L. and Bengio, S. (2018). Discrete autoencoders for sequence models. *ArXiv*, abs/1801.09797.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

[Kipf et al., 2020] Kipf, T., van der Pol, E., and Welling, M. (2020). Contrastive learning of structured world models. *ArXiv*, abs/1911.12247.

[Krishnan et al., 2015] Krishnan, R., Shalit, U., and Sontag, D. (2015). Deep kalman filters. *ArXiv*, abs/1511.05121.

[Larsson et al., 2017] Larsson, G., Maire, M., and Shakhnarovich, G. (2017). Colorization as a proxy task for visual understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6874–6883.

[Lee et al., 2018] Lee, A. X., Zhang, R., Ebert, F., Abbeel, P., Finn, C., and Levine, S. (2018). Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*.

[Lee et al., 2017] Lee, H.-Y., Huang, J.-B., Singh, M., and Yang, M.-H. (2017). Unsupervised representation learning by sorting sequences. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[Lima and Teichrieb, 2016] Lima, J. P. and Teichrieb, V. (2016). An efficient global point cloud descriptor for object recognition and pose estimation. In *Proceedings of the 29th SIBGRAPI - Conference on Graphics, Patterns and Images*, Sao Jose dos Campos, Brazil.

[Lotter et al., 2016] Lotter, W., Kreiman, G., and Cox, D. (2016). Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*.

[Lynch et al., 2020] Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. (2020). Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR.

[Makhzani et al., 2015] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*.

[McInnes et al., 2018] McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

[Minderer et al., 2019] Minderer, M., Sun, C., Villegas, R., Cole, F., Murphy, K., and Lee, H. (2019). Unsupervised learning of object structure and dynamics from videos. *arXiv preprint arXiv:1906.07889*.

[Misra et al., 2016] Misra, I., Zitnick, C. L., and Hebert, M. (2016). Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer.

[Neitz et al., 2018] Neitz, A., Parascandolo, G., Bauer, S., and Schölkopf, B. (2018). Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *NeurIPS*.

[Noroozi and Favaro, 2016] Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer.

[Oord et al., 2018] Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

[Oprea et al., 2020] Oprea, S., Martinez-Gonzalez, P., Garcia-Garcia, A., Castro-Vargas, J. A., Orts-Escolano, S., Garcia-Rodriguez, J., and Argyros, A. (2020). A review on deep learning techniques for video prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Pathak et al., 2017] Pathak, D., Girshick, R., Dollár, P., Darrell, T., and Hariharan, B. (2017). Learning features by watching objects move. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2701–2710.

[Pathak et al., 2016] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544.

[Pertsch et al., 2020] Pertsch, K., Lee, Y., and Lim, J. J. (2020). Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*.

[Ranzato et al., 2007a] Ranzato, M., Boureau, Y.-L., LeCun, Y., et al. (2007a). Sparse feature learning for deep belief networks. *Advances in neural information processing systems*, 20:1185–1192.

[Ranzato et al., 2007b] Ranzato, M., Poultney, C., Chopra, S., LeCun, Y., et al. (2007b). Efficient learning of sparse representations with an energy-based model. *Advances in neural information processing systems*, 19:1137.

[Rusu et al., 2010] Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J. (2010). Fast 3d recognition and pose using the viewpoint feature histogram. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2155–2162.

[Serban et al., 2017] Serban, I., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A. C., and Bengio, Y. (2017). A hierarchical latent variable encoder-decoder model for generating dialogues. In *AAAI*.

[Sharma et al., 2019] Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2019). Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*.

[Srinivas et al., 2020] Srinivas, A., Laskin, M., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. In *ICML*.

[van den Oord et al., 2017] van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017). Neural discrete representation learning. In *NIPS*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

[Wang et al., 2020] Wang, W., Tang, Q., and Livescu, K. (2020). Unsupervised pre-training of bidirectional speech encoders via masked reconstruction. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6889–6893. IEEE.

[Watter et al., 2015] Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. A. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*.

[Yan et al., 2019] Yan, Q., Guo, S., Chen, D., Yang, Z., and Chen, F. (2019). Transferable environment model with disentangled dynamics. *IEEE Access*, 7:106848–106860.

[Yang et al., 2018] Yang, Y., Feng, C., Shen, Y., and Tian, D. (2018). Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215.

[Zhang et al., 2018] Zhang, A., Satija, H., and Pineau, J. (2018). Decoupling dynamics and reward for transfer learning. *ArXiv*, abs/1804.10689.

[Zhao et al., 2019] Zhao, Y., Birdal, T., Deng, H., and Tombari, F. (2019). 3d point capsule networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1009–1018.

[Zhuang et al., 2021] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2021). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.