Learning by Watching and Learning by Doing

Daniel Gordon

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Ali Farhadi, Chair

Dieter Fox, Chair

Roozbeh Mottaghi

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Learning by Watching and Learning by Doing

Daniel Gordon

Co-Chairs of the Supervisory Committee:
Professor Ali Farhadi
Computer Science and Engineering

Professor Dieter Fox
Computer Science and Engineering

When we are babies, we learn how to see by watching how the world changes and by interacting with it. Can we use these same signals to train vision models? In this thesis, we outline several works which use these paradigms as a basis for learning algorithms. First, we explore learning by watching in which video data is directly used to learn about the visual world. Second, we tackle multiple challenging tasks in embodied environments in which agents learn by interacting with their surroundings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisors Ali Farhadi and Dieter Fox who have helped me immensely through grad school, especially through times of self-doubt. Their guidance, not only with research but also with life, has been extremely helpful and sincere. They have always had my best interests at heart. I would additionally like to acknowledge the members of my committee, Roozbeh Mottaghi and Ming-Ting Sun. Robert Pless also deserves special mention for mentoring me as an undergraduate and convincing me to go to grad school in the first place.

I would also like to thank my numerous labmates, all of whom inspire me to work harder and get smarter to keep up with their amazing talent. Every single one of them has helped me with research questions and to a person they have been kind and supportive throughout my time at the University of Washington.

Additionally, I would like to thank all my collaborators both at the University and elsewhere for giving me outside perspectives on research, and for helping me grow as a researcher and as a collaborator myself.

Next, I would like to acknowledge the support I've received from the University, from the Wissner-Slivka fellowship, the ARCS foundation, and Nvidia for funding all the whimsical ideas I had during my Ph.D.

Finally, I would like to thank my parents, my family, and my cats for their endless support, encouragement, and confidence in me.

DEDICATION

To Grandma Lois, Moose Tracks, and Allegro.

Chapter 1
INTRODUCTION

When we are babies, we learn about the world by watching it change over time. As we grow, we learn to interact with the world and adapt it to our needs. In general, humans learn to perceive and understand the world mostly by watching others and interacting with the environment. This process almost never resembles the snapshot-like supervised learning setup of tasks like the ImageNet challenge [45]. The flexibility of learning by observing and interacting with the world allows us to acquire knowledge which may be hard to explain but is nonetheless extremely useful; no one tells us how to segment and objects, we figure it out intuitively. When training artificial agents about the visual world, we can use the same organization to structure the learning setup.

We categorize these high-level approaches of learning about the visual world into three camps: **Single Image** learning like that in ImageNet, **Video** learning from observing raw video data, and **Active** learning by interacting with a virtual world via visual simulation environments. Each method provides its own set of benefits and drawbacks.

Single Image Learning: Image data is prevalent and fairly straightforward to label. Yet the data is somewhat unrealistic in that the world is dynamic. Techniques trained on single images are often applied frame-by-frame (without any temporal reasoning) on sequential visual data which ignores all temporal relationships between frames.

Video Learning: Video data is also prevalent, however these datasets generally take significantly more resources to label (*i.e.* data diversity requires many videos, but the task may require per-frame labels). Additionally, neither single-image nor video learning can infer how to affect the world since the datasets are static. In some cases, such as object tracking, network outputs like crop locations in the previous frame may affect the future inputs, but the underlying data remains

Single Image Learning	Video Learning	Active Visual Learning
	Example Tasks	
Image Classification	Activity Recognition	Point-Goal Navigation
Object Detection	Video Object Detection	Semantic Navigation
Semantic Segmentation	Object Tracking	Active Tracking
Visual Question Answering	Video Question Answering	Interactive Question Answering
Image Captioning	Video Captioning	Vision and Language Navigation
	Pros and Cons	
- Passive	- Passive	+ Active
- Data is static	= Outputs may affect future inputs	+ Data changes based on decisions made
- Independent decision-making	+ Sequential decision-making	+ Sequential decision-making
- Single-Image Understanding	+ Multi-Image Understanding	+ Multi-Image Understanding
+ Data is easy to gather	+ Data is easy to gather	- Environments are costly to create
- Data is expensive to label	-- Data is very expensive to label	++ Simulator provide dense labels
++ Data diversity is very high	+ Data diversity is high	- Data diversity is low
+ Trained on real visual data	+ Trained on real visual data	- Often trained on simulated visual data

Table 1.1: **Comparison of Passive and Active Visual Learning.** We list several tasks which fall into each camp as well as pros and cons for both types of tasks/datasets.

unchangeable. Visual skills learned from diverse static data are likely to be broadly applicable, but may be limited in scope to understanding the world without knowing how to use this understanding.

Active Visual Learning: Conversely, visual virtual environments are expensive to create but allow for a much richer set of interactions. This provides the ability to learn high-level reasoning and long-term planning but sacrifices some generalization capability.

The work in this thesis explores these trade-offs. More example tasks and trade-offs can be seen in Table 1.1. We seek to strike a balance between these various tasks. For example, in Chapter 2, we use large collections of unlabeled video data to learn high-level representations which allows us to maximize data diversity with 0 labeling cost. Conversely, when learning in simulation environments, we restructure our network architectures to avoid overfitting and maximize the generalization capabilities of our models. Below, we give an overview of common ways of leaning from videos and learning from interactions as well as how we employ them in our works.

1.1 Learning From Videos

Video data is abundant. 500 hours of YouTube videos are uploaded every minute [204]. Video captures the temporal nature of the real world which single images lack. Because of this, video is in many ways the most natural form of data for computer vision. Virtually every single-image task can be applied to videos [178, 181, 211], however not all video tasks apply to single images, such as tracking or video summarization. Yet these video datasets are usually significantly less popular than their single-image counterparts simply due to scale issues. Consider two options. In one case, 1 million random images are labeled with some property (object detection for example). In another case 1 million frames from 1000 videos (1000 frames per video) are labeled with the same property. We would likely expect a method trained on the first dataset to generalize much better than one trained on the second simply because it has seen a much larger variety of data. This is frequently the issue with video datasets: they lack diversity and scale because the labeling cost would be too high. The end results is that most researchers opt to use single-image datasets and are thus forced to ignore temporal consistency. However, numerous real-world challenges such as occlusions and motion blur are simply unsolvable on a single-frame level. Applying temporally-aware techniques in these domains can yield significantly better results [57, 139, 229].

1.1.1 Recurrent Neural Networks

Many techniques have been developed for applying convolutional neural networks to video data. Foremost among them is using recurrent neural networks (RNNs) to accumulate understanding over time. RNNs are a natural fit for video data because of their sequential nature. The most basic RNN can be written as:

$$h^t = \phi(\mathbf{W}[x^t; h^{t-1}] + b) \tag{1.1}$$

where t is the sequence index, x^t is the current input, h^{t-1} is the previous output, $[;]$ is the concatenation operator, $\mathbf{W}$ is a learned weight matrix, b is a learned bias vector, and ϕ is some nonlinear "squashing" function (often the hyperbolic tangent). Due to the squashing function ϕ, this formulation suffers from vanishing gradients for long sequences. Basic RNNs are never used in practice

for this reason. Instead, variants such as LSTMs [90] and GRUs [33] have been developed. Both of these methods learn multiplicative "gates" via the sigmoid function (which ranges from 0 to 1), allowing a fraction of the original data to be let through and combined with some prior memory state rather than squashing the entire output. This prevents the features from being squashed multiple times, eliminating the vanishing gradient problem. Recurrent neural networks may be applied to vector inputs as in Chapters 3 and 6, but they may also be applied spatially by transforming the matrix multiplications into convolutions [193].

1.1.2 Late Fusion

An alternative to the above methods is to simply concatenate multiple images together across the channel dimension. For example two $256 \times 256 \times 3$ images become one $256 \times 256 \times 6$ tensor which is input into a network. Doing this before input (known as Early Fusion) restricts the receptive field of the features which can be learned across the temporal axis. Instead, many works opt for some sort of Late Fusion [19, 50, 98, 247] including networks which cross domains such as RGBD networks [54]. We find late fusion to be effective for tracking and employ it in Chapter 3, though we use skip connections to retain both low-level and high-level visual features.

At the extreme, the features are never concatenated, but are instead compared using simple mathematical operations which enforces structure over the learned latent space [170]. We use this technique in Chapter 2 as well as for the ego-motion loss in Chapter 5.

1.1.3 3D Convolution

Another common method for handling video data is to use "3D convolutions" (which are actually 4D - $T \times C \times H \times W$) [25, 217]. In contrast to simply concatenating multiple images over the channel dimensions, this retains the temporal axis for the entire network structure, allowing for both low-level and high-level features to be learned. As the network deepens, the temporal receptive field expands allowing for long-term connections to be learned. 3D convolutions have often proven simpler to train, although they require a larger computational and memory (GPU RAM)

burden than RNNs. Using multiple streams for 3D CNNs such as one at short temporal scales and one at fast [57], one video and one optical flow [25], and one spatial and one temporal [198] have all proven effective, frequently outperforming RNN-based methods. Still, these methods are limited in that they cannot learn over very long time-scales due to their fixed width nature. Although 3D convolutions have gained in popularity over RNNs, we do not use them in our works on long, variable-length sequences such as for tracking or for long-term embodied tasks.

1.2 Learning From Interaction

Learning from static data can only go so far. In contrast to passive learning, Active Visual Learning requires an agent to directly affect the world, changing its environment via physical interactions. To facilitate large-scale and repeatable learning, research in this space relies extensively on simulation environments. Using these environments, we are able to create agents with a wide variety of behaviors. Like babies, agents learn by interacting with the world, viewing the result, and then adjusting their understanding. This enables a scope of learning much beyond single independent decision-points such as Interactive Question Answering or Visual Task Planning.

In interactive scenarios, it is often difficult to specify labels. For example finding the optimal action in many seemingly-simple video games such as Super Mario, and Tetris are all known to be NP-Hard [8, 44]. It is frequently significantly easier to specify a reward function based on if some goal was achieved. The shift from specifying labels to specifying rewards requires a similar shift from supervised learning to reinforcement learning (RL).

In supervised learning, the goal of an algorithm is to find parameters which minimize some loss function (often equivalent to minimizing errors in the labels). In reinforcement learning, however, the goal is to find a policy which maximizes the reward given to an agent. To simplify this learning process, we often make the Markov assumption which states that given the current state, the future is independent from the past. Thus, algorithms can make decisions solely based on the current state of the agent, rather than needing to consider any prior occurrences. Often this is phrased as

maximizing the expected discounted future reward as estimated by some function.

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}(f(x,\theta),y) \qquad \text{Supervised Learning Objective} \tag{1.2}$$

$$\underset{\pi}{\operatorname{argmax}} \ \sum_{i=0}^{\infty} \gamma \cdot r(p(s_i,\pi(s_i))) \qquad \text{Reinforcement Learning Objective} \tag{1.3}$$

In Equation 1.2, $\mathcal{L}$ is the loss function, f is the learned prediction function, θ is the parameters for the function, x is the input, and y is the label. In Equation 1.3, s_i is the world state at time i, π is the policy which maps from the current state to the current action, p is the probabilistic state-action transition function which maps from the current state and the current action to the next state, r is the reward function, and γ is the discount factor.

Using this reward-based formulation, many tasks which were previously unsolvable have been conquered. Many visually simple yet otherwise complex games such as Go, Starcraft, and DOTA have seen deep RL models surpass human performance [18, 196, 220] without relying on human labels. In the visual domain, success has been somewhat more limited due to several complicating factors. Firstly, observations based off of realistic 3D images (or the real-world) are significantly more complex than 2D video games. Secondly, in many of these games the state is entirely observable, whereas embodied environments with first-person cameras are partially observable. And thirdly, in the case of these games, the training and testing scenarios are exactly the same whereas in embodied tasks, agents must generalize to unseen environments. Still, given significant processing power, large steps have been made on interactive tasks such as point-goal navigation [230].

In our works, we frequently employ RL for solving long-term planning problems, many of which are NP-Complete similar to the Traveling Salesman problem, and all of which are partially observable due to the agent's egocentric view of the environment. Typically, our reward structure encourages success via large rewards, heavily penalizes failure, and provides small time penalties to encourage speedy task completion.

1.3 Thesis Overview

This thesis covers works which use temporal information to learn about the world and works which learn by directly interacting with the environment. It is split into two parts: **Learning By Watching** and **Learning By Doing**. In Part I, we present two methods for using video data to learn about the world. Chapter 2 proposes a method for learning to understand the world by watching it change over time. Chapter 3 presents an object tracking algorithm trained by observing entire video clips rather than single images. In Part II, we explore various methods of learning by interacting with the (virtual) world directly. Chapter 4 and Chapter 5 explore two structures which enforce a decomposition between visual understanding of the environment and using that understanding to accomplish some task in order to better generalize and adapt to new environments. Chapter 6 and Chapter 7 use hierarchical structures to combine low-level visual understanding with reasoning and planning to answer questions about the environment and perform household tasks.

Part I

LEARNING BY WATCHING

In this section, we describe our works on learning by observing how the world changes in videos.

Chapter 2

WATCHING THE WORLD GO BY: REPRESENTATION LEARNING FROM UNLABELED VIDEOS

2.1 Introduction

The world seen through our eyes is constantly changing. As we move through and interact with the world, we see much more than a single static image: objects rotate revealing occluded regions, deform, the surroundings change, and we ourselves move. Our internal visual systems are constantly seeing temporally coherent images. Yet many popular computer vision models learn representations which are limited to inference on single images, lacking temporal context. Visual representations learned from static images will be inherently limited to an understanding of the world as many unrelated static snapshots.

This is especially true of recent unsupervised learning techniques [14, 30, 80, 85, 88, 154, 214, 235], all of which train on a set of highly-curated, well-balanced data: ImageNet [45]. Scaling up single-image techniques to larger, less-curated datasets like Instagram-1B [143] has not provided large improvements in performance [80]. There is only so much that can be learned from a single image: no amount of artificial augmentation can show a new view of an object or what might happen next in a scene. This dichotomy can be seen in Figure 2.1.

In order to move beyond this limitation, we argue that video supplies significantly more semantically meaningful content than a single image. With video, we can see how the world changes, find connections between images, and more directly observe the underlying scene. Prior work using temporal cues has shown success in learning from unlabeled videos [155, 228, 201], but has not been able to surpass supervised pretraining. On the other hand, single image techniques have shown improvements over state-of-the-art by using Noise Contrastive Estimation [75] (NCE). In this chapter, we merge the two concepts with Video Noise Contrastive Estimation (VINCE), a

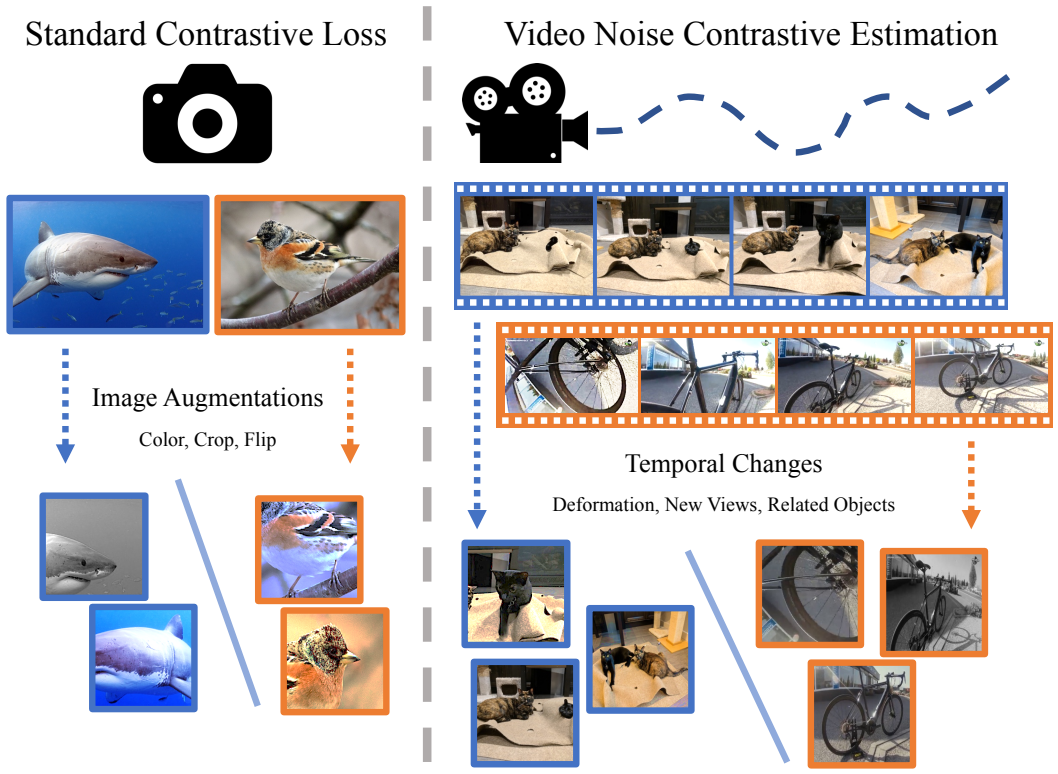


Figure 2.1: **Augmentation Comparison.** The standard unsupervised learning setup learns to separate multiple augmentations of the same image. Our method uses truly novel views and temporal consistency which single images cannot provide.

method for using unlabeled videos as a basis for learning visual representations. Instead of predicting whether two feature vectors come from the same underlying image, we task our network with predicting whether two images originate from the same video. Not only does this allow our method to learn how a single object might change, it also enables learning which things might be in a scene together, e.g. cats are more likely to be in videos with dogs than with sharks. Additionally, we generalize the NCE technique to operate on multiple positive pairs from a single source. To facilitate this learning, we construct Random Related Video Views (R2V2), a set 960,000 frames from 240,000 uncurated videos. Using our learning technique, we achieve across-the-board improvements over the recent Momentum Contrast method [80] as well as over a network pretrained on supervised ImageNet on diverse tasks such as scene classification, activity recognition, and object tracking.[1]

[1]Code and the Random Related Video Views dataset are available at `https://github.com/danielgordon10/vince`.

2.2 Related Work

2.2.1 Noise Contrastive Estimation (NCE)

The NCE loss [75] is at the center of many recent representation learning methods [14, 30, 80, 85, 88, 154, 214, 235]. Similar to the triplet loss [28], the basic principle behind NCE is to maximize the similarity between an anchor data point and a positive data point while minimizing similarity to all other (negative) points.

A challenge for using NCE in an unsupervised fashion is devising a way to construct positive pairs. Pairs should be different enough that a network learns a non-trivial representation, but structured enough that the learned representation is useful for downstream tasks. A standard approach used by [14, 30, 80] is to generate the pairs via artificial data augmentation techniques such as color jitter, cropping, and flipping. Contrastive Multiview Coding [214] uses multiple "views" of a single source image such as intensity (L), color (ab), depth, or segmentation, training separate encoders for each view. PIRL [154] uses the jigsaw technique [164] to break the image into non-overlapping regions and learns a shared representation for the full image and the shuffled image patches. Similarly, Contrastive Predictive Coding (CPC) [85] uses crops of an image as "context" and predicts features for the unseen portions of the image. We provide a more natural data augmentation by using multiple frames from a single video. As a video progresses, the objects in the scene, the background, and the camera itself may move, providing new views. Whereas augmentations on an image are constrained by a single snapshot in time, using different frames from a single video gives entirely new information about the scene. Additionally, rather than restricting our method to only use two frames from a video, we generalize the NCE technique to use many images from a single video, resulting in more computational reuse and a better final representation (AMDIM [14] similarly makes multiple comparisons per pair, but each anchor has only one positive).

2.2.2 Unsupervised Learning Using Video Cues

In contrast with supervised learning which requires hand-labeling, self-supervised and unsupervised learning acquire their labels for free. These techniques can create datasets which are orders

of magnitude larger than comparable fully-supervised datasets. Whereas self-supervised learning requires extra setup during data generation [67, 172, 190], unsupervised learning can use existing data without the need for any specific generation constraints. Unsupervised single image methods such as auto-encoders [121], colorization [245], GANs [68], jigsaw [164], and NCE [235] rely on properties of the images themselves and can be applied to arbitrary image datasets. However these image datasets cannot represent temporal information, nor can they show novel object views or occlusions.

Video data automatically provides temporal cohesion which can be used as additional supervisory signal to learn these phenomena. There is a long history of using videos for low level [136, 150, 201] and high-level tasks [155, 228]. One of the most common unsupervised setups is using the present to predict the future. The Natural Language Processing community has embraced language modeling as an unsupervised task which has resulted in numerous breakthroughs [47, 152, 175]. However, similar systems applied to unlabeled videos have not revolutionized computer vision. These representations still underperform supervised methods due to several issues. Primarily, neighboring video frames do not change nearly as much as neighboring words in a sentence, so a network which learns the identity function would perform well at next frame prediction. Additionally, words are reused and can thus be tokenized in an effective way whereas images never repeat, especially between two disparate video sources.

To avoid these issues, many have opted for other methods. Anand *et al*. [10] use the NCE loss to discriminate between temporally near frames and temporally far frames of ATARI gameplay but do not compare across games. Han *et al*. [77] also use the NCE loss and the CPC technique on a 3D-ResNet to learn spatio-temporal features. Aside from the NCE approach, other works have proposed alternative video training tasks. Misra *et al*. [155] shuffle the frames of a video and train a network to predict whether they are correctly temporally ordered. Wang *et al*. [228] and Vondrick *et al*. [224] use cycle consistency and color as a form of tracking points from one frame onto another. Earlier work from Wang *et al*. [227] uses hand-crafted features to track patches of a video and learn a correspondence between the patches. Our approach is inspired by these works but focuses on learning a semantic representation of the entire scene based on what is present in

a single frame from the video. If a network can consistently represent visually dissimilar images from the same video with similar vectors, then not only has it learned how to recognize what is in each image, but it can also represent what might happen in the past or future of that scene.

2.3 Representation Learning from Videos

In order to learn a semantically meaningful representation, we exploit the natural augmentations provided by unlabeled videos. In this section, we first outline the dataset generation process. We then describe the learning algorithm used to train our representation.

2.3.1 Dataset

Using ImageNet as a basis for representation learning has shown remarkable success both with supervised pretraining as well as unsupervised learning. However, even without labels, the images of ImageNet have been hand selected and are unnaturally balanced. To improve learned representations using existing techniques may require significantly larger datasets [80], but obtaining data with similar properties automatically and at scale is not practical. Instead, we turn to unlabeled videos as a source of additional supervision.

In order to train on a diverse set of realistic video frames, we collect a new dataset which we call Random Related Video Views (R2V2). We use the following fast and automated procedure to generate the images in our dataset. Using this procedure, we are able to construct R2V2 in under a day on a single machine.

1. Use YouTube Search to find videos for a set of queries, and download the top K videos licensed under the Creative Commons. In practice we use the ImageNet 1K classes.

2. Filter out videos with static images using a simple threshold over the percent of pixels which change between two frames. This removes videos of static images, which is common for music uploads.

Dataset	Number of Images (Train)	Number of Videos (Train)	Number of Categories	Mean Image Size
midrule ImageNet 1K [45]	1.3 M	0	1000	(428, 406)
YouTube 8M [4]	0	3.7 M	3862	-
Kinetics 400 [109]	0	0.22 M	400	-
GOT-10k [97]	1.4 M	>9 K	563	(1600, 912)
R2V2 (Ours)	0.96 M	0.24 M	-	(467, 280)

Table 2.1: **Comparison of various image and video datasets.** While we have neither the most images nor the most videos, we provide good diversity between videos which is crucial for learning a strong, generic image representation. GOT-10k [97] training set contains 9,000 video clips, but multiple clips may originate from a single source video.

3. Pick a random point in the video and extract T images with a gap of G seconds between each image. In practice $T = 4$ and $G = 5$.

Using ImageNet synsets for search queries provides reasonable visual diversity, but could be substituted with another set of queries. While we acknowledge that using YouTube's Search feature is not truly random, this procedure resulted in significantly more diverse samples than using existing datasets like YouTube8M [4] which is heavily unbalanced with unnatural videos like "Video Games" and "Cartoons." We do no additional data cleaning to ensure that the videos or extracted images actually contain the search term (many do not), nor do we search for "high interest" video segments as in Misra *et al*. [155]. We also discard the search term itself as a form of supervision. We find that a gap of 5 seconds between each saved image typically results in visually distinct but semantically related images. Too much shorter results in images which are less individually distinct, and too much longer may result in large and unpredictable changes. A sample from each dataset can be seen in Appendix A.3.

We compare R2V2 with other popular datasets in Table 2.1. Because our dataset is constructed automatically, we can easily gather more data (more frames per video, more videos overall). In this chapter we limit the scale to roughly that of comparable datasets.

2.3.2 Noise Contrastive Estimation (NCE) Learning

Given a dataset of diverse video frames, we learn a representation which takes advantage of the structure of the data. We choose the Noise Contrastive Estimation technique [75] which has been popular in many recent works [14, 30, 80, 85, 88, 154, 214, 235], augmented with temporal supervision.

The standard NCE implementation (used in [30, 14, 88]) uses the following procedure. First, a batch of anchor images A are selected. Second, a batch of positive images P are selected, one for each anchor. Positive matches for one example are reused as negatives for the other samples without the need to recompute the features. The NCE loss for a single batch is shown in equation 2.1 where $sim(X, Y)$ is any similarity metric between the two inputs. Gradients flow through the positive pairs (pulling the vectors together) as well as the negative pairs (pushing the vectors away from each other).

$$\mathcal{L}_{\text{NCE}} = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{sim(A_i,P_i)}}{\sum_{j=1}^{n}e^{sim(A_i,P_j)}} \tag{2.1}$$

As in other works, we use the cosine similarity of the feature embeddings of the data points (as seen in equation 2.2) as the similarity metric due to its computational efficiency [14, 30, 80, 85, 154, 235]. The similarity is rescaled by a temperature vector τ to create peaked softmax distributions. f and g are neural networks.

$$sim(X, Y) = \tau * \frac{f(X)}{\|f(X)\|} \cdot \frac{g(Y)}{\|g(Y)\|} \tag{2.2}$$

2.3.3 Multi-Frame NCE

All recent works perform some sort of transformation on a single image to create Anchor-Positive pairs for the NCE loss [14, 30, 80, 85, 154, 235]. We refer to this as "Same Frame." We differ from these works by using multiple images from a single video to form our pairs. This allows our network to see truly different views, deformations, similar objects, and larger scene changes. Additionally, this encodes temporal consistency as the semantic contents of a video are unlikely to

change suddenly. For example in a video of two cats playing, the camera may focus on one cat, or may even pan to a previously unseen dog, but it is unlikely to pan to a shark. Note that in practice, we select frames with replacement, so it is still possible to pair an image with itself, making our potential pairs a strict superset of those in prior works.

2.3.4 Memory Banks and Momentum Contrast

NCE-based methods benefit greatly from large pools of negatives because this increases the likelihood of finding at least one hard negative for each positive example. In some works [154, 235], negatives are sampled from a large memory bank which was filled with earlier outputs of the network. The NCE loss can be modified to use negatives from prior batches as shown in equation 2.3 for a memory bank of negatives $N_{1...m}$.

$$\mathcal{L}_{\text{NCE}} = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{sim(A_i,P_i)}}{e^{sim(A_i,P_i)} + \sum_{j=1}^{m} e^{sim(A_i,N_j)}} \tag{2.3}$$

As the network trains, its output distribution will shift. A potential issue when using a memory bank is the network learns a simple classifier between the current distribution and an old one. Momentum Contrast (MoCo) [80] alleviates this issue by using a quickly updating primary network (f in equation 2.2) and a slowly updating secondary network (g). f is updated based on the NCE loss in equation 2.3 and g is updated using a momentum rule $g \leftarrow \alpha g + (1-\alpha)f$. The memory bank is filled with previous outputs from the slowly changing network g, reducing the likelihood that f will be able to learn a simple recent batch/old batch classifier. For more details, see [80].

2.3.5 Multi-Pair NCE

By using MoCo, we increase the number of useful negatives without a large computational cost. Yet MoCo only uses n positive pairs per batch of size n. We can further increase the number of positives per batch (while holding batch size constant) by simply selecting v videos and k samples from each video where $k = \frac{n}{v}$. By computing the pairwise similarity between each pair, we reuse

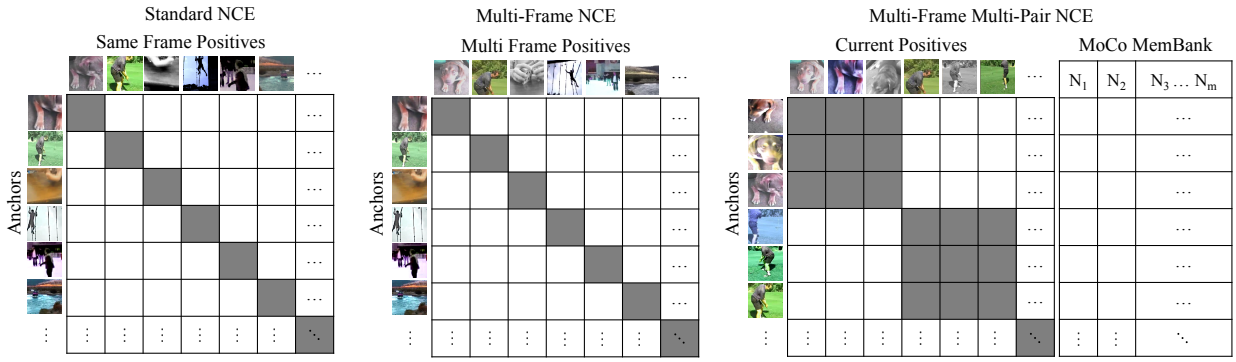


Figure 2.2: **Comparison of NCE Loss Setups.** Left: Standard NCE using "Same Frame" where all correct pairs come from the same image. Middle: Standard NCE using "Multi-Frame" where correct pairs come from the same video. Right: Multi-Frame Multi-Pair NCE which uses more than one positive pair per video, resulting in more positives per batch. The gray boxes indicate the true match pairs. The MoCo Memory Bank adds more negatives for each anchor.

each positive sample k times, resulting in $k^2v = \frac{n^2}{v}$ positives per batch. In the extreme, every sample from a batch could belong to the same class, resulting in n^2 positives, however this causes noisier, more extreme gradients which makes training unstable. Using a simple block-diagonal mask as shown in Figure 2.2 and Algorithm 1, we can efficiently compute the similarities and NCE loss both between elements of the batch and across a memory bank, achieving a large number of positive comparisons per batch while retaining a large negative size. In practice, we notice no meaningful computational cost to this approach. We refer to this full method using Multi-Pair on video data and the Multi-Frame learning procedure as Video Noise Contrastive Estimation (VINCE).

Using clusters of positives has the additional benefit of forcing each feature to match with multiple other features at once. For videos, this means a representation for a single image will be pulled towards some global video feature, resulting in a more consistent representation over a video. For single images, this more strongly enforces invariance to data augmentation.

Algorithm 1 Python-style pseudo code for Multi-Pair NCE.

```
def multi_pair_nce(
        f_output,                                          # [v, k, d] output of f encoder
        g_output,                                          # [v, k, d] output of g encoder
        moco_mem,                                          # [m, d]
        mask,                                              # [n, (n + m)] block diagonal boolean matrix
        temperature):                                      # [1]

    f_output = f_output.reshape(v * k, d)                  # [n, d]
    g_output = g_output.reshape(v * k, d)                  # [n, d]
    compare = concatenate((g_output, moco_mem), axis=0)    # [(n + m), d]
    similarities = matmul(f_output, compare.T)             # [n, (n + m)]
    similarities /= temperature
    pos_similarities = similarities[mask]                  # [n, k]
    neg similarities = similarities[!mask]                 # [n, (n + m - k)]
    exp_pos_sim = exp(pos_similarities)                    # [n, k]
    normalizing_constant = broadcast(                      # [n, k]
        reduce_sum(exp(neg_similarities), axis=1),
                   shape(numerator))))
    score = exp_pos_sim / (exp_pos_sim + normalizing_constant)
    loss = -mean(log(score))
    return loss
```

In practice we use the Log-Sum-Exp trick for numerical stability but omit here for clarity. `broadcast` repeats the input until it matches the provided dimensions. `!mask` flips the booleans of each point in the mask.

2.4 Experiments

We evaluate our method (**VINCE**) on both single-image and temporal tasks by freezing our learned representation and adding a small network (in most cases a single linear layer) for adaptation to new end-tasks. Our learned representation transfers well to a variety of visual tasks, especially tasks which require temporal reasoning. To show this, we compare with multiple strong baselines:

- **MoCo-IN**: Network pretrained on ImageNet [45] using the MoCo algorithm.
- **MoCo-R2V2**: Network pretrained on R2V2 using the MoCo algorithm. This uses exactly the same data as VINCE but the Same Frame technique described in 2.3.3. We also prevent multiple images from the same video being in the MoCo Memory Bank at the same time.
- **Sup-IN**: Network pretrained on fully supervised ImageNet.

To validate the benefits of unsupervised, uncleaned video, we additionally compare against an unsupervised, uncleaned image dataset. Since ImageNet itself required time, effort, and money to

create, we construct a new static image dataset analogous to our video dataset. Specifically, we search Google Images for the ImageNet synsets and download the top K results for each category. We refer to this as **MoCo-G**).

2.4.1 Target Tasks

We compare each method on several diverse end-tasks using both ResNet18 [83] backbones ResNet50 backbones. More training implementation details can be found in A.1. Results for these tasks are shown in Table 2.2. We train all end-task models using Adam [116] and a shared learning rate schedule per task. For each dataset, we use standard data augmentation approaches (crop, flip, color jitter except for tracking). One overall trend to note is the relative gain over MoCo-R2V2. If the single-frame algorithm performed as well as our multi-frame method on temporal tasks, it would indicate minimal temporal understanding. However, the relative gain of VINCE over MoCo-R2V2 on Kinetics (13.85%) and tracking (13.33% and 15.38%) are higher than those on single-frame tasks, showing that our method can incorporate temporal cues.

ImageNet:

For this task, we use our frozen learned representations, adding a single linear layer after the global average pool. Although none of the methods match the fully supervised performance of ResNet18 on ImageNet, they do achieve reasonable performance given only single linear layer. It is unsurprising that MoCo pretrained on ImageNet images (MoCo-IN) outperforms our method (0.447 vs 0.400) due to the domain shift between pretrain and end-task. However MoCo pretrained on R2V2 (MoCo-R2V2) suffers nearly a $2\times$ drop in accuracy (8.9%) compared to our method (4.7%), indicating that pretraining on multi-frame matching provides a clear benefit over single frame pairs. Our method, which has never seen an image from ImageNet before, still learns a representation which generalizes well to this new type of data. Even drawing images from a similar class distribution (MoCo-G) does not outperform our method.

		Image Classification ImageNet[45]	Scene Classification SUN Scenes[238]	Action Recognition Kinetics 400[109]	Tracking OTB 2015[234]	
	Trained Layer(s)	Linear	Linear	LSTM → Linear	1x1 Conv	
	Metric	Accuracy (Top 1)	Accuracy (Top 1)	Accuracy (Top 1)	Precision	Success
ResNet18	Sup-IN	**0.696**	0.491	0.207	0.557	0.396
	MoCo-IN	0.447	0.487	0.336	0.583	0.429
	MoCo-G	0.393	0.444	0.313	0.551	0.413
	MoCo-R2V2	0.358	0.450	0.318	0.555	0.403
	VINCE (Ours)	0.400	**0.495**	**0.362**	**0.629**	**0.465**
	Relative Gain over MoCo-R2V2	11.91%	9.93%	13.85%	13.33%	15.38%
ResNet50	Sup-IN	**0.762**	0.593	0.305	**0.458**	**0.320**
	MoCo-V2-IN (our impl.)	0.652	0.608	0.459	0.300	0.260
	MoCo-R2V2	0.536	0.581	0.456	0.386	0.299
	VINCE (Ours)	0.544	**0.611**	**0.491**	0.402	0.300
	Relative Gain over MoCo-R2V2	1.36%	5.29%	7.72%	4.15%	0.33%

Table 2.2: **Comparison of representation performance across a variety of end tasks.** We show improvements over MoCo trained on the same data on all tasks, and outperform MoCo trained on ImageNet as well as supervised pretraining on ImageNet on all tasks but ImageNet itself (and tracking for ResNet50). Each representation uses the same ResNet convolutional backbone, sharing weights across all tasks. Linear (for Kinetics LSTM → Linear) classifiers are the only learned weights for each end task.

SUN Scenes:

SUN Scenes is a classification dataset in which each image is categorized into one of 397 possible scene types such as airplane cabin, bedroom, and coffee shop. Again we train a single linear layer on top of each pretrained network. This data is quite similar to ImageNet in that each image is well-curated and contains single, unambiguous subjects. As such, the ImageNet fully supervised baseline transfers quite well to SUN Scenes. However VINCE outperforms Sup-IN by a small margin. Again we note a large improvement of VINCE over MoCo-R2V2 (0.495 vs. 0.450).

This shows that our method learns to recognize not just the main subject of an image but also the surrounding scene, which requires a richer understanding of the world.

Kinetics 400:

This dataset consists of 10 second clips from YouTube videos and action labels for each segment. We first download each video and subsample each clip to one frame per second (10 frames per clip). We train a single layer LSTM [90] followed by a single linear layer to predict the action category for each segment. Kinetics acts as a crucial test to evaluate whether our model learns temporal cues. VINCE greatly outperform all other methods, whereas traditional baselines such as fine-tuning supervised ImageNet do not adapt well at all. This shows that contrary to popular belief, representations pretrained on ImageNet many not be a good fit for other visual domains, especially on temporal tasks.

Object Tracking Benchmark (OTB) 2015:

OTB 2015 is a popular tracking dataset. Given an initial bounding box around an arbitrary object in the first image, a model must locate the object in the following frames. We use the SiamFC [19] tracking algorithm on top of our learned representation. SiamFC first crops the initial bounding box and extracts spatial features using a CNN. For each frame, it localizes the object by extracting spatial features on the full frame and convolving the template features with the full image features. This process is similar to template matching [23] but in deep feature space. For a more complete explanation, see [19].

To extract these features, we use the outputs of each model from before the average pooling. We additionally use dilated convolutions rather than strides for the second and third ResNet18 block to preserve spatial information even though the initial representation was pretrained using strides. We add a single 1x1 convolution layer to each representation. As OTB 2015 is only a set of test videos, we train on the GOT-10k dataset [97], a dataset of 9000 training clips and 1.4 million images.

OTB is evaluated using two metrics – precision and success. Precision measures the percentage of frames where the (normalized) center error is less than a certain threshold, using an area-under-the-curve evaluation. Similarly, success measures the percentage of frames where the Intersection over Union is more than a certain threshold, again using the area-under-the-curve.

A representation which works well for SiamFC would have the property that the cross correlation of two images of the same object is high, but the cross correlation of two different objects, or a poorly cropped image of the same object, is low. Pretraining our representations on multiple frames from the same video coincides quite well with the first objective, however since we use cropped data augmentations, the representations tend to be somewhat invariant to poorly-cropped candidates. Still, the models perform quite well across a variety of difficult tracking instances. Our ResNet18 model transfers significantly better than all other methods indicating a clear benefit to using temporal cues during pretraining. We find, somewhat unexpectedly, that the ResNet18 network fares better than the ResNet50. A likely explanation for this is that the original SiamFC method uses AlexNet [126] with no padding in a fully-convolutional manner. When using ResNet, padding must be applied to keep the outputs the same dimensionality as the inputs. Thus, at training time the network may latch onto zero-padding cues which will not be applicable at test time. This becomes more of an issue the larger the receptive field which is why ResNet50 struggles but ResNet18 is somewhat less affected.

2.4.2 Method Ablation

We validate the effectiveness of Multi-Frame (Sec. 2.3.3) and Multi-Pair (Sec. 2.3.5) learning by ablating the number of images from each video used in a batch of comparisons Due to computational constraints, we only perform ablations on ResNet18. The results of this ablation are shown in Table 2.3. The first row is equivalent to the procedure done in MoCo [80] i.e. the anchor and positive pairs are two data augmentations of the same image. The second row uses the MoCo procedure as well, however the anchors and positives may be from different images from the same video. The third row uses our Multi-Pair NCE method taking 4 positives and 4 anchors from each video, resulting in 16 positive pairs. A pictorial representation can be seen in Figure 2.2. Note that

Images Per Video	Test Task ImageNet	SUN Scene	Kinetics 400	OTB 2015 Precision	OTB 2015 Success
1: Same Frame	0.358	0.450	0.318	0.555	0.403
2: Multi-Frame	0.381	0.478	**0.361**	0.622	**0.464**
8: Multi-Frame Multi-Pair	**0.400**	**0.495**	**0.362**	**0.629**	**0.465**

Table 2.3: **Method ablation for VINCE.** We compare using one source image with two augmentations (the standard approach), two different images, or a set of different images. Using Multi-Frame results in a large boost across the board. Multi-Frame Multi-Pair further increases the power of the representation. Note that all methods use the entire dataset, but only Multi-Frame methods use multiple images from a video within one batch.

when selecting images for row 2 and 3, we use sampling with replacement, making our method a strict super-set of MoCo.

We observe across-the-board improvements from both modifications to the MoCo approach. The majority of the improvement comes from using two non-identical frames for matching, but we still gain an additional improvement from using Multi-Pair NCE. Our intuition is that using the Multi-Pair NCE creates gradients that pull each feature towards a global video representation whereas the standard NCE remains more instance-based, only moving a representation in one direction at a time. Thus, we would expect the Multi-Pair NCE features to be more holistically semantic whereas the standard NCE may retain more uniquely identifying features. In fact, we observe a larger performance gap on the more semantic ImageNet and SUN Scene tasks. In contrast, because the Kinetics model uses an LSTM to reason over all input images at once, instance-level features are equally useful as global video features for overall accuracy.

2.4.3 Pretraining Data Ablation

In Table 2.4 we explore the effect of different pretraining datasets on end-task performance. For each experiment, we use VINCE but use video data from three different sources: our method of

	Test Task				
Pretraining Data	ImageNet	SUN Scene	Kinetics 400	OTB 2015 Precision	OTB 2015 Success
R2V2 IN-Queries	**0.400**	**0.495**	0.362	0.629	0.465
R2V2 YT8M URLs	0.367	0.478	0.343	**0.667**	**0.492**
Kinetics 400 URLs	0.368	**0.494**	**0.390**	0.612	0.456

Table 2.4: **Pretraining data ablation for VINCE.** Each method uses exactly the same training setup, only substituting one data source for another. Since R2V2 uses ImageNet search queries, it outperforms the others on ImageNet. Similarly, pretraining on Kinetics 400 videos results in better end performance on Kinetics.

searching ImageNet synset queries, using the URLs from YouTube 8M [4], and using the URLs from Kinetics 400 [109]. Again, we only test ResNet18. Our YouTube8M(YT8M) pretraining data uses the same filtering procedure as R2V2 and contains 5.8 million images from 1.4 million videos. As noted in Table 2.2 MoCo-IN results, using the same dataset for pretraining as the end-task results in a boost in performance on that specific task but does not indicate that the representation will be better on all tasks. We see this trend is true again when pretraining on Kinetics data. Similarly, since R2V2 uses ImageNet synset for search queries, pretraining on it performs better on ImageNet than the other less-aligned datasets.

In general, this would indicate that given a large enough set of diverse videos, pretraining directly on the unlabeled source data would result in the best performing representation on that data. If this is not possible, then pretraining on a large external source of data may still result in a useful representation. It also indicates that the VINCE method works well on a variety of different pretraining datasets. The increased performance on tracking when using YT8M data could be explained by it simply having access to a larger number of video sources and frames. For generic object tracking, class diversity may be less important than number of samples because the class identity is ignored.

Figure 2.3: **Nearest Neighbors.** Results for a sampling of query images from R2V2 and ImageNet using various models. VINCE shows a clear understanding of each image and finds highly relevant neighbors.

2.4.4 Qualitative Results

We additionally provide two qualitative analyses to better understand the success and failure cases of VINCE: Nearest Neighbors, and t-SNE.

Nearest Neighbors:

We additionally query ImageNet Val and a set of test videos for nearest neighbor matches, taking at most one neighbor per video. We visualize the top 5 neighbors for VINCE, MoCo-R2V2, and MoCo-IN in Figure 2.3. We observe that VINCE seems to understand the semantics of an image more than MoCo-R2V2 and MoCo-IN. For instance, although MoCo-R2V2 and MoCo-IN find other control panels and buttons in query 1, they do not make the scene-level connection to car interiors as well as VINCE does. Query 2 shows an interesting quirk case of our method. Rather than matching the semantics of the image, VINCE relies on the news logo as a differentiating feature due to its discriminative nature. Each image in VINCE's query 2 results is from a separate

Figure 2.4: **t-SNE embedding of images from R2V2 test set.**

video, but from the same news source. For the ImageNet queries, despite never seeing ImageNet inputs during pretraining, VINCE is able to find good matches as well as MoCo-IN which was trained using only ImageNet inputs.

t-SNE:

Using a set of held-out video frames, we project the 64-D embedding space from VINCE to 2D using t-SNE [142] and visualize the formed clusters in Figure 2.4. Not only does this assist in verifying the quality of the embedding, it also serves as a visual method for evaluating the diversity of the dataset itself. The largest of the clusters seems to be the face cluster. YouTube is full of

videos of people looking and talking directly to a camera, and our random subset reflects this pattern. Other interesting, yet unexpected clusters emerge as well such as cats (YouTube loves cats), hands (demo videos), and food (cooking videos).

2.5 Conclusions

In this chapter we introduced Video Noise Contrastive Estimation, a process for using unlabeled videos to learn an unsupervised image representation. By training on multiple images from the same video instance, we learn from more natural changes such as deformation and viewpoint change rather than 2D artificial augmentations. To learn from a large variety of diverse video clips, we collect Random Related Video Views in a completely automated fashion. We show across-the-board improvements over the recently proposed MoCo [80] technique on a wide variety of tasks, and we believe Video Noise Contrastive Estimation will extend to other unsupervised methods such as SimCLR [30] and PIRL [154] as well as other end-tasks. As representation learning techniques improve, we believe that videos – rather than images – will prove an invaluable resource for pushing the state-of-the-art forward.

Chapter 3

RE^3: *RE*AL-TIME *RE*CURRENT *RE*GRESSION NETWORKS FOR VISUAL TRACKING OF GENERIC OBJECTS

In the previous chapter, we described a method for learning how to see from scratch using only video cues. In this chapter, we similarly seek to learn the low-level yet important visual task of object tracking, again extracting cues directly from video.

3.1 Introduction

One of the first visual skills babies learn is object tracking - the ability to follow an object as it moves over time [2]. At three months, no one is teaching us how to track, we simply learn by watching objects move. Although this basic skill is innate to humans, we still struggle to create robust artificial object tracking systems. Current generic 2D image tracking predominantly relies on learning a tracker online. A popular paradigm for tracking algorithms is tracking-by-detection: training an object-specific detector, and optionally updating it with the object's new appearance at every frame. While effective, this is somewhat unnatural as it disregards temporal cues like optical flow. Additionally, performing an appearance update takes a significant amount of time and computational resources, and not updating the appearance will accuracy to decrease.

We propose the ***Re*al-time, *Re*current, *Re*gression-based tracker**, or **Re**^{3}: a fast, accurate network for object tracking that addresses these disadvantages. Prior work has shown that a pretrained neural network can learn a robust tracker that functions on previously unseen objects [19, 84]. However, instead of freezing the network as in [19, 84] or adjusting the network parameters via online training [163], Re^3 learns to store and modify relevant object information in the recurrent parameters. By overcoming the need for any re-training, Re^3 efficiently tracks and updates itself simultaneously. This much more closely match the way humans track objects.

By incorporating information from large collections of images and videos, our network learns representations that capture the important features of the tracked object. The goal of this process is to teach the network how any given object is likely to change over time so these transformations can be embedded directly into the network. This shifts the computational burden offline, making Re^3 extremely fast and computationally cheap during inference, an important quality for algorithms operating on mobile robots with limited processing power. Because of our large variety of training data, we found our pretrained network can be directly applied to a variety of new environments such as drone videos, cellphone videos, and robot-mounted platforms, and due to the low computational cost, Re^3 could be run on embedded systems while remaining real-time. Our results show that recurrent networks are well suited for object tracking, as they can be fast, accurate, and robust to occlusions. Re^3 achieves competitive results on multiple tracking benchmarks, showing especially good performance during occlusions, all while running at 150 frames per second.[1]

3.2 Related Work

Object tracking has been studied in great depth by the robotics and computer vision community. In many cases, systems target objects with known 3D models or objects of a limited set of classes. DART [191] requires a depth camera and a predefined articulated model, but produces fine-grained pixelwise labels. Ondruska *et al*. [166] use planar laser scans and a recurrent network to track people under heavy occlusions. Their method succeeds because priors on likely human trajectories are quite strong. KITTI [62], a popular vision and robotics benchmark suite, only tests performance on tracking cars and people. We focus on the harder problem of tracking arbitrary objects given only an initial bounding box. Generic object tracking represents a new challenge for convolutional neural networks. Most deep learning algorithms rely on having millions of examples to function, learning invariance to high-level concepts; object detection algorithms expect the network to learn what a person looks like, but not to differentiate between two people. Trackers, on the other hand, are often given only a single initial example and must specialize in order to track that specific

[1] Code available at `https://gitlab.com/danielgordon10/re3-tensorflow`
Video available at `https://youtu.be/RByCiOLlxug`

target object. Because of the difficulty of adapting traditional deep methods to tracking, deep learning has only recently started to be used in tracking algorithms. In 2015, MDNet [163], a deep method, won the The Visual Object Tracking challenge (VOT) [124] for the first time. The VOT reports [123, 124, 125] present a succinct overview of many other generic object trackers. Those most related to ours can be categorized into three sub-groups: online-trained, offline-trained, and hybrid trackers.

3.2.1 Online-trained trackers

The most prevalent type of trackers operate entirely online, continually learning features of the object of interest as new frames arrive. This includes keypoint-based and part-based trackers [53], correlation based methods [86], and direct classification methods [79]. These methods often rapidly train a classifier to differentiate between the object of interest, the background, and possible occluders. Discriminative Scale Space Tracker (DSST) [35], the winner of the VOT 2014 challenge [123], uses this approach. DSST learns discriminative correlation filters for different scale and translation amounts. Because online trackers must train on frames as they arrive, they tend to directly trade off speed with model complexity.

3.2.2 Offline-trained trackers

The success of deep learning is often attributed in part to its ability to utilize massive amounts of training data better than other machine learning methods. Offline trackers such as [19] and [84] employ this technique to great success. Because they are trained entirely offline, the networks are fast to evaluate at test time, allowing both methods to operate at faster than real-time speeds. However, this underscores a large problem with offline trackers: they do not adapt to what they are seeing. Instead of incorporating information from an entire track, they learn a similarity function between pairs of frames. Held *et al*. [84] use only a single frame history, meaning any amount of occlusion will confuse the tracker. Bertinetto *et al*. [19] rely solely on the initial frame for appearance information and try to detect the object in all subsequent frames, meaning large appearance changes,

even if gradual, would be difficult to track. T-CNN [107] focuses on the detection and tracking problem in video by finding temporally coherent object detections, but they do not adapt the model using visual information from prior detections. Offline trackers' capabilities are fundamentally limited because they cannot adapt to new information.

3.2.3 Hybrid trackers

Hybrid trackers attempt to solve the problems with online and offline trackers by taking the best from both. MDNet, the winner of the VOT 2015 challenge, trained an image classification network offline, and then learned a per-object classifier online [163]. Similar approaches were taken by other top competitors [36]. Still, the complexity of their online training techniques limited their methods to taking seconds to process each frame.

Our contribution, Re^3, is a hybrid tracker, but prioritizes offline learning and limits online adaptation to recurrent state updates. Although we make this trade-off, our method is substantially different from purely offline trackers because we use information from previous frames to make future predictions. This lets us model temporal dependencies between sequential images and reason about occlusions. Other recurrent trackers such as [52] and [60] use attention-based recurrent neural networks. These techniques have only been shown to work on simple datasets such as tracking MNIST digits. To our knowledge, we are the first to demonstrate successful tracking in natural videos using recurrent neural networks.

3.3 Recurrent Object Tracking

Our tracking pipeline, depicted in Figure 3.1, consists of convolutional layers to embed the object appearance, recurrent layers to remember appearance and motion information, and a regression layer to output the location of the object. We train this network on a combination of real videos and synthetic data. At test time, unlike MDNet [163], we do not update the network itself; we instead let the recurrent parameters represent the tracker state which can be updated with a single forward pass. In this way, the tracker learns to use new observations to update the appearance and

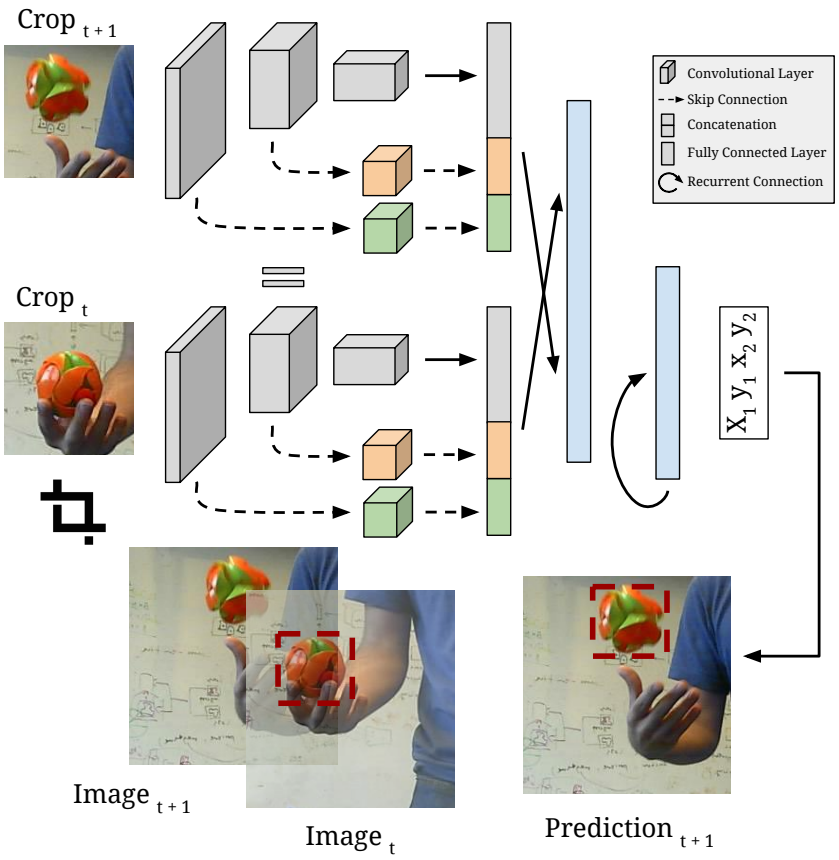


Figure 3.1: **Re^3 Network Structure.** Image crop pairs are fed in at each timestep. Both crops are centered around the object's location in the previous frame, and padded to two times the width and height of the object. Before every pooling stage, we add a skip layer to preserve high-resolution spatial information. The weights from the two image streams are shared. The output from the convolutional layers feeds into a single fully connected layer and an LSTM. The network predicts the top left and bottom right corners of the new bounding box.

motion models, but no extra computational cost is spent on online training.

3.3.1 Object Appearance Embedding

The task of generic object tracking in video sequences starts with an initial bounding box around an object, with the goal of keeping track of that object for the remainder of the video. For each frame of the video, the tracker must locate the object as well as update its internal state so it can continue tracking in future frames. A primary subtask in this framework is translating raw pixels into a higher-level feature vector representation. Many object trackers, like [53] rely on extracting

appearance information from the object pixels using hand-crafted features. We choose to learn the feature extraction directly by using a convolutional pipeline that can be trained fully end-to-end on a large amount of data.

Network Inputs: Similar to [84], at each frame, we feed the network a pair of crops from the image sequence. The first crop is centered at the object's location in the previous image, whereas the second crop is in the *same* location, but in the *current* image. The crops are each padded to be twice the size of the object's bounding box to provide the network with context. This padding offers a reasonable trade-off between speed, resolution, and search region size. If the bounding box at frame j had centers (X_c^j, Y_c^j) and width and height W^j, H^j, both crops would be centered at (X_c^j, Y_c^j) with width and height $2W^j$ and $2H^j$. By feeding a pair of crops, the network can directly compare differences in the two frames and learn how motion affects the image pixels. Though this method does not guarantee the object to be in the crop, if our first crop was in the correct location, the object would have to move more than 1.5 times its width and height in a single frame to be fully out of the crop, which is quite unlikely. The crops are warped to be 227×227 pixels before being input into the network. We experimentally determined that preserving the aspect ratio of the source images hurts performance because it forces the network to directly regress the aspect ratio rather than regress changes to the ratio. The pair of image features are concatenated at the end of the convolutional pipeline (late fusion) rather than at the beginning to allow the network to fully separate out the differences between the two images.

Skip Connections: The hierarchical structure of convolutional networks extracts different levels of information from different layers [242]; the lowest layers of image classification networks output features like edge maps, whereas the deeper layers capture high-level concepts such as animal noses, eyes, and ears [242]. Rather than only using the outputs from the last layer of the network, we represent the object's appearance using low, mid, and high level features. We use skip connections when spatial resolution decreases to give the network a richer appearance model. In this way, the network can differentiate a person (high level concept) wearing a red (low level concept) shirt

from a person wearing a blue shirt.

The skip connections are each fed through their own $1 \times 1 \times C$ convolutional layers where C is chosen to be less than the number of input channels. This reduces the dimensionality of the layers with higher spatial resolutions to keep computational cost low. As the spatial resolution is halved, C is doubled. All skip connection outputs and the final output are concatenated together and fed through a final fully-connected layer to further reduce the dimensionality of the embedding space that feeds into the recurrent pipeline.

3.3.2 Recurrent Specifications

Recurrent networks tend to be more difficult to train than typical feed-forward networks, often taking more iterations to converge and requiring more hyperparameter selection. We present a method of training a recurrent tracking network which translates the image embedding into an output bounding box while simultaneously updating the internal appearance and motion model. We also describe techniques that lead to faster convergence and better-performing networks.

Recurrent Structure: Using the prior work of Greff *et al*. [70], we opt for a two-layer, factored LSTM (the visual features are fed to both layers) with peephole connections. We find that this outperforms a single layer LSTM even given a deeper convolutional network and larger embedding space. The two layer LSTM is likely able to capture more complex object transformations and remember longer term relationships than the single layer LSTM. The exact formulation is shown in Equations 3.1-3.6 where t represents the frame index, x^t is the current input vector, y^{t-1} is the previous output (or recurrent) vector, $\mathbf{W}$, $\mathbf{R}$, and $\mathbf{P}$ are weight matrices for the input, recurrent, and peephole connections respectively, b is the bias vector, ϕ is the hyperbolic tangent function, σ is the sigmoid function, and $\cdot$ is point-wise multiplication. A forward pass produces both an output vector y^t, which is used to regress the current coordinates, and the cell state c^t, which holds important memory information. Both y^t and c^t are fed into the following forward pass, allowing for information to propagate forward in time.

$$z^t = \phi(\mathbf{W}_z x^t + \mathbf{R}_z y^{t-1} + b_z) \quad \text{LSTM input} \quad (3.1)$$

$$i^t = \sigma(\mathbf{W}_i x^t + \mathbf{R}_i y^{t-1} + \mathbf{P}_i c^{t-1} + b_i) \quad \text{Input gate} \quad (3.2)$$

$$f^t = \sigma(\mathbf{W}_f x^t + \mathbf{R}_f y^{t-1} + \mathbf{P}_f c^{t-1} + b_f) \quad \text{Forget gate} \quad (3.3)$$

$$c^t = i^t \cdot z^t + f^t \cdot c^{t-1} \quad \text{Cell state} \quad (3.4)$$

$$o^t = \sigma(\mathbf{W}_o x^t + \mathbf{R}_o y^{t-1} + \mathbf{P}_o c^t + b_o) \quad \text{Output gate} \quad (3.5)$$

$$y^t = o^t \cdot \phi(c^t) \quad \text{LSTM output} \quad (3.6)$$

The output and cell state vectors update as the object appearance changes. Figure 3.2 shows a t-SNE [142] plot of the LSTM states our tracker produces for each frame from the the VOT 2014 [123] videos. Because the LSTM states are initialized to 0 at the start of each video, the embeddings of the first few frames from each track are clustered together. As each video progresses, the LSTM state is transformed, resulting in many long, thin paths that follow the ordering of the frames in the original video. Certain points in the sequences with significant occlusion are circled, demonstrating that are embedding does not change drastically during occlusions even though the image pixels look quite different. The gaps in the sequences are mostly due to fast movement which tend to cause a rapid appearance change in the second crop.

Network Outputs: The second LSTM's outputs are fed into a fully-connected layer with four output values representing the top left and bottom right corners of the object box in the crop coordinate frame, as is done in [84]. By regressing these coordinates, we can directly handle size and aspect ratio changes. Similar to [84], we use an L1 loss on the outputs to encourage exact matches the ground truth and limit potential drift.

Unrolling during training: Recurrent networks generally take many more iterations to converge than comparable feed-forward networks. This is likely because the inputs are all fed in sequentially, and then one or many outputs and losses are produced. This means the loss must propagate through many noisy intermediate states, causing the gradients to fluctuate. However, for tracking, each

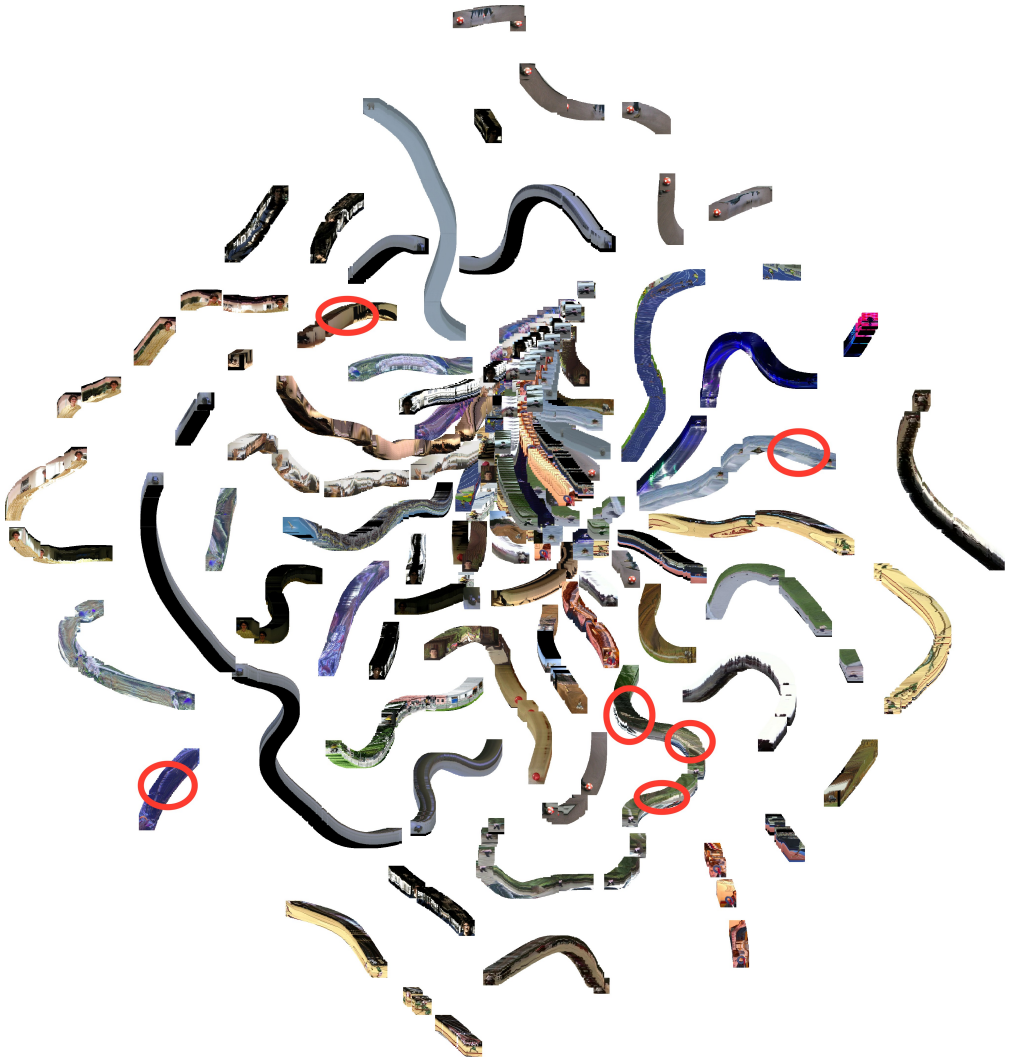


Figure 3.2: **t-SNE embedding of LSTM states from VOT 2014 data.** The cell and output states are concatenated together to form the feature vector. Rather than forming clusters, the states form paths indicating that as the images change during a video, the LSTM states change in a similar fashion. Circled portions of the embedding indicate occlusion.

input is directly paired with an immediate output. Thus, we can use a training curriculum that begins with few unrolls, and slowly increases the time horizon that the network sees to teach it longer-term relationships. Without the shorter unroll step, the network may take exponentially longer to train, or may simply never converge. Specifically, we initially train the network with only two unrolls and a mini-batch size of 64. After the loss plateaus, we double the number of unrolls and halve the mini-batch size until a maximum unroll of 32 timesteps and a mini-batch size of 4. Using this curriculum, we do not find it necessary to clip gradients.

Learning to Fix Mistakes: Recurrent networks are often trained by feeding ground truth outputs into the future timesteps rather than the network's own predictions [59, 221]. However, if we always provide the network with ground-truth crops, at test time it quickly accumulates more drift than it has ever encountered, and loses track of the object. To counteract this, we employ a regime that initially relies on ground-truth crops, but over time the network uses its own predictions to generate the next crops. We initially only use the ground truth crops, and as we double the number of unrolls, we increase the probability of using predicted crops to first 0.25, then subsequently 0.5 and 0.75. This method is similar to the one proposed in [17], however we make our random decision over the whole sequence rather than at every step independently.

3.3.3 Training Procedure

We use a combination of real and synthetic data to train our deep network. This results in our tracker being able to work on a large variety of object types, allowing us to successfully track across multiple datasets.

Training from Video Sequences: We train Re^3 on two large object tracking datasets: the training set from the ILSVRC 2016 Object Detection from Video dataset (Imagenet Video) [181] and the Amsterdam Library of Ordinary Videos 300++ (ALOV) [199]. In its training set alone, Imagenet Video provides 3862 training videos with 1,122,397 images, 1,731,913 object bounding boxes, and 7911 unique object tracks. This is by far the largest object tracking dataset we are aware of, however it only contains videos for 30 object categories. ALOV consists of 314 videos. We do not use the 7 videos that also occur in VOT 2014 [123] in order to avoid training on the test set. The remaining dataset comprises 307 videos and 148,319 images, each with a single object.

Training from Synthetic Sequences: Recently, many deep methods have supplemented their training sets with simulated or synthetic data [84, 177]. Due to the large variety of objects labeled in 'object detection in image' datasets, we construct synthetic videos from still images to show the network new types of objects. We use images from the Imagenet Object Detection dataset to fill

this role [181]. We discard objects that are less than 0.1^2 of the total image area due to lack of detail, resulting in 478,807 object patches.

To generate simulated data, we randomly sample over all images for an object to track. We use random patches from the same image as occluder patches. The full image serves as the background for the scene. The object, background, and occluders are taken from the same image in order to keep our simulated images close to the real image manifold. We then simulate tracks for the object and occluders, at each timestep modifying an initial speed, direction, and aspect ratio with Gaussian noise. This data adds diversity to the types of objects that the network sees, as categories like "person," which are common in many tracking datasets, are absent in Imagenet Video [181].

Tracking at test time: To generate test-time predictions, we feed crops to the network from each sequential frame. After every 32 iterations, we reset the LSTM state. This is necessary because we train on sequences with a maximum length of 32 frames, and without this reset, the LSTM parameters tend to diverge from values the network has seen before. Rather than resetting the LSTM state to all zeros, we use the output from the first forward pass. This maintains an encoding of the tracked object, while allowing us to test on sequences much longer than the number of training unrolls. We also notice that the reset helps the model recover from drifts by using a well-centered crop embedding.

Implementation Details: We use Tensorflow [3] to train and test our networks. Unless otherwise noted, we use the CaffeNet convolutional pipeline initialized with the CaffeNet pretrained weights for our convolutional layers. The skip connections occur after "norm1," "norm2," and "conv5," with 16, 32, and 64 channels respectively. Each skip layer has a PReLU nonlinearity [82]. The embedding fully-connected layer has 2048 units, and the LSTM layers have 1024 units each. We initialize all new layers with the MSRA initialization method [82]. We use the the ADAM gradient optimizer [116] with the default momentum and weight decay and an initial learning rate of 10^{-5}, which we decrease to 10^{-6} after 10,000 iterations and continue for approximately 200,000 iterations which takes roughly one week. All layers, including the pretrained ones, are updated with

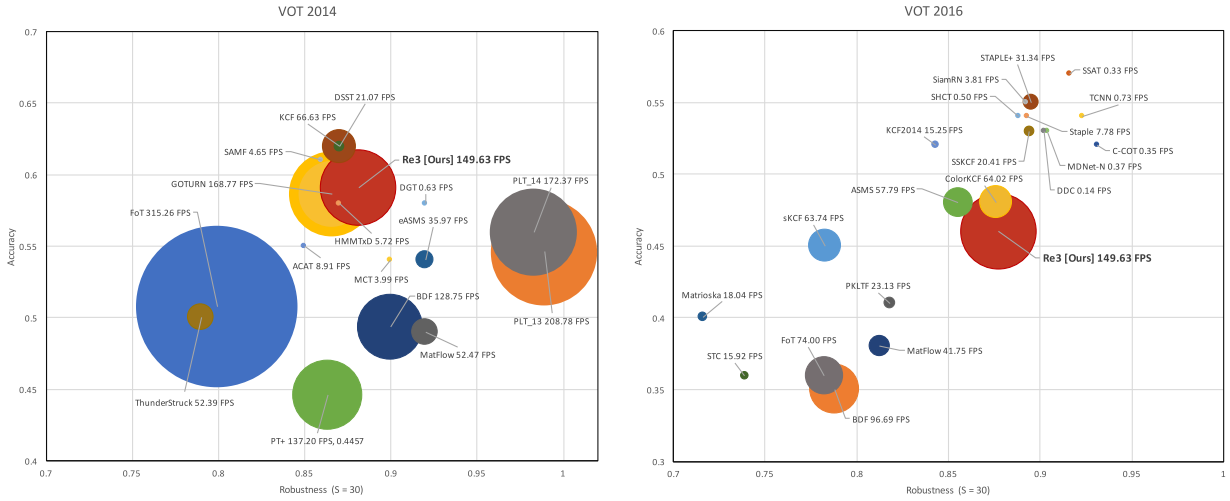


Figure 3.3: **Speed/Accuracy Tradeoff.** We compare Re^3 to other trackers on the VOT 2014 [123] and VOT 2016 [125] test suites. The size of the point indicates the speed of the tracker. Those below 3 FPS are enlarged to be visible. Speeds are taken directly from the VOT 2014 [123] and VOT 2016 [125] result reports. The VOT authors have stated that speed differences between years can be due to different code, different machines, and other confounding factors. For detailed analysis of other trackers' performance, please view [123, 125].

this learning rate. During training, we randomly mirror entire tracks with probability 0.5. All tests were carried out using an Intel Xeon CPU E5-2696 v4 @ 2.20GHz and an Nvidia Titan X (Pascal). For timing purposes, we ignore disk read speeds as they are independent of the tracking algorithm used.

3.4 Experiments

We compare Re^3 to other tracking methods on several popular tracking datasets in terms of both overall performance and robustness to occlusion. On all datasets, we are among the fastest, most accurate, and most robust trackers. We initially demonstrate our effectiveness by testing on a standard tracking benchmark, the Visual Object Tracking 2014 and 2016 (VOT 2014 and VOT 2016) challenges [123, 125], where we outperform other real-time trackers and are competitive with other deep methods. Next, we show results on the ILSVRC 2016 Object Detection from Video challenge (Imagenet Video) [181] comparing with other real-time trackers. We then examine

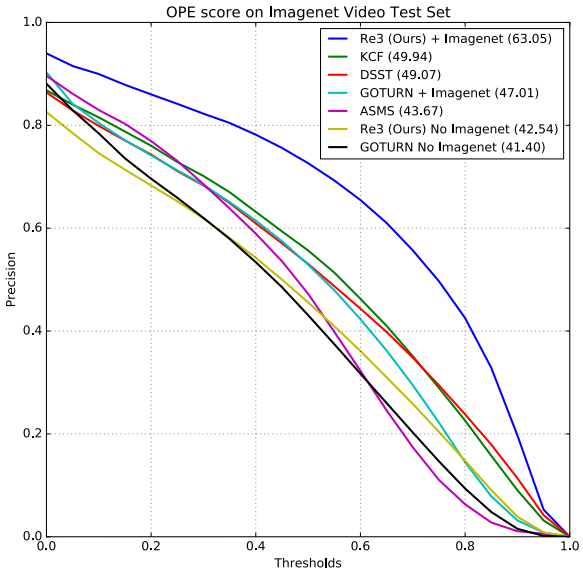


Figure 3.4: **Real-time trackers evaluated on the Imagenet Video test set.** Area under the curve (AUC) is shown for each method. We compare results with [86, 34, 84, 223] with code provided by [222, 63, 84, 223] respectively.

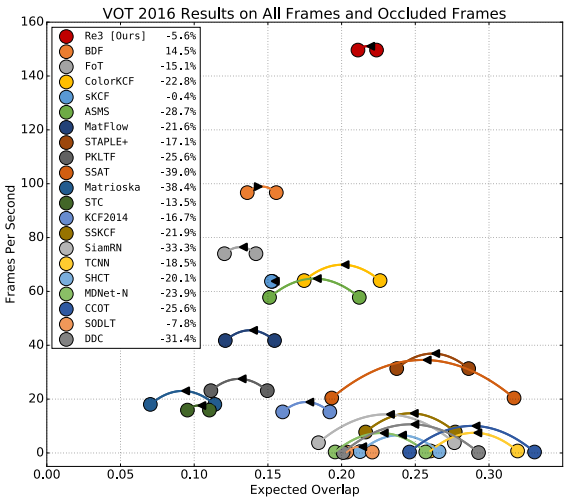


Figure 3.5: **Expected overlap of trackers on all frames and occluded frames.** The arrow indicates that BDF improves during occlusion whereas all other methods degrade. For further analysis of compared trackers, please view [125].

our performance during occlusion with specific experiments on occluded data. Additionally, we perform an ablation study to understand the contributions of each part to the overall success of the method. Finally, we examine qualitative results on novel video domains such as drone footage and cellphone video.

3.4.1 VOT 2014 and 2016

The VOT 2014 and 2016 object tracking test suite [123, 125] consists of 25 and 60 videos respectively made with the explicit purpose of testing trackers. Many of the videos contain difficulties such as large appearance change, heavy occlusions, and camera motion. Trackers are compared in terms of accuracy (how well the predicted box matches with the ground truth) and robustness (how infrequently a tracker fails and is reset). More details about these criteria can be found in [123]. Figure 3.3 compares Re^3 with other trackers submitted to the VOT 2014 and 2016 challenges [123, 125] as well as with Held *et al.*[84]. We show the 10 fastest trackers as well as the 10

most accurate trackers from each year.

Figure 3.3 Left shows our full model trained using all of the available training data. We are among the most accurate methods overall, and among the most robust of the real-time methods, likely due to the LSTM's ability to directly model temporal changes, allowing the network to adapt without much computational overhead.

Figure 3.3 Right compares our results against more modern trackers on the more difficult VOT 2016 test set [125]. For training this model, we omit the ALOV data entirely since there is a large overlap between the two video sets. We later explore the detrimental effect this has on our network's performance in the ablation analysis (model H in Table 3.1). Re^3 is 450x faster than the best methods [37, 125], while scoring only 20% and 5% lower in terms of relative accuracy and robustness. On both datasets, Re^3 offers an attractive trade-off of speed, accuracy, and robustness, especially in time-critical or computationally limited scenarios.

3.4.2 Imagenet Video

The Imagenet Video validation set consists of 1309 individual tracks and 273,505 images [181]. It is the largest dataset we test on, and it offers significant insights into the success cases and failure cases of our tracker. We use Imagenet Video to evaluate our performance against other open-source real-time trackers. Each tracker is initialized with the first frame of each test sequence and is not reset upon losing track. Each individual bounding box is evaluated against the ground truth for that track at various IOU thresholds. Figure 3.4 shows our method outperforming other real-time trackers over all thresholds by a wide margin, though only our method and GOTURN [84] + Imagenet were trained with the Imagenet Video training set. We also train a version of our network without using the Imagenet Video training data, only using a combination of ALOV [199] and simulated data. This performs significantly worse, most likely because LSTMs tend to take more data to train than comparable feed forward methods and this omits 90% of our training data. With sufficient training data, our method outperforms other methods trained on the same data.

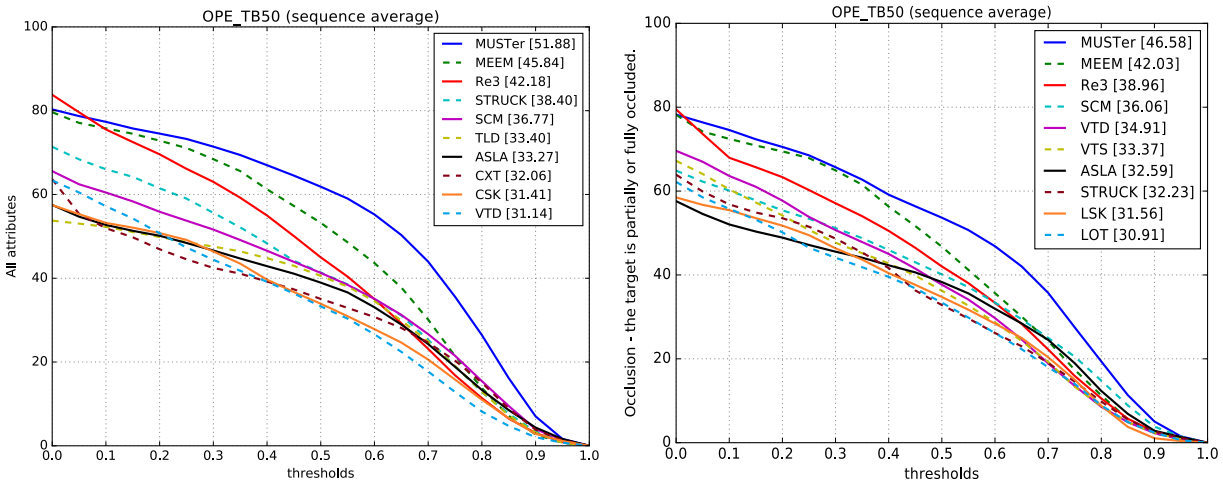


Figure 3.6: **Evaluation on the OTB benchmark.** We examine performance both overall (left) and during occluded frames (right) using the One Pass Evaluation (OPE) criterion explained in [234]. The legend shows area under the curve (AUC) for each method. Relative to other trackers, we suffer a smaller loss in accuracy due to occlusion. For detailed analysis of other trackers' performance, please view [234].

3.4.3 Online Object Tracking benchmark

The Online Object Tracking benchmark (OTB) [234] is a widely used benchmark in tracking literature consisting of 50 challenging tracking videos of various objects. The One Pass Evaluation (OPE) criteria on OTB is equivalent to the evaluation we perform on Imagenet Video. In Figure 3.6, we show results competitive with the provided baselines from the OTB website [234], even though we again omit the ALOV training data.

3.4.4 Robustness to Occlusion

We present two additional experiments showing that Re^3 performs comparatively well during occlusions. LSTMs can implicitly learn to handle occlusions because the structure of an LSTM can ignore information via the input and forget gates. This contrasts many other methods which assume all observations are useful, and may update their internal representation to include the occluder's

	Network Structure and Training Method	Speed (FPS)	VOT 2014 Accuracy	VOT 2014 # Drops	VOT 2014 Robustness	VOT 2014 Average	Imagenet Video Accuracy	Imagenet Video # Drops	Imagenet Video Robustness	Imagenet Video Average
A	Feed Forward Network (GOTURN) [84]	168.77	0.61	35	0.90	0.756	0.55	471	0.95	0.750
B	A + Imagenet Video Training	168.77	0.55	41	0.89	0.718	0.56	367	0.96	0.760
C	One Layer LSTM	**213.27**	0.48	67	0.82	0.651	0.49	738	0.92	0.706
D	C + Self-training	**213.27**	0.57	43	0.88	0.726	0.6	450	0.95	0.776
E	D + Simulated Data	**213.27**	0.6	38	0.89	0.747	0.65	359	0.96	0.806
F	E + Skip Layers	160.72	0.62	29	**0.92**	0.769	0.69	320	**0.97**	0.828
G	Full Model (F with two LSTM layers)	149.63	0.66	29	**0.92**	0.789	0.68	257	**0.97**	0.826
H	Full Model No ALOV	149.63	0.6	28	**0.92**	0.761	**0.71**	**233**	**0.97**	**0.842**
I	Full Model No Imagenet Video	149.63	0.58	61	0.82	0.700	0.52	1096	0.88	0.700
J	Full Model No LSTM Reset	149.63	0.54	47	0.87	0.705	0.61	539	0.94	0.775
K	Full Model with GoogleNet [208] conv layers	77.29	**0.68**	**27**	**0.92**	**0.802**	0.69	274	**0.97**	0.830

Table 3.1: **Ablation Study.** Average represents the arithmetic mean of accuracy and robustness, providing a single score to each method. Results on VOT 2014 [123] differ slightly from the VOT test suite, as they consider bounding boxes with a rotation angle, and we take the outermost points on these boxes as the ground truth labels.

appearance. In these experiments, we compare both the quality of track during occlusions as well as the difference in performance between overall scores and scores during occluded frames.

First, we examine our performance on the VOT 2016 test set [125]. Figure 3.5 shows the expected overlap measure of the same trackers from Figure 3.3 Right. Expected overlap represents the trackers' accuracy and robustness as a single number by performing many trials on subsets of the original data (more details available in [124]). Each tracker has two points on the graph: the first for overall expected overlap, and the second for expected overlap during occlusion. Re^3 performs nearly as well as the top performers from VOT 2016 [125] however at a much higher frame rate, and outperforms many on occluded frames. Re^3's performance degrades slightly during occlusions, but many of the other trackers drop in accuracy by more than 25%. sKCF [200] also barely changes, and BDF [147] actually improves during occlusions, however we outperform both of these methods on all frames and on occluded frames specifically.

We also evaluate how Re^3's performance degrades during occlusions across various IOU thresholds on OTB [234]. Similar to the previous experiment, Figure 3.6 compares the performance of trackers during all frames, and only during occluded frames. Again, we suffer a smaller loss

in accuracy of 7.6 in relative percentage compared to other top methods (MUSTer [94] 10.2%, MEEM [243] 8.3%, STRUCK [79] 16.1%). The performance on both datasets under occlusion illustrate that our LSTM-based method offers significant robustness to occlusion - one of the most difficult challenges in object tracking.

3.4.5 Ablation Study

Table 3.1 examines how various changes to the network affect the speed and performance of Re^3 on the VOT 2014 and Imagenet Video test sets [123, 181]. The difference between model A and C is that model A has three fully-connected layers with 4096 outputs each, whereas C has one fully-connected layer with 2048 outputs, and one LSTM layer with 1024 outputs. Despite the small change, simply adding an LSTM to an existing tracker without any modification in the training procedure hinders performance. Self-training, learning to correct previous mistakes and prevent drift (model D), is clearly necessary when training a recurrent tracker. Other modifications tend to add slight improvements in both accuracy and robustness. At the expense of speed, we can attain even better results. Model K uses the GoogleNet [208] architecture to embed the images, but is twice as slow. Model H, which was trained only on Imagenet Video [181] and simulated data, shows that by training on a fixed set of classes, performance improves on those classes but drops significantly on new objects (VOT 2014 [123]). Model I illustrates the need for a large training dataset, which seems especially important in terms of robustness. Model J shows the importance of resetting the LSTM state, as without the reset the network is much more affected by both parameter and model drift.

3.4.6 Qualitative Results

We test our network on a variety of important and challenging never-before-seen domains in order to gauge Re^3's usefulness to robotic applications. With a single initialization frame, our network performs remarkably well on challenging tasks such as shaky cellphone and car dashcam footage of a moving target, drone footage of multiple people simultaneously, and surveillance video of objects

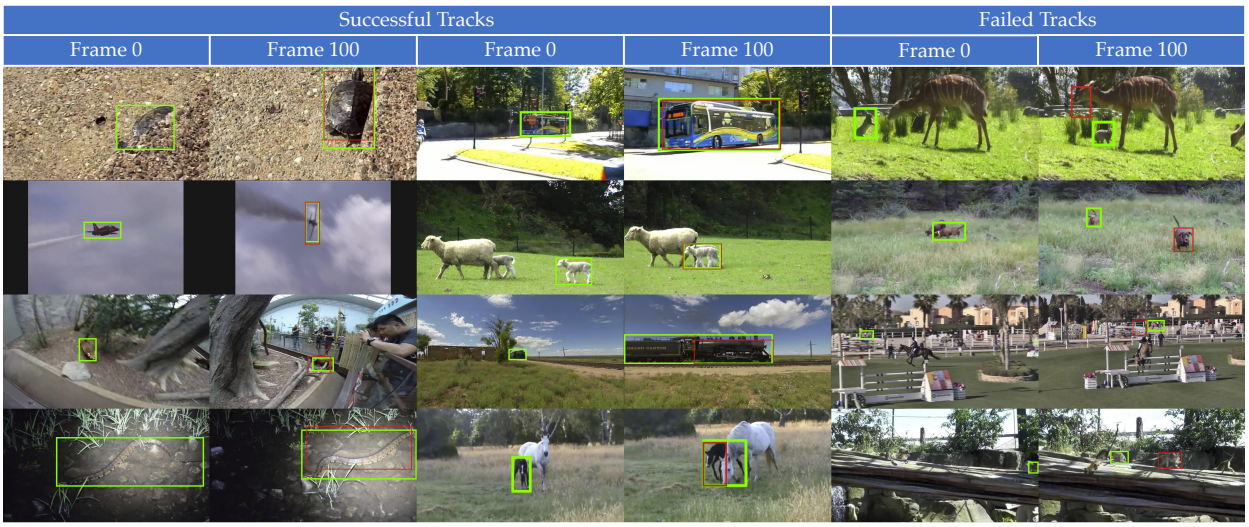


Figure 3.7: **Qualitative Results.** Examples of our tracker on the Imagenet Video [181] dataset. We show the initial frame and the 100th frame. Green represents the ground truth box and red represents Re^3's output. We include challenging examples of scale, appearance, and aspect ratio change as well as occlusion. On the right, we show failures due to latching onto a foreground object, confusion with a similar object, latching onto an occluder, and an ambiguous initial bounding box.

as small as 15×20 pixels. These results are shown in `https://youtu.be/RByCiOLlxug`. We also include excerpts of our performance on challenging frames from Imagenet Video [181] in Figure 3.7.

3.5 Conclusion

In this chapter, we presented the first algorithm that uses a recurrent neural network to track generic objects in a variety of natural scenes and situations. Recurrent models offer a new, compelling method of tracking due to their ability to learn from many examples offline and to quickly update online when tracking a specific object. Because they are end-to-end-trainable, recurrent networks can directly learn robustness to complex visual phenomena such as occlusion and appearance change. Our method demonstrates increased accuracy, robustness, and speed over comparable

trackers, especially during occlusions. We showed how to efficiently and effectively train a recurrent network to learn from labeled videos and synthetic data. Ultimately we have shown that recurrent neural networks have great potential in the fast generic object tracking domain, and can be beneficial in robotics applications that need real-time performance on a limited computational budget.

Part II

LEARNING BY DOING

Part I described two works which passively used data to learn about the world. This approach is clearly limited in what it can learn. To extend beyond passive learning, we must allow the agent to interact with the world. In this section, we outline multiple works on learning by interacting with simulated visual environments.

Chapter 4

VISUAL SEMANTIC PLANNING USING DEEP SUCCESSOR REPRESENTATIONS

4.1 Introduction

Humans demonstrate levels of visual understanding that go well beyond current formulations of mainstream vision tasks (e.g. object detection, scene recognition, image segmentation). A key element to visual intelligence is the ability to interact with the environment and *plan* a sequence of actions to achieve specific goals; This, in fact, is central to the efficacy of agents in dynamic environments.

Visual semantic planning, the task of interacting with a visual world and predicting a sequence of actions that achieves a desired goal, involves addressing several challenging problems. For example, imagine the simple task of `putting the bowl in the microwave` in the visual dynamic environment depicted in Figure 4.1. A successful plan involves first finding the bowl, navigating to it, then grabbing it, followed by finding and navigating to the microwave, opening the microwave, and finally putting the bowl in the microwave.

The **first** challenge in visual planning is that performing each of the above actions in a visual dynamic environment requires deep visual understanding of that environment, including the set of possible actions, their preconditions and effects, and object affordances. For example, to *open a microwave* an agent needs to know that it should be in front of the microwave, and it should be aware of the state of the microwave and not try to open an already opened microwave. Long explorations that are required for some tasks imposes the **second** challenge. The variability of visual observations and possible actions makes naïve exploration intractable. To *find a cup*, the agent might need to search several cabinets one by one. The **third** challenge is emitting a sequence of actions such that the agent ends in the goal state and the effects of the preceding actions meet

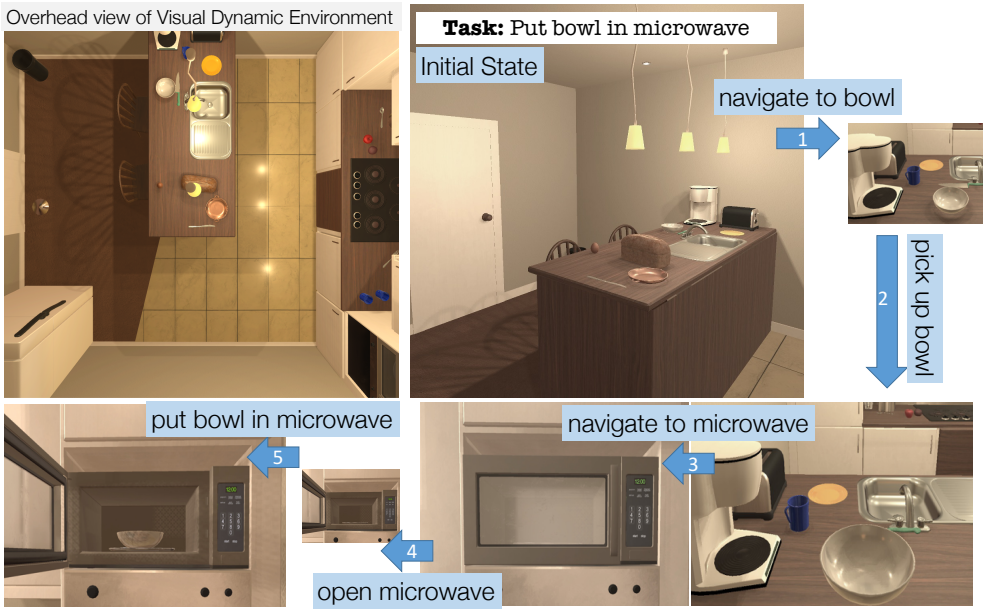


Figure 4.1: **Visual Semantic Planning.** Given a task and an initial configuration of a scene, our agent learns to interact with the scene and predict a sequence of actions to achieve the goal based on visual inputs.

the preconditions of the proceeding ones. Finally, a satisfactory solution to visual planning should enable cross task transfer; previous knowledge about one task should make it easier to learn the next one. This is the **fourth** challenge.

In this chapter, we address *visual semantic planning* as a policy learning problem. We mainly focus on high-level actions and do not take into account the low-level details of motor control and motion planning. **Visual Semantic Planning** (VSP) is the task of predicting a sequence of semantic actions that moves an agent from a random initial state in a visual dynamic environment to a given goal state.

To address the first challenge, one needs to find a way to represent the required knowledge of objects, actions, and the visual environment. One possible way is to learn these from still images or videos [56, 218, 226]. But we argue that learning high-level knowledge about actions and their

preconditions and effects requires an active and prolonged interaction with the environment. In this chapter, we take an interaction-centric approach where we learn this knowledge through interacting with the *visual dynamic environment*. Learning by interaction on real robots has limited scalability due to the complexity and cost of robotics systems [172, 173, 206]. A common treatment is to use simulation as *mental rehearsal* before real-world deployment [16, 112, 131, 236, 249]. For this purpose, we use the AI2-THOR framework [120], extending it to enable interactions with objects, where an action is specified as its pre- and post-conditions in a formal language.

To address the second and third challenges, we cast VSP as a policy learning problem, typically tackled by reinforcement learning [51, 72, 114, 137, 157, 195]. To deal with the large action space and delayed rewards, we use imitation learning to bootstrap reinforcement learning and to guide exploration. To address the fourth challenge of cross task generalization [130], we develop a deep predictive model based on successor representations [42, 128] that decouple environment dynamics and task rewards, such that knowledge from trained tasks can be transferred to new tasks with theoretical guarantees [15].

In summary, we address the problem of visual semantic planning and propose an interaction-centric solution. Our proposed model obtains near optimal results across a spectrum of tasks in the challenging AI2-THOR environment. Our results also show that our deep successor representation offers crucial transferability properties. Finally, our qualitative results show that our learned representation can encode visual knowledge of objects, actions, and environments.[1]

4.2 Related Work

4.2.1 Task planning

Task-level planning [49, 58, 105, 202, 203] addresses the problem of finding a high-level plan for performing a task. These methods typically work with high-level formal languages and low-dimensional state spaces. In contrast, visual semantic planning is particularly challenging due to the high dimensionality and partial observability of visual input. In addition, our method facilitates

[1] Video available at `https://youtu.be/_2pYVw6ATKo`

generalization across tasks, while previous methods are typically designed for a specific environment and task.

4.2.2 Perception and interaction

Our work integrates perception and interaction, where an agent actively interfaces with the environment to learn policies that map pixels to actions. The synergy between perception and interaction has drawn an increasing interest in the vision and robotics community. Recent work has enabled faster learning and produced more robust visual representations [6, 146, 172] through interaction. Some early successes have been shown in physical understanding [46, 131, 134, 159] by interacting with an environment.

4.2.3 Deep reinforcement learning

Recent work in reinforcement learning has started to exploit the power of deep neural networks. Deep RL methods have shown success in several domains such as video games [157], board games [195], and continuous control [137]. Model-free RL methods (e.g., [137, 156, 157]) aim at learning to behave solely from actions and their environment feedback; while model-based RL approaches (e.g., [43, 189, 210]) also estimate a environment model. Successor representation (SR), proposed by Dayan [42], can be considered as a hybrid approach of model-based and model-free RL. Barreto *et al*. [15] derived a bound on value functions of an optimal policy when transferring policies using successor representations. Kulkarni *et al*. [128] proposed a method to learn deep successor features. In this chapter, we propose a new SR architecture with significantly reduced parameters, especially in large action spaces, to facilitate model convergence. We propose to first train the model with imitation learning and fine-tune with RL. It enables us to perform more realistic tasks and offers significant benefits for transfer learning to new tasks.

4.2.4 Learning from demonstrations

Expert demonstrations offer a source of supervision in tasks which must usually be learned with copious random exploration. A line of work interleaves policy execution and learning from expert demonstration that has achieved good practical results [40, 180]. Recent works have employed a series of new techniques for imitation learning, such as generative adversarial networks [89, 135], Monte Carlo tree search [73] and guided policy search [132], which learn end-to-end policies from pixels to actions.

4.2.5 Synthetic data for visual tasks

Computer games and simulated platforms have been used for training perceptual tasks, such as semantic segmentation [78], pedestrian detection [148], pose estimation [168], and urban driving [7, 29, 177, 179]. In robotics, there is a long history of using simulated environments for learning and testing before real-world deployment [117]. Several interactive platforms have been proposed for learning control with visual inputs [16, 112, 120, 131, 236]. Among these, AI2-THOR [120] provides high-quality realistic indoor scenes. Our work extends AI2-THOR with a new set of actions and the integration of a planner.

4.3 Interactive Framework

To enable interactions with objects and with the environment, we modify the AI2-THOR framework [249] to better-suit the requirements of Visual Semantic Planning. Our modifications includes new object states, and a discrete description of the scene in a planning language [58].

4.3.1 Scenes

In this chapter, we focus on *kitchen* scenes, as they allow for a variety of tasks with objects from many categories. We limit our experiments in AI2-THOR to 10 individual kitchen scenes. Each scene contains an average of 53 distinct objects with which the agent can interact. The scenes are developed using the Unity 3D game engine.

Figure 4.2: **AI2-THOR Example Images.** These demonstrate the state changes before and after an object interaction from each of the six action types in our framework. Each action changes the visual state and certain actions may enable further interactions such as opening the fridge before taking an object from it.

4.3.2 Objects and Actions

We categorize the objects by their affordances [65], i.e., the plausible set of actions that can be performed. For the tasks of interest, we focus on two types of objects: 1) *items* that are small objects (mug, apple, etc.) which can be picked up, held, and moved by the agent to various locations in the scene, and 2) *receptacles* that are large objects (table, sink, etc.) which are stationary and can hold a fixed capacity of *items*. A subset of *receptacles*, such as fridges and cabinets, are *containers*. These *containers* have doors that can be opened and closed. The agent can only put an *item* in a *container* when it is open. We assume that the agent can hold at most one *item* at any point. We further define the following actions to interact with the objects:

1. Navigate {*receptacle*}: moving from the current location of the agent to a location near the *receptacle*;
2. Open {*container*}: opening the door of a *container* in front of an agent;
3. Close {*container*}: closing the door of a *container* in front of an agent;
4. Pick Up {*item*}: picking up an *item* in field of view;

5. Put {*receptacle*}: putting a held item in the *receptacle*;
6. Look Up and Look Down: tilting the agent's gaze 30 degrees up or down.

In summary, we have six action types, each taking a corresponding type of action arguments. The combination of actions and arguments results in a large action set of 80 per scene on average. Figure 4.2 illustrates example scenes and the six types of actions in our framework. Our definition of action space makes two important abstractions to make learning tractable: 1) it abstracts away from navigation, which can be tackled by a subroutine using existing methods such as [249]; and 2) it allows the model to learn with semantic actions, abstracting away from continuous motions, e.g., the physical movement of a robot arm to grasp an object. A common treatment for this abstraction is to "fill in the gaps" between semantic actions with callouts to a continuous motion planner [105, 202]. It is evident that not all actions can be performed in every situation. For example, the agent cannot pick up an *item* when it is out of sight, or put a tomato into fridge when the fridge door is closed. To address these requirements, we specify the preconditions and effects of actions. Next we introduce an approach to declaring them as logical rules in a planning language. These rules are only encoded in the environment but not exposed to the agent. Hence, the agent must learn them through interaction.

4.3.3 Planning Language

The problem of generating a sequence of actions that leads to the goal state has been formally studied in the field of automated planning [64]. Planning languages offer a standard way of expressing an automated planning problem instance, which can be solved by an off-the-shelf planner. We use STRIPS [58] as the planning language to describe our visual planning problem.

In STRIPS, a planning problem is composed of a description of an initial state, a specification of the goal state(s), and a set of actions. In visual planning, the initial state corresponds to the initial configuration of the scene. The specification of the goal state is a boolean function that returns true on states where the task is completed. Each action is defined by its precondition (conditions that must be satisfied before the action is performed) and postcondition (changes caused by the action). The STRIPS formulation enables us to define the rules of the scene, such as object affordances and

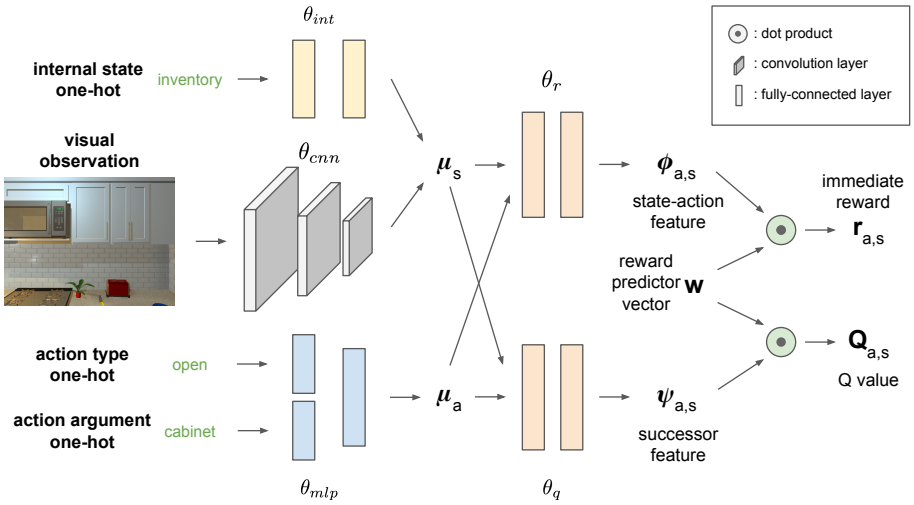


Figure 4.3: **Successor Representation (SR) Architecture Overview.** Our network takes in the current state as well as a specific action and predicts an immediate reward $r_{a,s}$ as well as a discounted future reward $Q_{a,s}$, performing this evaluation for each action. The learned policy π takes the argmax over all Q values as its chosen action.

causality of actions.

4.4 Decomposition using Successor Representation

We first outline the basics of policy learning in Sec. 4.4.1. Next we formulate the *visual semantic planning* problem as a policy learning problem and describe our model based on successor representation. Later we propose two protocols of training this model using imitation learning (IL) and reinforcement learning (RL). To this end, we use IL to bootstrap our model and use RL to further improve its performance.

4.4.1 Successor Representation

We formulate the agent's interactions with an environment as a *Markov Decision Process* (MDP), which can be specified by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ are the sets of states and actions. For $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $p(s'|s,a)$ defines the probability of transiting from the state s to the next state $s' \in \mathcal{S}$ by taking action a. $r(s,a)$ is a real-value function that defines the expected immediate reward of taking action a in state s. For a state-action trajectory, we define the future discounted *return* $R = \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)$, where $\gamma \in [0,1]$ is called the discount factor, which trades off the importance of immediate rewards versus future rewards.

A *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ defines a mapping from states to actions. The goal of policy learning is to find the optimal policy π^* that maximizes the future discounted return R starting from state s_0 and following the policy π^*. Instead of directly optimizing a parameterized policy, we take a value-based approach. We define a state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ under a policy π as

$$Q^\pi(s,a) = \mathbb{E}^\pi[R|s_0 = s, a_0 = a], \tag{4.1}$$

i.e., the expected episode return starting from state s, taking action a, and following policy π. The Q value of the optimal policy π^* obeys the Bellman equation [206]:

$$Q^{\pi^*}(s,a) = \mathbb{E}^{\pi^*}[r(s,a) + \gamma \max_{a'} Q(s',a')] \tag{4.2}$$

In deep Q networks [157], Q functions are approximated by a neural network $Q(s,a|\theta)$, and can be trained by minimizing the ℓ_2-distance between both sides of the Bellman equation in Eq. (4.2). Once we learn Q^{π^*}, the optimal action at state s can be selected by $a^* = \arg\max_a Q^{\pi^*}(s,a)$.

Successor representation (SR), proposed by Dayan [42], uses a similar value-based formulation for policy learning. It differs from traditional Q learning by factoring the value function into a dot product of two components: a reward predictor vector $\mathbf{w}$ and a predictive successor feature $\psi(s,a)$. To derive the SR formulation, we start by factoring the immediate rewards such that

$$r(s,a) = \phi(s,a)^T \mathbf{w}, \tag{4.3}$$

where $\phi(s, a)$ is a *state-action feature*. We expand Eq. (4.1) using this reward factorization:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) | s_0 = s, a_0 = a] \\ &= \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i \phi(s_i, a_i)^T \mathbf{w} | s_0 = s, a_0 = a] \\ &= \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i \phi(s_i, a_i) | s_0 = s, a_0 = a]^T \mathbf{w} \\ &= \psi^\pi(s,a)^T \mathbf{w} \end{aligned} \tag{4.4}$$

We refer to $\psi(s, a)^\pi = \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i \phi_{s,a} | s_0 = s, a_0 = a]$ as the *successor features* of the pair (s, a) under policy π.

Intuitively, the successor feature $\psi^\pi(s, a)$ summarizes the environment dynamics under a policy π in a state-action feature space, which can be interpreted as the expected future "feature occupancy". The reward predictor vector $\mathbf{w}$ induces the structure of the reward functions, which can be considered as an embedding of a task. Such decompositions have been shown to offer several advantages, such as being adaptive to changes in distal rewards and apt to option discovery [128]. A theoretical result derived by Barreto *et al.* implies a bound on performance guarantee when the agent transfers a policy from a task t to a similar task t', where task similarity is determined by the ℓ_2-distance of the corresponding $\mathbf{w}$ vectors between these two tasks t and t' [15]. Successor representation thus provides a generic framework for policy transfer in reinforcement learning.

4.4.2 Our Model

We formulate the problem of *visual semantic planning* as a policy learning problem. Formally, we denote a task by a Boolean function $t : \mathcal{S} \to \{0, 1\}$, where a state s completes the task t iff $t(s) = 1$. The goal is to find an optimal policy π^*, such that given an initial state s_0, π^* generates a state-action trajectory $\mathcal{T} = \{(s_i, a_i) \,|\, i = 0 \dots T\}$ that maximizes the sum of immediate rewards $\sum_{i=0}^{T-1} r(s_i, a_i)$, where $t(s_{0 \dots T-1}) = 0$ and $t(s_T) = 1$.

We parameterize such a policy using the successor representation (SR) model from the previous section. We develop a new neural network architecture to learn ϕ, ψ and $\mathbf{w}$. The network archi-

tecture is illustrated in Fig. 4.3. In AI2-THOR, the agent's observations come from a first-person RGB camera. We also pass the agent's internal state as input, expressed by one-hot encodings of the held object in its inventory. The action space is described in Sec. 4.3.2. We start by computing embedding vectors for the states and the actions. The image is passed through a 3-layer convolutional encoder, and the internal state through a 2-layer MLP, producing a state embedding $\mu_s = f(s; \theta_{cnn}, \theta_{int})$. The action $a = [a_{type}, a_{arg}]$ is encoded as one-hot vectors and passed through a 2-layer MLP encoder that produces an action embedding $\mu_a = g(a_{type}, a_{arg}; \theta_{mlp})$. We fuse the state and action embeddings and generate the state-action feature $\phi_{s,a} = h(\mu_s, \mu_a; \theta_r)$ and the successor feature $\psi_{s,a} = m(\mu_s, \mu_a; \theta_q)$ in two branches. The network predicts the immediate reward $r_{s,a} = \phi_{s,a}^T \mathbf{w}$ and the Q value under the current policy $Q_{s,a} = \psi_{s,a}^T \mathbf{w}$ using the decomposition from Eq. (4.3) and (4.4).

4.4.3 Imitation Learning

Our SR-based policy can be learned in two fashions. First, it can be trained by imitation learning (IL) under the supervision of the trajectories of an optimal planner. Second, it can be learned by trial and error using reinforcement learning (RL). In practice, we find that the large action space in AI2-THOR makes RL from scratch intractable due to the challenge of exploration. The best model performance is produced by IL bootstrapping followed by RL fine-tuning. Given a task, we generate a state-action trajectory:

$$\mathcal{T} = \{(s_0, a_0), \{(s_1, a_1), \ldots, (s_{T-1}, a_{T-1}), (s_T, \emptyset)\} \tag{4.5}$$

using the planner from the initial state-action pair (s_0, a_0) to the goal state s_T (no action is performed at terminal states). This trajectory is generated on a low-dimensional state representation in the STRIPS planner (Sec. 4.3.3). Each low-dimensional state corresponds to an RGB image, i.e., the agent's visual observation. During training, we perform *input remapping* to supervise the model with image-action pairs rather than feeding the low-dimensional planner states to the network. To fully explore the state space, we take planner actions as well as random actions off the optimal plan. After each action, we recompute the trajectory. This process of generating train-

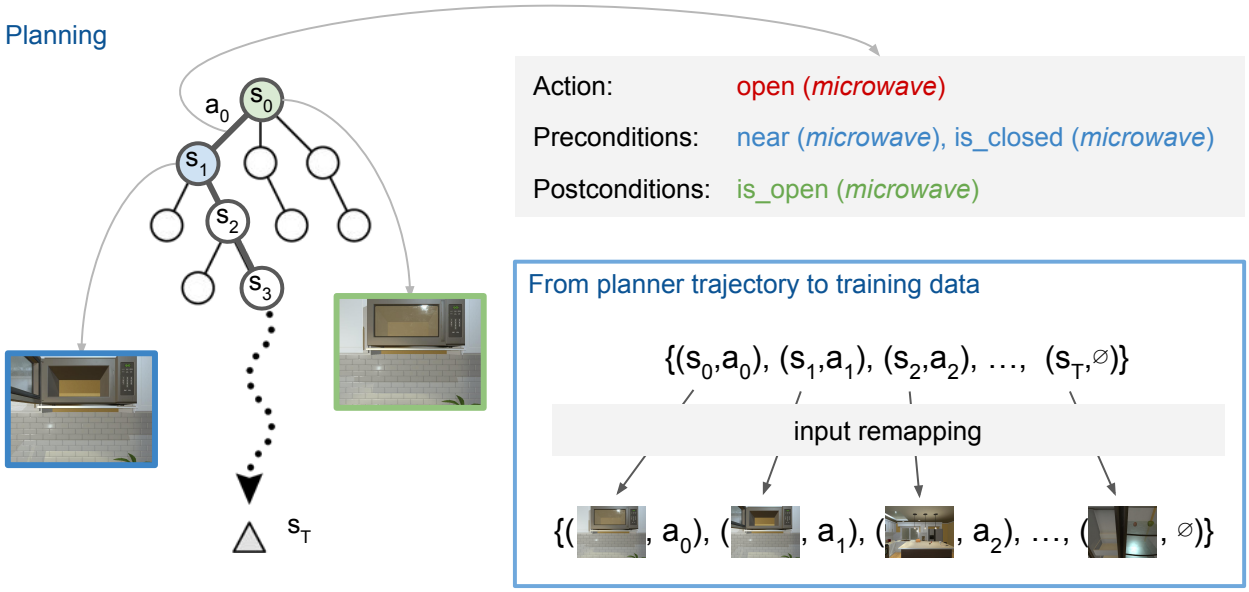


Figure 4.4: **STRIPS Planner Setup.** We use a planner to generate a trajectory from an initial state-action pair (s_0, a_0) to a goal state s_T. We describe each scene in a STRIPS-based planning language, where actions are specified by their pre- and post-conditions (see Sec. 4.3.3). We perform input remapping, illustrated in the blue box, to obtain the image-action pairs from the trajectory as training data. After performing an action, we update the plan and repeat.

ing data from a planner is illustrated in Fig. 4.4. Each state-action pair is associated with a true immediate reward $\hat{r}_{s,a}$. We use the mean squared loss function to minimize the error of reward prediction:

$$\mathcal{L}_r = \frac{1}{T}\sum_{i=0}^{T-1}(\hat{r}_{s,a} - \phi_{s,a}^T \mathbf{w})^2. \tag{4.6}$$

Following the REINFORCE rule [206], we use the discounted return along the trajectory $\mathcal{T}$ as an unbiased estimate of the true Q value: $\hat{Q}_{s,a} \approx \sum_{i=0}^{T-1} \gamma^i \hat{r}_{s,a}$. We use the mean squared loss to minimize the error of Q prediction:

$$\mathcal{L}_Q = (\hat{Q}_{s,a} - \psi_{s,a}^T \mathbf{w})^2 \tag{4.7}$$

The final loss on the planner trajectory $\mathcal{T}$ is the sum of the reward loss and the Q loss: $\mathcal{L}_{\mathcal{T}} = \mathcal{L}_r + \mathcal{L}_Q$. Using this loss signal, we train the whole SR network on a large collection of planner trajectories starting from random initial states.

4.4.4 Reinforcement Learning

When training our SR model using RL, we can still use the mean squared loss in Eq. (4.6) to supervise the learning of reward prediction branch for ϕ and $\mathbf{w}$. However, in absence of expert trajectories, we would need an iterative way to learn the successor features ψ. Rewriting the Bellman equation in Eq. (4.2) with the SR factorization, we can obtain an equality on ϕ and ψ:

$$\psi^{\pi^*}(s,a) = \mathbb{E}^{\pi^*}[\phi(s,a) + \gamma\psi(s',a')] \tag{4.8}$$

where $a' = \arg\max_a \psi(s',a)^T\mathbf{w}$. Similar to DQN [157], we minimize the ℓ_2-loss between both sides of Eq. (4.8):

$$L_{SR} = \mathbb{E}^{\pi}[(\phi_{s,a} + \gamma\psi_{s',a'} - \psi^{\pi}_{s,a})^2] \tag{4.9}$$

We use a similar procedure to Kulkarni *et al*. [128] to train our SR model. The model alternates training between the reward branch and the SR branch. At each iteration, a mini-batch is randomly drawn from a replay buffer of past experiences [157] to perform one SGD update.

4.4.5 Transfer with Successor Features

A major advantage of successor features is its ability to transfer across tasks by exploiting the structure shared by the tasks. Given a fixed state-action representation ϕ, let M_ϕ be the set of all possible MDPs induced by ϕ and all instantiations of the reward prediction vectors $\mathbf{w}$. Assume that π_i^* is the optimal policy of the i-th task in the set $\{M_i \in M_\phi | i = 1, \dots n\}$. Let M_{n+1} to be a new task. We denote $Q_{n+1}^{\pi_i^*}$ as the value function of executing the optimal policy of the task M_i on the new task M_{n+1}, and $\tilde{Q}_{n+1}^{\pi_i^*}$ as an approximation of $Q_{n+1}^{\pi_i^*}$ by our SR model. Given a bound on the approximations such that

$$|Q_{n+1}^{\pi_i^*}(s,a) - \tilde{Q}_{n+1}^{\pi_i^*}(s,a)| \leq \epsilon \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, i = 1, \dots, n,$$

we define a policy π' for the new task M_{n+1} using $\tilde{Q}_{1,\ldots,n}$, where $\pi'(s) = \arg\max_a \max_i \tilde{Q}^{\pi_i^*}_{n+1}(s,a)$. Theorem 2 in Barreto *et al*. [15] implies a bound of the gap between value functions of the optimal policy π^*_{n+1} and the policy π':

$$Q^{\pi^*_{n+1}}_{n+1}(s,a) - Q^{\pi'}_{n+1}(s,a) \leq \frac{2\phi_m}{1-\gamma}(\min_i ||\mathbf{w}_i - \mathbf{w}_{n+1}|| + \epsilon),$$

where $\phi_m = \max_{s,a} ||\phi(s,a)||$. This result serves the theoretical foundation of policy transfer in our SR model. In practice, when transferring to a new task while the scene dynamics remain the same, we freeze all model parameters except the single vector $\mathbf{w}$. This way, the policy of the new task can be learned with substantially higher sample efficiency than training a new network from scratch.

4.4.6 Implementation Details

We feed a history of the past four observations, converted to grayscale, to account for the agent's motions. We use a time cost of -0.01 to encourage shorter plans and a task completion reward of 10.0. We train our model with imitation learning for 500k iterations with a batch size of 32, and a learning rate of 1e-4. We also include the successor loss in Eq. (4.9) during imitation learning, which helps learn better successor features. We subsequently fine-tune the network with reinforcement learning with 10,000 episodes.

4.5 Experiments

We evaluate our model using the extended AI2-THOR framework on a variety of household tasks. We compare our method against standard reinforcement learning techniques as well as with non-successor based deep models. The tasks compare the different methods' abilities to learn across varying time horizons. We also demonstrate the SR network's ability to efficiently adapt to new tasks. Finally, we show that our model can learn a notion of object affordance by interacting with the scene.

Method	Easy		Medium		Hard	
	Success Rate	Mean (Std.) Episode Length	Success Rate	Mean (Std.) Episode Length	Success Rate	Mean (Std.) Episode Length
Random Action	1.00	696.33 (744.71)	0.00	-	0.04	2827.08 (927.84)
Random Valid Action	1.00	64.03 (68.04)	0.02	3897.50 (548.50)	0.36	2194.83 (1401.72)
A3C [156]	0.96	101.12 (151.04)	0.00	-	0.04	2674.29 (4370.40)
CLS-MLP	1.00	**2.42** (0.70)	0.65	256.32 (700.78)	0.65	475.86 (806.42)
CLS-LSTM	1.00	2.86 (0.37)	**0.80**	314.05 (606.25)	**0.66**	136.94 (523.60)
SR IL (ours)	1.00	2.70 (1.06)	**0.80**	32.32 (29.22)	0.65	**34.25** (63.81)
SR IL + RL (ours)	1.00	2.57 (1.04)	**0.80**	**26.56** (3.85)	-	-
Optimal planner	1.00	2.36 (1.04)	1.00	12.10 (6.16)	1.00	14.13 (9.09)

Table 4.1: **Results of evaluating the model on the easy, medium, and hard tasks.** For each task, we evaluate how many out of the 100 episodes were completed (success rate) and the mean and standard deviation for successful episode lengths. The numbers in parentheses show the standard deviations. We do not fine-tune our SR IL model for the hard task.

4.5.1 Quantitative Evaluation

We examine the effectiveness of our model and baseline methods on a set of tasks that require three levels of planning complexity in terms of optimal plan length.

Experiment Setup We explore the two training protocols introduced in Sec. 4.4 to train our SR model:

1. **RL**: we train the model solely based on trial and error, and learn the model parameters with RL update rules.

2. **IL**: we use the planner to generate optimal trajectories starting from a large collection of random initial state-action pairs. We use the imitation learning methods to train the networks using supervised losses.

Empirically, we find that training with reinforcement learning from scratch cannot handle the large action space. Thus, we report the performance of our SR model trained with imitation learning (SR IL) as well as with additional reinforcement learning fine-tuning (SR IL + RL).

We compare our SR model with the state-of-the-art deep RL model, A3C [156], which is an advantage-based actor-critic method that allows the agent to learn from multiple copies of simulation while updating a single model in an asynchronous fashion. A3C establishes a strong baseline for reinforcement learning. We further use the same architecture to obtain two imitation learning (behavior cloning) baselines. We use the same A3C network structure to train a softmax classifier that predicts the planner actions given an input. The network predicts both the action types (e.g., Put) and the action arguments (e.g., *apple*). We call this baseline CLS-MLP. We also investigate the role of memory in these models. To do this, we add an extra LSTM layer to the network before action outputs, called CLS-LSTM. We also include simple agents that take random actions and take random valid actions at each time step.

Levels of task difficulty We evaluate all of the models with three levels of task difficulty based on the length of the optimal plans and the source of randomization:

1. **Level 1 (Easy)**: Navigate to a *container* and toggle its state. A sample task would be `go to the microwave and open it if it is closed, close it otherwise.` The initial location of the agent and all *container* states are randomized. This task requires identifying object states and reasoning about action preconditions.

2. **Level 2 (Medium)**: Navigate to multiple *receptacles*, collect *items*, and deposit them in a *receptacle*. A sample task here is `pick up three mugs from three cabinets and put them in the sink.` Here we randomize the agent's initial location, while the item locations are fixed. This task requires a long trajectory of correct actions to complete the goal.

3. **Level 3 (Hard)**: Search for an *item* and put it in a *receptacle*. An example task is `find the apple and put it in the fridge.` We randomize the agent's location as well as the location of all items. This task is especially difficult as it requires longer-term memory to account for partial observability, such as which cabinets have previously been checked.

We evaluate all of the models on 10 easy tasks, 8 medium tasks, and 7 hard tasks, each across 100 episodes. Each episode terminates when a goal state is reached. We consider an episode fails if it does not reach any goal state within 5,000 actions. We report the episode success rate and mean episode length as the performance metrics. We exclude these failed episodes in the mean episode length metric. For the easy and medium tasks, we train the imitation learning models to mimic the optimal plans. However for the hard tasks, imitating the optimal plan is infeasible, as the location of the object is uncertain. In this case, the target object is likely to hide in a cabinet or a fridge which the agent cannot see. Therefore, we train the models to imitate a plan which searches for the object from all the *receptacles* in a fixed order. For the same reason, we do not perform RL fine-tuning for the hard tasks.

Table 4.1 summarizes the results of these experiments. Pure RL-based methods struggle with the medium and hard tasks because the action space is so large that naïve exploration rarely, if ever, succeeds. Comparing CLS-MLP and CLS-LSTM, adding memory to the agent helps improving success rate on medium tasks as well as completing tasks with shorter trajectories in hard tasks. Overall, the SR methods outperform the baselines across all three task difficulties. Fine-tuning the SR IL model with reinforcement learning further reduces the number of steps towards the goal. More qualitative results can be found in the supplementary video.

4.5.2 Task Transfer

One major benefit of the successor representation decomposition is its ability to transfer to new tasks while only retraining the reward prediction vector $\mathbf{w}$, while freezing the successor features. We examine the sample efficiency of adapting a trained SR model on multiple novel tasks in the same scene. We examine policy transfer in the hard tasks, as the scene dynamics of the searching policy retains, even when the objects to be searched vary. We evaluate the speed at which the SR model converges on a new task by fine-tuning the $\mathbf{w}$ vector versus training the model from scratch. We take a policy for searching a bowl in the scene and substituting four new items (lettuce, egg, container, and apple) in each new task. Fig. 4.5 shows the episode success rates (bar chart) and the successful action rate (line plot). By fine-tuning $\mathbf{w}$, the model quickly adapts to new tasks,

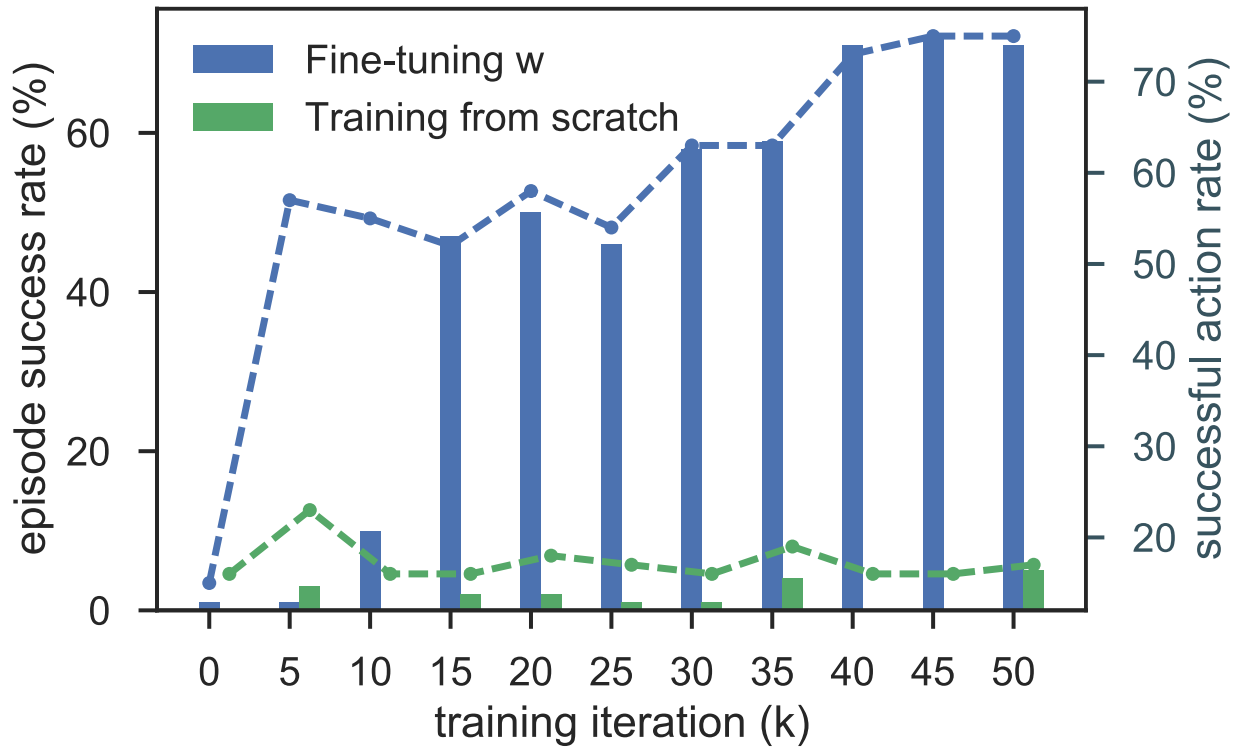


Figure 4.5: **Success Rates Over Time.** We compare updating $\mathbf{w}$ with retraining the whole network for new hard tasks in the same scene. By using successor features, we can quickly learn an accurate policy for the new item. Bar charts correspond to the episode success rates, and line plots correspond to successful action rate.

yielding both high episode success rate and successful action rate. In contrast, the model trained from scratch takes substantially longer to converge. We also experiment with fine-tuning the entire model, and it suffers from similar slow convergence.

4.5.3 Learning Affordances

An agent in an interactive environment needs to be able to reason about the causal effects of actions. We expect our SR model to learn the pre- and post-conditions of actions through interaction,

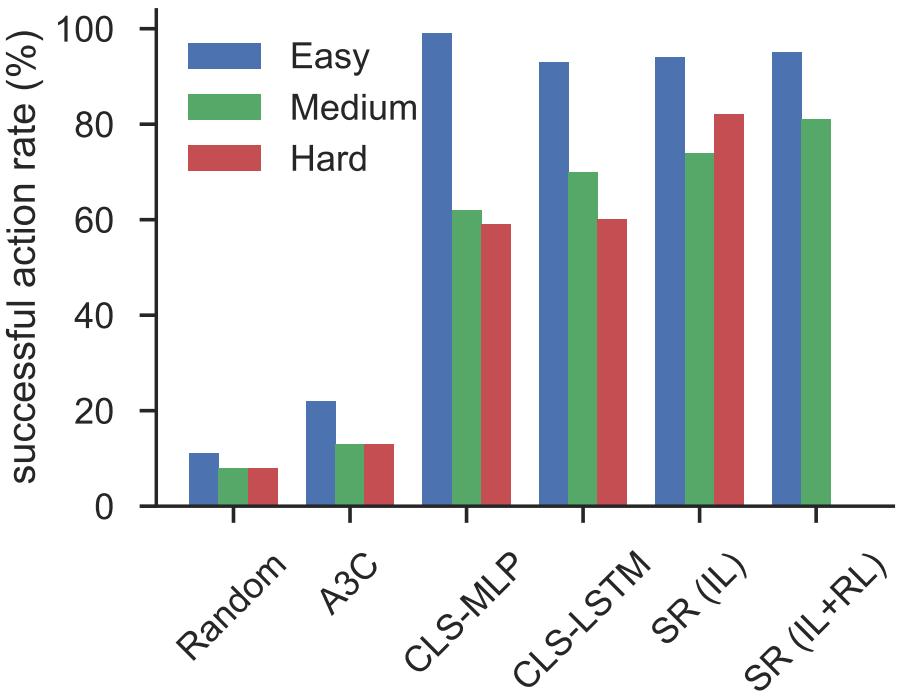


Figure 4.6: **Transfer Comparison.** We compare the different model's likelihood of performing a successful action during execution. A3C suffers from the large action space due to naïve exploration. Imitation learning models are capable of differentiating between successful and unsuccessful actions because the supervised loss discourages the selection of unsuccessful actions.

such that it develops a notion of affordance [65], i.e., which actions can be performed under a circumstance. In the real world, such knowledge could help prevent damages to the agent and the environment caused by unexpected or invalid actions.

We first evaluate each network's ability to *implicitly* learn affordances when trained on the tasks in Sec. 4.5.1. In these tasks, we penalize unnecessary actions with a small time penalty, but we do not explicitly tell the network which actions succeed and which fail. Fig. 4.6 illustrates that a standard reinforcement learning method cannot filter out unnecessary actions especially given delayed rewards. Imitation learning methods produce significantly fewer failed actions because they can directly evaluate whether each action gets them closer to the goal state.

We also analyze the successor network's capability of *explicitly* learning affordances. We train

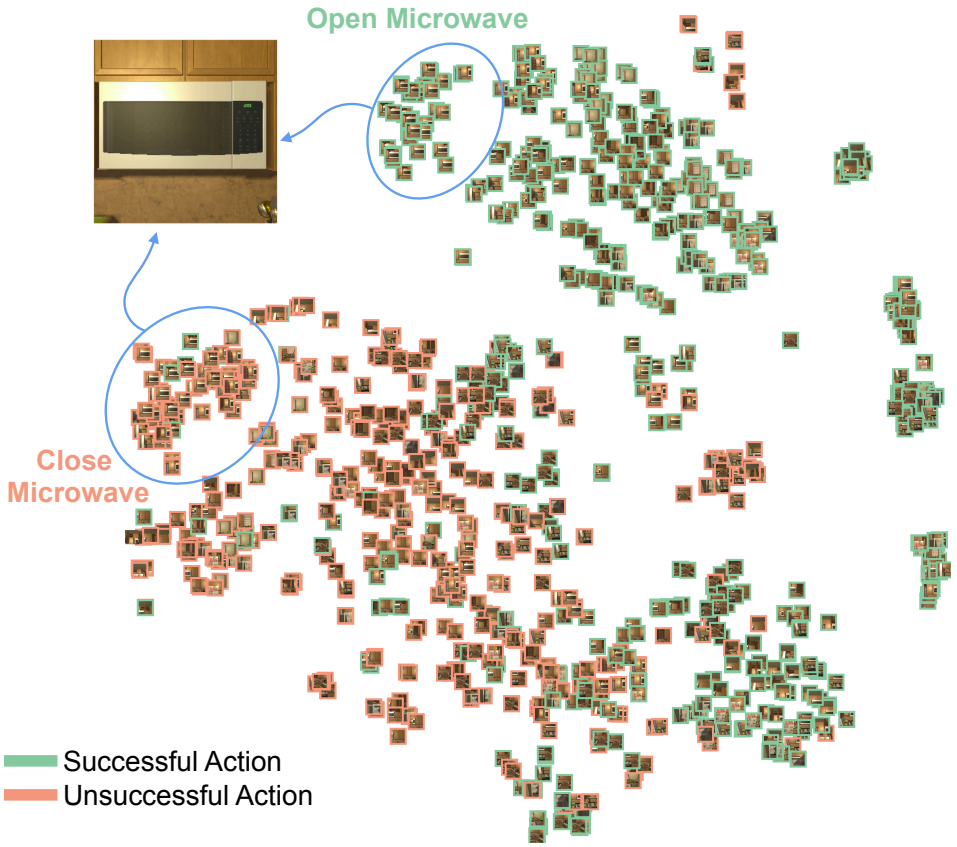


Figure 4.7: **Visualization of a t-SNE embedding of the state-action vector $\phi_{s,a}$ for a random set of state-action pairs.** Successful state-action pairs are shown in green, and unsuccessful pairs in orange. The two blue circles highlight portions of the embedding with very similar images but different actions. The network can differentiate successful pairs from unsuccessful ones.

our SR model with reinforcement learning, by executing a completely random policy in the scene. We define the immediate reward of issuing a successful action as $+1.0$ and an unsuccessful one as -1.0. The agent learns in 10,000 episodes. Fig. 4.7 shows a t-SNE [142] visualization of the state-action features $\phi_{s,a}$. We see that the network learns to cluster successful state action pairs (shown in green) separate from unsuccessful pairs (orange). The network achieves an ROC-AUC of 0.91 on predicting immediate rewards over random state-action actions, indicating that the model can differentiate successful and unsuccessful actions by performing actions and learning from their outcomes.

4.6 Conclusions

In this chapter, we argue that visual semantic planning is an important next task in computer vision. Our proposed solution shows promising results in predicting a sequence of actions that change the current state of the visual world to a desired goal state. We have examined several different tasks with varying degrees of difficulty and show that our proposed model based on deep successor representations achieves near optimal results in the challenging AI2-THOR environment. We also show promising cross-task knowledge transfer results, a crucial component of any generalizable solution. Our qualitative results show that our learned successor features encode knowledge of object affordances, and action preconditions and post-effects. Our next steps involve exploring knowledge transfer from AI2-THOR to real-world environments as well as examining the possibilities of more complicated tasks with a richer set of actions.

Chapter 5

SPLITNET: SIM2SIM AND TASK2TASK TRANSFER FOR EMBODIED VISUAL NAVIGATION

In the previous chapter, we used the successor representation to decompose the network into a "dynamics" branch and a "policy" branch to decouple scene dynamics from planning-task policy. Similarly, in this chapter we decompose the network into a "perception" branch and a "policy" branch in order to allow for better transfer and faster adaptation to new visual and policy needs.

5.1 Introduction

The recent deep learning revolution in perception systems has enabled robots to better-understand their surroundings, navigate efficiently and safely, and perform a large variety of tasks in complex environments. A practical application of the recent successes in Deep Reinforcement Learning is to train robots with minimal supervision to perform these tasks. Yet poorly-trained agents can easily injure themselves, the environment, or others. These concerns, as well as the difficulty in parallelizing and reproducing experiments at a low cost, have drawn research interest towards simulation environments [26, 120, 237, 239, 187].

However no simulator perfectly replicates reality, and agents trained in simulation often fail to generalize to the real-world. Transferring learned policies from simulation to the real-world (Sim2Real) has become an area of broad interest [171, 182, 215] yet there still exists a sizable performance gap for most algorithms. Furthermore, Sim2Real transfer reintroduces safety and reproducibility concerns. To mitigate this, we explore the related task of Sim2Sim, transferring policies between simulators, for embodied visual navigation (Figure 5.1). Transferring between simulators incurs a similar "reality gap" as between simulation and reality, due to differences in data collection and rendering. Learning to transfer between simulation environments serves as an

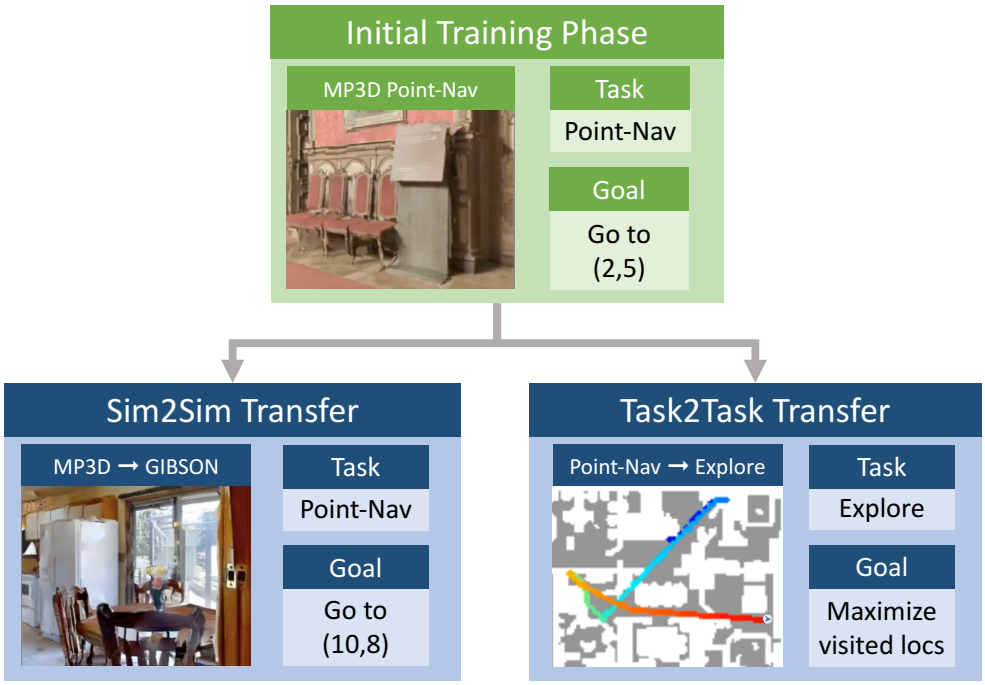


Figure 5.1: **SplitNet Decomposition.** We decompose learning of visual navigation tasks into learning of a visual encoder and learning of an embodied task decoder. Through this decomposition we enable fast transfer to new visual environments and transfer to new embodied tasks.

encouraging preliminary step towards true Sim2Real transfer.

To enable Sim2Sim transfer we propose SplitNet, a composable model for embodied visual tasks which allows for the sharing and reuse of information between different visual environments. SplitNet enables transfer across different embodied tasks (Task2Task), meaning our model can learn new skills quickly and adapt to the ever-changing requirements of end users. Our key insight is to observe that embodied visual tasks are naturally decomposable into visual representation learning to extract task agnostic salient information from the visual input, and policy learning to interpret the visual representation and determine a proper action for the agent. Rather than learning these components solely independently or completely tied, we introduce an algorithm for learning these embodied visual tasks which benefits both from the scalability and strong in-domain, on-task performance of an end-to-end system and from the generalization and fast adaptability of modular systems.

SplitNet incorporates auxiliary visual tasks, such as depth prediction, as a source of interme-

diate supervision which guides the visual representation learning to extract information from the images extending beyond the initial embodied task. We demonstrate that initial pre-training of the visual representation on such auxiliary visual tasks produces a more robust initialization than the standard approach of pre-training on auxiliary visual datasets (e.g. ImageNet [45]) which may not be from an embodied perspective. Then we showcase the composability of our model by illustrating its ability to selectively adapt only the visual representation (when moving to a new visual environment) or only its policy (when moving to a new embodied task).

We center our evaluation on adapting between different simulators of varying fidelity and between different embodied tasks. Specifically, our experiments show that compared to end-to-end methods, SplitNet learns more transferable visual features for the task of visual point-to-point navigation, reduces overfitting to small samples from a new target simulator, and adapts faster and better to novel embodied tasks.

In summary, our contributions are **1.** a principled way to decouple perception and policy (in our case navigation), **2.** showing that this technique improves performance on the primary task *and* facilitates transfer to new environments and tasks, **3.** describing specifically how to update the weights for new tasks or new environments using significantly less data to adapt.[1]

5.2 Related Work

This chapter introduces a learning approach for transferring visual representations between environments and for transferring policy information between different embodied tasks. The most related lines of work focus on adaptation and transfer of visual representations, deep reinforcement learning (especially from visual inputs), and transferring from simulation to the real-world (Sim2Real) both for visual and embodied tasks.

---

[1] Code available at `https://github.com/facebookresearch/splitnet`
Video available at `https://youtu.be/AKQE9RyOIMY`

5.2.1 Visual Transfer and Adaptation

Many works have explicitly studied techniques for increasing the reusability of learned information across different visual tasks. Domain adaptation research has mainly focused on reusing a representation even as the input distribution changes, with most work focusing on representation alignment through explicit statistics [141, 205] or through implicit discrepancy minimization with a domain adversarial loss [61, 219]. A related line of work focuses on sharing between two image collections through direct image-to-image transfer [174, 248], whereby a mapping function is learned to take an image from one domain and translate it to mimic an image from the second domain [22, 91, 140, 209].

In parallel, many works focus on reusing learned representations for solving related visual tasks. The most prevalent such technique is simply using the first representation parameters as initialization for learning the second, termed finetuning [66]. A recent study proposed a technique for computing the similarity between a suite of visual tasks to create a Taskonomy [241] which may be used to determine, given a new task, which prior tasks should be used for the initialization before continued learning. This method focuses on "passive" visual understanding tasks such as recognition, reconstruction and depth estimation and does not delve into learning representations for "active" tasks such as embodied navigation where an agent must both understand the world and directly use its understanding for some underlying task.

Overall, much of the prior work has focused on representation learning for visual recognition. In contrast, this chapter studies transfer of visuomotor policies for embodied tasks and decomposes the problem into transfer of visual representations for embodied imagery (Sim2Sim) and transfer of policies across various downstream embodied tasks (Task2Task).

5.2.2 Visual RL Tasks

In parallel with the development of deep representation learning for passive visual tasks, there has also been a plethora of recent research on policy learning from visual inputs inspired by the success of end-to-end visuomotor policy learning [132, 133, 156]. Much of the success here comes from

training on large-scale [133] data, frequently made possible by extensive use of simulation environments [74, 156, 170, 249]. These techniques often leverage the additional supervision and auxiliary tasks given by the simulators to bootstrap their learning [153, 188]. Perceptual Actor [188] specifically examines how 20 different pretraining tasks affect the learning speed and accuracy of a visual navigation policy as compared to random initialization. Others use unsupervised [99] or self-supervised [170] learning as an additional signal in domains with sparse rewards. We build on these approaches by explicitly separating the auxiliary learning from the policy layers to ensure a decoupling of the weights which enables better transfer to new environments.

For increased task generalization, others (including us in Chapter 4) have proposed using the successor representation [244] which decomposes the reward and Q-functions into a state-action feature $\phi_{s,a}$, a successor feature $\psi_{s,a}$ and a task reward vector w. This decomposes the network into one which learns the dynamics of the environment separate from the specified task, which allows for faster transfer to new tasks by only retraining the task embedding w. Our proposed method allows quick transfer to new tasks as well as new environments.

5.2.3 Sim2Real

Significant progress has been made on adapting between simulated and real imagery for visual recognition, especially in the context of semantic segmentation in driving scenes [91, 92, 96, 246]. These techniques build on the visual domain adaptation methods described above. In parallel, there has been work on transferring visual policies learned in simulation to the real-world, but often limited to simple visual domains [182, 215] which bear little resemblance to the complexity of true real-world scenes. Rusu *et al*. [182] train a network in simulation before initializing a new network which receives outputs from the simulation-trained network as well as real-world inputs. Yet their evaluation is limited to simple block picking experiments with no complex visual scenes. Peng *et al*. [171] use randomization over the robot dynamics to learn robust policies, but do not use visual inputs in simulation or reality and only perform simple puck-pushing tasks. Sadeghi *et al*. [184] also uses randomization of textures, lighting, and furniture placement in a simulation environment for Sim2Real transfer of drone flight. Tobin *et al*. [215] and

Sadeghi *et al*. [185] randomize colors, textures, lighting, and camera pose as a form of augmentation of the simulated imagery to better generalize to real-world imagery for picking tasks. [215] focuses on primitive geometric objects for picking tasks and does not decouple visual feature learning from policy learning which limits the transferrability of their method to new tasks. [185] shows similar benefits of decoupling perception and policy for Sim2Real transfer, but do not explore transfer to new tasks. Two recent method [160, 183], uses semantic segmentation and obstacle detection as an intermediate objective to aid in transferring learned driving policies from simulation to the real-world. While we do not transfer our policies to real robots, we focus on visually diverse scenes which better match the complexity of the real-world then the simplistic setups of many of the prior policy transfer approaches. Similar to Müller *et al*. [160] we use auxiliary intermediate objectives to aid in transfer, but in our case focus on a set of auxiliary visual and motion tasks which generalize to many downstream embodied tasks and propose techniques to selectively transfer either across visual environments or across embodied tasks.

5.3 Decoupled Perception and Policy

Solving complex visual planning problems frequently requires different types of abstract understanding and reasoning based on the visual inputs. In order to learn compact representations and generalizable policies, it is often necessary to go beyond the end-to-end training paradigm. This is especially true when the initial learning setting (source domain) and current learning setting (target domain) have sufficiently different visual properties (e.g. differing visual fidelity as seen in Figure 5.1 *left*) or different objectives (e.g. transfer from one task to another as in Figure 5.1 *right*). In this section we outline the learning tasks we use, and our strategy for training a network which transfers to new visual domains and new embodied tasks.

5.3.1 Embodied Tasks

In this chapter, we focus on the following three visual navigation tasks which require memory, planning, and geometric understanding: Point-to-Point Navigation (Point-Nav), Scene Exploration

(Exploration), and Run Away from Location (Flee). In our experiments, all tasks share a discrete action space: Move Forward by 0.25 meters and Rotate Left/Right by 10 degrees.

1. **Point-to-Point Navigation (Point-Nav)** An agent is directed to go to a point via a constantly updating tuple of (angle to goal, distance to goal). The agent succeeds if it ends the episode within a fixed radius of the goal. In our experiments we use a success radius of 0.2 meters and the agent is spawned anywhere from 1 to 30 meters from the goal. The agent is provided with a one-hot encoding of its previous action. Since the agent is given the distance to the goal, learning the Stop action is trivial, so we disregard it.

2. **Scene Exploration (Exploration)** We discretize the world-space into 1 meter cubes and count the number of distinct cubes visited by the agent during a fixed duration. This task differs from Point-to-Point Navigation in that no absolute or relative spatial locations are provided to the agent. This prohibits agents from learning to detect collisions by comparing location values from two timesteps, requiring them to visually detect collisions. The agent still receives a one-hot encoding of its previous action.

3. **Run Away from Location (Flee)** The goal of this task is to maximize the geodesic distance from the start location and the agent's final location in episodes of fixed length. As in Exploration, no spatial locations are given to the agent.

5.3.2 Decomposing the Learning Problem

For visual navigation tasks, an agent must **understand what it sees** and it must use the perceived world to **decide what to do**. Thus, we decompose visual navigation into the subtasks of (1) encoding the visual information and (2) using the encoded information to navigate. At each time t the agent receives an egocentric image I_t from the environment and must return a navigation action a_t in order to accomplish the task. Instead of learning actions directly from pixels, we break the decision-making into two stages. First, a function $\mathcal{F}$ processes the image I_t producing a feature embedding $\phi_t = \mathcal{F}(I_t)$. Next, the features are decoded into an action $a_t = \mathcal{G}(\phi_t)$. Our goal is

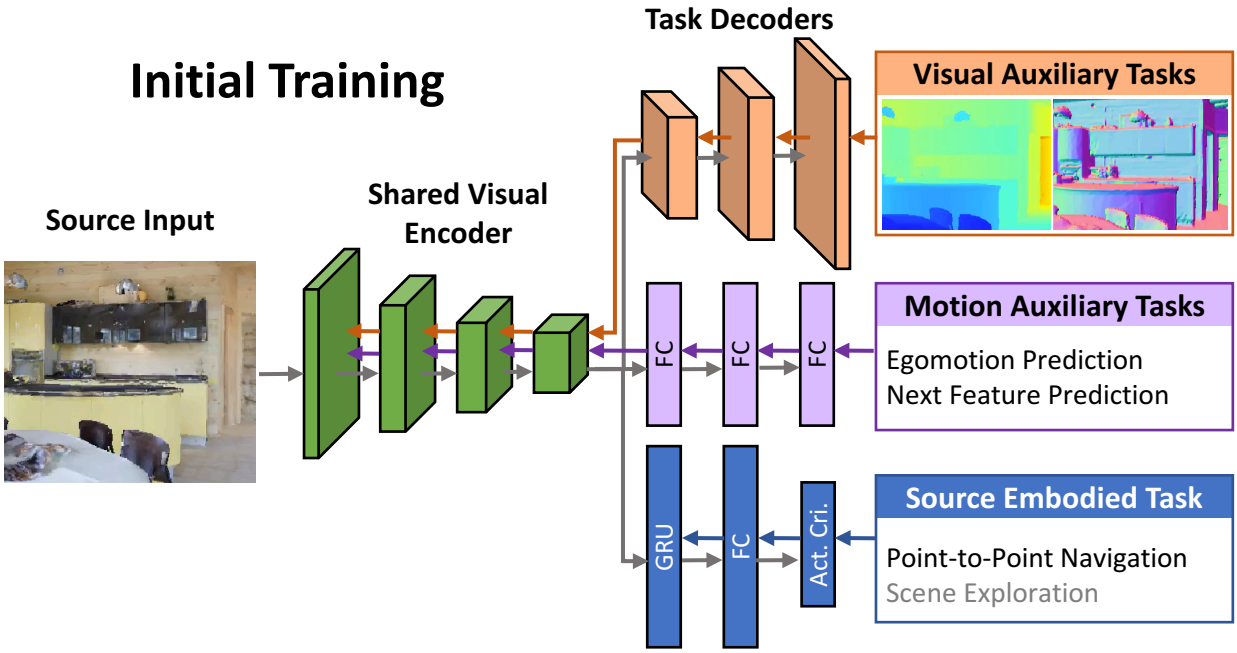


Figure 5.2: **SplitNet initial learning on source data and task.** Given source visual inputs, the visual encoder is trained using auxiliary visual and motion based tasks. Next, the policy decoder is trained on the source embodied tasks with a fixed visual encoder. Gradients from the embodied task (depicted as blue arrows) are stopped before the shared visual encoder to ensure decoupling of the policy and perception.

to learn features ϕ_t which extract salient information for completing navigation tasks and which generalize to new environments. Rather than passively expecting the end-to-end training to result in transferable features, we directly optimize portions of the network with distinct objectives to produce representations which are highly semantically meaningful and transferable.

5.3.3 Visual Encoder

Visual understanding comes in many forms and is highly dependent on the desired end task. In the case of visual navigation, the agent must convert pixel inputs into an implicit or explicit geomet-

ric understanding of the environment's layout. To encapsulate these ideas, we train a bottleneck encoder-decoder network supervised by several auxiliary visual and motion tasks. Each task uses a shared encoder, and produces a general purpose feature, ϕ_t. This feature is then used as input to learn a set of task specific decoders.

Auxiliary Visual Tasks: We encourage the shared encoder to extract geometric information from the raw visual input by augmenting the learning objective with the following auxiliary visual tasks: (1) prediction of depth through a depth decoder, $\mathcal{D}$, (2) prediction of surface normals through a surface normal decoder, $\mathcal{S}$, and (3) RGB reconstruction through a reconstruction decoder, $\mathcal{R}$ (sample outputs are shown in Appendix D.4). For an input image I_t with ground truth depth, D_t and ground truth surface normals S_t, the learning objective for each of these auxiliary visual decoders is as follows:

$$\mathcal{L}_D = \sum_{pixels} \|\mathcal{D}(\phi_t) - D_t\|_1 \tag{5.1}$$

$$\mathcal{L}_S = 1 - \sum_{pixels} \frac{\mathcal{S}(\phi_t) \cdot S_t}{\|\mathcal{S}(\phi_t)\|_2 * \|S_t\|_2} \tag{5.2}$$

$$\mathcal{L}_R = \sum_{pixels} \|\mathcal{R}(\phi_t) - I_t\|_1 \tag{5.3}$$

We use the ℓ_1 loss for reconstruction and depth to encourage edge sharpness. We use the cosine loss for the surface normals as it is a more natural fit for an angular output.

Auxiliary Motion Tasks: We additionally encourage the visual encoder to extract information which may be generically useful for future embodied tasks by adding the following auxiliary motion tasks: (1) predict the egomotion (discrete action) of the agent with motion decoder $\mathcal{E}$, and (2) forecast the next features given the current features and a one-hot encoding of the action performed with motion decoder $\mathcal{P}$. For a visual encoding ϕ_t at time t, previous encoding ϕ_{t-1}, and action a_t that causes the agent to move from I_{t-1} to I_t, the learning objective for each of these auxiliary

motion decoders is as follows:

$$\mathcal{L}_E = -\sum_{a \in A} p(a_t = a) \log(\mathcal{E}(\phi_t, \phi_{t-1})) \tag{5.4}$$

$$\mathcal{L}_P = 1 - \sum_{features} \frac{\mathcal{P}(\phi_{t-1}, a_t) \cdot \phi_t}{\|\mathcal{P}(\phi_{t-1}, a_t)\|_2 * \|\phi_t\|_2} \tag{5.5}$$

We use the cross-entropy loss as we use a discrete action space, and we use the cosine loss for next feature prediction as it directly normalizes for scale which stops the network from forcing all the features arbitrarily close to 0.

All objectives affecting the learning of the visual encoder can be summarized in the joint loss:

$\mathcal{L} = \lambda_R \mathcal{L}_R + \lambda_D \mathcal{L}_D + \lambda_S \mathcal{L}_S + \lambda_E \mathcal{L}_E + \lambda_P \mathcal{L}_P$

where $\lambda_R, \lambda_D, \lambda_S, \lambda_E, \lambda_P$ are scalar hyperparameters which control the trade-off between the various tasks in this multi-task learning objective.

Rather than expecting our network to learn to extract geometric information decoupled from the policy decoders, we force the visual representation to contain this information directly. This decreases the likelihood of overfitting to training environments and thus increases the likelihood that our model generalizes to unseen environments.

5.3.4 Policy Decoder

Our policy decoder takes as input the visual features ϕ_t and learns to predict a desired action, a_{t+1}, supervised by a reward signal provided by the desired task. To avoid purely reactive policies, we employ a GRU [33] to add temporal context. The output of the policy layers predicts a probability distribution over the discretized action space and a value estimate for the current state. The probability distribution is sampled to determine which action to perform next.

When training the policy decoder, we fix our visual encoder and optimize only the policy decoder weights for the chosen task i.e. **gradients do not propagate from the source task to the visual layers** (see Figure 5.2 for an illustration of the gradient flow from the embodied task loss). This prevents policy information from leaking into the visual representation, ensuring the visual

encoder generalizes well for many tasks. For the task of Point-to-Point Navigation we use two training strategies: **BC** and **BC, PPO**.

BC: We train the agent using behavioral cloning (BC) where the ground truth represents the action which would maximally decrease the geodesic distance between the current position and the goal. This is trained in a "student-forcing" regime i.e. the agent executes actions based on its policy, but evaluates the actions using the ground truth.

BC, PPO: We initialize the agent with the weights from the BC setting and update only the policy layers using the PPO algorithm [192] with a shaped reward based on the geodesic distance to the goal, $Geo(P, G)$:

$$r_t^{pointnav} = Geo(P_{t-1}, G) - Geo(P_t, G) + \lambda_T \tag{5.6}$$

where P_t is the agent's location at time t, G is the goal location, and λ_T is a small constant time penalty.

5.3.5 Selective Transfer to New Domains and Tasks

Adapting to new Visual Domains By decomposing the learning task into a perceptual encoder and a policy decoder, each supervised by their own objectives, our model is able to learn more transferable visual features than end-to-end methods. Furthermore, our model can quickly adapt its perceptual understanding with auxiliary visual and motion based training in the target environment without needing to modify the policy. Figure 5.3 illustrates the visual encoder adaptation learning procedure. Given a small sample of data and tasks in the target domain, we backpropagate gradients through the policy decoder[2] and the auxiliary task layers **but freeze the weights for all but the shared visual encoder**. By doing so, our model can quickly adapt its perception without overfitting the policy to the small sample.

[2]Without propagating gradients through the policy decoder, the encoder feature representation shifts and no longer matches the policy decoder.

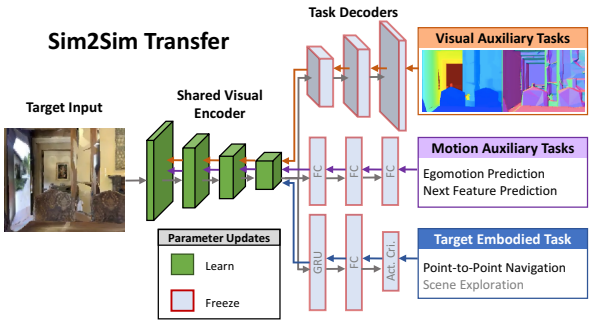


Figure 5.3: **SplitNet visual domain transfer.** When the target visual inputs differ from the source visual inputs while the desired embodied task remains fixed, our model updates the shared visual encoder using only auxiliary visual and motion based learning tasks. All decoder weights are frozen (to prevent overfitting), but gradients propagate through all decoder layers to the encoder.

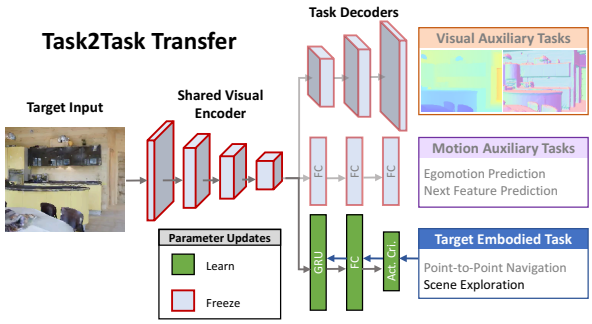


Figure 5.4: **SplitNet task transfer.** When learning a new embodied task for the same visual inputs as in the source initial learning, our model fixes the shared visual encoder and updates the policy decoder using the new target embodied loss.

Adapting to new Tasks When transferring to a new embodied task operating in the same visual space, our model only needs to update the policy decoder parameters (see Figure 5.4). While reusing lower-level features for new tasks by replacing and retraining the final layers is a common technique in deep learning [66, 104, 69] our model naturally decouples perception and reasoning offering a clear solution as to which layers to freeze or finetune. Rich perceptual features often transfer to tasks which require different reasoning assuming that the representation encodes the necessary information for the new task. By using auxiliary tasks to inform the updates to the visual encoder, we aim to encourage learning of intermediate features that capture semantically meaningful information which should better transfer to new tasks than arbitrary latent features. For example, the latent features from a purely end-to-end learned model may represent a variety of different (sometimes spurious) correlations, while our features must contain enough information to reconstruct depth and surface normals etc., so the network should be able to, for instance, avoid obstacles using the exact same features. While this implies that the selection of an appropriate

auxiliary task affects the success of our method, if necessary our network can still be trained end-to-end using the pretrained weights as initialization.

In the specific cases of transferring from Point-Nav to Exploration or Flee we initialize the model with the weights from the **BC, PPO** setting and update the policy decoder layers using PPO with the new reward functions:

$$r_t^{explore} = \|Visited_t\| - \|Visited_{t-1}\| + \lambda_T \tag{5.7}$$

$$r_t^{flee} = Geo(P_t, P_{t_0}) - Geo(P_{t-1}, P_{t_0}) + \lambda_T \tag{5.8}$$

where $\|Visited_t\|$ represents how many unique spatial locations the agent has visited at time t.

5.4 Experiments

To evaluate visual navigation tasks we use the Habitat scene renderer [187] on scenes from the near-photo-realistic 3D room datasets Matterport 3D (referred to as MP3D) [26] and Gibson [237] as well as a third 3D navigation dataset (referred to as IndoorEnv).

5.4.1 Baselines

We compare our results against traditional end-to-end (E2E) training algorithm results for all experiments. These can be trained via the PPO algorithm, via behavioral cloning (BC), or pretrained with BC and finetuned with PPO. One common technique across deep learning is to pretrain models on ImageNet [45], and finetune the entire network on the desired task, which we also include as a baseline. We do not freeze any weights when training E2E methods. We additionally include blind (but learned) agents for each task and random-action agents to benchmark task difficulty. For Point-Nav, we also include a Blind Goal Follower which aligns itself in the direction of goal vector and moves forward, realigning after it collides with obstacles.

5.4.2 Generalization to Unseen Environments

The ability for an algorithm to generalize to unseen environments represents its effectiveness in real-world scenarios. To begin analyzing our model, we experiment with the standard protocol

Method	IndoorEnv		MP3D [26]		Gibson [237]	
	SPL	Success	SPL	Success	SPL	Success
Random	0.012	0.027	0.011	0.016	0.046	0.028
Blind Goal Follower	0.199	0.203	0.155	0.158	0.325	0.319
Blind BC	0.159	0.323	0.232	0.382	0.351	0.603
Blind BC, PPO	0.291	0.377	0.317	0.471	0.427	0.643
Blind PPO	0.258	0.371	0.313	0.463	0.538	0.822
E2E PPO	0.324	0.529	0.322	0.477	0.634	0.831
E2E BC	0.343	0.548	0.459	0.737	0.509	0.824
E2E BC, PPO	0.393	0.593	0.521	0.733	0.606	0.869
ImageNet Pretrain, E2E BC	0.280	0.499	0.315	0.552	0.548	0.843
ImageNet Pretrain, E2E BC, PPO	0.338	0.440	0.450	0.539	0.642	0.737
SplitNet BC	0.421	0.687	0.517	0.808	0.584	0.865
SplitNet BC, PPO	**0.560**	0.703	**0.716**	0.844	**0.701**	0.855

Table 5.1: **Performance on Unseen Environments.** Blind methods are not provided with visual input but still receive an updated goal vector. "BC, PPO" methods are first trained with a softmax loss to take the best next action and are finetuned with the PPO algorithm.

of training and evaluating on data from the same simulator, partitioning the scenes into train and test. We compare performance for the Point-Nav task on three simulators (IndoorEnv, MP3D [26], Gibson [237]) evaluating in never-before-seen scenes. We use the SPL metric proposed in [11] which can be stated as

$$SPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{\ell_i}{\max(p_i, \ell_i)} \tag{5.9}$$

where S_i is a success indicator for episode i, p_i is the path length, and ℓ_i is the shortest path length. This combines the accuracy (success) of a navigation method with its efficiency (path length) where 1.0 would be an oracle agent.

Effective policies generalize by understanding the geometry of the scenes rather than trying to localize into a known map based on the visual inputs. SplitNet outperforms all other methods by a wide margin on all three environments (shown in Table 5.1). Surprisingly, pretraining on ImageNet does not offer better generalization, likely because the features required for ImageNet are sufficiently different from those needed to navigate effectively (note, the convolutional weights trained on ImageNet *are not frozen* during BC and PPO training). This is true even compared to

	Number Train Scenes		Test Data		Number Train Scenes		Test Data	
	IndoorEnv	MP3D (Train)	MP3D (Val)		MP3D	Gibson (Train)	Gibson (Val)	
	(*Source*)	(*Target*)	SPL	Success	(*Source*)	(*Target*)	SPL	Success
Source E2E BC, PPO	990	0	0.257	0.412	61	0	0.609	0.866
Source SplitNet BC, PPO	990	0	**0.376**	0.539	61	0	**0.651**	0.764
Target E2E BC	0	1	0.211	0.321	0	1	0.396	0.589
Target E2E Finetune	990	1	Failure	Failure	61	1	Failure	Failure
Target SplitNet Transfer	990	1	**0.447**	0.596	61	1	**0.686**	0.822
Target E2E BC	0	10	0.259	0.463	0	10	0.501	0.782
Target E2E Finetune	990	10	0.401	0.612	61	10	0.667	0.870
Target SplitNet Transfer	990	10	**0.531**	0.681	61	10	**0.727**	0.854
Target E2E BC, PPO	0	All (61)	0.521	0.733	0	All (72)	0.606	0.869
Target SplitNet BC, PPO	0	All (61)	**0.716**	0.844	0	All (72)	**0.701**	0.855

Table 5.2: **Performance transferring across simulation environments (Sim2Sim).** Our method, SplitNet, significantly outperforms the end-to-end (E2E) baseline at the task of transferring across simulated environments. For reference, we also report the performance of a source only trained model (top two rows) or a target only trained model (bottom two rows). "Failure" indicates that performance on the target data decreases after finetuning.

E2E without pretraining on ImageNet.

As qualitative intuition about the performance of the various methods, we depict the policies for a subset of methods on an example MP3D episode from in Figure 5.5. For a fixed start (blue diamond) and goal (green star) location, we show the output trajectory from each method where the trajectory color (ranging from blue to red) denotes the number of steps so far. If a policy failed to reach the goal, the final destination is denoted with a red "x". From this visualization we can see that SplitNet using BC and PPO successfully completes the task and does so with the shortest overall path. At the beginning of the episode SplitNet BC is stuck behind the wall, but eventually is able to navigate away from the wall and reach the target.

We further analyze the performance of SplitNet compared to baselines as a function of the geodesic distance between the starting and goal locations in Figure 5.6. This distance is highly correlated with the difficulty of an episode. Unsurprisingly, all methods degrade as the starting

location is moved further from the goal location, but SplitNet retains its advantage over baselines irrespective of the episode difficulty. Additionally, we see the performance gap widen over the more difficult episodes, meaning we handle difficult episodes better than the baselines.

5.4.3 Transfer Across Simulators

We now study the ability for our method to transfer between visual environments for the fixed task of Point-Nav. We denote *Source* to be the initial simulator in which we train our model using both BC and PPO and denote this initial model as "Source SplitNet BC, PPO." The baseline source model that uses end-to-end training is denoted as "Source E2E BC, PPO." We then compare our method for transfer to the new simulator *Target*, described in Section 5.3 and denoted as "Target SplitNet Transfer," against the end-to-end baseline finetuned on the target, "Target E2E Finetune." For reference, we also present the performance of training an end-to-end model using only the

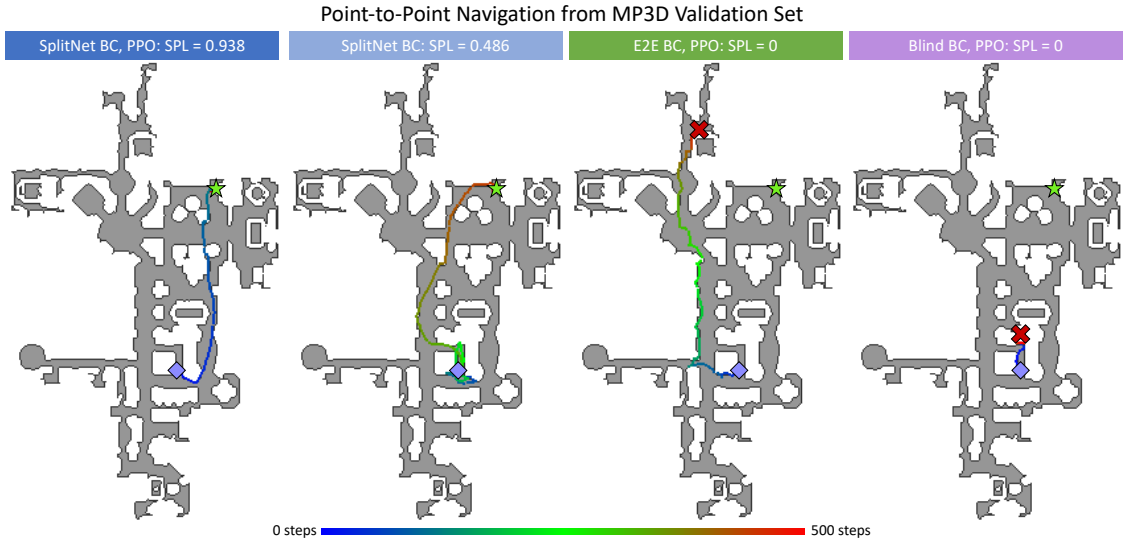


Figure 5.5: **Qualitative comparison of Point-Nav policies on MP3D validation.** An exemplar validation episode (fixed start and end location) and the predicted trajectories from baselines and SplitNet.

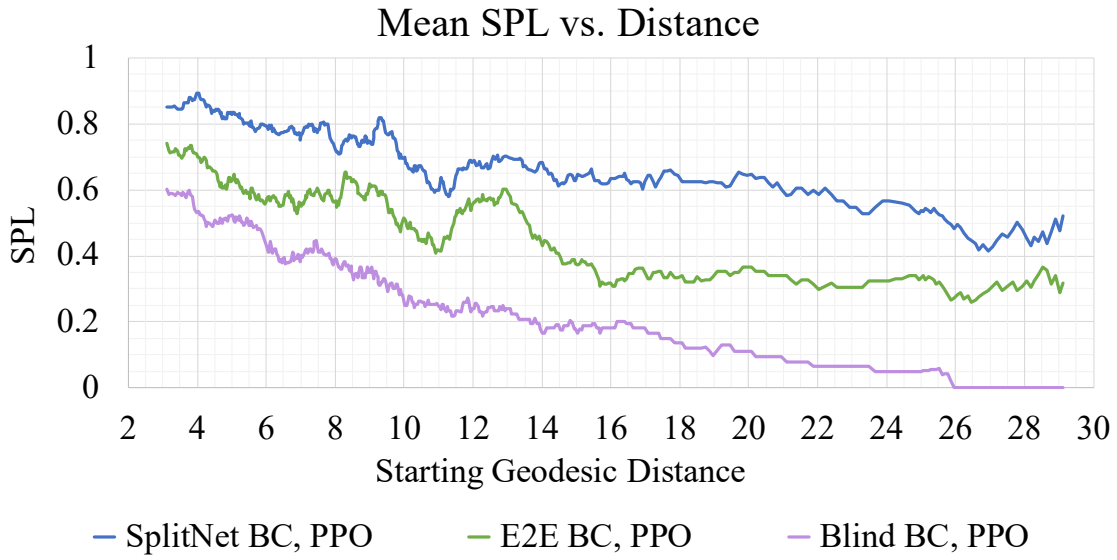


Figure 5.6: **MP3D Point-Nav Performance vs episode difficulty.** We compare our method, SplitNet, to end-to-end (E2E) and blind learned baselines and report SPL performance as a function of starting geodesic distance from the goal. SplitNet outperforms on all starting distances, especially on the more difficult episodes.

available target data, denoted as "Target E2E BC."

Table 5.2 reports our main results for this Sim2Sim transfer problem as a function of the amount of available target scenes during training. We report performance for the two transfer settings of IndoorEnv→MP3D and MP3D→Gibson. These simulators differ in terms of complexities of differing rendering appearance (as seen in Figure 5.1), different environment construction methods (synthetic vs. depth-scan reconstruction), and different environment size. Again, SplitNet outperforms all baselines across all experiments in terms of the SPL metric and performs better or comparable to the baseline in terms of success for all transfer setups. Even with no extra data, our initially learned network is more generalizable to new environments, especially those which are significantly different in appearance (IndoorEnv→MP3D). Of note, in both cases, SplitNet given 10 scenes from the target dataset matches or outperforms the end-to-end baseline SPL given the entire target dataset.

SplitNetModel	Layers Finetuned	Number Target Scenes	SPL	Success
Transfer IndoorEnv → MP3D (train): Eval MP3D (val)				
Source Only	-	-	0.376	0.539
Finetune Target	V+P	1	0.435	0.586
Finetune Target	V	1	**0.447**	0.596
Finetune Target	V+P	10	0.400	0.552
Finetune Target	V	10	**0.531**	0.681
Transfer MP3D → Gibson (train): Eval Gibson (val)				
Source Only	-	-	0.651	0.764
Finetune Target	V+P	1	Failure	Failure
Finetune Target	V	1	**0.686**	0.822
Finetune Target	V+P	10	Failure	Failure
Finetune Target	V	10	**0.727**	0.854

Table 5.3: **Ablation of SplitNet Sim2Sim transfer strategy.** SplitNet only updates the visual encoder ("V") and freezes the policy decoder ("P") when finetuning the source model. In contrast, finetuning both V+P on the target leads to degraded performance.

Note, that our approach to visual environement transfer includes finetuning only the visual encoder in the target environment and leaving the policy decoder fixed. One may wonder whether this is the optimal approach or whether our method would benefit from target updates to the policy decoder as well. To answer this question, in Table 5.3 we report performance comparing the initial source SplitNet performance to that of finetuning either only the visual encoder ("V") which is our proposed approach or finetuning both the visual encoder and policy decoder ("V+P"). Interestingly, we found that allowing updates to both the visual encoder and policy decoder in the target environment lead to significant overfitting which resulted in failed generalization to the unseen scenes from the validation sets. This confirms the benefit of our split training approach.

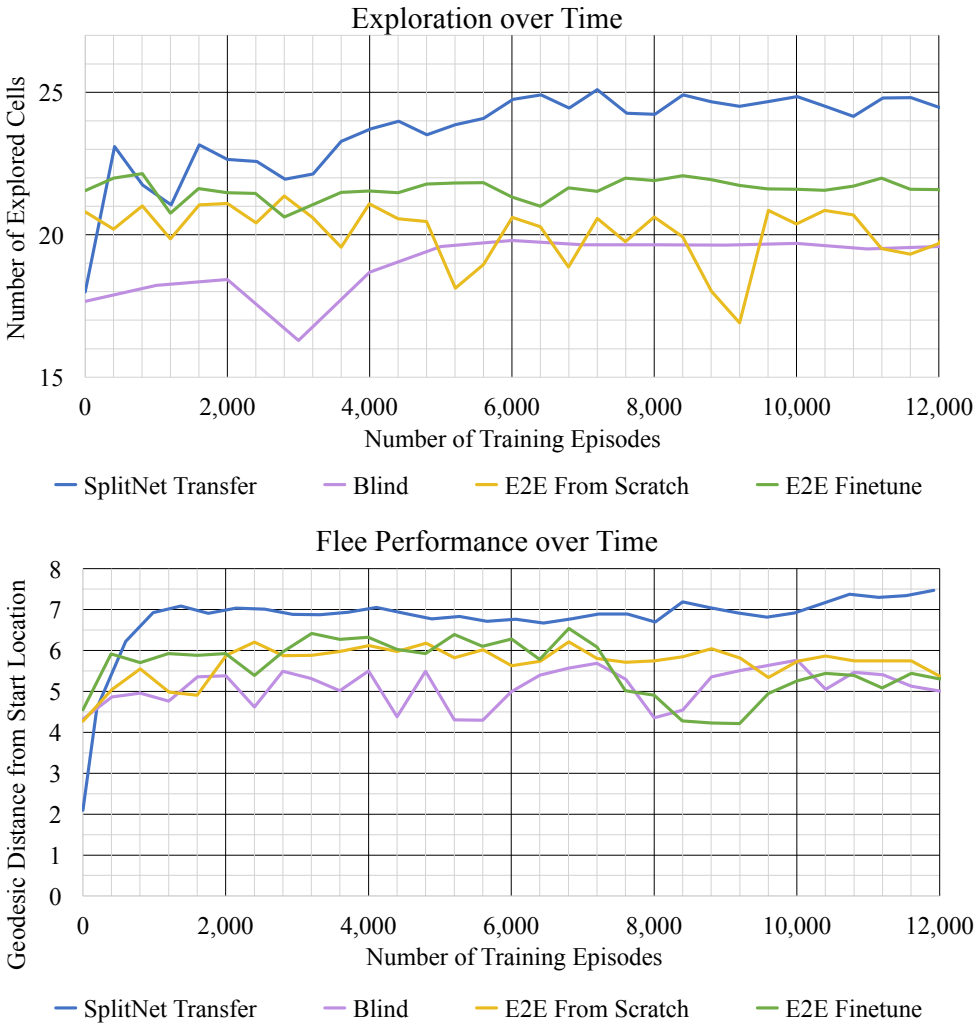


Figure 5.7: **IndoorEnv Task2Task performance as a function of target training episodes.** SplitNet Transfer and E2E Transfer are first trained on IndoorEnv Point-Nav, but SplitNet only updates the policy layers whereas E2E updates the entire network. E2E from scratch is randomly initialized a episode 0. The Blind method only receives its previous action as input and is randomly initialized. Oracle agents perform at 33.5 and 19.5 respectively.

5.4.4 Transfer Across Tasks

We test the ability for SplitNet to learn a new task by first training the network on Point-Nav and using our approach to transfer the model to the novel tasks of Exploration and Flee. All three tasks require the ability to transform 2D visual inputs into 3D scene geometry, but the decisions about what to do based on the perceived geometry are drastically different. Since SplitNet decouples the policy from the perception, it learns features which readily transfer to a novel, but related task.

Figure 5.7 shows that SplitNet immediately begins to learn effective new policies, outperform-

ing the other methods almost right away. In Exploration, our method is able to reuse its understanding of depth to quickly learn to approach walls, then turn at the last second and head off in a new direction. For the Flee task, our method identifies long empty hallways and navigates down those away from the start location. None of the other methods learn robust obstacle-avoidance behavior or geometric scene understanding. Instead they latch on to simple dataset biases such as "repeatedly move forward then rotate." Oracle agents perform at 33.5 and 19.5 respectively, but are not directly comparable in that they benefit from knowing the environment layout before beginning the episode.

5.4.5 Analysis of Auxiliary Objectives

Our method was designed as a solution for generalization on a downstream embodied task. However, SplitNet also learns outputs for the auxiliary visual and motion tasks. While our goal is not to surpass state-of-the-art performance on these auxiliary tasks it is still useful to verify that the visual encodings match our expectations. We therefore include several examples which show the auxiliary outputs in Appendix D.4. In our our initial experiments, we found depth and normal estimation to be the most important auxiliary task as they most directly translate to navigation understanding (free space, geometry, etc.).

5.5 Conclusion

This chapter introduces SplitNet, a method for decomposing embodied learning tasks to enable fast and accurate transfer to new environments and new tasks. By disentangling the visual encoding of the state from the policy for a task, we learn more robust features which can be frozen or adapted based on the changed domain. Our model matches the performance of end-to-end methods even with six times less data. We believe SplitNet may prove to be a useful stepping stone in transferring networks from simulation environments onto robots in the real-world.

Chapter 6

IQA: VISUAL QUESTION ANSWERING IN INTERACTIVE ENVIRONMENTS

The prior two chapters used network structures and gradient flows to decompose the learned models. This chapter as well as the next opt for fully separated hierarchical models as they solve the combined tasks of object detection, navigation, and planning. This decomposition allows us to train each sub-task separately, and combine them in the end for better accuracy and generalizability. Although end-to-end solutions are easier to train, hierarchical systems often result in better performance because they can specialize in each subtask and solve each task well.

6.1 Introduction

A longstanding goal of the artificial intelligence community has been to create agents that can perform manual tasks in the real world and can communicate with humans via natural language. For instance, a household robot might be posed the following questions: *Do we need to buy more milk?* which would require it to navigate to the kitchen, open the fridge and check to see if there is sufficient milk in the milk jug, or *How many boxes of cookies do we have?* which would require the agent to navigate to the cabinets, open several of them and count the number of cookie boxes. Towards this goal, Visual Question Answering (VQA), the problem of answering questions about visual content, has received significant attention from the computer vision and natural language processing communities. While there has been a lot of progress on VQA, research by and large focuses on answering questions passively about visual content, i.e. without the ability to interact with the environment generating the content. An agent that is only able to answer questions passively is limited in its capacity to aid humans in their tasks.

We introduce **Interactive Question Answering (IQA)**, the task of answering questions that

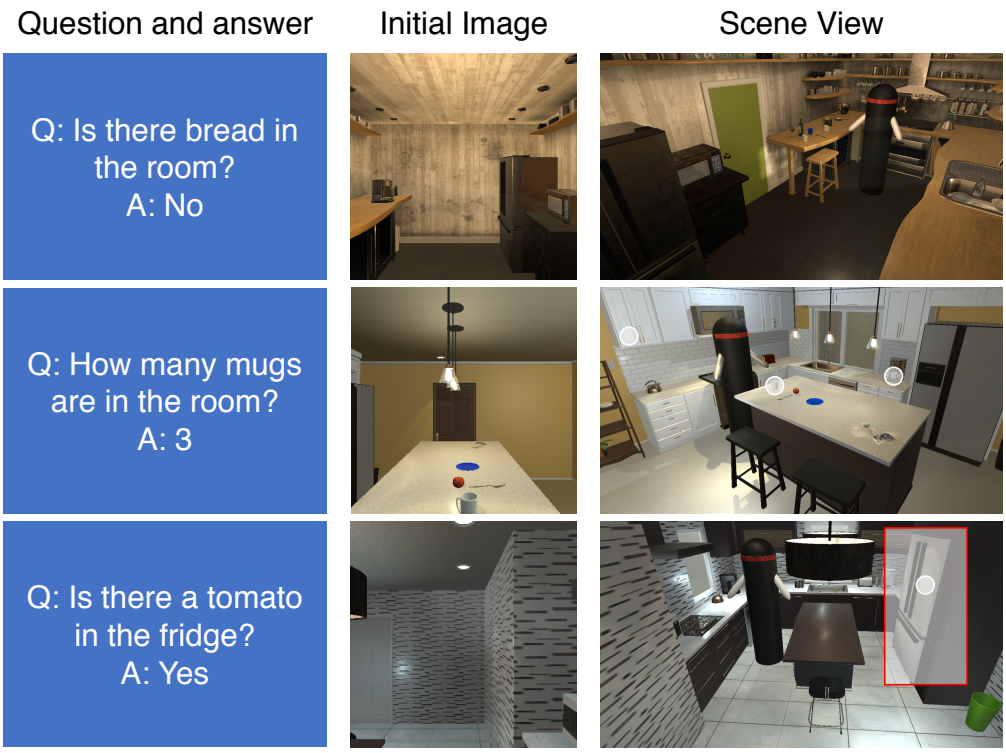


Figure 6.1: **Samples from IQUAD V1.** Each row shows a question paired with the agent's initial view and a scene view of the environment (which is not provided to the agent). In the scene view, the agent is shown in black, and the locations of the objects of interest for each question are outlined. Note that none of the questions can be answered accurately given only the initial image.

require the agent to interact with a dynamic environment. IQA poses several key challenges in addition to the ones posed by VQA. **First**, the agent must be able to navigate through the environment. **Second**, it must acquire an understanding of its environment including objects, actions, and affordances. **Third**, the agent must be able to interact with objects in the environment (such as opening the microwave, picking up books, etc.). **Fourth**, the agent must be able to plan and execute a series of actions in the environment conditioned on the questions asked of it.

To address these challenges, we propose HIMN (Hierarchical Interactive Memory Network). Figure 6.2 provides an overview of HIMN. Akin to past works on hierarchical reinforcement learning, HIMN is factorized into a hierarchy of controllers, allowing the system to operate, learn, and

reason across multiple time scales while simultaneously reducing the complexity of each individual subtask. A high level controller, referred to as the *Planner* chooses the task to be performed (for example, navigation / manipulation / answering / etc.) and generates a command for the chosen task. Tasks specified by the *Planner* are executed by a set of low level controllers (*Navigator*, *Manipulator*, *Detector*, *Scanner* and *Answerer*) which return control to the *Planner* when a task termination state is reached. Since these subtasks are fairly independent, we can pretrain each controller independently, while assuming oracle versions of the remaining controllers. Our experiments show that this factorization enables higher accuracy and generalization to unseen environments.

Several question types require the agent to remember where it has been and what it has seen. For example, *How many pillows are in this house?* requires an agent to navigate around the rooms, open closets and keep track of the number of pillows it encounters. For sufficiently complex spaces, the agent needs to hold this information in memory for a long time. This motivates the need for an explicit external memory representation that is filled by the agent as it interacts with its environment. This memory must be both spatial and semantic so it can represent *what* is *where*. We propose a new recurrent layer formulation: Egocentric Spatial GRU (esGRU) to represent this memory (Sec 6.4.1).

Training and evaluating interactive agents in the real world is currently prohibitive from the standpoint of operating costs, scale and research reproducibility. A far more viable alternative is to train and evaluate such agents in realistic simulated environments. Towards this end, we present the Interactive Question Answering Dataset (IQUAD V1) built upon AI2-THOR [120], a photo-realistic customizable simulation environment for indoor scenes integrated with the Unity [1] physics engine. IQUAD V1 consists of over 75,000 multiple choice questions, each question accompanied by a unique scene configuration.

We evaluate HIMN on IQUAD V1 using a question answering accuracy metric and show that it outperforms a baseline based on a common architecture for reinforcement learning used in past work. We evaluate in both familiar and unfamiliar environments to show that our semantic model generalizes well across scenes.

In summary, our contributions include: (a) proposing Interactive Question Answering, the task

of answering questions that require the agent to interact with a dynamic environment, (b) presenting the Hierarchical Interactive Memory Network, a question answering model factorized into a high level *Planner*, a set of low level controllers and a rich semantic spatial memory, (c) the Egocentric Spatial GRU, a new recurrent layer to represent this memory and (d) a new dataset IQUAD V1 towards the task of IQA.[1]

6.2 Related Work

6.2.1 Visual Question Answering (VQA)

VQA has seen significant progress over the past few years, owing to the design of deep architectures suited for this task and the creation of large VQA datasets to train these models [233]. These include datasets of natural images [5, 122, 144, 225], synthetic images [5, 13, 102, 106, 110], natural videos [101, 211], synthetic videos [115, 161] and multimodal contexts [111]. Some of these use questions written by humans [5, 110, 111, 122] and others use questions that are generated automatically [13, 102, 106]. IQUAD V1 is set in a photo-realistic simulation environment and uses automatically generated questions. In contrast to the aforementioned datasets that only require the agent to observe the content passively, IQUAD V1 requires the agent to interact with a dynamic environment.

The first deep architectures designed for VQA involved using an RNN to encode the question, using a CNN to encode the image and combining them using fully connected layers to yield the answer [5, 145].

More recently, modular networks [13, 95, 103] that construct an explicit representation of the reasoning process by exploiting the compositional nature of language have been proposed. Similar architectures have also been applied to the video domain with extensions such as spatiotemporal attention [101, 161]. Our proposed approach to question answering allows the agent to interact with its environment and is thus fundamentally different to past QA approaches. However, we note

[1] Code and IQUAD V1 are available at
`https://github.com/danielgordon10/thor-iqa-cvpr-2018`
Video available at `https://youtu.be/pXd3C-1jr98`

that approaches such as visual attention and modularity can easily be combined with our model to provide further improvements.

6.2.2 Hierarchical Reinforcement Learning (HRL)

RL algorithms have been employed in a wide range of problems including locomotion [118], obstacle detection [151] and autonomous flight [119, 184]. Of particular relevance to our approach is the area of hierarchical reinforcement learning (HRL), which consists of a high level controller and one or more low level controllers. The high-level controller selects a subtask to be executed and invokes one of the low level controllers. The advantage of HRL is that it allows the model to operate at multiple levels of temporal abstraction. Early works proposing HRL algorithms include [48, 169, 207]. More recent approaches include [127] who propose hierarchical-DQN with an intrinsically motivated RL algorithm, [212] who use HRL to create a lifelong learning system that has the ability to reuse and transfer knowledge from one task to another, and [165] who use HRL to enable zero shot task generalization by learning subtask embeddings that capture correspondences between similar subtasks. Our use of HRL primarily lets us learn at multiple time scales and its integration with the semantic memory lets us divide the complex task of IQA into more concrete tasks of navigation, detection, planning etc. that are easier to train.

RL techniques have also recently been applied to QA tasks, most notably by [103] to train a program generator that constructs an explicit representation of the reasoning process to be performed and an execution engine that executes the program to predict the answer.

6.2.3 Visual Navigation

The majority of visual navigation techniques fall into three categories: offline map-based, online map-based, and map-less approaches. Offline map-based techniques [20, 21, 113, 167] require the complete map of the environment to make any decisions about their actions, which limits their use in unseen environments. Online map-based methods [41, 55, 162, 197, 216, 231] often construct the map while exploring the environment. The majority of these approaches use the

computed map for navigation only, whereas our model constructs a rich semantic map which is used for navigation as well as planning and question answering. Map-less approaches [76, 100, 138, 186, 249] which use techniques such as obstacle avoidance and feature matching, depend upon implicit representations of the world to perform navigation, and lack long-term memory capabilities. Recently Gupta *et al*. [74] proposed a joint architecture for a *mapper* that produces a spatial memory and a *planner* that can plan paths. The similarities between our works lie in the usage of a hierarchical system and a spatial memory. In contrast to their work, navigation is not the end goal of our system, but a subtask towards question answering, and our action space is more diverse as it includes interaction and question answering.

6.2.4 Visual Planning

To answer questions such as *Do I need to buy milk?* an agent needs to plan a sequence of actions to explore and interact with the environment. A large body of research on planning algorithms [49, 58, 105, 202, 203] use high-level formal languages. These techniques are designed to handle low-dimensional state spaces but do not scale well to high-dimensional state spaces such as natural images.

Other relevant work includes visual navigation [249] and visual semantic planning (Chapter 4) which both use the AI2-THOR environment [120]. The former tackles navigation, and the latter focuses on high level planning and assumes an ideal low level task executor; in contrast, our model trains low level and high level controllers jointly. Also, both these approaches do not generalize well to unseen scenes, whereas our experiments show that we do not overfit to previously encountered environments. Finally, these methods lack any sort of explicit map, whereas we construct a semantic map which helps us navigate and answer questions.

Recently Chaplot *et al*. [27] and Hill *et al*. [87] have proposed models to complete navigation tasks specified via language (e.g. *Go to the red keycard*) and trained their systems in simulated 3D environments. These models show the ability to generalize to unseen instructions of seen concepts. In contrast, we tackle several question types that require a variety of navigation behaviours and interaction, and the environment we use is significantly more photo-realistic. In our experiments,

we compare our proposed HIMN model to a baseline system (A3C in Section 6.5) that very closely resembles the model architectures proposed in [27] and [87].

6.2.5 Visual Learning by Simulation

There has been an increased use of simulated environments and game platforms to train computer vision systems to perform tasks such as learning the dynamics of the world [158, 159, 232], semantic segmentation [78], pedestrian detection [148], pose estimation [168] and urban driving [7, 29, 177, 179]. Several of these are also interactive making them suitable to learn control, including [16, 112, 120, 131, 236]. We choose to use AI2-THOR [120] in our work since it provides a photo-realistic and interactive environment of real world scenes, making it very suitable to train IQA systems that might be transferable to the real world.

6.3 Learning Framework

6.3.1 Actionable Environment

Training and evaluating interactive agents in the real world is currently prohibitive from the standpoint of operating costs, scale, time, and research reproducibility. A far more viable alternative is to use simulated environments. However, the framework should be visually realistic, allow interactions with objects, and have a detailed model of the physics of the scene so that agent movements and object interactions are properly represented. Hence, we adopt the AI2-THOR environment [120] for our purposes. AI2-THOR is a photo-realistic simulation environment of 120 rooms in indoor settings, tightly integrated with a physics engine. Each scene consists of a variety of objects, from furniture such as couches, appliances such as microwaves and smaller objects such as crockery, cutlery, books, fruit, etc. Many of these objects are actionable such as fridges which can be opened, cups which can be picked up and put down, and stoves which can be turned on and off.

Interactive Question Answering Dataset Statistics		
	Train	Test
Existence	25,600	640
Counting	25,600	640
Spatial Relationships	25,600	640
Rooms	25	5
Total scene configurations (s.c.)	76,800	1,920
Avg # objects per (s.c.)	46	41
Avg # interactable objects (s.c.)	21	16
Vocabulary Size	70	70

Table 6.1: **Dataset Statistics.** This table shows the statistics of our proposed dataset in a variety of question types, objects and scene configurations.

6.3.2 Interactive Question Answering Dataset

IQUAD V1 is a question answering dataset built upon AI2-THOR [120]. It consists of over 75,000 multiple choice questions for three different question types (table 6.1 shows more detailed statistics). Each question is accompanied by a scene identifier and a unique arrangement of movable objects in the scene. Figure 6.1 shows three such examples. The wide variety of configurations in IQUAD V1 prevent models from memorizing simple rules like "apples are always in the fridge" and render this dataset challenging. IQUAD V1 consists of several question types including: Existence questions (*Is there an apple in the kitchen?*), Counting questions (*How many forks are present in the scene?*), and Spatial Relationship questions (*Is there lettuce in the fridge? / Is there a cup on the counter-top?*). Questions, ground truth answers, and answer choices are generated automatically. Since natural language understanding is not a focus of this dataset, questions are generated using a set of templates written down a priori. IQUAD V1 is a balanced dataset that prevents models from obtaining high accuracies by simply exploiting trivial language and scene configuration biases. Similar to past balanced VQA datasets [69], each question is associated with multiple scene

configurations that result in different answers to the question. We split the 30 kitchen rooms into 25 train and 5 test, and have 1024 unique (question, scene configuration) pairs for each (room, question type) pair in train, and 128 in test. An episode is finished when the *Answerer* is invoked. We evaluate different methods using Top-1 accuracy.

6.3.3 Agent and Objects

The agent in our environments has a single RGB camera mounted at a fixed height. An agent can perform one of five navigation actions (move ahead 25 cm, rotate 90 degrees left or right, look up or down 30 degrees). We assume a grid-world floor plan that ensures that the agent always moves along the edges of a grid and comes to a stop on a node in this grid. The agent can perform two interaction actions (open and close) to manipulate objects. A wide variety of objects (fridges, cabinets, drawers, microwaves, etc.) can be interacted with. If there are multiple items in the current viewpoint which can be opened or closed, the environment chooses the one nearest to the center of the current image. The success of each action depends on the current state of the environment as well as the agent's current location. For instance, the agent cannot open a cabinet that is more than 1 meter away or is not in view, or is already open, and it cannot walk through a table or a wall.

6.4 Hierarchical Interactive Memory Networks

We propose the HIMN (Hierarchical Interactive Memory Network) framework, consisting of a hierarchy of controllers that operate at multiple levels of temporal abstraction and a rich semantic memory that aids in navigation, interaction, and question answering. Figure 6.2 provides an overview of HIMN. We now describe each of HIMN's components in greater detail.

6.4.1 Spatial Memory

Several question types require the agent to keep track of objects that it has seen in the past along with their locations. For complex scenes with several locations and interactable objects, the agent

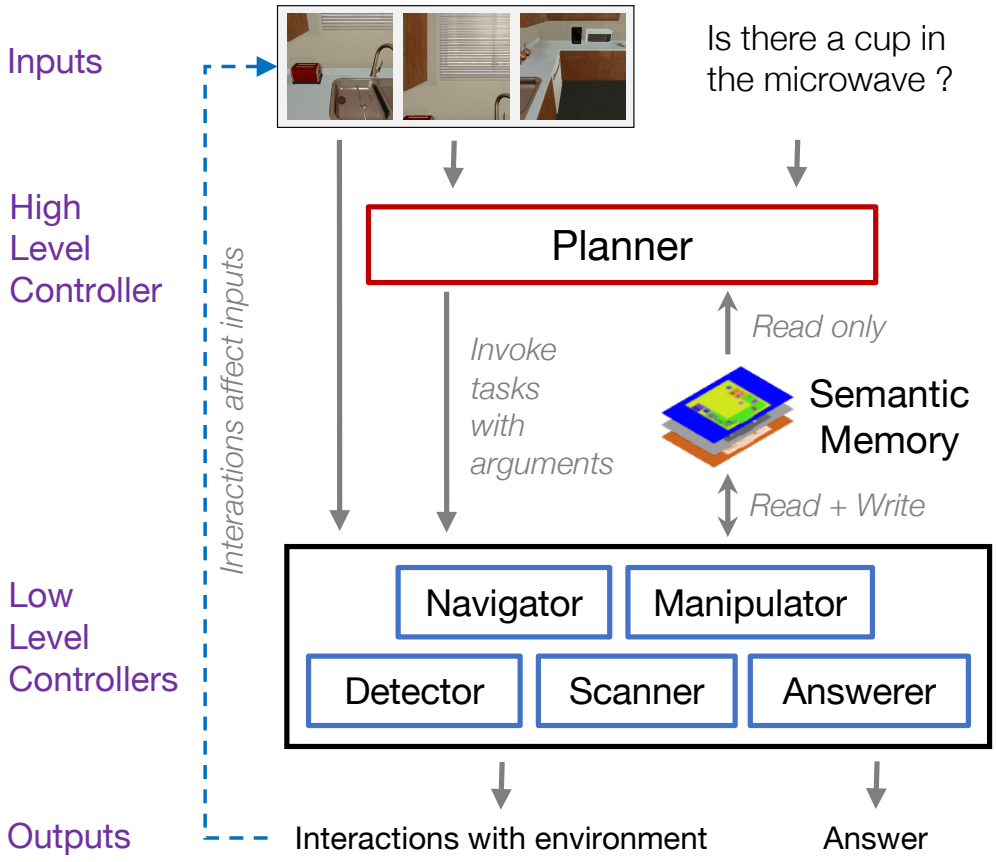


Figure 6.2: **An overview of the Hierarchical Interactive Memory Network.**

needs to hold this information in memory for a long duration. This motivates the need for an explicit external memory representation that is filled by the agent on the fly and can be accessed at any time. To address this, HIMN uses a rich semantic spatial memory that encodes a semantic representation of each location in the scene. Each location in this memory consists of a feature vector encoding object detection probabilities, free space probability (a 2D occupancy grid), coverage (has the agent inspected this location before), and navigation intent (has the agent attempted to visit this location before). We propose a new recurrent layer formulation: Egocentric Spatial GRU (esGRU) to represent this memory, illustrated in Figure 6.3. The esGRU maintains an external global spatial memory represented as a 3D tensor. At each time step, the esGRU swaps in

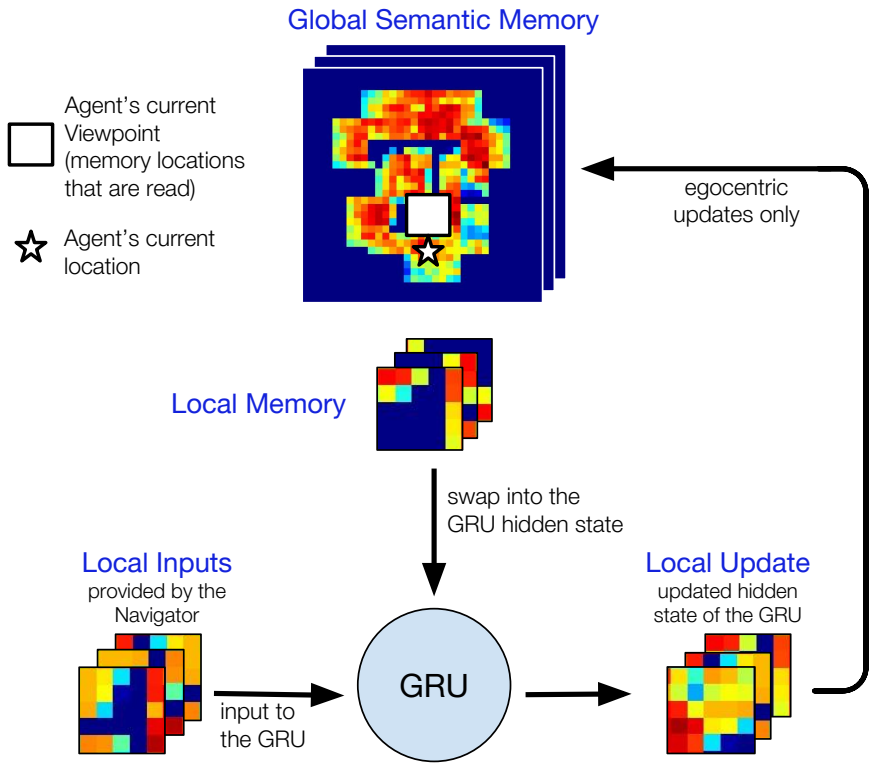


Figure 6.3: **An overview of the Egocentric Spatial GRU (esGRU).** The esGRU only allows writing to a local window within the memory, dependent on the agent's current location and viewpoint.

local egocentric copies of this memory into the hidden state of the GRU, performs computations using current inputs, and then swaps out the resulting hidden state into the global memory at the predetermined location. This speeds up computations and prevents corrupting the memory at locations far away from the agent's current viewpoint. When navigating and answering questions, the agent can access the full memory, enabling long-term recall from observations seen hundreds of states prior. Furthermore, only low level controllers have read-write access to this memory. Since the *Planner* only makes high level decisions, without interacting with the world at a lower level, it only has read access to the memory.

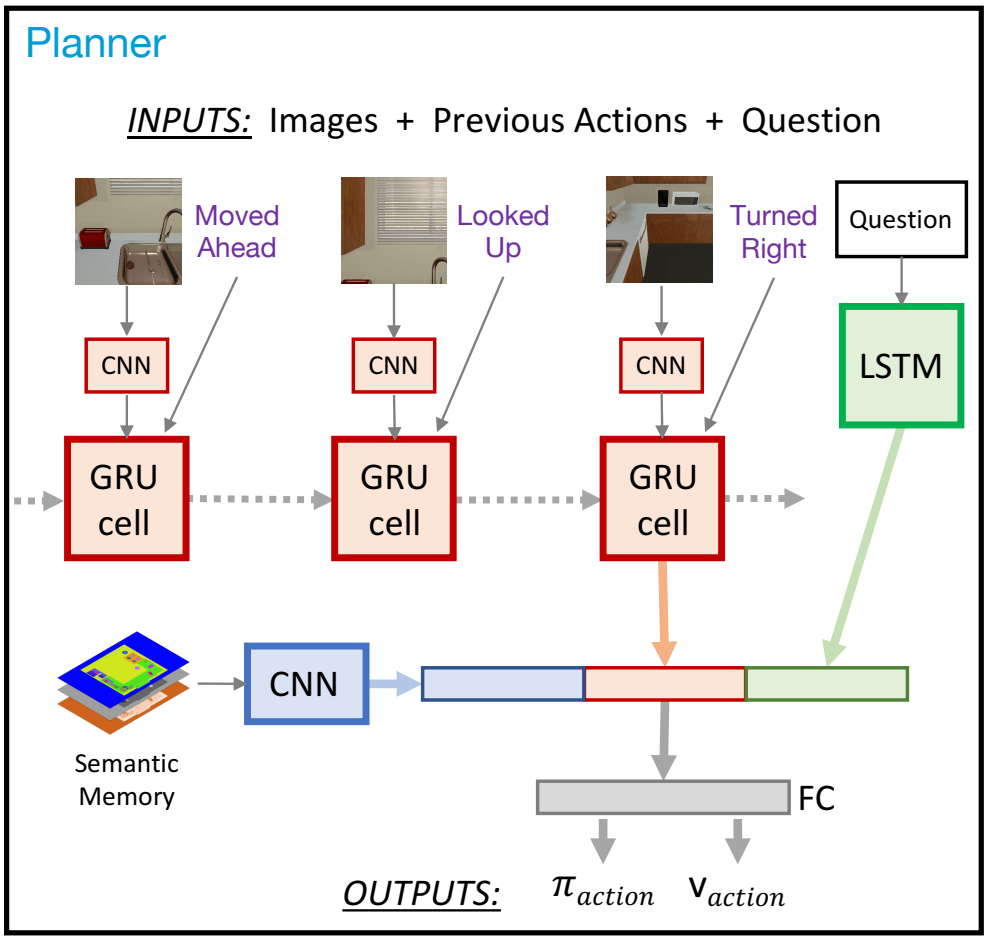


Figure 6.4: **Schematic representation of the *Planner*.**

6.4.2 Planner

The high level *Planner* invokes low level controllers in order to explore the environment, gather knowledge needed to answer the given question, and answer the question. We frame this as a reinforcement learning problem where the agent must issue the fewest possible commands that result in a correct answer. The agent must learn to explore relevant areas of the scene based on learned knowledge (e.g. apples are often in the fridge, cabinets are openable, etc.), the current memory state (e.g. the fridge is to the left), current observations (e.g. the fridge is closed) and the question. At every timestep, the *Planner* chooses to either invoke the *Navigator* providing a relative location in a 5x5 grid in front of the agent, invoke the *Scanner* with a direction such as up or left, invoke the *Manipulator* with open/close commands on a nearby object, or invoke the

Answerer for a total of 32 discrete actions. It does this by producing a policy π consisting of probabilities π_i for each action, and a value v for the current state. π and v are learned using the A3C algorithm [156]. Figure 6.4 shows a schematic of the *Planner*. It consists of a GRU which accepts at each time step the current viewpoint (encoded by a CNN) and the previous action. The *Planner* has read only access to the semantic memory centered around the agent's current location. The output of this GRU is combined with the question embedding and an embedding of the nearby semantic spatial memory to predict π and v. The agent receives a fixed reward/penalty based on answering correctly/incorrectly. It is also provided a constant time penalty to encourage efficient explorations of the environment and quick answering, as well as a penalty for attempting to perform invalid actions. The agent is also given intermediate rewards for increasing the "coverage" of the environment, effectively training the network to maximize the amount of the room it has explored as quickly as possible. Finally, at each time step, the *Planner* also predicts which high level actions are viable given the current world state. In many locations in the scenes, certain navigation destinations are unreachable or there are no objects to interact with. Predicting possible actions at each timestep allows gradients to propagate through all actions rather than just the chosen action. This leads to higher accuracies and faster convergence (see section 6.5.2 for more details).

6.4.3 Low level controllers

Navigator: The *Navigator* is invoked by the *Planner* which also provides it with the relative coordinates of the target location. Given a destination specified by the *Planner* and the current estimate of the room's occupancy grid, the *Navigator* runs A* search to find the shortest path to the goal. As the *Navigator* moves through the environment, it uses the esGRU to produce a local (5x5) occupancy grid given the current visual observation. This updates the global occupancy estimate, and prompts a new shortest-path computation. This is a fully supervised problem and can be trained with the standard sigmoid-cross-entropy. The *Navigator* also invokes the *Scanner* to obtain a wide angle view of the environment. Given that the requested destination may be outside the bounds of the room or otherwise impossible (e.g. at a wall or other obstacle), the *Navigator*'s network also predicts a termination signal, and returns control to the *Planner* when the prediction

passes a certain threshold.

Scanner: The *Scanner* is a simple controller which captures images by rotating the camera up, down, left, or right while maintaining the agent's current location. The *Scanner* calls the *Detector* on every new image.

Detector: Object detection is a critical component of HIMN given that all questions in IQUAD V1 involve one or more objects in the room. We use YOLOv3 [176] fine-tuned on the AI2-THOR training scenes as an object detector. We estimate the depth of an object using the FRCN depth estimation network [129] and project the probabilities of the detected objects onto the ground plane. Both of these networks operate at real-time speeds, which is necessary since they are invoked on every new image. The detection probabilities are incorporated into the spatial memory using a moving average update rule. We also perform experiments where we substitute the trained detector and depth estimator with oracle detections. Detections provided by the environment still requires the network to learn affordances. For instance, the network must learn that *microwaves can be opened*, *apples can be in fridges*, etc.

Manipulator: The *Manipulator* is invoked by the *Planner* to manipulate the current state of an object. For example, opening and closing the microwave. This leads to a change in the visual appearance of the scene. If the object is too far away or out of view, the action will fail.

Answerer: The *Answerer* is invoked by the *Planner* to answer the question. It uses the current image, the full spatial memory, and the question embedding vector to predict answer probabilities a_i for each possible answer to the question. The question vector is tiled to create a tensor with the same width and height as the spatial memory. These are depthwise concatenated with the spatial memory and passed through 4 convolution and max pool layers followed by a sum over the spatial layers. This output vector is fed through two fully connected layers and a softmax over all possible answers. After the *Answerer* is invoked, the episode ends, whether the answer was correct or not.

6.4.4 Training

The full system is trained jointly. However, since the individual tasks of the controllers are mostly independent, we are able to pretrain them separately. Our initial analysis showed that this leads to faster convergence and better accuracy than training end-to-end from scratch. We outline our training procedures below.

Planner: To pretrain the *Planner*, we assume a perfect *Navigator* and *Detector* by using the ground truth shortest path for the *Navigator* and the ground truth object information for the *Detector*.

Navigator: We pretrain the *Navigator* by using pairs of random starting points and goal locations.

Answerer: The *Answerer* is pretrained by using ground-truth partial semantic maps which contain enough information to answer the current question correctly.

Detector: The *Detector* is pretrained by fine-tuning YOLOv3 [176] on the AI2-THOR training scenes. It is trained to identify small object instances which may repeat in multiple scenes (apples, forks, etc.) as well as large object instances which are unique to each scene (e.g. each fridge model will only exist in one scene).

Scanner and Manipulator: There are no trainable parameters for these controllers in our current setup. Their behavior is predefined by the AI2-THOR environment.

Joint Training After all trainable controllers are pretrained, we update the model end-to-end.

6.5 Experiments

We evaluate HIMN on the IQUAD V1 dataset, using Top-1 question answering accuracy. An initial baseline of Most likely answer per Question-Type (MLA) shows that the dataset is exactly balanced. Additionally, because we construct the data such that each generated question has a scene configuration for each answer possibility, there is no possible language bias in the dataset. The learned baseline that we compare to (A3C), is based on a common architecture for reinforcement learning used in past works including for visual semantic planning (Chapter 4) and task oriented language grounding [27, 87]. We extend this for the purpose of question answering. Since HIMN has access to object detections provided by either the environment or YOLO [176], we also provide

Model	Existence		Counting		Spatial Relationships	
	Accuracy	Length	Accuracy	Length	Accuracy	Length
Most Likely Answer Per Q-type (MLA)	50	-	25	-	50	-
A3C with ground truth (GT) detections	48.59	332.41	24.53	998.32	49.84	578.71
HIMN with YOLO [176] detections	**68.47**	318.33	**30.43**	926.11	**58.67**	516.23
Human (small sample)	90	58.40	80	81.90	90	43.00

Table 6.2: **Method Performance.** This tables compares the test accuracy and episode lengths of question answering across different models and question types.

detections to the baseline. For the baseline model, at each time-step, the raw RGB image observed by the agent is concatenated depth wise with object detections (one channel per object class). This tensor is passed through convolutional layers and fed into a GRU. The question is passed through an LSTM. The output of the LSTM and GRU are concatenated, and passed through two fully connected layers to produce probabilities π_i for each action and a value v. The output of the first fully connected layer is also passed to an answering module that consists of two more fully connected layers with a softmax on the space of all possible answers. The model is trained using the A3C algorithm for action probabilities and a supervised loss on the answers. We also provide a human baseline of random questions on each question type.

Table 6.2 shows the test accuracies and the average episode lengths for the proposed HIMN model and baselines for each question type. HIMN significantly outperforms the baselines on all question types, both with YOLO object detections as well as ground truth object detections. Surprisingly, the A3C baseline performs slightly worse than random chance even with ground truth detections. We conjecture that this is because there is no explicit signal for when to answer, and no persistence of object detections. The A3C model is not able to associate object detections with the question, and thus has no reason to remember detections for long periods of time. Because HIMN does not overwrite its entire spatial memory at each timestep, object detections persist for much longer, and the *Answerer* can better learn the associations between the questions and the objects.

Model	Existence		Counting		Spatial Relationships	
	Accuracy	Length	Accuracy	Length	Accuracy	Length
HIMN with YOLO [176] detections	68.47	318.33	30.43	926.11	58.67	516.23
HIMN with GT detection	86.56	679.70	35.31	604.79	70.94	311.03
HIMN with GT detection and oracle navigator (HIMN-GT)	**88.60**	618.63	**48.44**	871.12	**72.50**	475.55
HIMN-GT Question not given to planner	50.00	150.60	24.50	293.33	50.25	118.09
HIMN-GT No loss on invalid actions	49.84	659.28	24.84	911.46	50.00	613.50

Table 6.3: **Ablation experiments on the HIMN model**.

HIMN further benefits from a spatial memory in counting and spatial relationship questions because these require much more spatial reasoning than existence questions. Additionally, because A3C does not learn to answer questions, it also does not learn to efficiently explore the environments, as most of the reward comes from answering questions correctly. HIMN, on the other hand, traverses much more of the environment, only answering when it is confident that it has sufficiently explored the room. This indicates that HIMN (which uses an explicit semantic spatial memory with egocentric updates) is more effective than A3C (which uses a standard fully-connected GRU) at (a) Estimating when the environment has been sufficiently explored, given the question (b) Keeping track of past observations for much longer durations, which is important in determining answers for questions that require a thorough search of the environment, and (c) Keeping track of multiple object instances in the scene, which may be observed several time steps apart (which is crucial for answering counting questions).

6.5.1 Ablation Analysis

We perform four ablative experiments on our network structure and inputs, shown in table 6.3. First, we use the ground truth object detections and depth instead of YOLO [176] and FRCN depth [129]. This adds a dramatic improvement to our model owing primarily to the fact that without detection mistakes, the *Answerer* can be more accurate and confident. Secondly, we substitute our learned navigation controller with an oracle *Navigator* that takes the shortest path in the en-

Model	Percentage of invalid actions		
	Existence	Counting	Spatial Relationships
A3C with GT detections	32.75	34.55	32.63
HIMN No loss on invalid actions	56.27	53.43	51.93
HIMN with YOLO [176] detections	6.07	5.95	6.68
HIMN with GT detections	6.49	5.71	5.66
HIMN-GT	**1.79**	**2.02**	**1.27**
Human	5.99	6.47	3.49

Table 6.4: **Percentage of invalid actions across different models.** Lower is better.

vironment. When the optimal *Navigator* is provided, HIMN further improves. This is because the *Planner* can more accurately direct the agent through the environment, allowing it to be more efficient and more thorough at exploring the environment. It also takes fewer invalid actions (as seen in table 6.4), indicating that it is less likely to get stuck in parts of the room. In our third ablative experiment, we remove the question vector from the input of the *Planner*, only providing it to the *Answerer*, which results in random performance. This shows that the *Planner* utilizes the question to direct the agent towards different parts of the room to gather information required to answer the question. For instance any question about an object in the fridge requires the planner to know the fridge needs to be opened. If the planner is not told the question, it has no reason to open the fridge, and instead will likely choose to continue exploring the room as exploration often gives more reward than opening an object. Also, some questions can be answered soon after an object is observed (*e.g.*Existence), whereas others require longer explorations (*e.g.*Counting). Having access to the questions can clearly help the *Planner* in these scenarios. Tables 6.3 shows that HIMN does in fact explore the environment for longer durations for Counting questions than for Existence and Spatial Relationship questions. In our final ablation experiment, we remove the loss on invalid actions. If we do not apply any loss on these actions and only propagate gradients through the chosen action, the agent suffers from the difficulty of exploring a large action space and again performs at random chance.

Model	Existence		Counting		Spatial Relationships	
	Seen	Unseen	Seen	Unseen	Seen	Unseen
HIMN with YOLO [176] detections	73.68	68.47	36.26	30.43	60.71	58.67
HIMN with GT detections	94.00	86.56	42.38	35.31	73.38	70.94

Table 6.5: **Accuracy of question answering across different models on Seen and Unseen environments.**

6.5.2 Invalid Actions

Table 6.4 shows the percentage of invalid actions taken by the different methods. Failed actions are due to navigation failures (failing to see an obstacle) or interaction failures (trying to interact with something too far away or otherwise impossible). There is a clear benefit to including a loss on the invalid actions both in terms of QA accuracy, as can be seen in table 6.3, as well as in terms of percentage of invalid actions performed, shown in table 6.4. All models in table 6.4 are penalized for every invalid action they attempt, but this only provides feedback on a single action at every timestep. With the addition of a supervised loss on all possible actions, the percentage of invalid actions performed is nearly an order of magnitude lower. By directly training our agent to recognize affordances (valid actions), we are able to mitigate the difficulties posed by a large action space, allowing the *Planner* to learn much more quickly. The validity loss also serves as an auxiliary task which has been shown to aid the convergence of RL algorithms [99]. By replacing the learned *Navigator* with an oracle, we observe that the majority of failed actions are due to navigation failures. We believe that with a smaller step size, we would further reduce the navigation errors at the expense of longer trajectories.

6.5.3 Generalization in Unseen Environments

One benefit of HIMN over other RL architectures is that encoding semantic information into a spatial map should generalize well in both seen and unseen environments. Thus, in table 6.5, we compare HIMN's performance on seen and unseen environments. Unseen environments tests the agent with questions that occur in 5 never-before-seen rooms, whereas the seen environments

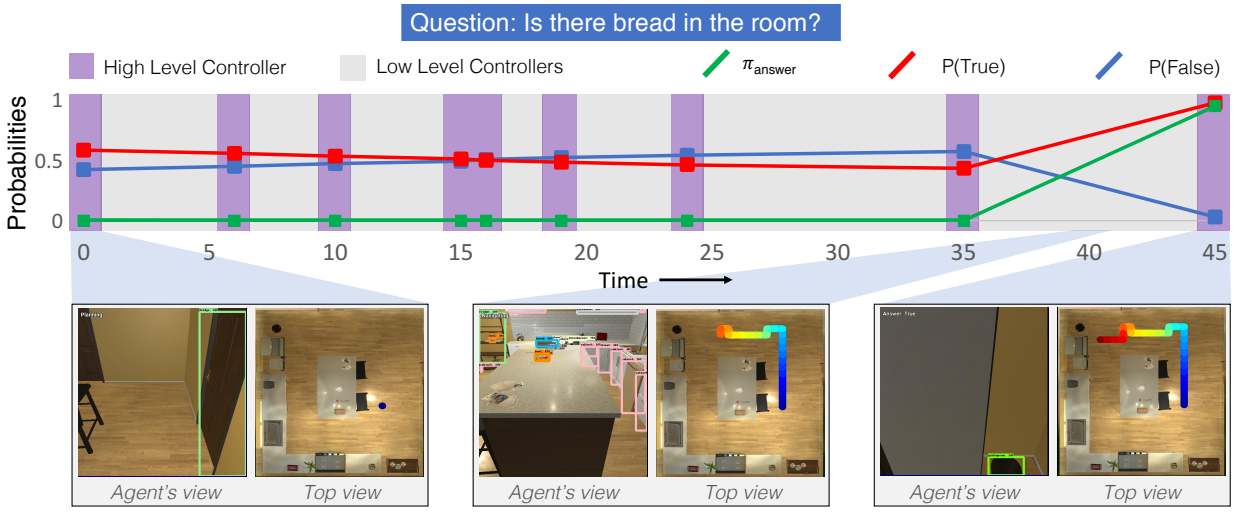


Figure 6.5: **Sample Trajectory.** This shows the trajectory for answering the existence question: *Is there bread in the room?* The purple sections indicate a *Planner* step, and the gray sections indicate a lower level controller such as the *Navigator* is controlling the agent. For a more detailed explanation, refer to section 6.5.4.

use the 25 training rooms but never-before-seen object placements and corresponding questions. Despite our relatively small number of training rooms, table 6.5 shows that our method only loses up to a few percentage points of accuracy when tested on unseen environments. This contrasts with many other end-to-end RL methods which learn deep features that tend to limit their applicability outside a known domain [249]. Note that all experiments in previous sections were only performed on unseen environments.

6.5.4 Qualitative Results

Figure 6.5 shows a sample run of HIMN for the question "Is there bread in the room." Initially, $P(True)$ and $P(False)$ both start near 50%. The *Planner* begins searching the room by navigating around the kitchen table. During the initial exploration phase, bread is not detected, and $P(False)$ slowly increases. At timestep 39, the *Navigator* invokes the *Detector*, which sees the bread and

incorporates it into the semantic spatial map. However, the *Navigator* does not return control to the *Planner*, as it has not yet reached the desired destination. Upon returning at timestep 45, the *Planner* reads the spatial map, sees the bread, and immediately decides it can answer the question. Thus π_{answer} and $P(True)$ both increase to nearly 100%. For more examples, please see our video at `https://youtu.be/pXd3C-1jr98`.

6.5.5 Limitations

Although HIMN performs quite well, it still has several obvious limitations. As can be seen in the human experiments in table 6.2, HIMN is still fairly inefficient at exploring the environment. In the following chapter, we will investigate more traditional planning algorithms to reduce the time spent on exploration. Secondly, our templated language model is quite simple, and would not extend to arbitrary questions. In follow-up work, Shridhar *et al*. [194] extend this framework to incorporate natural language provided by humans for complex task planning.[2]

6.6 Conclusion

In this chapter, we pose a new problem of Interactive Question Answering for several question types in interactive environments. We propose the Hierarchical Interactive Memory Network, consisting of a factorized set of controllers, allowing the system to learn from long trajectories. We also introduce the Egocentric Spatial GRU for updating spatial memory maps. The effectiveness of our proposed model is demonstrated on a new benchmark dataset built upon a high-quality simulation environment for this task. This dataset still presents several challenges to our model and baselines and warrants future research.

[2]The author of this thesis is a co-author on [194]

Chapter 7

WHAT SHOULD I DO NOW? SYMBOLIC PLANNING AND REINFORCEMENT LEARNING IN THE VISUAL WORLD

In the previous chapter we outlined a method for solving interactive question answering (IQA) tasks hierarchically. In this chapter, we augment this hierarchy with a symbolic planner to more directly and effectively solve some of the long-term planning constraints present in IQA. Additionally, this enables us to solve more complex semantic tasks that were initially proposed in Chapter 4 across multiple environments (with the same weights) and including the navigation component which was omitted in Chapter 4.

7.1 Introduction

An important goal in developing systems with visual understanding is to create agents that interact intelligently with the visual world. Teaching these agents about the world requires several steps. An agent must initially learn simple behaviors such as navigation and object affordances. Then, it should combine several learned skills together to accomplish longer term goals. As the task complexity increases, the agent must plan farther in the future. Yet when training end-to-end neural networks, agents must discover these skills without any explicit supervision. In practice, this often results in the agents making obvious mistakes and learning potentially suboptimal policies. With even a small amount of planning, many of these mistakes can be easily avoided. This contrast is illustrated by Figure 7.1.

Recently, researchers have predominantly trained interactive agents using deep learning techniques on raw visual data. Neural Networks learn to extract meaningful features from large sources of raw data. Yet as task complexity increases, and long-term planning is required, these systems often have difficulty learning reliable policies. Conversely, many robotics systems still favor plan-

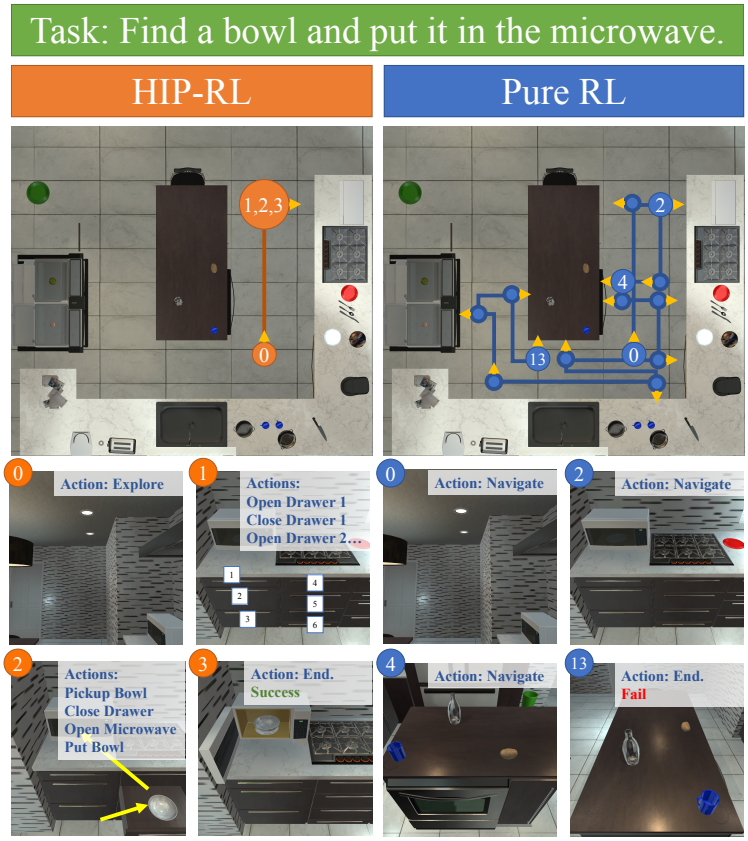


Figure 7.1: **Sample Visual Semantic Planning task execution for "Put a bowl in the microwave."** The Pure RL system wanders around the room, does not open any drawers, and ends the episode after many steps, failing the task. In contrast, HIP-RL methodically accomplishes the task. At hierarchical step $t = 0$, HIP-RL explores the room. At $t = 1$, the Meta-Controller invokes the Planner which creates a plan to check all the drawers. The Object Detector also sees the microwave and saves its location. At $t = 2$ a bowl is found, and the Planner updates its plan. At $t = 3$ the agent puts the bowl in the microwave.

ning and search techniques such as RRT* [108]. When sensor data is clean, planning algorithms offer better generalization and may provide optimality guarantees which are beneficial for ensuring safety in dangerous environments. Yet most planning algorithms operate on (nearly) perfect high-level states rather than raw sensor input. This is especially true for task planning algorithms which use binary values as state representations (e.g. the apple *is* in the fridge). As such, deep learning and task planning have complementary benefits and drawbacks: deep learning can extract information from raw (pixel) data but has difficulty in long-term planning, and task planning re-

quires preprocessed data but can use the high-level information to construct long-horizon plans. To benefit from both, we propose Hierarchical Planning and Reinforcement Learning (HIP-RL), a method for merging these techniques. Others have come to similar conclusions [71, 240] in simple RL setups. We build on these ideas, extending the concept to complex visual domains.

HIP-RL has several advantages over both planning and Deep RL techniques.

1. By using a symbolic planner, HIP-RL avoids obviously incorrect or inefficient actions, increasing the accuracy and speed over comparable pure RL systems.

2. By using CNNs, HIP-RL learns from raw pixels which may otherwise hinder a symbolic planner.

3. HIP-RL trains using an order of magnitude fewer training episodes than comparable Deep RL algorithms.

4. HIP-RL can smoothly trade off hand-defined rules for execution time without sacrificing accuracy.

One potential drawback of a planning-based approach is the need for external knowledge to be hand-coded into the system. However, adding this supervision can be less expensive than data-driven approaches and can generalize across datasets and across tasks. Yet if desired, HIP-RL can trade predefined rules for increased execution time if, for example, certain knowledge is more probabilistic in nature.

To demonstrate the efficacy of our algorithm, we apply HIP-RL to a variety of tasks. First, we apply HIP-RL to the task of Interactive Question Answering (IQA), an extension of Visual Question Answering where an agent navigates in and interacts with an unknown environment in order to answer questions. We apply HIP-RL to IQUAD V1 (Chapter 6) and EQA V1 [38] and achieve state-of-the-art results on both datasets. We additionally show that HIP-RL is able to perform complex Visual Semantic Planning (VSP) tasks such as `Put a bowl in the microwave`

including navigation which was omitted in Chapter 4, outperforming learning-only and planning-only baselines. In summary, we find that using symbolic planning and visual learning together results in higher accuracy, more efficient exploration, and faster training than other methods.[1]

7.2 Related Work

Much of the related work from Chapter 6 is still relevant here, specifically Hierarchical Reinforcement Learning (6.2.2), Visual Planning (6.2.4), and Deep RL in Virtual Visual Environments (6.2.5).

7.2.1 Planning and Learning

Although both learning and planning have significant amounts of prior work, there have been relatively few attempts at merging the two – especially for complex visual problems. Yet when planning and learning are combined, the results are often greater than either method alone. [71] proposes using a symbolic planner to examine future world states and inform a sparse reward function. Their PLANQ system outperforms a standard Q-learner, but their work only explores simple grid-world problems. [240] outlines PEORL, a method for using a symbolic planner to discover subtasks or "options" for a hierarchical RL learner to execute. They outperform other HRL agents which do not use a planner as well as flat RL agents, but again only focus on fully-observable grid-world problems. AlphaZero combines Deep RL for board state evaluation and Monte Carlo Tree Search for planning and finding high-value future board positions [196]. However these techniques are not feasible in a partially observed visually dynamic environment. Planning in stochastic environments is often solved using Partially Observable Markov Decision Processes. Although they are frequently used to great success [9, 24], POMDPs assume a known noise and transition model, which is not applicable to deep feature extraction. To overcome these limitations, we use the Metric-FF Planner [93] to plan high-level trajectories and deep networks to solve the low-level visually complex tasks.

---

[1] Video available at `https://youtu.be/0TtWJ_0mPfI`

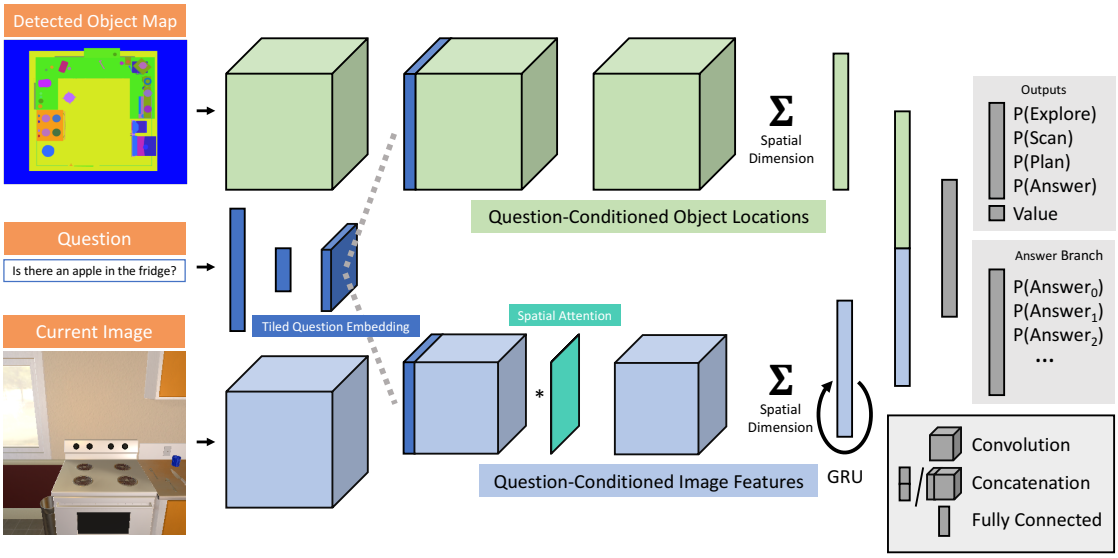


Figure 7.2: **Overview of the network architecture for the Meta-Controller and question-answerer used for IQA.** *For VSP the question is replaced with the task and the answer branch is omitted, but all else is unchanged.*

7.3 Hierarchical Planning and Reinforcement Learning (HIP-RL)

In order to accomplish complex tasks, a system must be able to plan long action trajectories. To operate in a visual world, a system must learn to understand a dynamic visual environment. Thus, to learn to plan in a visual environment, we combine Deep RL with Symbolic Planning. HIP-RL consists of a hierarchical Meta-Controller, several sub-controllers, and a shared memory. In this section we outline the individual components of HIP-RL as well as how they work together to solve complex learning and planning tasks.

7.3.1 HIP-RL Algorithm

The HIP-RL algorithm flow is described in Algorithm 2 with an example execution is shown in Figure 7.1. In this example, the Meta-Controller M selects the sub-controller $C_{explore}$ which navigates to a new location (the inner loop). Then the Meta-Controller calls C_{plan} which makes and executes a plan to put the bowl in the microwave. Finally, the Meta-Controller calls C_{stop} to end the episode. We now describe each component of HIP-RL in more detail.

Algorithm 2 HIP-RL Procedure

Require: Environment E, Task $task$, Meta-Controller M, Sub-Controllers $C_{stop}, C_{plan}, C_{scan}, C_{explore}$, Perception system V

$stop \leftarrow False$
$mem \leftarrow \emptyset$
while not $stop$ **do**
  $C_i, subtask_i \leftarrow M(task, mem)$
  if $C_i = C_{stop}$ **then**
    $stop \leftarrow True$
  else
    $action \leftarrow C_i(subtask_i, mem)$
    while $action \neq done$ **do**
      $observation \leftarrow E(action)$
      $mem \leftarrow mem \cup V(observation)$
      $action \leftarrow C_i(subtask_i, mem)$

7.3.2 Memory Component

Crucial to our method is a memory component accessible to the Meta-Controller as well as the sub-controllers. The memory contains all perceptual and interactive information such as navigable locations, object positions, and which areas have been visited. The learning-based controllers read the memory as a series of top-down spatial maps. For the symbolic planner, the spatial map is discretized and thresholded to construct logical and numeric statements such as "an apple is in position (3,4)" or "the fridge is open." In this chapter we make the simplifying assumption of perfect localization, however we do not assume any prior knowledge of the environment's map. The memory is filled by a free-space mapper (see section 7.3.7) and an object detector (see section 7.3.8) run on the visual observations at every timestep.

7.3.3 Hierarchical Meta-Controller

Given a high-level task, the Meta-Controller encodes the task, reads the memory, and decides which of the sub-controllers to invoke. The Meta-Controller learns to trade off between the length of an episode and the confidence of ending an episode. We train this behavior using the A3C rein-

forcement learning algorithm [156] where the "action space" is the choice of which sub-controller to call. The Meta-Controller is trained using a sparse reward for answering a question correctly, a large penalty for answering incorrectly, and a small constant time penalty for each hierarchical step. This prevents the agent from falling into the local minimum of failing early to avoid the time penalty as failure has an even larger penalty. At both training and test time we sample the sub-controller from the output distribution of the Meta-Controller.

We visualize the Meta-Controller's architecture in the context of Interactive Question Answering in Figure 7.2. The network takes as input the detected object map, the question, and the current image. The question embedding is concatenated with the spatial map and visual features to condition the features on the question. Both the detection map features and the image features are transformed into a single vector by summing over the spatial dimension. The image features additionally use a GRU [33] to add temporal context. The map and image features are concatenated and fed into a fully-connected layer. The network outputs a probability distribution over the actions, a value estimate for the state, and a distribution over the answers for the question. For VSP, the "Question" input is replaced with a task like `Put a bowl in the microwave`, and there is no answer branch in the output, but all else remains the same.

7.3.4 Planner

Given a subtask, the Planner creates and executes a sequence of **actions** to accomplish a specific **goal**. The Planner uses a Planning Domain Definition Language (PDDL) [149] solver which returns a set of actions or that the goal is impossible based on the current observations. PDDL specifies states using boolean and numeric "fluents" (an apple is in the fridge, location A and B are 10 steps apart). **Actions** consist of *preconditions* necessary for the action to be possible, and *effects* which modify the values of the fluents caused by executing that action. For example, the `Open` action takes the preconditions that an object must be openable but closed and that the agent must be near the object, and has the effect of setting the object's `closed` property to `False`. For IQA, the Planner's actions are "Navigate to a position," "Open an object," and "Close an object." VSP additionally adds "Pick up an object" and "Put down an object." **Goals** specify a set of cri-

teria necessary for completing some task. Given a small set of actions such as opening/closing and picking up/putting down objects, the agent is able to accomplish many diverse tasks. Even complex tasks which take hundreds of individual actions or have multiple constraints like `Put all the mugs into the sink` or `Take the apple from the table and put it in the fridge` can be encoded. When using the PDDL Solver's output, the Planner takes the next action, updates the memory given the new observation, and replans in an open-loop fashion. This allows the Planner to recover from incorrect initial detections or false negatives which may be corrected over time.

An example Planner sequence for the question `Is there a bowl in a drawer?` is as follows. The Meta-Controller invokes the Planner with the goal `All drawers have been checked or a bowl is in a drawer`. The memory contains three drawers and the microwave. The Planner outputs a plan to check each drawer. The first drawer is empty, so the plan continues. In the second drawer, the bowl is found, and the Planner returns control to the Meta-Controller.

Grouping the multi-step output from the Planner into a single decision made by the Meta-Controller gives HIP-RL several advantages over pure RL solutions. Primarily, this significantly reduces the number of decisions made by the model in a single episode, simplifying credit assignment resulting in more stable and faster convergence. Furthermore, because the Planner guarantees that the goal will be reached (or ruled impossible), HIP-RL can be more thorough and efficient in its exploration. In the examples above, if the agent had only checked a single drawer and moved on, it would incorrectly assume the answer was "no."

In this chapter we use the Metric-FF PDDL solver [93]. Metric-FF uses hill-climbing on relaxed plans (plans in which contradictions are ignored) as a heuristic to estimate the distance to the goal. For numeric values, the relaxation takes the form of ignoring any non-monotonically increasing effects that an action may have. Finding the absolute shortest solution to PDDL problems is NP-Complete, but in practice, Metric-FF usually returns nearly optimal plans in around 100 milliseconds. We include a sample PDDL state and action domain in Appendix F.

Human Knowledge: While the symbolic planner requires certain properties be defined manually, this is no more expensive than labeling an equivalent training dataset. Each rule (e.g. fridges can be opened) only needs to be defined once, whereas in a data-driven approach each rule would require dozens (or more) of examples in order to be learnable. In simulated datasets, this can be further automated by reading the properties from the simulator itself. By using a symbolic planner to simplify the learning, we avoid the need for other human intervention like behavioral cloning or reward shaping. Additionally, hand-coded rules transfer to new environments and new problems more naturally than end-to-end network weights which often entangle the learned policy with the environment understanding. In our work, we use the same planner rules for three different problem setups, only changing the goal definition. This is akin to defining a reward function for a new RL problem. Finally, we can trade off defining rules with execution time or labeled instances, essentially relying less on the planning and more on the learning (see section 7.5.5).

7.3.5 Explorer

The Meta-Controller uses the Explorer to gather information about an environment. In this chapter, the Explorer is not learned; instead, it uses the memory's occupancy map and picks the location/orientation which maximizes new views while minimizing the distance from the agent's current location. It then invokes the Navigator with this chosen location.

7.3.6 Scanner

The Scanner issues a predefined sequence of commands to the environment to obtain a 360° view of its surroundings. Specifically, it performs three full rotations at +30°, 0°, and -30° with the horizon, stopping every 90° to run object detection. It is often useful to call the Scanner after the Explorer, but we leave this to the Meta-Controller to learn.

7.3.7 Navigation

We use the navigator from Section 6.4.3 as it shows reliable performance for going to locations specified in a relative coordinate frame.

7.3.8 Object Detection

The object detector predicts object masks (using Mask-RCNN [81]) as well as pixelwise depths (using FRCN depth [129]). These predictions are projected into a global coordinate frame and merged with prior detections. We merge detections by joining the bounding cube around two detections if their 3D intersection is above a certain threshold.

7.3.9 Training Procedure

We train the networks in several stages in order to simplify and stabilize the learning procedure.

1. We train the esGRU navigator to navigate between random points in the environment using supervised free-space maps. We additionally train the MaskRCNN object detector and FRCN depth estimator using random views from the training environments.

2. We pretrain the Meta-Controller using ground truth object detection and ignoring navigation by allowing the agent to teleport to its requested destination.

3. We use the learned navigator, detector, and depth estimator, and optimize the Meta-Controller end-to-end, allowing it to directly learn to handle the noise of the other components.

7.4 Tasks

We focus on two tasks: Interactive Question Answering (IQA) and Visual Semantic Planning (VSP). Both tasks require complex visual and spatial reasoning as well as multi-step planning. To stay consistent with prior work, we use only RGB input. This makes the task of navigation and mapping significantly harder than if depth were also provided.

7.4.1 Interactive Question Answering (IQA)

We evaluate our agent on IQUAD V1 (see section 6.3.2 for more details) and EQA V1 [38]. IQUAD V1 provides 75,600 training questions in 25 training rooms, and 1920 test questions in 5 rooms not available during training time (unseen). EQA V1 consists of 7129 training questions in 648 train houses and 853 validation questions in 68 unseen houses. [39] shows results with the agent starting 10, 30, or 50 steps away from the goal, yet only their results for 10 steps outperform the language baselines of [213]. As such, we limit our experiments to starting 10 steps away. Both datasets use templated language.

7.4.2 Visual Semantic Planning (VSP)

We also use the AI2-THOR environment [120] for semantic planning tasks such as `Put the apple in the sink` where the agent must first find a specific object somewhere in the scene and then move it somewhere else. Each task specifies one of 13 small objects (e.g. mug, fork, potato), and one of 6 target locations (e.g. fridge, cabinet, sink). In contrast with Chapter 4, we train one network to accomplish all tasks rather than one for each object-location pair. Additionally, we include navigation as a subtask of planning which Chapter 4 omits. Compared with IQA, VSP contains only a single task type (find and move a single object) but uses a larger action space. Although we only test a single task type, we include more complex task examples in the supplemental video. VSP is not applicable in the EQA environments as the environments do not contain movable objects. We use the same train/test split of environments as in IQUAD V1, and construct 25,200 training tasks, 640 test tasks in unseen rooms. During dataset generation, we verify that each task is possible, yet cannot be completed by the empty plan, e.g. for the task `Put a mug in a cabinet,` there exists at least one mug and at least one cabinet, but no mugs start in cabinets. This data will be made available upon publication.

Method	IQUAD V1 (Section 6.3.2)		EQA V1 [38]		VSP	
	Accuracy	SSPL	Accuracy	SSPL	Success	SPL
Language Only [213]	41.7%	0	48.8%	0	-	-
HIMN (Section 6.4)	52.52%	0.015	-	-	-	-
NMC [39]	-	-	53.58%	-	-	-
Planner Only	56.91%	**0.105**	49.53%	0.002	11.41%	0.059
HIP-RL	**65.99%**	0.086	**58.41%**	**0.007**	**46.01%**	**0.189**
HIMN with GT Detections	64.27%	0.042	-	-	-	-
Planner Only with GT Detections	74.53%	**0.251**	54.15%	**0.025**	43.44%	0.245
HIP-RL with GT Detections	**81.25%**	0.177	**65.28%**	0.017	**73.75%**	**0.362**
HIMN with GT Detections and Teleport	69.85%	0.046	-	-	-	-
Planner Only with GT Detections and Teleport	68.07%	0.091	49.64%	0.004	31.72%	0.268
HIP-RL with GT Detections and Teleport	**83.30%**	**0.325**	**65.52%**	**0.033**	**81.88%**	**0.549**

Table 7.1: **Comparison of accuracy and efficiency on unseen environments.** In IQUAD V1/VSP finding true shortest path for the SPL metric is a traveling salesman problem. Instead, a shortest path estimate for IQUAD V1 and VSP is computed using the Metric-FF planner given the positions of all large objects (fridges, cabinets, etc.) a priori. For EQA V1 the shortest path is found using an oracle with preexisting knowledge of the location of the single target object.

7.5 Experiments

We compare HIP-RL across the datasets outlined in Section 7.4 using existing state-of-the-art methods as baselines, the unimodal baselines from [213], and pure planning baselines. All tests are done in environments the agent has never seen before. On each dataset, we measure the accuracy/success of our method as well as the "efficiency" of each method. [11] proposes the Success weighted by (normalized inverse) Path Length (SPL) which combines accuracy and episode length into a single metric for evaluating embodied agents.

$$SPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{\ell_i}{\max(p_i, \ell_i)} \tag{7.1}$$

where S_i is a success indicator for episode i, p_i is the agent's path length, and ℓ_i is the ground-truth shortest path length. Whereas in pure navigation tasks such as those in [11] the SPL metric

is directly applicable, it does not extend well to question answering; in navigation tasks, it is impossible to be successful without moving, however even a guessing QA agent will succeed sometimes.[2] To address this issue, we propose Shifted SPL (SSPL) which is defined as

$$SSPL = \frac{\mu - b}{1 - b} * SPL \tag{7.2}$$

where μ is the average accuracy of the method and b is the average accuracy of a blind agent (an agent which only receives the question as input). SSPL directly accounts for dataset construction biases by subtracting the accuracy of a learned baseline rather than simply the most common answer or random chance accuracy. We use the "Language Only" baselines presented in [213] as b.

7.5.1 Baselines

On all datasets we include (at least) one pure-learning and one pure-planning baseline. The "Planner Only" baseline uses the same Plan/Act/Observe/Replan loop as HIP-RL but does not include any hierarchical decision making. If at the start of the episode the plan is empty (for example if the agent starts looking at a wall), we rotate the agent until the plan is not empty. The "Planner Only" baseline can be seen as a lower-bound on the number of steps that pure-planning agent could take. While this baseline is simpler than many existing planning-based robotics approaches [32, 105], no existing approaches have been applied to IQA or VSP. We feel it would take significant extra work to include a planning only baseline that was not similar in approach to our work, but recognize that a more finely-tuned system may outperform our baseline. We also use the "Language Only" baselines from [213] which attempt to answer the question without any observations, effectively learning the language bias of the dataset. For VSP, we introduce a "Learning Only" baseline which removes the Planner from HIP-RL and adds reward shaping to encourage certain interactions like looking at and picking up the object of the task. Even after significant training time, this method fails to learn a working policy.

[2] Thomason *et al.* [213] shows large biases exist in EQA V1 [38] and in MatterPort3D [12], but not in IQUAD V1.

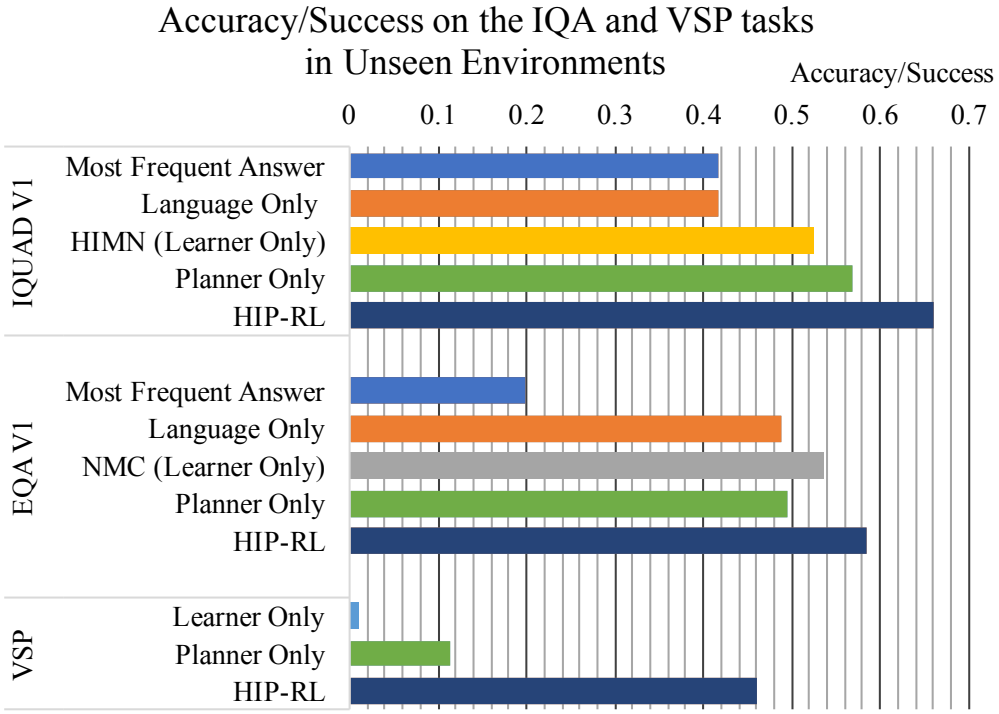


Figure 7.3: **Accuracy of various methods on each of the tasks.** In all cases, HIP-RL achieves state-of-the-art performance. We include "Learner Only" and "Planner Only" results for each experiment to show that combining both their strengths is better than either alone.

7.5.2 Results

We test our system for accuracy on IQUAD V1, EQA V1, and the new VSP dataset, achieving state-of-the-art performance on all tasks. The results are shown in Figure 7.3. In IQUAD V1 and VSP the "Planner Only" baseline outperforms the "Learning Only" baseline. Because the ground-truth trajectories are significantly long and contain many interactions (especially in VSP), pure RL systems struggle to learn robust policies. A fundamental issue with reinforcement learning is that it must "luck into" good solutions randomly before it can improve, which can be very rare in complex multi-step tasks with sparse rewards. Planning simplifies this by directly solving objectives rather than making guesses and observing rewards. Yet planning suffers from myopia in that it assumes perfect and complete global information, leading it to ignore unobserved parts of

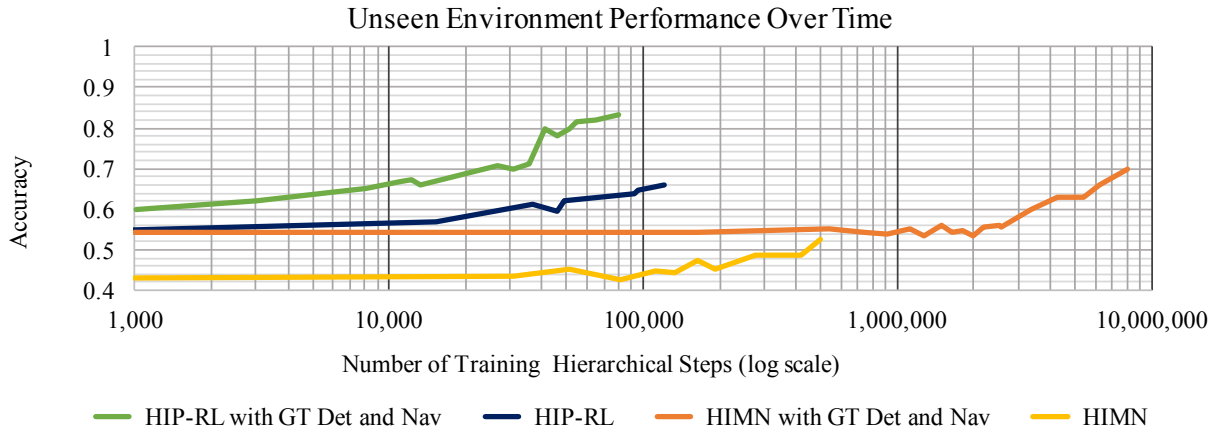


Figure 7.4: **Learning speed of HIP-RL and HIMN with and without ground truth information.** HIMN's Answerer is pretrained on fully observed rooms whereas HIP-RL does not require pretraining.

the environment. In practice, it frequently answers too soon (i.e. before it has observed enough to answer definitively), resulting in trajectories which are shorter than the ground-truth shortest path. By combining both strategies, HIP-RL achieves the exploration tendencies of RL along with the goal-oriented direct problem solving of planning.

7.5.3 Episode Efficiency

Table 7.1 also lists SPL/SSPL scores for each method. Note that the episode lengths used to compute SPL/SSPL include every interaction with the environment (turn left, open, move ahead each count as one action), not just hierarchical actions. While HIP-RL improves over HIMN, there is still a large gap between the shortest path estimate. Some inefficiency due to exploration is unavoidable, but there are also cases where the agent explores even after it could answer. For example, this often occurs in IQUAD V1 on counting questions where the agent is not sure that it has sufficiently checked everywhere where the object could be.

7.5.4 Ablation

We further explore the effect that various sources of inaccuracy have on HIP-RL by substituting the object detector with Ground Truth (GT) information and navigation with a "teleport" command, shown in Table 7.1. Adding GT detections greatly improves our accuracy across the board. Since the tasks are object-centric, if an object is misidentified or not detected, the Answerer/Planner will have difficulty making good decisions. Mask-RCNN produces high-quality results on large datasets, yet in 3D environments, there are frequently only a few object models and textures which are partitioned between the train and test splits. In this setup, Mask-RCNN struggles to detect objects in unseen environments. We provide additional evidence of the performance gap between seen and unseen environments in Appendix F.5. We believe that in scenarios with more diverse training examples, HIP-RL with detection would approach similar performance.

In contrast, using "teleport" does not improve accuracy dramatically. Teleport does not tell the agent where to go, only how to get there; thus good high-level planning still matters for success. Using teleport does nearly double the efficiency. We observe this is frequently from the beginning of episodes when the map is empty and the navigator unknowingly goes down dead ends or takes inefficient paths. Interestingly, using the navigation system instead of GT navigation helps the "Planner Only" method by giving it more varied locations to run the detector.

7.5.5 Removing Prior Information

In some cases, it may be more desirable to learn environment characteristics than to define prior information. HIP-RL can smoothly trade off the cost of enumerating logical priors with the evaluation time of the method. Without placement information, HIP-RL degrades gracefully, retaining the original accuracy while, understandably, executing longer trajectories. Table 7.2 shows the specific case of removing the prior knowledge of where objects can be located for IQUAD V1. The accuracies of HIP-RL with and without this information are very similar indicating that this extra information helps HIP-RL be faster, but is not required for accuracy. One could also build a more complex learning system at the expense of more labeled data.

Method	Uses Object Location Rules	IQUAD V1 Accuracy	IQUAD V1 Episode Length
HIP-RL	✓	65.99%	357.690
HIP-RL		65.57%	476.680
HIP-RL + GT Nav	✓	65.68%	180.702
HIP-RL + GT Nav		67.14%	284.447
HIP-RL + GT Det	✓	81.25%	297.238
HIP-RL + GT Det		80.16%	522.603
HIP-RL + GT Det and Nav	✓	83.30%	132.191
HIP-RL + GT Det and Nav		84.43%	271.966

Table 7.2: **Comparison of Accuracy and Speed with and without priors on where objects can begin.**

7.5.6 Learning Speed

Figure 7.4 compares the convergence speed of HIP-RL on the IQUAD V1 task with the previous state-of-the-art model, HIMN. After only 26,000 hierarchical steps, HIP-RL with ground truth information matches the final performance of HIMN with GT at 8 million hierarchical steps, a 300 fold gain. After 120,000 hierarchical steps, HIP-RL (without ground truth) converges to its maximum performance compared to HIMN which takes 500,000 iterations and achieves significantly worse performance. HIP-RL trains much faster than traditional RL algorithms because the Planner simplifies the learning by being able to return good, thorough trajectories even upon the network's initialization. This correlates well with how humans learn, using our knowledge of the world to shorten the time it takes to solve a new problem. Being thorough early on is especially useful for question answering where an algorithm may get confusing feedback if it answers too soon; for example, for the question `Is there a bowl in the room?` if an agent does not see a bowl and answers "no" but the correct answer is "yes," the network will receive contradictory learning signals. By using a symbolic planner, we ensure more thorough exploration so this case is much less likely to occur, even at the beginning of training.

7.6 Conclusion

In this chapter, we presented HIP-RL, a method for combining the benefits of Deep Visual Reinforcement Learning and Symbolic Planning. We demonstrate its effectiveness at increasing accuracy and efficiency while simultaneously decreasing training time. Though this exact implementation may not be applicable to many other tasks, we believe the high level idea of joining planning techniques and deep reinforcement learning can be useful in a broader array of tasks. In general, we observe that using planning algorithms to assist in performing "good" actions results in improved accuracy, test-time efficiency, and training speed. We are excited about the potential impacts of visual agents and their ability to learn to interact more intelligently with the world.

Chapter 8
DISCUSSION

In this thesis we describe multiple techniques for learning from the environment itself rather than from specifically defined knowledge bases. This shift away from traditional supervised learning allows our models to learn not just what we think it should, but anything that it finds useful. By letting a model directly observe and interact with the environment, we enable a richer, deeper understanding of the world. We presented two potential paradigms for learning in this fashion: learning by watching via direct video understanding, and learning by doing via interacting with visual simulation environments.

8.1 Summary

The most basic way of learning passively about the dynamic visual world is to use unsupervised learning on large collections of video data. Chapter 2 introduces the technique of Video Noise Contrastive Estimation for learning using the similarities of individual frames of a video. By forcing the representations from distant video frames to be similar, we pretrain a network which is capable of understanding complex semantic concepts about the world such as objects, activities, and scenes.

Chapter 3 further develops the idea of learning directly from video frames. For the task of generic object tracking most algorithms train a detector on a single frame and then repeatedly apply the detector frame-by-frame, ignoring any temporal context. Sidestepping this tracking-by-detection meta-algorithm, we opt instead to directly learn from videos using recurrent neural networks to regress where the object of interest moves. This allows our network to quickly adapt its underlying object model without needing to retrain and directly incorporates temporal cues into the prediction.

In the second part of this thesis, we opt for an even more direct approach for learning about the world - learning through interaction. Chapter 4 introduces the high-level task of **Visual Semantic Planning** (without navigation) in which an agent must learn to perform complex long-horizon tasks purely through interacting with the world. We show that while learning these tasks, the agent additionally learns object affordances and can quickly learn to perform new tasks.

Chapter 5 focuses on the low-level task of **Navigation** (which is required for any embodied agent). Using structured gradient flows and auxiliary tasks, we decompose the learning problem into two steps, perception and reasoning. By providing this additional grounding, we learn weights which generalize better to new environments and tasks than typical end-to-end systems.

Chapters 6 and 7 tie the low-level and high-level tasks together hierarchically in order to solve the challenging tasks of **Interactive Question Answering** and **Visual Semantic Planning with Navigation**. Chapter 6 introduces an initial model known as the Hierarchical Interactive Memory Network for solving these very long-horizon tasks like answering the question "How many apples are in this room?" Chapter 7 further refines this method by incorporating a symbolic planner to handle even more complex tasks like "Put all the apples in the sink." We show that incorporating a symbolic planner allows for faster convergence, better accuracy, and shorter episodes.

8.2 Future Directions

Although significant progress has been made towards general intelligence which can directly interact with the world, there is still a large gap in performance between human capabilities and their artificial counterparts. Further research in several directions is likely to yield significantly more intelligent agents.

8.2.1 Language and Vision

Natural language provides additional information and knowledge about the world which is often omitted from visual data. Incorporating natural language and visual systems together may yield results which outperform those systems trained on only one of the two modalities. Especially in

the interactive domain, language may prove to be a very useful signal for grounding the visual understanding in concepts like visible objects, and relative relationships like "left of." Initial work of combining language and vision for complex embodied tasks such as Shridhar *et al*. [194] pushes in this direction and illustrates that there is significant room for improvement of even our best systems.

8.2.2 From Simulation to the Real World

Significant progress in simulation environments has spawned numerous new and exciting directions. Simulation enables a scale of learning which is simply not possible to perform in the real world. Additionally, simulation provides free, perfect labels, increases reproducibility, and reduces safety concerns. However, simulated agents cannot actually be helpful to real-world humans; we will need agents which are also successful in reality. As simulator graphics and physics improve, the effectiveness of simulated agents at transferring to the real world also increases. Yet there is still a reality gap. Reducing the friction between training in simulation and evaluating in the real world is a necessary and exciting future endeavor. One potential way of accomplishing this would be to pretrain a perception system on unsupervised tasks such as those in Chapter 2, and then learn a policy with these features which operates well in both simulation and in the real world. This would likely result in higher performance in transferring *back* to reality since the visual system would be primarily (or entirely) trained on real-world data without requiring any additional labelling.

8.2.3 Self-Supervised Learning

Unsupervised learning techniques such as Video Noise Contrastive Estimation allow us to take advantage of existing data at massive scales without needing to gather labels. It also extends flexibility to the model since we never tell it exactly what to learn. However, this setup is still passive and limited in this regard. Extending these or similar techniques to an embodied agent which can gather its own data would allow for an even greater understanding of the world. An agent which can actively decide what data to gather as its learning would be better able to tackle new unseen

problems and learn specific area knowledge if required. Techniques such as Curiosity [170] may provide the next step forward for enabling agents to figure out their own strengths and weaknesses and adapt to overcome these knowledge gaps.

8.3 Concluding Remarks

In this thesis we have presented several methods for learning about the visual world. However there is still significant room to improve the visual capabilities of artificial agents. The rapidly developing field of computer vision presents many interesting, challenging, yet ultimately solvable tasks. This is an encouraging time for artificial visual intelligence.

BIBLIOGRAPHY

[1] Unity software. `https://unity3d.com`.

[2] Milestones in visual development, June 2007.

[3] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *Computing Resource Repository (CoRR)*, 2016.

[5] Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C Lawrence Zitnick, Devi Parikh, and Dhruv Batra. Vqa: Visual question answering. *International Journal on Computer Vision (IJCV)*, 2017.

[6] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *Computing Resource Repository (CoRR)*, 2016.

[7] James Little Alireza Shafaei and Mark Schmidt. Play and learn: Using video games to train computer vision models. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *British Machine Vision Conference (BMVC)*, pages 26.1–26.13. BMVA Press, September 2016.

[8] Greg Aloupis, Erik D Demaine, Alan Guo, and Giovanni Viglietta. Classic nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[9] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal on Robotics Research (IJRR)*, 35(14):1760–1778, 2016.

[10] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm. Unsupervised state representation learning in atari. In *Neural Information Processing Symposium (NeurIPS)*, pages 8766–8779, 2019.

[11] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *Computing Resource Repository (CoRR)*, 2018.

[12] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *Computer Vision and Pattern Recogntion (CVPR)*, 2018.

[13] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. *Computer Vision and Pattern Recogntion (CVPR)*, 2016.

[14] Philip Bachman, R. Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *Neural Information Processing Symposium (NeurIPS)*, pages 15509–15519, 2019.

[15] André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *Computing Resource Repository (CoRR)*, 2016.

[16] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013.

[17] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Neural Information Processing Symposium (NeurIPS)*, pages 1171–1179, 2015.

[18] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *Computing Resource Repository (CoRR)*, 2019.

[19] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision (ECCV)*, pages 850–865. Springer, 2016.

[20] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 1989.

[21] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 1991.

[22] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Computer Vision and Pattern Recogntion (CVPR)*, 20117.

[23] Kai Briechle and Uwe D. Hanebeck. Template matching using fast normalized cross correlation. In *SPIE Defense + Commercial Sensing*, 2001.

[24] Panpan Cai, Yuanfu Luo, David Hsu, and Wee Sun Lee. Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *Robotics Science and Systems (RSS)*, 11, 2018.

[25] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 6299–6308, 2017.

[26] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *International Conference on 3D Vision (3DV)*, 2017.

[27] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *Computing Resource Repository (CoRR)*, 2017.

[28] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research (JMLR)*, 11(Mar):1109–1135, 2010.

[29] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015.

[30] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *Computing Resource Repository (CoRR)*, 2020.

[31] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning, 2020.

[32] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In *International Conference on Robotics and Automation (ICRA)*, pages 447–454. IEEE, 2016.

[33] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[34] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference (BMVC)*. BMVA Press, 2014.

[35] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg. Discriminative scale space tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(8):1561–1575, 2017.

[36] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Convolutional features for correlation filter based visual tracking. In *International Conference on Computer Vision (ICCV) Workshops*, pages 58–66, 2015.

[37] Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *European Conference on Computer Vision (ECCV)*, pages 472–488. Springer, 2016.

[38] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Computer Vision and Pattern Recogntion (CVPR)*, volume 5, page 6, 2018.

[39] Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural Modular Control for Embodied Question Answering. In *Conference on Robot Learning (CoRL)*, 2018.

[40] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[41] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *International Conference on Computer Vision (ICCV)*, 2003.

[42] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 1993.

[43] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference in Machine Learning (ICML)*, pages 465–472, 2011.

[44] Erik D Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is hard, even to approximate. In *International Computing and Combinatorics Conference*, pages 351–363. Springer, 2003.

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 248–255. Ieee, 2009.

[46] Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to perform physics experiments via deep reinforcement learning. *Computing Resource Repository (CoRR)*, 2016.

[47] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2019.

[48] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.*, 2000.

[49] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *International Workshop on Safety, Security & Rescue Robotics (SSRR)*, 2009.

[50] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[51] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference in Learning Representations (ICLR)*, 2017.

[52] Samira Ebrahimi Kahou, Vincent Michalski, Roland Memisevic, Christopher Pal, and Pascal Vincent. Ratm: Recurrent attentive tracking model. In *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 10–19, 2017.

[53] Mario Edoardo Maresca and Alfredo Petrosino. The matrioska tracking algorithm on ltdt2014 dataset. In *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 706–711, 2014.

[54] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. In *International Conference on Intelligent Robotics and Systems (IROS)*, pages 681–687. IEEE, 2015.

[55] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision (ECCV)*, 2014.

[56] Alireza Fathi and James M Rehg. Modeling actions through state changes. In *Computer Vision and Pattern Recogntion (CVPR)*, 2013.

[57] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *International Conference on Computer Vision (ICCV)*, pages 6202–6211, 2019.

[58] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 1971.

[59] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *International Conference on Computer Vision (ICCV)*, pages 4346–4354, 2015.

[60] Quan Gan, Qipeng Guo, Zheng Zhang, and Kyunghyun Cho. First step toward model-free, anonymous object tracking with recurrent neural networks. *Computing Resource Repository (CoRR)*, 2015.

[61] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In David Blei and Francis Bach, editors, *International Conference in Machine Learning (ICML)*, pages 1180–1189. JMLR Workshop and Conference Proceedings, 2015.

[62] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recogntion (CVPR)*, 2012.

[63] Georg Nebehay. Dsst: Discriminative scale space tracker. `https://github.com/gnebehay/DSST`.

[64] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. 2004.

[65] James J Gibson. *The ecological approach to visual perception: classic edition*. 2014.

[66] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recogntion (CVPR)*, 2014.

[67] Clement Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Computer Vision and Pattern Recogntion (CVPR)*, July 2017.

[68] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Symposium (NeurIPS)*, 2014.

[69] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Computer Vision and Pattern Recogntion (CVPR)*, 2017.

[70] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2016.

[71] Matthew Grounds and Daniel Kudenko. Combining reinforcement learning with symbolic planning. In *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pages 75–86. Springer, 2005.

[72] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[73] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Neural Information Processing Symposium (NeurIPS)*, 2014.

[74] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 2616–2625, 2017.

[75] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[76] H Haddad, Maher Khatib, Simon Lacroix, and Raja Chatila. Reactive navigation in outdoor environments using potential fields. In *International Conference on Robotics and Automation (ICRA)*, 1998.

[77] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding. In *International Conference on Computer Vision (ICCV) Workshops*, 2019.

[78] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding real world indoor scenes with synthetic data. In *Computer Vision and Pattern Recogntion (CVPR)*, 2016.

[79] Sam Hare, Stuart Golodetz, Amir Saffari, Vibhav Vineet, Ming-Ming Cheng, Stephen L Hicks, and Philip HS Torr. Struck: Structured output tracking with kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(10):2096–2109, 2016.

[80] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *Computing Resource Repository (CoRR)*, 2019.

[81] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *International Conference on Computer Vision (ICCV)*, pages 2980–2988. IEEE, 2017.

[82] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 770–778, 2016.

[84] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision (ECCV)*, pages 749–765. Springer, 2016.

[85] Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *Computing Resource Repository (CoRR)*, 2019.

[86] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(3):583–596, 2015.

[87] Felix Hill, Karl Moritz Hermann, Phil Blunsom, and Stephen Clark. Understanding grounded language learning agents. *Computing Resource Repository (CoRR)*, 2017.

[88] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[89] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Neural Information Processing Symposium (NeurIPS)*, 2016.

[90] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[91] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International Conference in Machine Learning (ICML)*, 2018.

[92] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *Computing Resource Repository (CoRR)*, 2016.

[93] Jörg Hoffmann. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)*, 20:291–341, 2003.

[94] Zhibin Hong, Zhe Chen, Chaohui Wang, Xue Mei, Danil Prokhorov, and Dacheng Tao. Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 749–758, 2015.

[95] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. *Computing Resource Repository (CoRR)*, 2017.

[96] Haoshuo Huang, Qixing Huang, and Philipp Krahenbuhl. Domain transfer through deep activation matching. In *European Conference on Computer Vision (ECCV)*, 2018.

[97] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, page 1–1, 2019.

[98] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 2462–2470, 2017.

[99] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference in Learning Representations (ICLR)*, 2017.

[100] Andrew Jaegle, Stephen Phillips, and Kostas Daniilidis. Fast, robust, continuous monocular egomotion computation. In *International Conference on Robotics and Automation (ICRA)*, 2016.

[101] Yunseok Jang, Yale Song, Youngjae Yu, Youngjin Kim, and Gunhee Kim. Tgif-qa: Toward spatio-temporal reasoning in visual question answering. *Computing Resource Repository (CoRR)*, 2017.

[102] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Fei fei Li, C. Lawrence Zitnick, and Ross B. Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *Computing Resource Repository (CoRR)*, 2016.

[103] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Fei fei Li, C. Lawrence Zitnick, and Ross B. Girshick. Inferring and executing programs for visual reasoning. *Computing Resource Repository (CoRR)*, 2017.

[104] Jonathan Long*, Evan Shelhamer*, and Trevor Darrel. Fully convolutional networks for semantic segmentation. In *Computer Vision and Pattern Recogntion (CVPR)*, 2015.

[105] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation (ICRA)*, pages 1470–1477. IEEE, 2011.

[106] Samira Ebrahimi Kahou, Adam Atkinson, Vincent Michalski, Ákos Kádár, Adam Trischler, and Yoshua Bengio. Figureqa: An annotated figure dataset for visual reasoning. *Computing Resource Repository (CoRR)*, 2017.

[107] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, et al. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017.

[108] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal on Robotics Research (IJRR)*, 30(7):846–894, 2011.

[109] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *Computing Resource Repository (CoRR)*, 2017.

[110] Aniruddha Kembhavi, Michael Salvato, Eric Kolve, Min Joon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *European Conference on Computer Vision (ECCV)*, 2016.

[111] Aniruddha Kembhavi, Minjoon Seo, Dustin Schwenk, Jonghyun Choi, Ali Farhadi, and Hannaneh Hajishirzi. Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension. In *Computer Vision and Pattern Recogntion (CVPR)*, 2017.

[112] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[113] Dongsung Kim and Ramakant Nevatia. Symbolic navigation with a generic map. *Autonomous Robots*, 1999.

[114] H. Jin Kim, Michael I. Jordan, Shankar Sastry, and Andrew Y. Ng. Autonomous helicopter flight via reinforcement learning. In *Neural Information Processing Symposium (NeurIPS)*, 2004.

[115] Kyung-Min Kim, Min-Oh Heo, Seong-Ho Choi, and Byoung-Tak Zhang. Deepstory: Video story qa by deep embedded memory networks. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2017.

[116] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computing Resource Repository (CoRR)*, 2015.

[117] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal on Robotics Research (IJRR)*, 2013.

[118] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation (ICRA)*, 2004.

[119] Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *The International Journal on Robotics Research (IJRR)*, 2008.

[120] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *Computing Resource Repository (CoRR)*, 2017.

[121] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[122] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei fei Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal on Computer Vision (IJCV)*, 2016.

[123] Matej Kristan et al. The visual object tracking vot2014 challenge results. In *European Conference on Computer Vision (ECCV) Workshops*, 2014.

[124] Matej Kristan et al. The visual object tracking vot2015 challenge results. In *International Conference on Computer Vision (ICCV) Workshops*, pages 1–23, 2015.

[125] Matej Kristan et al. The visual object tracking vot2016 challenge results. In *European Conference on Computer Vision (ECCV) Workshops*, 2016.

[126] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Symposium (NeurIPS)*, 2012.

[127] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Neural Information Processing Symposium (NeurIPS)*, pages 3675–3683, 2016.

[128] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *Computing Resource Repository (CoRR)*, 2016.

[129] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *International Conference on 3D Vision (3DV)*, 2016.

[130] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Computing Resource Repository (CoRR)*, 2016.

[131] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. In *International Conference in Machine Learning (ICML)*, pages 430–438, 2016.

[132] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, 2016.

[133] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal on Robotics Research (IJRR)*, 37(4-5):421–436, 2018.

[134] Wenbin Li, Seyedmajid Azimi, Ales Leonardis, and Mario Fritz. To fall or not to fall: A visual approach to physical stability prediction. *Computing Resource Repository (CoRR)*, 2016.

[135] Yunzhu Li, Jiaming Song, and Stefano Ermon. Inferring the latent structure of human decision-making from raw visual inputs. *Computing Resource Repository (CoRR)*, 2017.

[136] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. Learning the depths of moving people by watching frozen people. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 4521–4530, 2019.

[137] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. 2015.

[138] Chris Linegar, Winston Churchill, and Paul Newman. Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera. In *International Conference on Robotics and Automation (ICRA)*, 2016.

[139] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 5686–5695, 2018.

[140] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Neural Information Processing Symposium (NeurIPS)*, 2016.

[141] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In *International Conference in Machine Learning (ICML)*, 2015.

[142] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research (JMLR)*, 9(Nov):2579–2605, 2008.

[143] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *European Conference on Computer Vision (ECCV)*, pages 181–196, 2018.

[144] Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *Neural Information Processing Symposium (NeurIPS)*, 2014.

[145] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *International Conference on Computer Vision (ICCV)*, 2015.

[146] Mohsen Malmir, Karan Sikka, Deborah Forster, Javier R Movellan, and Garison Cottrell. Deep q-learning for active recognition of germs: Baseline performance on a standardized dataset for active learning. In *British Machine Vision Conference (BMVC)*, 2015.

[147] Mario Edoardo Maresca and Alfredo Petrosino. Clustering local motion estimates for robust and efficient object tracking. In *European Conference on Computer Vision (ECCV)*, pages 244–253. Springer, 2014.

[148] Javier Marin, David Vázquez, David Gerónimo, and Antonio M López. Learning appearance in virtual scenarios for pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR)*, 2010.

[149] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[150] Simon Meister, Junhwa Hur, and Stefan Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[151] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference in Machine Learning (ICML)*, 2005.

[152] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Neural Information Processing Symposium (NeurIPS)*, pages 3111–3119, 2013.

[153] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference in Learning Representations (ICLR)*, 2017.

[154] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. *Computing Resource Repository (CoRR)*, 2019.

[155] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision (ECCV)*, pages 527–544. Springer, 2016.

[156] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference in Machine Learning (ICML)*, pages 1928–1937, 2016.

[157] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[158] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *Computer Vision and Pattern Recogntion (CVPR)*, 2016.

[159] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "What happens if..." learning to predict the effect of forces in images. In *European Conference on Computer Vision (ECCV)*, 2016.

[160] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. In *European Conference on Computer Vision (ECCV)*, 2018.

[161] Jonghwan Mun, Paul Hongsuck Seo, Ilchae Jung, and Bohyung Han. Marioqa: Answering questions by watching gameplay videos. *Computing Resource Repository (CoRR)*, 2016.

[162] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 2015.

[163] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 4293–4302, 2016.

[164] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision (ECCV)*, pages 69–84. Springer, 2016.

[165] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Communicating hierarchical neural controllers for learning zero-shot task generalization. 2016.

[166] Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016.

[167] Giuseppe Oriolo, Marilena Vendittelli, and Giovanni Ulivi. On-line map building and navigation for autonomous mobile robots. In *International Conference on Robotics and Automation (ICRA)*, 1995.

[168] Jeremie Papon and Markus Schoeler. Semantic pose using deep networks trained on synthetic rgb-d. In *International Conference on Computer Vision (ICCV)*, 2015.

[169] Ronald E. Parr and Stuart J. Russell. Reinforcement learning with hierarchies of machines. In *Neural Information Processing Symposium (NeurIPS)*, 1997.

[170] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference in Machine Learning (ICML)*, volume 2017, 2017.

[171] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[172] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision (ECCV)*, 2016.

[173] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *International Conference on Robotics and Automation (ICRA)*, 2016.

[174] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference in Learning Representations (ICLR)*, 2016.

[175] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[176] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 2018.

[177] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision (ECCV)*, pages 102–118. Springer, 2016.

[178] Anna Rohrbach, Marcus Rohrbach, Niket Tandon, and Bernt Schiele. A dataset for movie description. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 3202–3212, 2015.

[179] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Computer Vision and Pattern Recogntion (CVPR)*, 2016.

[180] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635, 2011.

[181] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal on Computer Vision (IJCV)*, 115(3):211–252, 2015.

[182] Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning (CoRL)*, pages 262–270, 2017.

[183] Fereshteh Sadeghi. DIViS: Domain invariant visual servoing for collision-free goal reaching. In *Robotics Science and Systems (RSS)*, 2019.

[184] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. In *Robotics Science and Systems (RSS)*, 2017.

[185] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. Sim2real viewpoint invariant visual servoing by recurrent control. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 4691–4699, 2018.

[186] Parvaneh Saeedi, Peter D Lawrence, and David G Lowe. Vision-based 3-d trajectory tracking for unknown environments. *IEEE transactions on robotics*, 2006.

[187] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. *Computing Resource Repository (CoRR)*, 2019.

[188] Alexander Sax, Bradley Emi, Amir Roshan Zamir, Leonidas J. Guibas, Silvio Savarese, and Jitendra Malik. Mid-level visual representations improve generalization and sample efficiency for learning active tasks. *Computing Resource Repository (CoRR)*, 2018.

[189] Jürgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *IJCNN*, 1990.

[190] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters (RAL)*, 2(2):420–427, 2016.

[191] Tanner Schmidt, Richard A Newcombe, and Dieter Fox. Dart: Dense articulated real-time tracking. In *Robotics: Science and Systems*, 2014.

[192] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *Computing Resource Repository (CoRR)*, 2017.

[193] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Neural Information Processing Symposium (NeurIPS)*, pages 802–810, 2015.

[194] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *Computer Vision and Pattern Recogntion (CVPR)*, 2020.

[195] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[196] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[197] Robert Sim and James J Little. Autonomous vision-based exploration and mapping using hybrid maps and rao-blackwellised particle filters. In *International Conference on Intelligent Robotics and Systems (IROS)*. IEEE, 2006.

[198] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Neural Information Processing Symposium (NeurIPS)*, pages 568–576, 2014.

[199] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(7):1442–1468, 2014.

[200] Andres Solis Montero, Jochen Lang, and Robert Laganiere. Scalable kernel correlation filter with sparse feature integration. In *International Conference on Computer Vision (ICCV) Workshops*, pages 24–31, 2015.

[201] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference in Machine Learning (ICML)*, pages 843–852, 2015.

[202] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart J. Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *International Conference on Robotics and Automation (ICRA)*, 2014.

[203] Siddharth Srivastava, Lorenzo Riano, Stuart Russell, and Pieter Abbeel. Using classical planners for tasks with continuous operators in robotics. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

[204] Lesley Stahl. How does youtube handle the site's misinformation, conspiracy theories and hate?, Dec 2019.

[205] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision (ECCV)*, 2016.

[206] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 1998.

[207] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 1999.

[208] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[209] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *International Conference in Learning Representations (ICLR)*, 2017.

[210] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Neural Information Processing Symposium (NeurIPS)*, 2016.

[211] Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Movieqa: Understanding stories in movies through question-answering. *Computer Vision and Pattern Recogntion (CVPR)*, pages 4631–4640, 2016.

[212] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2017.

[213] Jesse Thomason, Daniel Gordon, and Yonatan Bisk. Shifting the baseline: Single modality performance on visual navigation & qa. In *NAACL-HLT 2019 Short Papers*, 2019.

[214] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *Computing Resource Repository (CoRR)*, 2019.

[215] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robotics and Systems (IROS)*, pages 23–30. IEEE, 2017.

[216] Masahiro Tomono. 3-d object map building using dense object models with sift-based recognition features. In *International Conference on Intelligent Robotics and Systems (IROS)*, 2006.

[217] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *International Conference on Computer Vision (ICCV)*, pages 4489–4497, 2015.

[218] Son D Tran and Larry S Davis. Event modeling and recognition using markov logic networks. In *European Conference on Computer Vision (ECCV)*, 2008.

[219] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Adversarial discriminative domain adaptation. In *Computer Vision and Pattern Recogntion (CVPR)*, 2017.

[220] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, page 2, 2019.

[221] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 3156–3164, 2015.

[222] Tomas Vojir. Tracking with kernelized correlation filters. `https://github.com/vojirt/kcf`.

[223] Tomas Vojir, Jana Noskova, and Jiri Matas. Robust scale-adaptive mean-shift for tracking. In *Scandinavian Conference on Image Analysis*, pages 652–663. Springer, 2013.

[224] Carl Vondrick, Abhinav Shrivastava, Alireza Fathi, Sergio Guadarrama, and Kevin Murphy. Tracking emerges by colorizing videos. In *European Conference on Computer Vision (ECCV)*, pages 391–408, 2018.

[225] Peng Wang, Qi Wu, Chunhua Shen, Anton van den Hengel, and Anthony R. Dick. Fvqa: Fact-based visual question answering. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[226] Xiaolong Wang, Ali Farhadi, and Abhinav Gupta. Actions ~ transformations. In *Computer Vision and Pattern Recogntion (CVPR)*, 2016.

[227] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *International Conference on Computer Vision (ICCV)*, pages 2794–2802, 2015.

[228] Xiaolong Wang, Allan Jabri, and Alexei A Efros. Learning correspondence from the cycle-consistency of time. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 2566–2576, 2019.

[229] Xuanhan Wang, Lianli Gao, Jingkuan Song, and Hengtao Shen. Beyond frame-level cnn: saliency-aware 3-d cnn with lstm for video action recognition. *IEEE Signal Processing Letters*, 24(4):510–514, 2016.

[230] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames, 2019.

[231] David Wooden. A guide to vision-based map building. *Robotics & Automation Magazine*, 2006.

[232] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Neural Information Processing Symposium (NeurIPS)*, 2015.

[233] Qi Wu, Damien Teney, Peng Wang, Chunhua Shen, Anthony R. Dick, and Anton van den Hengel. Visual question answering: A survey of methods and datasets. *Computing Resource Repository (CoRR)*, 2016.

[234] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(9):1834–1848, 2015.

[235] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 3733–3742, 2018.

[236] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. 4(6):2, 2000.

[237] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 9068–9079, 2018.

[238] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 3485–3492. IEEE, 2010.

[239] Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. Chalet: Cornell house agent learning environment. *Computing Resource Repository (CoRR)*, 2018.

[240] Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. Peorl: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 4860–4866. AAAI Press, 2018.

[241] Amir R. Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Computer Vision and Pattern Recogntion (CVPR)*. IEEE, 2018.

[242] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014.

[243] Jianming Zhang, Shugao Ma, and Stan Sclaroff. Meem: robust tracking via multiple experts using entropy minimization. In *European Conference on Computer Vision (ECCV)*, pages 188–203. Springer, 2014.

[244] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *International Conference on Intelligent Robotics and Systems (IROS)*, 2017.

[245] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision (ECCV)*, pages 649–666. Springer, 2016.

[246] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *International Conference on Computer Vision (ICCV)*, 2017.

[247] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Computer Vision and Pattern Recogntion (CVPR)*, pages 1851–1858, 2017.

[248] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision (ICCV)*, 2017.

[249] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.

Part III

APPENDIX

Appendix A

ADDITIONAL MATERIALS FOR CHAPTER 2

A.1 Implementation Details

For training our ResNet18 [83] representations we use the network up to the global average pooling followed by a fully connected layer (512 x 512), Leaky-ReLU and a final embedding layer (512 x 64). The features are L2-normalized and multiplied by $\tau = \frac{1}{0.07}$ as in MoCo [80]. We use 8 GPUs, a training batch size of 256 and a MoCo Memory Bank of size 65536 and a g-network momentum of 0.999. For multi-GPU training we employ the Shuffle-BN technique shuffling both the anchors and the positives to reduce the correlation between batches filled with multiple images from the same video. We use SGD with a learning rate of 0.03, momentum=0.9, and weight decay=0.0001. All of these hyperparameters are shared with MoCo [80]. All methods are trained for approximately 450k iterations. When selecting frames from a video, we pick with replacement. In addition to the natural augmentation, we perform standard data augmentation (color jitter, cropping, flipping) on the inputs. This prevents the network from relying too heavily on shared video statistics like mean frame color. After cropping, all images are resized to 224×224. It is worth noting that both VINCE and our data (R2V2) can be used with other network architectures or learning algorithms such as AMDIM [14], PIRL [154], or SimCLR [30]. We choose ResNet18 and MoCo due to implementation simplicity and relatively low computational constraints.

For ResNet50 training, we closely match the hyperparameters in MoCo v2 [31]. Specifically, we use the blur augmentation and stronger color augmentation and a cosine annealing learning rate for 286,000 iterations (200 epochs of ImageNet or equivalent on R2V2 regardless of the number of unique positives per batch). The batch size we use is 896 with an initial learning rate of 0.105 and an embedding dimensionality of 128. The temperature parameter used was $\tau = \frac{1}{0.2}$. We did not perform any hyperparameter searches for VINCE or R2V2, so these results may be suboptimal.

A.2 t-SNE

We provide the full resolution t-SNE [142] image for further inspection. Best viewed on a screen.

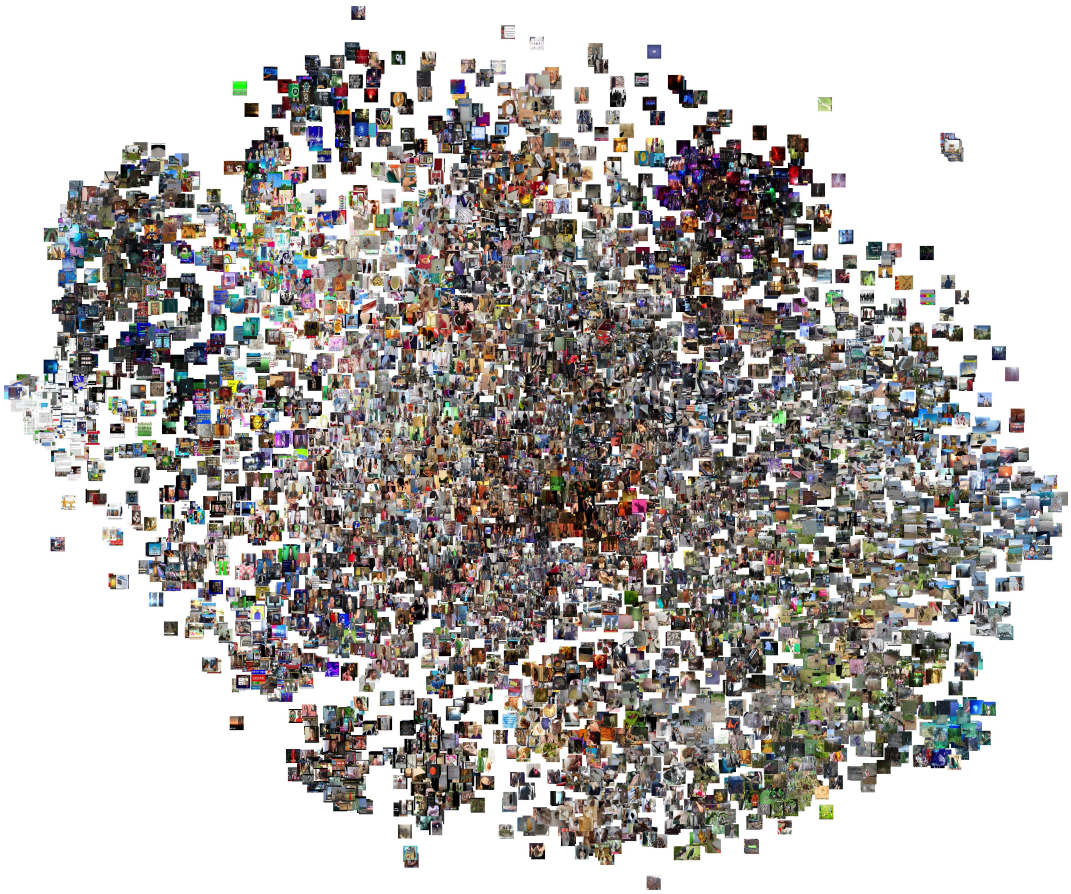


Figure A.1: **Full Resolution t-SNE embedding of images from R2V2 test set.**

A.3 Dataset Samples

Existing datasets such as YouTube8M [4] and Kinetics400 [109] provide a large number of YouTube links over a diverse set of videos. However, these dataset are highly unbalanced and contain many videos undesirable for learning strong visual representations. For example, the second most common category in YouTube8M, comprising 540k of the 3.7 million training videos is "Video Game," and the fifth is "Cartoon" with 240k. The category "Minecraft" itself has over

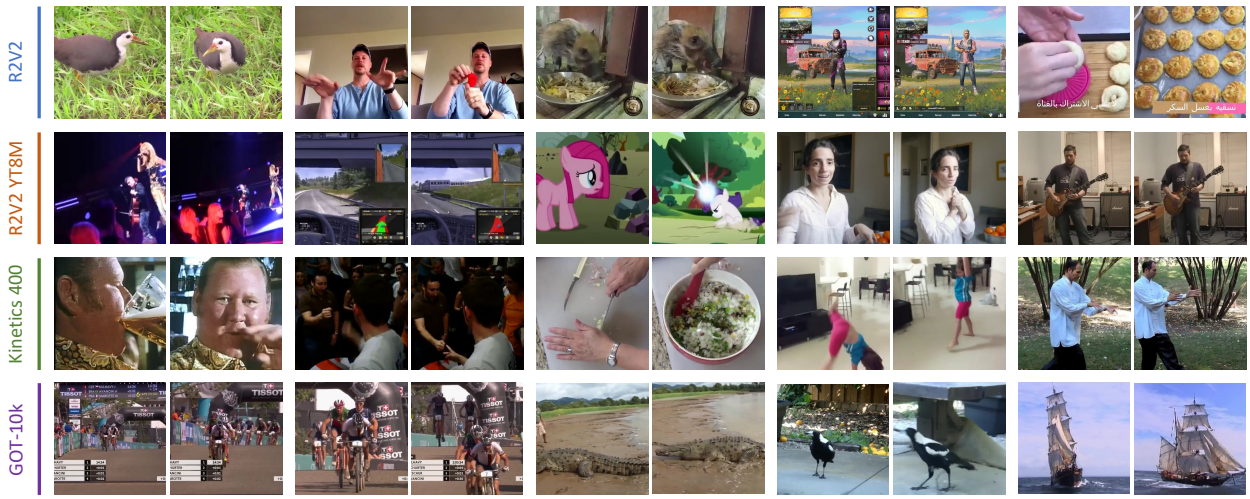


Figure A.2: **Random sampling of pairs of images from videos in each dataset.** In GOT-10k, sometimes different video clips are segments from the same original video as seen in the first and second sample. Images are square cropped for visualization purposes only.

57,000 videos in the dataset whereas the category "Pear" has only 138. Kinetics contains only videos of humans performing actions. Alternative datasets such as GOT-10k [97] provide a comparatively small number of videos but with dense annotations (in GOT-10k's case for object tracking).

A.3.1 Random Related Video Views Samples

We show more samples from R2V2 in Figure A.3. Videos each have four images 150 frames apart. Each separate video (outlined in blue) lists its corresponding YouTube link.

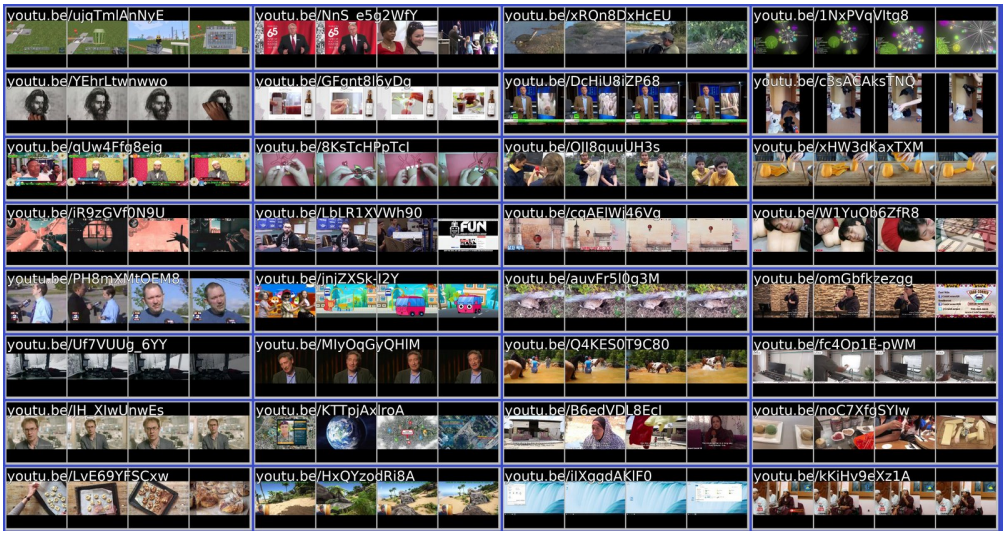
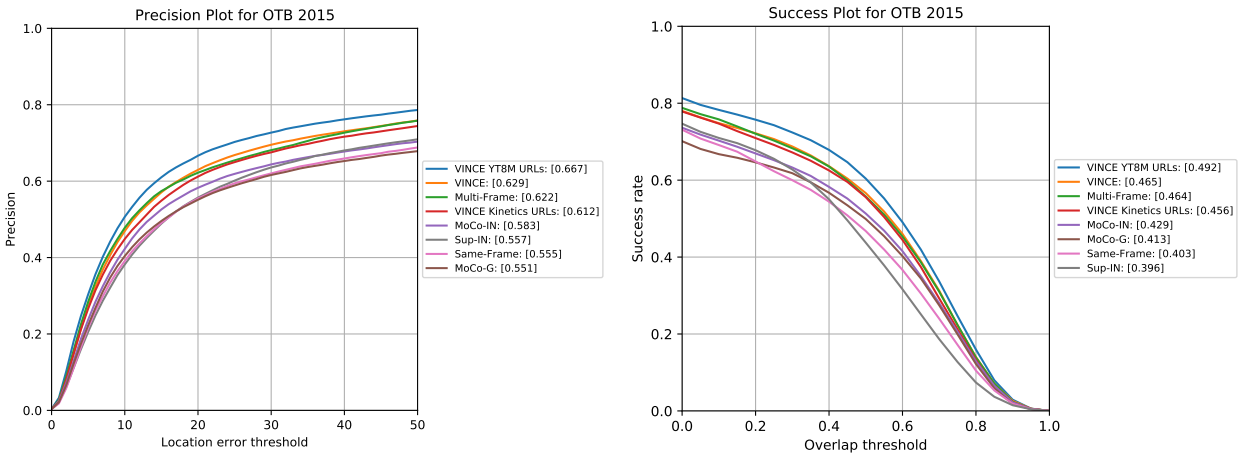


Figure A.3: **Sample from Random Related Video Views (train set).**

A.4 Precision and Success Plots for OTB 2015

We provide full breakdowns of the Precision and Success of each method on OTB 2015 [234] for the ResNet18 backbone. The values in the legend correspond to the (mean) area under the curve for each method.



(a) (b)

Figure A.4: **Precision (a) and Success (b) plots for OTB 2015 for various backbones.**

Appendix B

ADDITIONAL MATERIALS FOR CHAPTER 3

B.1 OPE Breakdown for OTB-50

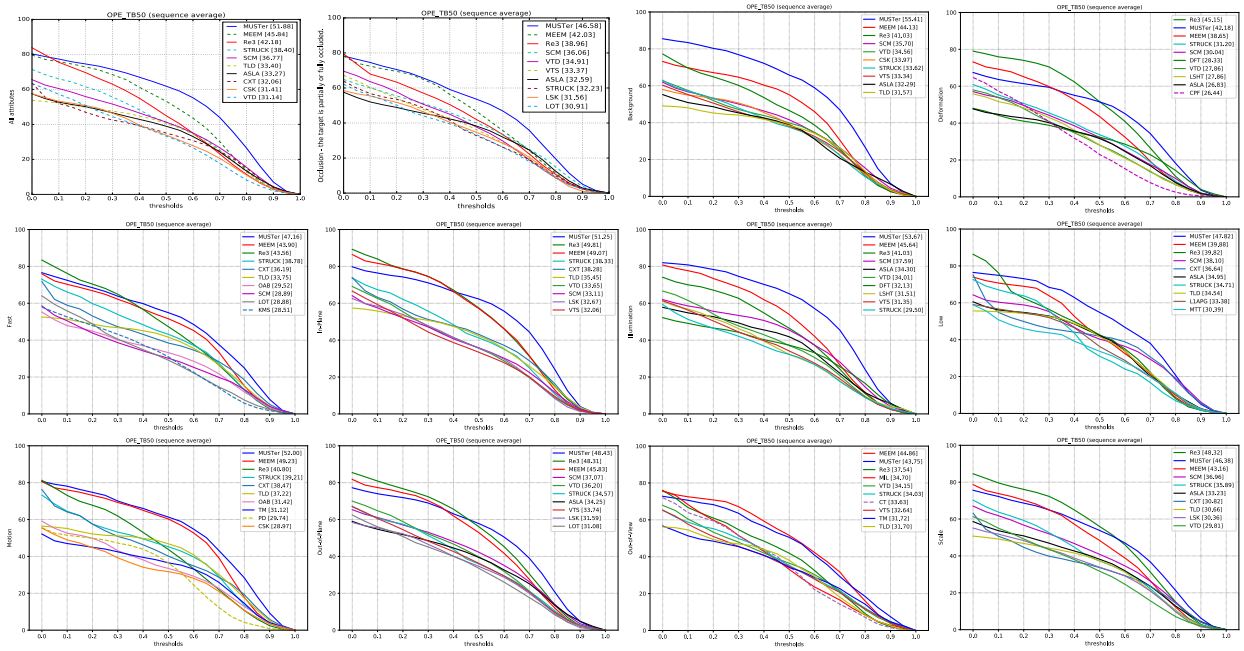


Figure B.1: **Full results of the OPE evaluation on the OTB-50 dataset.**

Here we show the breakdown by category provided by the OTB evaluation code.

B.2 Network Architecture

Here we provide a more detailed schematic of the network architecture used for all experiments (unless otherwise noted).

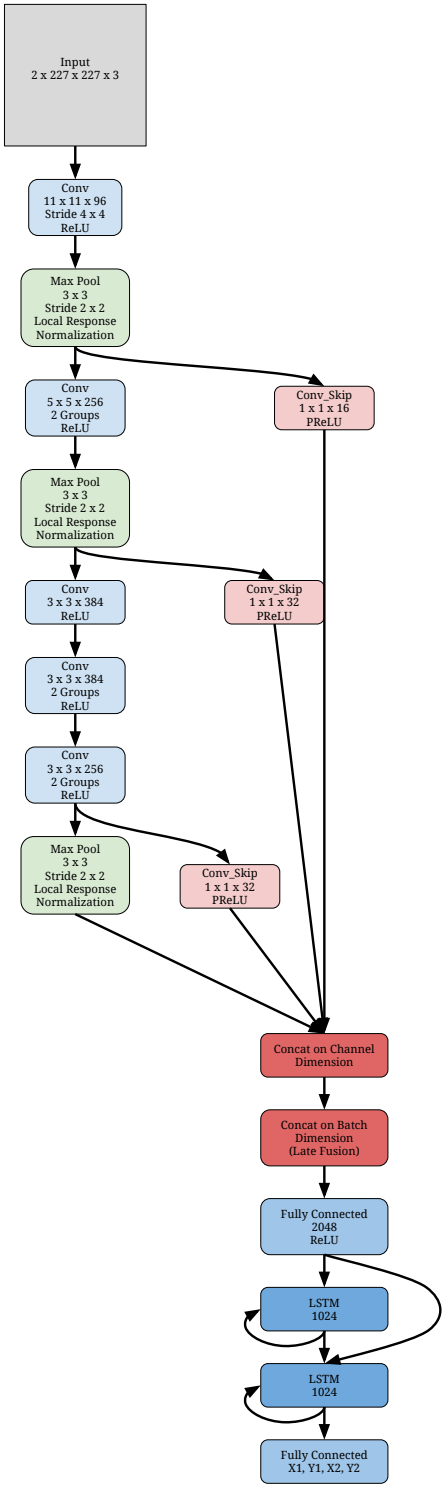


Figure B.2: **Re^3 Detailed Network Architecture.**

Appendix C

ADDITIONAL MATERIALS FOR CHAPTER 4

C.1 Experiment Details

C.1.1 Experiment Setup

We used the Adam optimizer [116] for learning our Successor Representation (SR) model with a learning rate of 1e-4 and a mini-batch size of 32. For the reinforcement learning experiments, we use the discounted factor $\gamma = 0.99$ and a replay buffer size of 100,000. The exploration term ϵ is annealed from 1.0 to 0.1 during the training process. We run an ϵ-greedy policy ($\epsilon = 0.1$) during evaluation. We use soft target updates ($\tau = 0.1$) after every episode. For the easy and medium tasks, we assign $+10.0$ immediate reward for task completion, -5.0 for invalid actions, and -1.0 for other actions (to encourage a shorter plan). For the hard task, we train our SR model to imitate a plan that searches all the *receptacles* for an object in a fixed order of visitation based on the spatial locations of the *receptacles*. We assign $+1.0$ immediate reward for task completion, and an episode terminates as failure if the agent does not follow the order of visitation in the plan.

C.1.2 Network Inputs

The input to the SR model consists of three components: action (action type and argument), agent's observation (image), and agent's internal state. The action type is encoded by a 7-dimensional one-hot vector, indicating one of the seven action types (Navigate, Open, Close, Pick Up, Put, Look Up, and Look Down). The action argument is encoded by a one-hot vector that has the same dimension as the number of interactable objects plus one. The first dimension denotes null argument used for Look Up and Look Down actions, and the other dimensions correspond to the index of each object. RGB images from the agent's first-person camera are preprocessed to

84×84 grayscale images. We stack four history frames to make an $84 \times 84 \times 4$ tensor as the image input to the convolutional networks. The agent's internal state is expressed by the agent's inventory, rotation, and viewpoint. The inventory is a one-hot vector that represents the index of the held *item*, with an extra dimension for null. The rotation is a 4-dimensional one-hot vector that represents the rotation of the agent (90 degree turns). The viewpoint is a 3-dimensional one-hot vector that represents the tiling angle of the agent's camera ($-30°$, $0°$, and $30°$).

C.1.3 Network Architecture

Here we describe the network architecture of our proposed SR model. The convolutional image encoder θ_{cnn} takes an $84 \times 84 \times 4$ image as input. The three convolutional layers are 32 filters of size 8×8 with stride 4, 64 filters of size 4×4 with stride 2, 64 filters of size 3×3 with stride 1. Finally a fully-connected layer maps the outputs from the convolutional encoder into a 512-d feature. The actions encoder θ_{mlp} and internal state encoder θ_{int} are both 2-layer MLPs with 512 hidden units. A concatenated vector of action, internal state, and image encodings is fed into two 2-layer MLPs θ_r and θ_q with 512 hidden units to produce the 512-d state-action feature $\phi_{s,a}$ and the successor feature $\psi_{s,a}$. We take the dot product of the 512-d reward predictor vector $\mathbf{w}$ and state-action features (successor features) to compute the immediate rewards (Q values). All the hidden layers use ReLU non-linearities. The final dot product layers of the immediate reward and the Q value produce raw values without any non-linearity.

Algorithm 3 Reinforcement Learning for Successor Representation Model

procedure RL-TRAINING
  Initialize replay buffer $\mathcal{D}$ to size N
  Initialize an SR network θ with random weights $\theta = [\theta_{int}, \theta_{cnn}, \theta_{mlp}, \theta_r, \theta_q, \mathbf{w}]$
  Make a clone of θ as the target network $\tilde{\theta}$
  for $i = 1$: *#episodes* **do**:
    Initialize an environment with random configuration
    Reset exploration term $\epsilon = 1.0$
    while not terminal **do**
      Get agent's observation and internal state s_t from the environment
      Compute $Q_{s_t,a} = f(s_t, a; \theta)$ for every action a in action space
      With probability ϵ select a random action a_t; otherwise, select $a_t = \arg\max_a Q_{s_t,a}$
      Execute action a_t to obtain the immediate reward r_t and the next state s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
      Sample a random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
      Compute $\tilde{r}_j$, ϕ_{s_j,a_j}, and ψ_{s_j,a_j} using θ for every transition j
      Compute gradients that minimize the mean squared error between r_j and $\tilde{r}_j$
      Compute $\phi_{s_{j+1},a}$, $\psi_{s_{j+1},a}$, and $\tilde{Q}_{s_{j+1},a}$ using $\tilde{\theta}$ for every transition j and every action a
      if s_{j+1} is a terminal state **then**:
        Compute gradients that minimize the mean squared error between ψ_{s_j,a_j} and ϕ_{s_j,a_j}
      else:
        Compute gradients that minimize the mean squared error between ψ_{s_j,a_j} and $\phi_{s_j,a_j} + \gamma\psi_{s_{j+1},a'}$
        where $a' = \arg\max_a \tilde{Q}_{s_{j+1},a}$
      Perform a gradient descend step to update θ
    Anneal exploration term ϵ
    Soft-update target network $\tilde{\theta}$ using θ

C.2 Algorithm Details

We describe the reinforcement learning procedure of the SR model in Algorithm 3. This training method follows closely with previous work on deep Q-learning [157] and deep SR model [128]. Similar to these two works, replay buffer and target network are used to stabilize training.

Figure C.1: **Screenshot of Scene #9.**

C.3 Action Space

The set of plausible actions in a scene is determined by the variety of objects in the scene. On average each scene has 53 objects (a subset of them are interactable) and the agent is able to perform 80 actions. Here we provide an example scene to illustrate the interactable objects and the action space.

Scene #9: 16 *items*, 23 *receptacles* (at 11 unique locations), and 15 *containers* (a subset of *receptacles*)

items: *apple*, *bowl*, *bread*, *butter knife*, *glass bottle*, *egg*, *fork*, *knife*, *lettuce*, *mug 1-3*, *plate*, *potato*, *spoon*, *tomato*

Scene	Easy	Medium	Hard
1	open / close *fridge*	put *lettuce, tomato* and *glass bottle* to the *sink*	find *bowl* and put in *sink*
2	open / close *cabinet*	put *apple, egg* and *glass bottle* to the *table top*	find *plate* and put in *cabinet*
3	open / close *microwave*	put *glass bottle, lettuce* and *apple* to the *table top*	find *lettuce* and put in *fridge*
4	open / close *cabinet*	put three *mug*s to the *fridge*	find *glass bottle* and put in *microwave*
5	open / close *fridge*	-	-
6	open / close *fridge*	-	-
7	open / close *cabinet*	put three *mug*s to the *table top*	-
8	open / close *fridge*	put *potato, tomato* and *apple* to the *sink*	find *lettuce* and put on *table top*
9	open / close *microwave*	put three *mug*s to the *table top*	find *glass bottle* and put in *fridge*
10	open / close *cabinet*	put *glass bottle, bread* and *lettuce* to *fridge*	find *bowl* and put in *sink*

Table C.1: **List of Tasks from Three Levels of Difficulty.**

receptacles: *cabinet 1-13*, *coffee machine*, *fridge*, *garbage can*, *microwave*, *sink*, *stove burner 1-4*, *table top*

containers: *cabinet 1-13*, *fridge*, *microwave*

actions: 80 actions in total, including 11 Navigation actions, 15 Open actions, 15 Close actions, 14 Pick Up actions, 23 Put actions, Look Up and Look Down.

We have fewer Navigation and Pick Up actions than the number of receptacles and items respectively, as we merge some adjacent receptacles to one location (navigation destination). We also merge picking up items from the same object category into one action. This reduces the size of the action space and speeds up learning. An important simplification that we made is to treat the Navigation actions as "teleports", which abstracts away from visual navigation of the agent. The actual visual navigation problem can be solved as an independent subroutine from previous work [249]. As discussed in Section 4.3.2, not all actions in the set can be issued given a certain circumstance based on affordance. We use the PDDL language to check if the preconditions of an action are satisfied before the action is sent to AI2-THOR for execution.

C.4 Tasks

We list the tasks that we have evaluated in the experiments in Table C.1. In summary, we evaluated tasks from three levels of difficulty, with 10 easy tasks, 8 medium tasks, and 7 hard tasks.

Appendix D

ADDITIONAL MATERIALS FOR CHAPTER 5

Dataset	Number of Train Scenes	Number of Train Episodes	Number of Val Scenes	Number of Val Episodes
IndoorEnv	990	898,267	905	99
MP3D	61	5,000,000	495	11
Gibson	72	4,932,479	1000	16

Table D.1: **Dataset Statistics.**

D.1 Dataset Details

We constructed the Point-Nav datasets for each of IndoorEnv, MP3D, and Gibson environments using a sampling-based method which filtered out easy episodes (those with $\frac{\text{euclidean distance}}{\text{geodesic distance}} < 1.1$). We additionally filter out episodes where there is no path between the start and goal location. The start points from these episodes were additionally used for the Exploration and Flee tasks, but the goal locations were ignored. Per-environment statistics are listen in Table D.1.

D.2 Cumulative Performance based on Starting Distance

To examine the balance of starting distances, Figure D.1 shows a breakdown of performance on all episodes which start closer than N meters. This additionally shows that 50% of the episodes start more than 9 meters away and nearly 25% start greater than 15 meters away.

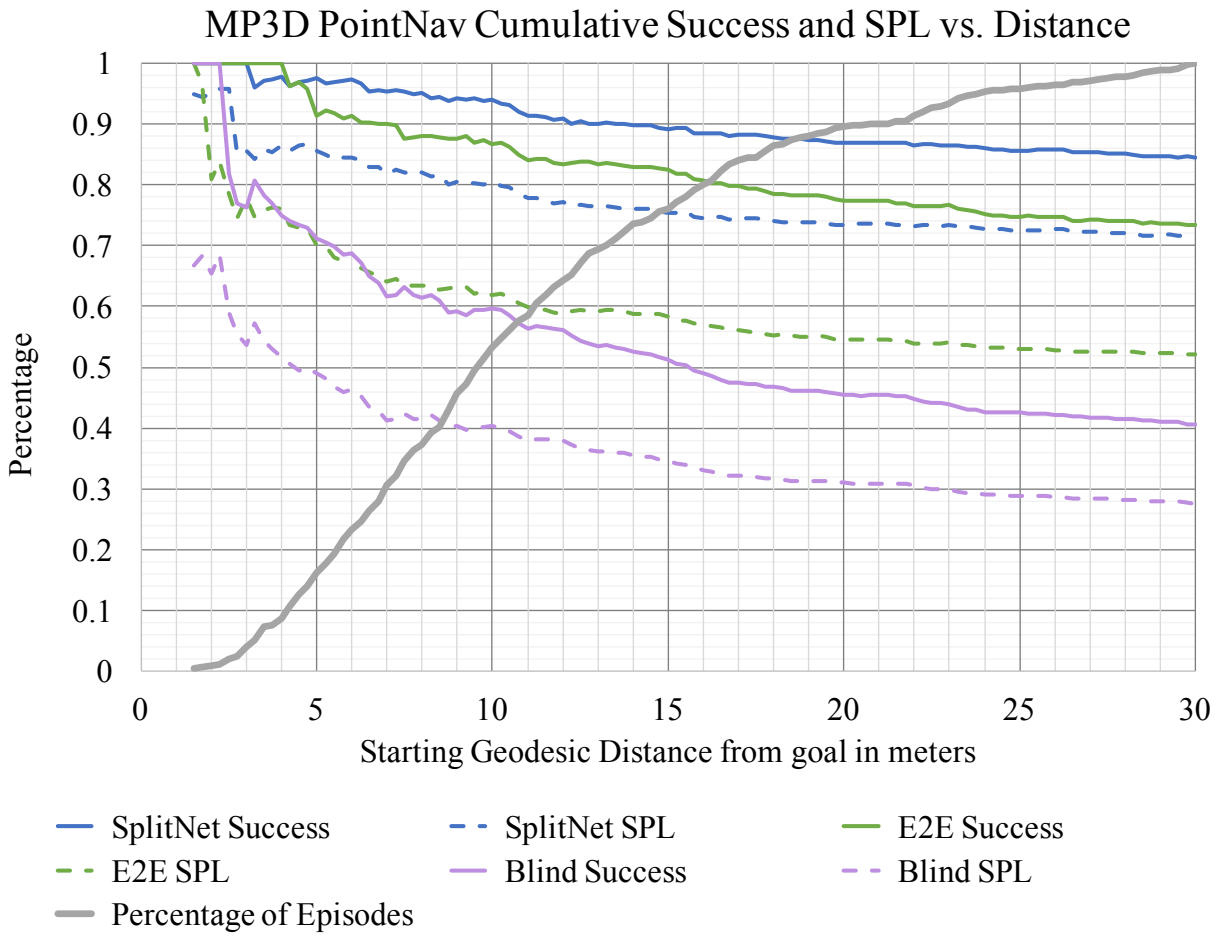


Figure D.1: **Cumulative Accuracy and SPL on MP3D dataset.**

D.3 Network Architecture

Figure D.2 shows the encoder-decoder architecture of SplitNet. The E2E method trains the blue and orange portions, and the blind agent trains only the orange. Additionally the Motion layers are only trained for SplitNet. Those are omitted for simplicity due to them operating on multiple timesteps.

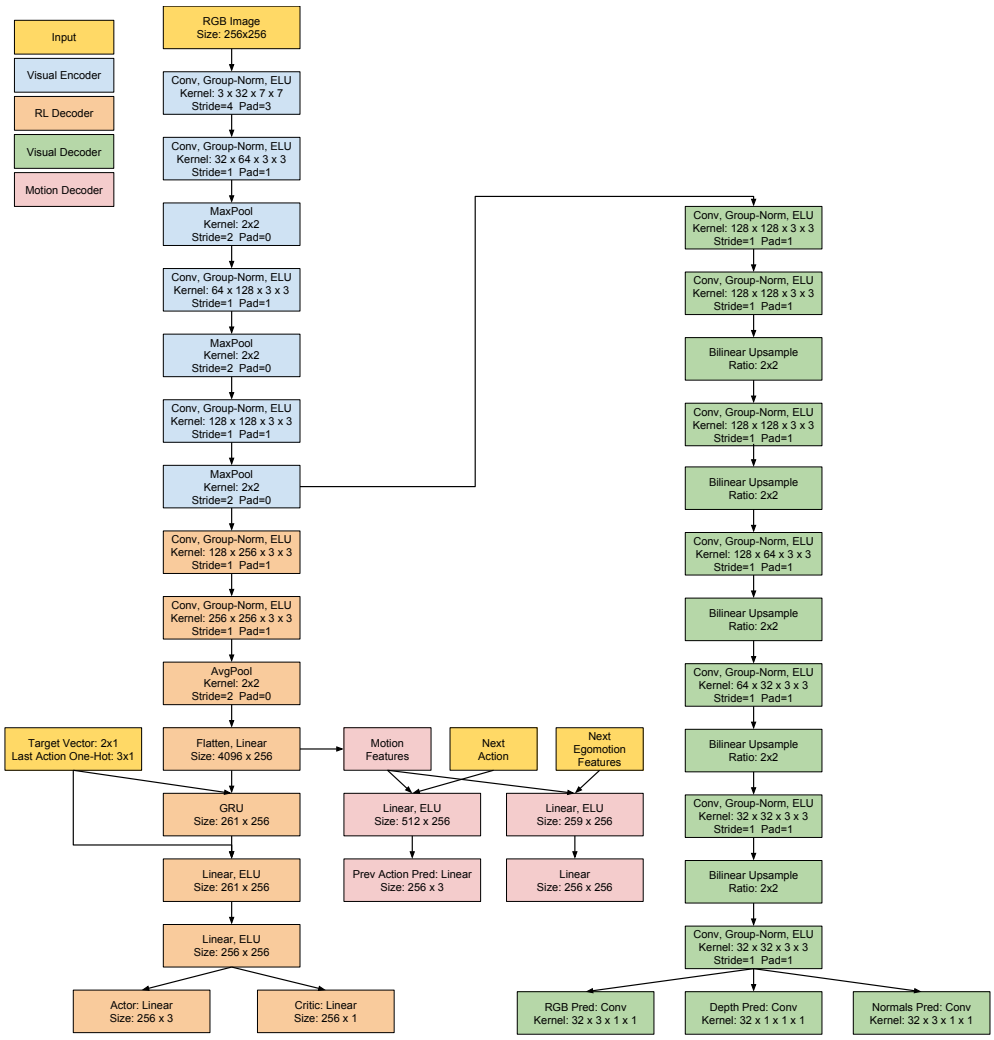


Figure D.2: **SplitNet Architecture.**

D.4 Auxiliary Outputs

To verify that our network learns to encode geometry and appearance information, we show the output of the RGB, depth, and surface normal decoders on test environments in Figure D.3. Learning these encoder-decoders, especially for depth, improves the network's ability to codify visual information into actionable representations. The decoders also allow us to see where the network makes mistakes as a method of debugging the failure modes.

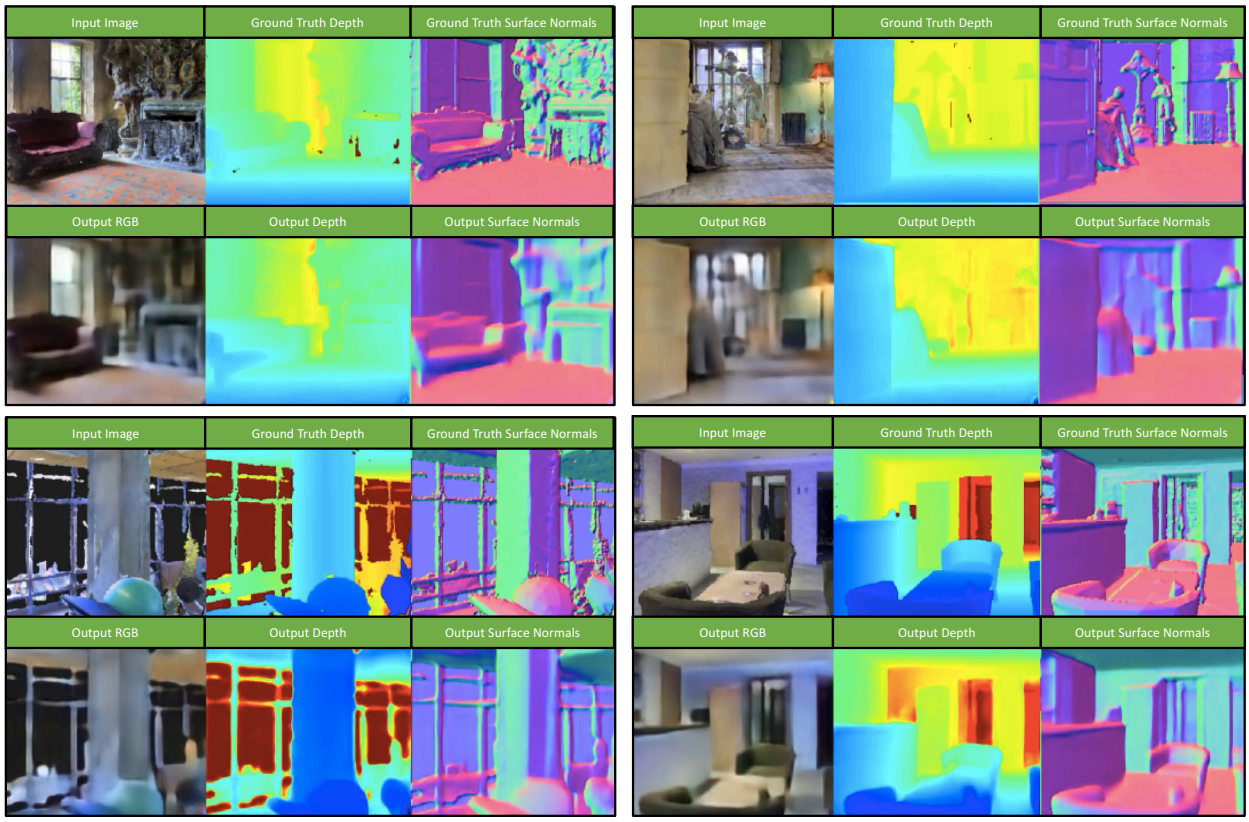


Figure D.3: **Example predictions of auxiliary outputs on unseen MP3D test environments.**

Appendix E

ADDITIONAL MATERIALS FOR CHAPTER 6

E.1 Full Accuracy Breakdown

	Accuracy of QA Per Answer							
	Existence		Counting				Spatial Relation	
Correct Answer	No	Yes	0	1	2	3	No	Yes
Always Answer Most Likely Value	57	0	27	0	0	0	52	0
A3C with no object detections	99.69	0.83	20.81	33.04	26.53	26.32	55.02	64.94
A3C with YOLO object detection	84.91	14.05	41.22	35.4	42.07	44.81	39.72	42.77
A3C with oracle object detections	100	1.92	34.29	11.11	35.14	23.08	64.21	68.09
HIMN	92.41	40	35.14	33.93	32.17	27.52	64.44	66.67
HIMN with YOLO object detection	92.14	25.62	48.32	53.57	48.30	50.66	50.93	52.89
HIMN with oracle navigator	100	37.6	36.24	53.57	51.02	42.11	70.82	72.29

Table E.1: **Results for each question category broken down by each possible answer.**

We separate the accuracies for different answers to explore potential biases in the questions as well as in the model behaviors, shown in Table E.1. We find that the questions are mostly balanced, yet Existence is slightly more likely to be false than true. This is a result of filtering out Existence questions where the object is placed in an unobservable location. Because of this, we notice that the A3C models, rather than learn how to explore the environment, simply exploits the bias in the questions. Our model, on the other hand is still quite likely to get true existence questions correct.

E.2 Network Architecture

The full HIMN network can be broken into several networks for navigation, planning, and answering. Their architectures are as follows:

E.2.1 Navigation Network

Inputs:

- Current image at $300 \times 300 \times 3$ resolution
- Previous Action One-Hot Vector
- Destination

Layers:

- Conv: $64 \times 7 \times 7$ kernels, stride 2, ELU activation
- Max Pool: 2×2, stride 2
- Conv: $128 \times 5 \times 5$ kernels, stride 1, ELU activation
- Max Pool: 2×2, stride 2
- Conv: $256 \times 3 \times 3$ kernels, stride 1, ELU activation
- Conv: $256 \times 3 \times 3$ kernels, stride 1, ELU activation
- Conv: $256 \times 3 \times 3$ kernels, stride 1, ELU activation
- Max Pool: 2×2, stride 2
- Conv: $512 \times 3 \times 3$ kernels, stride 1, ELU activation
- Conv: $512 \times 3 \times 3$ kernels, stride 1, ELU activation
- Conv: $512 \times 3 \times 3$ kernels, stride 1, ELU activation
- Max Pool: 2×2, stride 2
- FC1: Fully Connected on conv output: 1024 units, ELU activation
- FC2: Fully Connected on action one-hot: 32 units, ELU activation
- FC-Concat: Concatenate (FC1, FC2)
- GRU: 1024 units
- GRU-Concat: Concatenate(FC-Concat, GRU)
- Spatial GRU: $32 \times 5 \times 5$
- **Output** Path Weight: Conv: $1 \times 1 \times 1$, stride 1, activation = $min(max(1, 5 * e^x), 200)$
- Crop: Spatial GRU at Destination with 5×5 padding
- Conv: $32 \times 3 \times 3$ kernels, stride 1, ELU activation
- Conv: $32 \times 3 \times 3$ kernels, stride 1, ELU activation
- **Output** Terminal: Fully Connected, no activation.

E.2.2 Planner Network

Inputs:

- Current image at $300 \times 300 \times 3$ resolution
- Semantic map at $RoomW \times RoomH \times NumClasses + 5$
- Previous Action One-Hot Vector
- Question Encoding

Layers:

- Conv: $32 \times 7 \times 7$ kernels, stride 2, ReLU activation
- Max Pool: 2×2, stride 2
- Conv: $64 \times 5 \times 5$ kernels, stride 1, ReLU activation
- Max Pool: 2×2, stride 2
- Conv: $128 \times 3 \times 3$ kernels, stride 1, ReLU activation
- Max Pool: 2×2, stride 2
- Conv: $256 \times 3 \times 3$ kernels, stride 1, ReLU activation
- FC1: Fully Connected on conv output: 1024 units, ELU activation
- FC2: Fully Connected on action one-hot: 32 units, ELU activation
- FC-Concat: Concatenate (FC1, FC2)
- GRU: 1024 units
- GRU-Concat: Concatenate(FC-Concat, GRU)

- FCQ: Fully connected on Question Encoding: 64 units, ELU activation
- Tile FCQ: $RoomW \times RoomH$
- Concatenate(Semantic Map, Tile FCQ)
- Conv: $64 \times 1 \times 1$ kernels, stride 1, ELU activation
- Conv: $64 \times 11 \times 11$ kernels, stride 1, ELU activation
- Conv: $128 \times 3 \times 3$ kernels, stride 1, ELU activation
- Conv: $256 \times 3 \times 3$ kernels, stride 1, ELU activation
- Semantic Features: Conv: $256 \times 3 \times 3$ kernels, stride 1, ELU activation
- **Output**: Viability: Conv: $1 \times 1 \times 1$, stride 1, no activation
- Crop: 1×1 at Semantic Features(current spatial location)
- Concatenate (GRU-Concat, Crop)
- **Output**: V_{action}: Fully Connected on Concatenate
- **Output**: π_{action} Fully connected on Concatenate: 6 units

E.2.3 Answerer Network

Inputs:

- Semantic map at $RoomW \times RoomH \times NumClasses + 5$
- Question Encoding

Layers:

- Tile Question: $RoomW \times RoomH$
- Concatenate: (Semantic Map, Tile Question)
- Conv: $64 \times 1 \times 1$ kernels, stride 1, ELU activation
- Conv: $128 \times 3 \times 3$ kernels, stride 2, ELU activation
- Max Pool: 2×2, stride 2
- Conv: $128 \times 3 \times 3$ kernels, stride 2, ELU activation
- Max Pool: 2×2, stride 2
- Conv: $128 \times 3 \times 3$ kernels, stride 2, ELU activation
- Max Pool: 2×2, stride 2
- Spatial Sum
- Fully Connected: 128 units, ELU activation
- Fully Connected: 128 units, ELU activation
- **Output**: Answer: Fully Connected: NumAnswerClasses, no activation
- **Output**: V_{answer}: Fully Connected, no activation
- **Output**: π_{answer}: Fully Connected, no activation

E.3 Training details

To train the navigation network, we used the ADAM [116] optimization algorithm with the learning rate of 10^{-4}, and default Tensorflow constants. We trained with a batch size of 256 for 200,000 iterations. To train the planner and answerer, we use the RMSProp optimization algorithm with a learning rate of 10^{-3}. We use the learning curriculum described in section 6.4.4 for 5 million iterations of A3C (5 million distinct interactions with the environment).

Appendix F

ADDITIONAL MATERIALS FOR CHAPTER 7

F.1 PDDL Domain

Despite the complexity of our agent, the entire PDDL [149] domain can be specified on a single page since the rules are generic and templatized. Below is the PDDL Domain used for the work in Chapter 7.

```
(define (domain qa_vsp_task)
  (:requirements
    :adl
  )
  (:types
    agent
    location
    receptacle
    object
    rtype
    otype
  )

  (:predicates
    (atLocation ?a - agent ?l - location)
    (receptacleAtLocation ?r - receptacle ?l - location)
    (objectAtLocation ?o - object ?l - location)
    (openable ?r - receptacle)
    (opened ?r - receptacle)
    (inReceptacle ?o - object ?r - receptacle)
    (checked ?r - receptacle)
    (receptacleType ?r - receptacle ?t - rtype)
    (objectType ?o - object ?t - otype)
    (canContain ?t - rtype ?o - otype)
    (holds ?a - agent ?o - object)
    (holdsAny ?a - agent)
    (full ?r - receptacle)
  )

  (:functions
    (distance ?from ?to)
    (totalCost)
  )

  ;; agent goes to receptacle
```

```
(:action GotoLocation
  :parameters (?a - agent
               ?lStart - location
               ?lEnd - location)
  :precondition (atLocation ?a ?lStart)
  :effect (and
    (atLocation ?a ?lEnd)
    (not (atLocation ?a ?lStart))
    (forall (?r - receptacle)
      (when (and (receptacleAtLocation ?r ?lEnd)
                 (or (not (openable ?r)) (opened ?r)))
        (checked ?r)
      )
    )
    (increase (totalCost) (distance ?lStart ?lEnd))
  )
)

;; agent opens receptacle
(:action OpenObject
  :parameters (?a - agent
               ?l - location
               ?r - receptacle)
  :precondition (and
    (atLocation ?a ?l)
    (receptacleAtLocation ?r ?l)
    (openable ?r)
    (forall (?re - receptacle)
      (not (opened ?re)))
  )
  :effect (and
    (opened ?r)
    (checked ?r)
    (increase (totalCost) 1)
  )
)

;; agent closes receptacle
(:action CloseObject
  :parameters (?a - agent
               ?l - location
               ?r - receptacle)
  :precondition (and
    (atLocation ?a ?l)
    (receptacleAtLocation ?r ?l)
    (openable ?r)
    (opened ?r)
  )
  :effect (and
    (not (opened ?r))
    (increase (totalCost) 1)
  )
)
```

```
  ;; agent picks up object
  (:action PickupObject
    :parameters (?a - agent
                 ?l - location
                 ?o - object
                 ?r - receptacle)
    :precondition (and
      (atLocation ?a ?l)
      (objectAtLocation ?o ?l)
      (or (not (openable ?r)) (opened ?r))
      (inReceptacle ?o ?r)
      (not (holdsAny ?a))
    )
    :effect (and
      (not (inReceptacle ?o ?r))
      (holds ?a ?o)
      (holdsAny ?a)
      (increase (totalCost) 1)
    )
  )

  ;; agent puts down object
  (:action PutObject
    :parameters (?a - agent
                 ?l - location
                 ?ot - otype
                 ?o - object
                 ?r - receptacle)
    :precondition (and
      (atLocation ?a ?l)
      (receptacleAtLocation ?r ?l)
      (or (not (openable ?r)) (opened ?r))
      (not (full ?r))
      (objectType ?o ?ot)
      (holds ?a ?o)
    )
    :effect (and
      (inReceptacle ?o ?r)
      (full ?r)
      (not (holds ?a ?o))
      (not (holdsAny ?a))
      (increase (totalCost) 1)
    )
  )
)
```

F.2 Goal specifications

For IQUAD V1, we specify the goal state as either checking all locations where the question's subject could be, or knowing where the question's subject is. For example, for the question `Is there a mug in the microwave?` the Planner knows the agent must only check the microwave, but for the question `Is there a mug in the room?` the agent must check the microwave, the fridge, and all the drawers until it finds a mug. In general, every object can be in any location apart from a few notable exceptions such as silverware should not be in the microwave.

For EQA V1 [38], we state the goal as "the agent is looking at the object of interest if it knows where it is, or it has looked through all the doorways that it has seen." This encourages the Planner to explore various different rooms in the environment in the case where the object of interest is not located in the starting room. As PDDL is a natural fit for semantic planning, the goals are straightforward. For VSP the goals are simply specified as "object X is in receptacle Y."

F.3 PDDL Goal Example

Below is the `Find a mug` goal spec. for the question `Is there a mug in the room?`

```
(:goal
  (or
    ; Either there is a mug in the room or...
    (exists (?o - object)
      (objectType ?o MugType))
    (and
      (forall (?t - rtype)
        (forall (?r - receptacle) ; For all receptacles
          ; Either the receptacle cannot hold a mug
          ; Or the receptacle has been checked
          (or
            (not (and (canContain ?t MugType)
                      (receptacleType ?r ?t)))
            (checked ?r)
          )
        )
      )
    )
  )
)
```

Method	Existence		Counting		Spatial Relationship		Mean	
	Accuracy	Length	Accuracy	Length	Accuracy	Length	Mean Accuracy	Mean Length
HIMN	68.47%	318.33	30.43%	926.11	58.67%	516.23	52.52%	586.89
Planner Only	72.30%	**153.56**	35.94%	**226.48**	62.56%	**35.94**	56.93%	**138.66**
HIP-RL	**81.72%**	366.85	**45.00%**	547.8	**71.25%**	158.42	**65.99%**	357.69
HIMN with GT Detections	86.56%	679.7	35.31%	604.79	70.94%	311.03	64.27%	531.84
Planner Only with GT Detections	87.34%	**195.67**	53.75%	**259.3**	82.50%	**54.35**	74.53%	**169.77**
HIP-RL with GT Detections	**92.97%**	305.41	**66.25%**	427.56	**84.53%**	158.74	**81.25%**	297.24
HIMN with GT Detections and Nav	88.60%	618.63	48.44%	871.12	72.50%	475.55	69.85%	655.1
Planner Only with GT Detections and Nav	80.31%	**63.75**	44.06%	**85.28**	79.84%	**21.88**	68.07%	**56.97**
HIP-RL with GT Detections and Nav	**94.69%**	128.71	**66.72%**	183.11	**88.59%**	84.75	**83.33%**	132.19

Table F.1: **Breakdown of Accuracy and Episode Length for various methods on the three question types from IQUAD V1.**

F.4 Breakdown Per Question Type

Table F.1 shows how each method performs on each different question type from IQUAD V1. In all cases HIP-RL outperforms other methods. The Planner-Only results represent a lower bound for the episode length of a HIP-RL-like approach.

F.5 Generalization

One benefit of hierarchical models is they tend to generalize better as they force certain structure to be consistent between seen and unseen environments. Yet because the hierarchical models depend on the performance of individual components, the generalization performance of the constituent models directly affects the overall performance. In Table F.2 we explore the generalization of HIP-RL on IQUAD V1 and VSP by comparing performance on rooms seen during training time with never-before-seen rooms (EQA V1 only provides test questions for unseen environments). With ground truth detections, HIP-RL generalizes quite well, losing less than 10% raw performance in both cases and staying nearly as efficient step-wise. With Mask-RCNN [81] detections, there is a much larger gap. In AI2-THOR [120], there are frequently only as many examples of a particular class as there are scenes (e.g. all cabinets in one kitchen look similar, but they look very different

	IQUAD V1 Seen			VSP Seen		
	Accuracy	Length	SSPL	Success	Length	SPL
HIP-RL	77.75%	265.668	0.182	70.88%	245.301	0.384
HIP-RL + GT Det	87.04%	277.538	0.278	82.48%	170.22	0.504
	IQUAD V1 Unseen			VSP Unseen		
	Accuracy	Length	SSPL	Success	Length	SPL
HIP-RL	65.99%	357.69	0.086	46.01%	427.784	0.189
HIP-RL + GT Det	81.25%	297.238	0.177	73.75%	254.367	0.362

Table F.2: **Comparison of accuracy/success on seen and unseen environments.** HIP-RL is the full method, and HIP-RL + GT Det uses the ground truth detections.

to cabinets in another scene). Because there are so few different examples, Mask-RCNN struggles to detect objects in unseen environments from AI2-THOR. We believe that in scenarios with many more diverse training examples, HIP-RL with detection would approach the same level of generalization performance as without.