

Learning to Learn How to Learn: Self-Adaptive Visual Navigation Using Meta-Learning

Mitchell Wortsman¹, Kiana Ehsani², Mohammad Rastegari¹, Ali Farhadi^{1,2}, Roozbeh Mottaghi¹

¹ Allen Institute for AI (AI2), ² University of Washington

Abstract

Learning is an inherently continuous phenomenon. When humans learn a new task there is no explicit distinction between training and inference. After we learn a task, we keep learning about it while performing the task. What we learn and how we learn it varies during different stages of learning. Learning how to learn and adapt is a key property that enables us to generalize effortlessly to new settings. This is in contrast with conventional settings in machine learning where a trained model is frozen during inference. In this paper we study the problem of learning to learn at both training and inference time in the context of visual navigation. A fundamental challenge in navigation is generalization to unseen scenes. In this paper we propose a self-adaptive visual navigation method (SAVN) which learns to adapt to new environments without any explicit supervision. Our solution is a meta-reinforcement learning approach where an agent learns a self-supervised interaction loss that encourages effective navigation. Our experiments, performed in the AI2-THOR framework, show major improvements in both success rate and SPL for visual navigation in novel scenes.

1. Introduction

Learning is an inherently continuous phenomenon. We learn further about tasks that we have already learned and can learn to adapt to new environments by interacting in these environments. There is no hard boundary between the training and the testing phases while we are learning and performing tasks: we learn as we perform. This stands in stark contrast with many modern deep learning techniques, where the network is trained during the training phase and frozen during inference.

What we learn and how we learn it varies during different stages of learning. To learn a new task we often rely on explicit external supervision. After learning a task, we further learn as we adapt to new settings. This adaptation does not necessarily need explicit supervision; we often do this via interaction with the new environment.

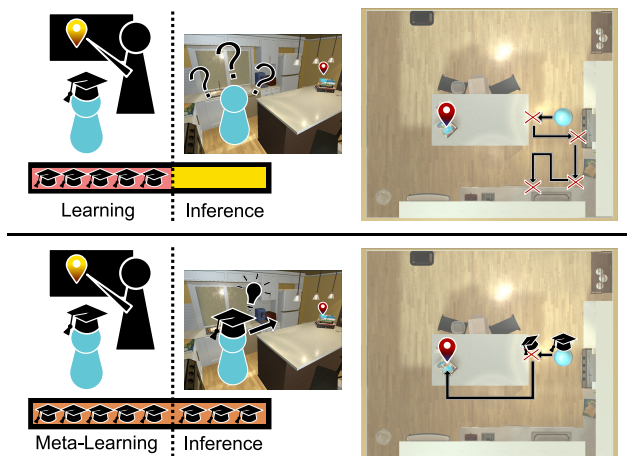


Figure 1. Traditional navigation approaches freeze the model after training (top row); this results in difficulties generalizing to unseen environments. In this paper, we propose a meta-reinforcement learning approach for navigation, where the agent learns to adapt to a novel environment in a self-supervised manner (bottom row). In this example, the agent learns to adapt itself to new environments when it collides with an object once and acts correctly afterwards. In contrast, a standard solution (top row) makes multiple mistakes of the same kind when performing the task in the new environment.

In this paper, we study the problem of learning to learn and adapt at both training and inference time in the context of visual navigation; one of the most crucial skills for any visually intelligent agent. The goal of visual navigation is to move towards certain objects or regions of an environment. A key challenge in navigation is generalizing to a scene that has not been observed during training, as the structure of the scene and appearance of objects are unfamiliar. In this paper we propose a self-adaptive visual navigation (SAVN) model which learns to adapt to a new environment during inference without any explicit supervision using an interaction loss (Figure 1).

Formally, our solution is a meta-reinforcement learning approach to visual navigation, where an agent learns to adapt through a self-supervised interaction loss that is learned during training. Our approach is inspired by gradi-

ent based meta-learning algorithms that learn quickly using a small amount of data [13]. In our approach, however, we learn quickly using a small amount of self-supervised interaction. In visual navigation, adaptation is possible without access to any reward function or positive example. As the agent trains, it learns a self-supervised loss that encourages effective navigation. During training, we encourage the gradients induced by the self-supervised loss to be similar to those we obtain from the supervised navigation loss. The agent is therefore able to adapt during inference, where explicit supervision is not available.

In summary, during both training and inference, the agent modifies its network while performing navigation. This approach differs from traditional reinforcement learning where the network is frozen during inference, and contrasts with supervised meta-learning as we learn to adapt to new environments during inference without access to rewards.

We perform our experiments using the AI2-THOR [23] framework. The agent aims to navigate to an instance of a given object category (e.g., *microwave*) using only visual observations. We show that SAVN outperforms the non-adaptive baseline in terms of both success rate (42.1 vs 32.9) and SPL (16.1 vs 13.7). Moreover, we demonstrate that learning a self-supervised loss provides improvement over hand-crafted self-supervised losses. Additionally, we show that our approach outperforms memory-augmented non-adaptive baselines.

2. Related Work

Deep Models for Navigation. Traditional navigation methods typically perform planning on a given map of the environment or build a map as the exploration proceeds [26, 40, 21, 24, 9, 4]. Recently, learning-based navigation methods (e.g., [50, 15, 27]) have become popular as they *implicitly* perform localization, mapping, exploration and semantic recognition end-to-end.

Zhu et al. [50] address target-driven navigation given a picture of the target. A joint mapper and planner has been introduced by [15]. [27] use auxiliary tasks such as loop closure to speed up RL training for navigation. We differ in our approach as we adapt dynamically to a novel scene. [37] propose the use of topological maps for the task of navigation. They explore the test environment for a long period to populate the memory. In our work, we learn to navigate without an exploration phase. [20] propose a self-supervised deep RL model for navigation. However, no semantic information is considered. [31] learn navigation policies based on object detectors and semantic segmentation modules. We do not rely on heavily supervised detectors and learn from a limited number of examples. [46, 44] incorporate semantic knowledge to better generalize to unseen scenarios. Both of these approaches dynamically up-

date their manually defined knowledge graphs. However, our model learns which parameters should be updated during navigation and how they should be updated. Learning-based navigation has been explored in the context of other applications such as autonomous driving (e.g., [7]), map-based city navigation (e.g., [5]) and game play (e.g., [43]). Navigation using language instructions has been explored by various works [3, 6, 17, 47, 29]. Our goal is different since we focus on using meta-learning to more effectively navigate new scenes using only the class label for the target.

Meta-learning. Meta-learning, or learning to learn, has been a topic of continued interest in machine learning research [41, 38]. More recently, various meta-learning techniques have pushed the state of the art in low-shot problems across domains [13, 28, 12].

Finn et al. [13] introduce Model Agnostic Meta-Learning (MAML) which uses SGD updates to adapt quickly to new tasks. This gradient based meta-learning approach may also be interpreted as learning a good parameter initialization such that the network performs well after only a few gradient updates. [25] and [48] augment the MAML algorithm so that it uses supervision in one domain to adapt to another. Our work differs as we do not use supervision or labeled examples to adapt.

Xu et al. [45] use meta-learning to significantly speed up training by encouraging exploration of the state space outside of what the actor’s policy dictates. Additionally, [14] use meta-learning to augment the agent’s policy with structured noise. At inference time, the agent is able to better adapt from a few episodes due to the variability of these episodes. Our work instead emphasizes self-supervised adaptation while executing a single visual navigation task. Neither of these works consider this domain.

Clavera et al. [8] consider the problem of learning to adapt to unexpected perturbations using meta-learning. Our approach is similar as we also consider the problem of learning to adapt. However, we consider the problem of visual navigation and adapt via a self-supervised loss.

Both [18] and [48] learn an objective function. However, [18] use evolutionary strategies instead of meta-learning. Our approach for learning a loss is inspired by and similar to [48]. However, we adapt in the same domain without explicit supervision while they adapt across domains using a video demonstration.

Self-supervision. Different types of self-supervision have been explored in the literature [1, 19, 11, 42, 49, 36, 34, 32]. Some works aim to maximize the prediction error in the representation of future states [33, 39]. In this work, we *learn* a self-supervised objective which encourages effective navigation.

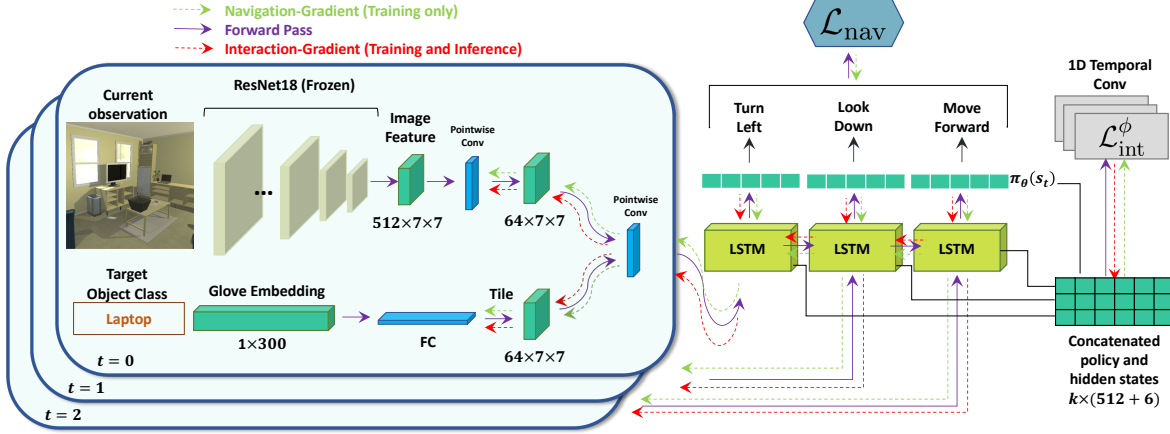


Figure 2. **Model overview.** Our network optimizes two objective functions, 1) self-supervised interaction loss $\mathcal{L}_{\text{int}}^{\phi}$ and 2) navigation loss \mathcal{L}_{nav} . The inputs to the network at each time t are the egocentric image and word embedding of the target object class. The network outputs a policy $\pi_{\theta}(s_t)$. During training, the interaction and navigation-gradients are back-propagated through the network, and the parameters of the self-supervised loss are updated at the end of each episode using navigation-gradients. During inference, the parameters of the interaction loss remain fixed while the rest of the network is updated using interaction-gradients. Note that the green color in the figure represents the intermediate and final outputs.

3. Adaptive Navigation

In this section, we begin by formally presenting the task and our base model without adaptation. We then explain how to incorporate adaptation and perform training and inference in this setting.

3.1. Task Definition

Given a target object class, e.g. *microwave*, our goal is to navigate to an instance of an object from this class using only visual observations.

Formally, we consider a set of scenes $\mathcal{S} = \{S_1, \dots, S_n\}$ and target object classes $\mathcal{O} = \{o_1, \dots, o_m\}$. A task $\tau \in \mathcal{T}$ consists of a scene S , target object class $o \in \mathcal{O}$, and initial spatial position p . We therefore denote each task τ by the tuple $\tau = (S, o, p)$. We consider disjoint sets of scenes for the training tasks $\mathcal{T}_{\text{train}}$ and testing tasks $\mathcal{T}_{\text{test}}$. We refer to the trial of a navigation task as an episode.

The agent is required to navigate using only the egocentric RGB images and the target object class (the target object class is given as a Glove embedding [35]). At each time t the agent takes an action a from the action set \mathcal{A} until the termination action is issued by the agent. We consider an episode to be successful if, within certain number of steps, the agent issues a termination action when an object from the given target class is sufficiently close and visible. If a termination action is issued at any other time, then the episode concludes and the agent has failed.

3.2. Learning

Before we discuss our self-adaptive approach we begin with an overview of our base model and discuss deep rein-

forcement learning for navigation in a traditional sense.

We let s_t , the egocentric RGB image, denote the agent's state at time t . Given s_t and the target object class, the network (parameterized by θ) returns a distribution over the actions which we denote $\pi_{\theta}(s_t)$ and a scalar $v_{\theta}(s_t)$. The distribution $\pi_{\theta}(s_t)$ is referred to as the agent's *policy* while $v_{\theta}(s_t)$ is the *value* of the state. Finally, we let $\pi_{\theta}^{(a)}(s_t)$ denote the probability that the agent chooses action a .

We use a traditional supervised actor-critic navigation loss as in [50, 27] which we denote \mathcal{L}_{nav} . By minimizing \mathcal{L}_{nav} , we maximize a reward function that penalizes the agent for taking a step while incentivizing the agent to reach the target. The loss is a function of the agent's policies, values, actions, and rewards throughout an episode.

The network architecture is illustrated in Figure 2. We use a ResNet18 [16] pretrained on ImageNet [10] to extract a feature map for a given image. We then obtain a joint feature-map consisting of both image and target information and perform a pointwise convolution. The output is then flattened and given as input to a Long Short-Term Memory network (LSTM). For the remainder of this work we refer to the LSTM hidden state and agent's internal state representation interchangeably. After applying an additional linear layer we obtain the policy and value. In Figure 2 we do not show the ReLU activations we use throughout, or reference the value $v_{\theta}(s_t)$.

3.3. Learning to Learn

In visual navigation there is ample opportunity for the agent to learn and adapt by interacting with the environment. For example, the agent may learn how to handle ob-

stacles it is initially unable to circumvent. We therefore propose a method in which the agent learns how to adapt from interaction. The foundation of our method lies in recent works which present gradient based algorithms for learning to learn (meta-learning).

Background on Gradient Based Meta-Learning. We rely on the meta-learning approach detailed by the MAML algorithm [13]. The MAML algorithm optimizes for fast adaptation to new tasks. If the distribution of training and testing tasks are sufficiently similar then a network trained with MAML should quickly adapt to novel test tasks.

MAML assumes that during training we have access to a large set of tasks $\mathcal{T}_{\text{train}}$ where each task $\tau \in \mathcal{T}_{\text{train}}$ has a small meta-training dataset $\mathcal{D}_{\tau}^{\text{tr}}$ and meta-validation set $\mathcal{D}_{\tau}^{\text{val}}$. For example, in the problem of k -shot image classification, τ is a set of image classes and $\mathcal{D}_{\tau}^{\text{tr}}$ contains k examples of each class. The goal is then to correctly assign one of the class labels to each image in $\mathcal{D}_{\tau}^{\text{val}}$. A testing task $\tau \in \mathcal{T}_{\text{test}}$ then consists of unseen classes.

The training objective of MAML is given by

$$\min_{\theta} \sum_{\tau \in \mathcal{T}_{\text{train}}} \mathcal{L}(\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_{\tau}^{\text{tr}}), \mathcal{D}_{\tau}^{\text{val}}), \quad (1)$$

where the loss \mathcal{L} is written as a function of a dataset and the network parameters θ . Additionally, α is the step size hyper-parameter, and ∇ denotes the differential operator (gradient). The idea is to learn parameters θ such that they provide a good initialization for fast adaptation to test tasks. Formally, Equation (1) optimizes for performance on $\mathcal{D}_{\tau}^{\text{val}}$ after *adapting* to the task with a gradient step on $\mathcal{D}_{\tau}^{\text{tr}}$. Instead of using the network parameters θ for inference on $\mathcal{D}_{\tau}^{\text{val}}$, we use the *adapted* parameters $\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_{\tau}^{\text{tr}})$. In practice, multiple SGD updates may be used to compute the adapted parameters.

Training Objective for Navigation. Our goal is for an agent to be continually learning as it interacts with an environment. As in MAML, we use SGD updates for this adaptation. These SGD updates modify the agent’s policy network as it interacts with a scene, allowing the agent to adapt to the scene. We propose that these updates should occur with respect to \mathcal{L}_{int} , which we call an *interaction loss*. Minimizing \mathcal{L}_{int} should assist the agent in completing its navigation task, and it can be learned or hand-crafted. For example, a hand-crafted variation may penalize the agent for visiting the same location twice. In order for the agent to have access to \mathcal{L}_{int} during inference, we use a self-supervised loss. Our objective is then to learn a good initialization θ , such that the agent will learn to effectively navigate in an environment after a few gradient updates using \mathcal{L}_{int} .

For clarity, we begin by formally presenting our method in a simplified setting in which we allow for a single SGD update with respect to \mathcal{L}_{int} . For a navigation task τ we let $\mathcal{D}_{\tau}^{\text{int}}$ denote the actions, observations, and internal state rep-

resentations (defined in Section 3.2) for the first k steps of the agent’s trajectory. Additionally, let $\mathcal{D}_{\tau}^{\text{nav}}$ denote this same information for the remainder of the trajectory. Our training objective is then formally given by

$$\min_{\theta} \sum_{\tau \in \mathcal{T}_{\text{train}}} \mathcal{L}_{\text{nav}}(\theta - \alpha \nabla_{\theta} \mathcal{L}_{\text{int}}(\theta, \mathcal{D}_{\tau}^{\text{int}}), \mathcal{D}_{\tau}^{\text{nav}}), \quad (2)$$

which mirrors the MAML objective from Equation (1). However, we have replaced the small training set $\mathcal{D}_{\tau}^{\text{tr}}$ from MAML with an interaction phase. The intuition for our objective is as follows: at first we interact with the environment and then we adapt to it. More specifically, the agent interacts with the scene using the parameters θ . After k steps an SGD update with respect to the self-supervised loss is used to obtain the *adapted* parameters $\theta - \alpha \nabla_{\theta} \mathcal{L}_{\text{int}}(\theta, \mathcal{D}_{\tau}^{\text{int}})$.

In domain adaptive meta-learning, two separate losses are used for adaptation from one domain to another [25, 48]. A similar objective to Equation (2) is employed by [48] for one-shot imitation from observing humans. Our method differs in that we are learning how to adapt in the same domain through self-supervised interaction.

As in [25], a first order Taylor expansion provides intuition for our training objective. Equation (2) is approximated by

$$\min_{\theta} \sum_{\tau \in \mathcal{T}_{\text{train}}} \mathcal{L}_{\text{nav}}(\theta, \mathcal{D}_{\tau}^{\text{nav}}) - \alpha \langle \nabla_{\theta} \mathcal{L}_{\text{int}}(\theta, \mathcal{D}_{\tau}^{\text{int}}), \nabla_{\theta} \mathcal{L}_{\text{nav}}(\theta, \mathcal{D}_{\tau}^{\text{nav}}) \rangle, \quad (3)$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product. We are therefore learning to minimize the navigation loss while *maximizing* the *similarity* between the gradients we obtain from the self-supervised interaction loss and the supervised navigation loss. If the gradients we obtain from both losses are similar, then we are able to continue “training” during inference when we do not have access to \mathcal{L}_{nav} . However, it may be difficult to choose \mathcal{L}_{int} which allows for similar gradients. This directly motivates *learning* the self-supervised interaction loss.

3.4. Learning to Learn How to Learn

We propose to *learn* a self-supervised interaction objective that is explicitly tailored to our task. Our goal is for the agent to improve at navigation by minimizing this self-supervised loss in the current environment.

During training, we both learn this objective and learn how to learn using this objective. We are therefore “learning to learn how to learn”. As input to this loss we use the agent’s previous k internal state representations concatenated with the agent’s policy.

Formally, we consider the case where \mathcal{L}_{int} is a neural network parameterized by ϕ , which we denote $\mathcal{L}_{\text{int}}^{\phi}$. Our training objective then becomes

Algorithm 1 SAVN-Training($\mathcal{T}_{\text{train}}, \alpha, \beta_1, \beta_2, k$)

```

1: Randomly initialize  $\theta, \phi$ .
2: while not converged do
3:   for mini-batch of tasks  $\tau_i \in \mathcal{T}_{\text{train}}$  do
4:      $\theta_i \leftarrow \theta$ 
5:      $t \leftarrow 0$ 
6:     while termination action is not issued do
7:       Take action  $a$  sampled from  $\pi_{\theta_i}(s_t)$ 
8:        $t \leftarrow t + 1$ 
9:       if  $t$  is divisible by  $k$  then
10:         $\theta_i \leftarrow \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}_{\text{int}}^{\phi}(\theta_i, \mathcal{D}_{\tau}^{(t,k)})$ 
11:       $\theta \leftarrow \theta - \beta_1 \sum_{i=1}^{\ell} \nabla_{\theta} \mathcal{L}_{\text{nav}}(\theta_i, \mathcal{D}_{\tau})$ 
12:       $\phi \leftarrow \phi - \beta_2 \sum_{i=1}^{\ell} \nabla_{\phi} \mathcal{L}_{\text{nav}}(\theta_i, \mathcal{D}_{\tau})$ 
13: return  $\theta, \phi$ 

```

$$\min_{\theta, \phi} \sum_{\tau \in \mathcal{T}_{\text{train}}} \mathcal{L}_{\text{nav}}(\theta - \alpha \nabla_{\theta} \mathcal{L}_{\text{int}}^{\phi}(\theta, \mathcal{D}_{\tau}^{\text{int}}), \mathcal{D}_{\tau}^{\text{nav}}) \quad (4)$$

and we freeze the parameters ϕ during inference. There is no explicit objective for the learned-loss. Instead, we simply encourage that minimizing this loss allows the agent to navigate effectively. This may occur if the gradients from both losses are similar. In this sense we are training the self-supervised loss to imitate the supervised \mathcal{L}_{nav} loss.

As in [48], we use one dimensional temporal convolutions for the architecture of our learned loss. We use two layers, the first with 10×1 filters and the next with 1×1 . As input we concatenate the past k hidden states of the LSTM and the previous k policies. To obtain the scalar objective we take the ℓ_2 norm of the output. Though we omit the ℓ_2 norm, we illustrate our interaction loss in Figure 2.

Hand Crafted Interaction Objectives. We also experiment with two variations of simple hand crafted interaction losses which can be used as an alternative to the learned loss. The first is a diversity loss $\mathcal{L}_{\text{int}}^{\text{div}}$ which encourages the agent to take varied actions. If the agent does happen to reach the same state multiple times it should definitely not repeat the action it previously took. Accordingly,

$$\mathcal{L}_{\text{int}}^{\text{div}}(\theta, \mathcal{D}_{\tau}^{\text{int}}) = \sum_{i < j \leq k} g(s_i, s_j) \log(\pi_{\theta}^{(a_i)}(s_j)), \quad (5)$$

where s_t is the agent's state at time t , a_t is the action the agent takes at time t , and g calculates the similarity between two states. For simplicity we let $g(s_i, s_j)$ be 1 if the pixel difference between s_i and s_j is below a certain threshold and 0 otherwise.

Additionally, we consider a prediction loss $\mathcal{L}_{\text{int}}^{\text{pred}}$ where the agent aims to predict the success of each action. The idea is to avoid taking actions that the network predicts will

Algorithm 2 SAVN-Inference($\mathcal{T}_{\text{test}}, \theta, \phi, \alpha, \beta, k$)

```

1: for mini-batch of tasks  $\tau_i \in \mathcal{T}_{\text{test}}$  do
2:    $\theta_i \leftarrow \theta$ 
3:    $t \leftarrow 0$ 
4:   while termination action is not issued do
5:     Take action  $a$  sampled from  $\pi_{\theta_i}(s_t)$ 
6:      $t \leftarrow t + 1$ 
7:     if  $t$  is divisible by  $k$  then
8:        $\theta_i \leftarrow \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}_{\text{int}}^{\phi}(\theta_i, \mathcal{D}_{\tau}^{(t,k)})$ 

```

fail. We say that the agent's action has failed if we detect sufficient similarity in two consecutive states. This may occur when the agent bumps into an object or wall. In addition to producing a policy π_{θ} over actions the agent also predicts the success of each action. For state s_t we denote the predicted probability that action a succeeds as $q_{\theta}^{(a)}(s_t)$. Instead of sampling an action from $\pi_{\theta}(s_t)$ we instead use $\tilde{\pi}_{\theta}(s_t) = \pi_{\theta}(s_t) * q_{\theta}(s_t)$ where $*$ denotes element-wise multiplication.

For $\mathcal{L}_{\text{int}}^{\text{pred}}$ we use a standard binary cross entropy loss between our success prediction $q_{\theta}^{(a)}$ and observed success. Using the same g from Equation (5) we write our loss as

$$\mathcal{L}_{\text{int}}^{\text{pred}}(\theta, \mathcal{D}_{\tau}^{\text{int}}) = \sum_{t=0}^{k-1} \mathcal{H}(q_{\theta}^{(a_t)}(s_t), 1 - g(s_t, s_{t+1})), \quad (6)$$

where $\mathcal{H}(\cdot, \cdot)$ denotes binary cross-entropy.

We acknowledge that in a non-synthetic environment it may be difficult to produce a reliable function g . Therefore we only use g in the hand-crafted variations of the loss.

3.5. Training and Inference

So far we have implicitly decomposed the agent's trajectory into an interaction and navigation phase. In practice, we would like the agent to keep adapting until the object is found during both training and inference. We therefore perform an SGD update with respect to the self-supervised interaction loss every k steps. We compute the interaction loss at time t by using the information from the previous k steps of the agent's trajectory, which we denote $\mathcal{D}_{\tau}^{(t,k)}$. Note that $\mathcal{D}_{\tau}^{(t,k)}$ is analogous to $\mathcal{D}_{\tau}^{\text{int}}$ in Equation (4). In addition, the agent should be able to navigate efficiently. Hence, we compute the navigation loss \mathcal{L}_{nav} using the the information from the complete trajectory of the agent, denoted by \mathcal{D}_{τ} .

For the remainder of this work we refer to the gradient with respect to \mathcal{L}_{int} as the *interaction-gradient* and the gradient with respect to \mathcal{L}_{nav} as the *navigation-gradient*. These gradients are illustrated in Figure 2 by red and green arrows, respectively. Note that we do not update the loss parameters ϕ via the interaction-gradient.

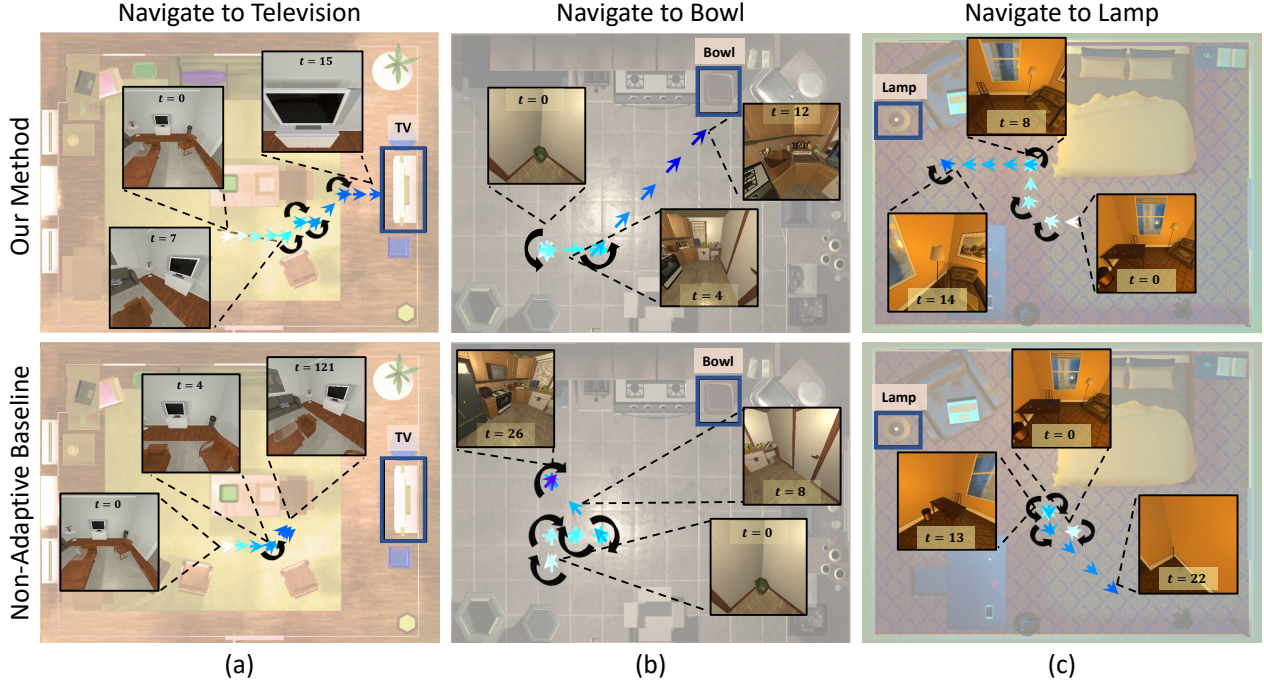


Figure 3. **Qualitative examples.** We compare our method with the non-adaptive baseline. We illustrate the trajectory of the agent from the bird’s eye view (white corresponds to the beginning of the trajectory and dark blue shows the end). Black arrows represent rotation. We also show the egocentric view of the agent at a few time steps. Our method learns from its mistakes (e.g., getting stuck behind an object) at inference time.

Algorithms 1 and 2 detail our method for training and inference, respectively. In Algorithm 1 we learn a policy network π_θ and a loss network parameterized by ϕ with step-size hyper-parameters α, β_1, β_2 . Recall that k is a hyper-parameter which prescribes the frequency of the interaction-gradients. If we are instead considering a hand-crafted self-supervised loss then we ignore ϕ and omit line 12.

Recall that the adapted parameters, which we denote θ_i in Algorithm 1 and 2, are implicitly a function of θ, ϕ . Therefore, the differentiation in lines 11 and 12 is well defined though it requires the computation of Hessian vector-products. We never compute more than 4 interaction-gradients due to computational constraints.

During inference we may adapt in a new environment with respect to the self-supervised interaction loss, but we no longer have access to \mathcal{L}_{nav} . Additionally, note that the shared parameter θ is not updated during inference, as detailed in Algorithm 2.

4. Experiments

Our goal in this section is to (1) evaluate our self-adaptive navigation model in comparison to non-adaptive baselines, (2) determine if the learned self-supervised objective provides any improvement over hand-crafted self-supervised losses, and (3) gain insight into how and why our method may be improving performance.

4.1. Experiment setup

We train and evaluate our models using the AI2-THOR [23] environment. AI2-THOR provides indoor 3D synthetic scenes in four room categories, kitchen, living room, bedroom and bathroom. For each room type, we use 20 scenes for training, 5 for validation and 5 for testing (a total of 120 scenes).

We choose a subset of target object classes as our navigation targets such that (1) they are not hidden in cabinets, fridges, etc., (2) they are not too large that they take a big portion of the room and are visible from most parts of the room (e.g., beds in bedrooms). We choose the following sets of objects for each type of room: 1) Living room: pillow, laptop, television, garbage can, box, and bowl. 2) Kitchen: toaster, microwave, fridge, coffee maker, garbage can, box, and bowl. 3) Bedroom: plant, lamp, book, and alarm clock. 4) Bathroom: sink, toilet paper, soap bottle, and light switch.

The target object classes for train, test and validation sets are the same, but the object instances are unique in terms of appearance, geometry and texture.

We consider the actions $\mathcal{A} = \{\text{MoveAhead}, \text{RotateLeft}, \text{RotateRight}, \text{LookDown}, \text{LookUp}, \text{Done}\}$. Horizontal rotation occurs in increments of 45 degrees while looking up and down change the camera tilt angle by 30 degrees. Done corresponds to the termination

action discussed in Section 3.1. The agent successfully completes a navigation task if this action is issued when an instance from the target object class is within 1 meter from the agent’s camera and within the field of view. This follows from the primary recommendation of [2], and contrasts with previous works such as [50] where the agent receives the termination signal from the environment. Note that if the agent ever issues the `Done` action when it has not reached a target object then we consider the task a failure.

4.2. Implementation details

We train our method and baselines until the success rate saturates on the validation set. We train one model across all scene types with an equal number of episodes per type using 12 asynchronous workers. For \mathcal{L}_{nav} , we use a reward of 5 for finding the object and -0.01 for taking a step. For each scene we randomly sample an object from the scene as a target along with a random initial position. For our interaction-gradient updates we use SGD and for our navigation-gradients we use Adam [22]. For step size hyper-parameters (α, β_1, β_2 in Algorithm 1) we use 10^{-4} and for k we use 6. Recall that k is the hyper-parameter which prescribes the frequency of interaction-gradients. We experimented with a schedule for k but saw no significant improvement in performance.

For evaluation we perform inference for 1000 different episodes (250 for each scene type). The scene, initial state of the agent and the target object are randomly chosen. All models are evaluated using the same set.

4.3. Evaluation metrics

We evaluate our method on unseen scenes using both Success Rate and Success weighted by Path Length (SPL). SPL was recently proposed by [2] and captures information about navigation efficiency. Success is defined as $\frac{1}{N} \sum_{i=1}^N S_i$ and SPL is defined as $\frac{1}{N} \sum_{i=1}^N S_i \frac{L_i}{\max(P_i, L_i)}$, where N is the number of episodes, S_i is a binary indicator of success in episode i , P_i denotes path length and L_i is the length of the optimal trajectory to any instance of the target object class in that scene. We evaluate the performance of our model both on all trajectories and trajectories where the optimal path length is at least 5. We denote this by $L \geq 5$ (L refers to optimal trajectory length).

4.4. Baselines

We compare our models with the following baselines:

Random agent baseline. At each time step the agent randomly samples an action using a uniform distribution.

Nearest neighbor (NN) baseline. At each time step we select the most similar visual observation (in terms of Euclidean distance between ResNet features) among scenes in training set which contain an object of the class we are

	All		$L \geq 5$	
	SPL	Success	SPL	Success
Random	2.94	7.80	0.07	0.16
NN	6.09	7.90	1.38	1.66
No Adaptation (A3C)	13.73	32.90	10.88	20.66
Ours - prediction loss	14.20	39.50	11.89	27.40
Ours - diversity loss	14.95	39.20	12.76	27.40
Ours - SAVN	16.13	42.10	13.19	30.54

Table 1. **Quantitative results of navigation.** We compare variations of our method with random, nearest neighbor and non-adaptive baselines. We consider two evaluation metrics, Success Rate and SPL. We provide results for all targets ‘All’ and a subset of targets whose optimal trajectory length is greater than 5.

searching for. We then take the action that is optimal in the train scene when navigating to the same object class.

No adaptation (A3C) baseline. The architecture for the baseline is the same as ours, however there is no interaction-gradient and therefore no interaction loss. The training objective for this baseline is then $\min_{\theta} \sum_{\tau \in \mathcal{T}_{\text{train}}} \mathcal{L}_{\text{nav}}(\theta, \mathcal{D}_{\tau})$ which is equivalent to setting $\alpha = 0$ in Equation (4). This baseline is trained using A3C [30].

4.5. Results

Table 1 summarizes the performance of our approach and the baselines. We consider three variations of our method, which include SAVN (learned self-supervised loss) and the hand-crafted prediction and diversity loss alternatives.

Our learned self-supervised loss outperforms all baselines by a large margin in terms of both success rate and SPL metrics. Most notably, we observe about 10% absolute improvement in success and 2.5 in SPL over the non-adaptive (A3C) baseline. The self-supervised objective not only learns to navigate more effectively but it also learns to navigate efficiently.

The models trained with hand-crafted exploration losses outperform our baselines by large margins in success, however, the SPL performance is not as impressive as with the learned loss. We hypothesize that minimizing these hand-crafted exploration losses are not as conducive to efficient navigation.

Failed actions. We now explore a behavior which sets us apart from the non-adaptive baseline. In the beginning of an episode the agent looks around or explores the free space in front of it. However, as the episode progresses, the non-adaptive agent might issue the termination action or get stuck. Our method (SAVN), however, learns to adapt and thus exhibits this pattern less frequently.

To examine this behavior we compute the ratio of actions which fail. Recall that an agent’s action has failed if two consecutive frames are sufficiently similar. Typically, this will occur if an agent collides with an object. As shown in Figure 4, our method experiences significantly fewer failed

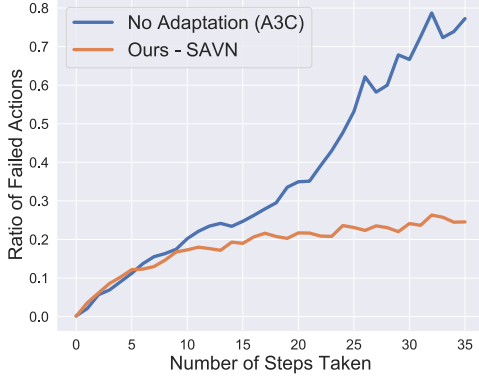


Figure 4. **Failed actions.** Our approach learns to adapt and not to take unsuccessful actions as the navigation proceeds.

actions than the baseline as the episode progresses.

Qualitative examples. Figure 3 qualitatively compares our method with the non-adaptive (A3C) baseline. In scenario (a) our baseline gets stuck behind the box and tries to move forward multiple times, while our method adapts dynamically and finds the way towards the television. Similarly in scenario (c), the baseline tries to move towards the lamp but after bumping into the bed 5 times and rotating 9 times, it issues Done in a distant location from the target. The supplementary video at the following link includes more examples: <https://youtu.be/-Ba6ZRMcxEE>.

4.6. Ablation Study

In this section we perform an ablation on our methods to gain further insight into our result.

Adding modules to the non-adaptive baseline. In Table 2 we experiment with the addition of various modules to our non-adaptive baseline. We begin by augmenting the baseline with an additional memory module which performs self-attention on the latest $k = 6$ hidden states of the LSTM. SAVN outperforms the memory-augmented baseline as well.

Additionally, we add the prediction loss detailed in Section 3.4 to the training objective. This experiment reveals that our result is not simply a consequence of additional losses. By using our training objective with the added hand-crafted prediction loss (referred to as ‘Ours - prediction loss’), we outperform the baseline non-adaptive model with prediction (referred to as ‘A3C w/ prediction loss’) by 4.7% for all trajectories and 5.8% for trajectories of at least length 5 in terms of success rate. As discussed in the Section 4.5, minimizing the hand-crafted objectives during the episode may not be optimal for efficient exploration. This may be why we show a boost in SPL for trajectories of at least length 5 but not overall. We run the same experiment with the diversity loss but find that the baseline model is unable to converge with this additional loss, as it may enter a bad local minimum.

	All		$L \geq 5$	
	SPL	Success	SPL	Success
No Adaptation (A3C)	13.73	32.90	10.88	20.66
A3C w/ mem	15.54	34.30	11.21	20.50
A3C w/ prediction loss	14.95	34.80	10.94	21.60
Ours - prediction loss	14.20	39.50	11.89	27.40
Ours - SAVN	16.13	42.10	13.19	30.54
A3C (GT obj)	31.34	44.40	16.77	26.05
Ours - SAVN (GT obj)	35.55	54.40	23.47	37.87

Table 2. **Ablation results.** We compare our approach with the non-adaptive baseline augmented with memory and our hand-crafted loss. We also provide the result when we use ground truth object information (bottom two rows).

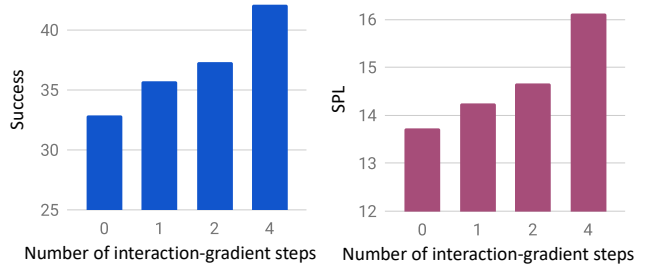


Figure 5. **Number of Gradients Ablation.** Our performance increases as more interaction-gradients are taken during training/inference.

Ablation of the number of gradients. To showcase the efficacy of our method we modify the number of interaction-gradient steps that we perform during the adaptation phase during training and inference. As discussed in Section 3.5, we never perform more than 4 interaction-gradients due to computational constraints. As illustrated by Figure 5, there is a clear increase in performance when more gradient updates are used, demonstrating the importance of the interaction-gradients.

Perfect object information. Issuing the termination action at the correct location plays an important role in our navigation task. We observe that SAVN still outperforms the baseline even when the termination signal is provided by the environment (referred to as ‘GT obj’ in Table 2).

5. Conclusions

We introduce a self-adaptive visual navigation agent (SAVN) that learns during both training and inference. During training the model learns a self-supervised interaction loss that can be used during inference when there is no supervision. Our experiments show that this approach outperforms non-adaptive baselines by a large margin. Furthermore, we show that the learned interaction loss performs better than hand-crafted alternatives. Additionally, we find that SAVN navigates more effectively than a memory-augmented non-adaptive baseline. We conjecture that this idea may be applied in other domains where the agents may learn from self-supervised interaction.

Acknowledgements: We would like to thank Marc Millstone and the AI2 Beaker team for providing a robust experiment platform and providing tremendous support. We also thank Luca Weihs and Eric Kolve for the discussions, and Chelsea Finn for her valuable suggestions.

References

- [1] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. In *ICCV*, 2015. 2
- [2] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir. On evaluation of embodied navigation agents. *arXiv*, 2018. 7
- [3] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Stnderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018. 2
- [4] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *ICRA*, 2010. 2
- [5] S. Brahmbhatt and J. Hays. Deepnav: Learning to navigate large cities. In *CVPR*, 2017. 2
- [6] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumathi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2018. 2
- [7] C. Chen, A. Seff, A. L. Kornhauser, and J. Xiao. Deep-driving: Learning affordance for direct perception in autonomous driving. In *ICCV*, 2015. 2
- [8] I. Clavera, A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt: Meta-learning for model-based control. *arXiv*, 2018. 2
- [9] M. Cummins and P. Newman. Probabilistic appearance based navigation and loop closing. In *ICRA*, 2007. 2
- [10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 3
- [11] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015. 2
- [12] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. rl^2 : Fast reinforcement learning via slow reinforcement learning. *arXiv*, 2016. 2
- [13] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 2, 4
- [14] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. *arXiv*, 2018. 2
- [15] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017. 2
- [16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3
- [17] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. M. Czarnecki, M. Jaderberg, D. Teplyashin, M. Wainwright, C. Apps, D. Hassabis, and P. Blunsom. Grounded language learning in a simulated 3d world. *arXiv*, 2017. 2
- [18] R. Houthoofd, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel. Evolved policy gradients. *arXiv*, 2018. 2
- [19] D. Jayaraman and K. Grauman. Learning image representations tied to ego-motion. In *ICCV*, 2015. 2
- [20] G. Kahn, A. Villafior, B. Ding, P. Abbeel, and S. Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *ICRA*, 2018. 2
- [21] K. Kidono, J. Miura, and Y. Shirai. Autonomous visual navigation of a mobile robot using a human-guided experience. *Robotics and Autonomous Systems*, 2002. 2
- [22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 7
- [23] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 2, 6
- [24] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. A. Fischler, and B. P. Gerkey. Outdoor mapping and navigation using stereo vision. In *ISER*, 2006. 2
- [25] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Learning to generalize: Meta-learning for domain generalization. In *AAAI*, 2018. 2, 4
- [26] L. H. Matthies and S. A. Shafer. Error modeling in stereo navigation. *IEEE J. Robotics and Automation*, 1987. 2
- [27] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017. 2, 3
- [28] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018. 2
- [29] D. K. Misra, J. Langford, and Y. Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017. 2
- [30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 7
- [31] A. Mousavian, A. Toshev, M. Fiser, J. Kosecka, and J. Davidson. Visual representations for semantic target driven navigation. In *ECCV Workshop on Visual Learning and Embodied Agents in Simulation Environments*, 2018. 2
- [32] A. Owens, P. Isola, J. H. McDermott, A. Torralba, E. H. Adelson, and W. T. Freeman. Visually indicated sounds. In *CVPR*, 2016. 2
- [33] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 2
- [34] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016. 2
- [35] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014. 3
- [36] L. Pinto, D. Gandhi, Y. Han, Y. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In *ECCV*, 2016. 2

- [37] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018. 2
- [38] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 1997. 2
- [39] B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv*, 2015. 2
- [40] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 1998. 2
- [41] S. Thrun and L. Pratt. *Learning to Learn: Introduction and Overview*. 1998. 2
- [42] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015. 2
- [43] Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *ICLR*, 2017. 2
- [44] Y. Wu, Y. Wu, A. Tamar, S. J. Russell, G. Gkioxari, and Y. Tian. Learning and planning with a semantic model. *arXiv*, 2018. 2
- [45] T. Xu, Q. Liu, L. Zhao, and J. Peng. Learning to explore with meta-policy gradient. In *ICML*, 2018. 2
- [46] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors. *arXiv*, 2018. 2
- [47] H. Yu, H. Zhang, and W. Xu. Interactive grounded language acquisition and generalization in a 2d world. In *ICLR*, 2018. 2
- [48] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. In *RSS*, 2018. 2, 4, 5
- [49] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *ECCV*, 2016. 2
- [50] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 2, 3, 7