THDA: Treasure Hunt Data Augmentation for Semantic Navigation

Oleksandr Maksymets[1*] Vincent Cartillier[2] Aaron Gokaslan[1,4] Erik Wijmans[2] Wojciech Galuba[1]
Stefan Lee[3] Dhruv Batra[1,2]
[1]Facebook AI Research [2]Georgia Institute of Technology [3]Oregon State University [4]Cornell University
Project Webpage: http://thda.maksymets.com

Abstract

Can general-purpose neural models learn to navigate? For PointGoal navigation ('go to $\Delta x, \Delta y$'), the answer is a clear 'yes' – mapless neural models composed of task-agnostic components (CNNs and RNNs) trained with large-scale model-free reinforcement learning achieve near-perfect performance [27]. However, for ObjectGoal navigation ('find a TV'), this is an open question; one we tackle in this paper. The current best-known result on ObjectNav with general-purpose models is 6% success rate [25].

First, we show that the key problem is overfitting. Large-scale training results in 94% success rate on training environments and only 8% in validation. We observe that this stems from agents memorizing environment layouts during training – sidestepping the need for exploration and directly learning shortest paths to nearby goal objects. We show that this is a natural consequence of optimizing for the task metric (which in fact penalizes exploration), is enabled by powerful observation encoders, and is possible due to the finite set of training environment configurations.

Informed by our findings, we introduce Treasure Hunt Data Augmentation (THDA) to address overfitting in ObjectNav. THDA inserts 3D scans of household objects at arbitrary scene locations and uses them as ObjectNav goals – augmenting and greatly expanding the set of training layouts. Taken together with our other proposed changes, we improve the state of art on the Habitat ObjectGoal Navigation benchmark by 90% (from 14% success rate to 27%) and path efficiency by 48% (from 7.5 SPL to 11.1 SPL).

1. Introduction

Consider an agent given a task such as *'Bring me a teapot'* in a new environment. In order to successfully perform the command, it first must navigate around the environment to find the teapot. This search subtask is referred to as ObjectNav [1, 3] and is illustrated in Fig. 1.

In this work, we examine if general-purpose neural models learn to navigate when given their goal specified as an object name. Specifically, models that are composed of navigation-agnostic general architectural components (CNNs, RNNs, fully-connected layers, *etc*.) and trained under an experimental setup that provides no inductive bias towards how humans believe this problem should be solved *e.g*. no map-like or spatial structural components in the agent, no mapping supervision, no auxiliary tasks – nothing other than the task of navigation to a goal object. We find this question interesting both from a scientific perspective (what *are* the fundamental limits of learnability?) and an engineering perspective (general-purpose architectures are widely applicable across tasks and any advances made are likely to have significant ripple effects).

In recent work studying PointGoal Navigation (or navigating to a relative waypoint), task-agnostic neural models trained with large-scale model-free reinforcement learning (RL) achieve near-perfect performance [27]. We find however that similar models are unable to achieve even non-trivial performance in unseen environments when applied directly to ObjectNav. We show that the key problem is extreme overfitting – even in the presence of standard tricks like early stopping. Large-scale training results in 97% success rate in training environments and only 8% success unseen environments from validation.[1]

We identify three key reasons for this poor generalization and demonstrate techniques to reduce their effect:

1) Overly rich sensors. Given that ObjectNav relies on semantic information, agents are often provided observations in the form of RGB-D images and a semantic segmentation of the current frame (via either ground-truth or a separately trained network). However, we argue that this sensor suite is 'too rich' for the task, *i.e*. makes memorizing environments

*Correspondence to maksymets@facebook.com

[1]We encourage the reader to fight hindsight bias at this point – it is *not at all obvious* that general-purpose neural networks trained with model-free reinforcement learning *should* be able to achieve such high performance on a large number of diverse training environments. We could very well have found ourselves in a world where both training and validation performance was poor, with approximation or optimization error to blame.

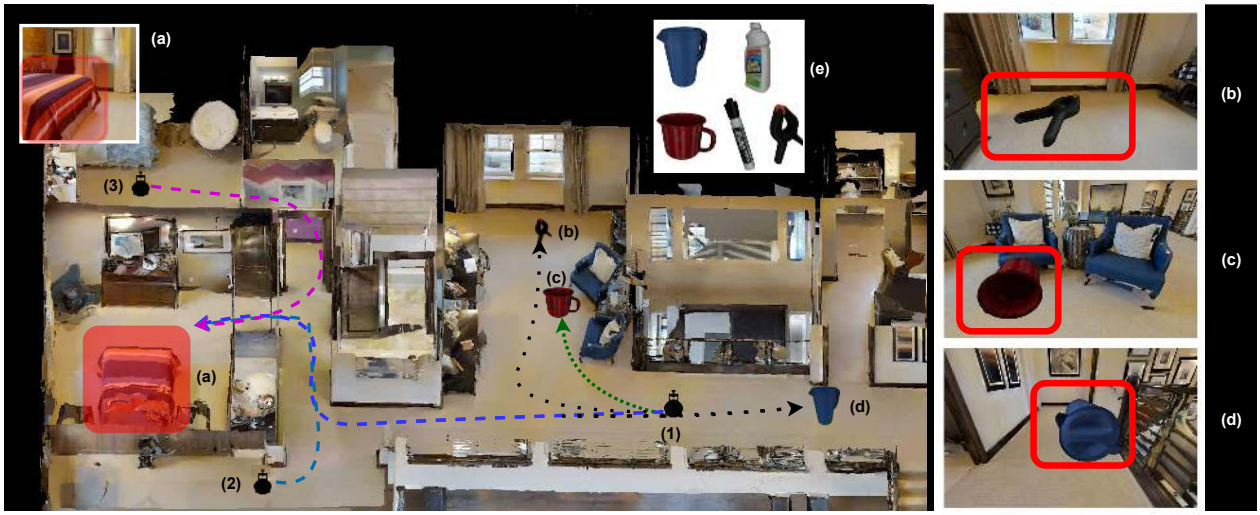

Figure 1: In ObjectNav, an agent is spawned (at locations 1,2,3) and asked to find the 'bed', shown in (a). We introduce the idea of Treasure Hunt Data Augmentation (THDA), where we insert common household objects from the YCB dataset (shown in (e)) at random locations in the house (b,c,d) and ask the agent to find them. In the original ObjectNav dataset (left half of figure), the agent's spawn position varies (1−3), but the goal location does not. In THDA, both the robot spawn position and the goal location changes. This increases the training data significantly by expanding the diversity of the (starting_position, target_category, target_position) tuples in the training data. We demonstrate empirical that this improve validation performance significantly on unseen scenes.

and goals during training easy. We empirically find that limiting the agent to a minimal sensor suite of a `Depth` sensor and a segmentation mask of the goal category reduces overfitting (when combined with our other ideas below).

2) Mismatch between reward and necessary behavior at inference. ObjectGoal navigation is fundamentally about exploration – when an agent is put into an unseen environment, there are few priors it can rely on for finding the object and thus it must explore. The typical reward used for training navigation agents encourages them to reach their goal as quickly as possible (called 'slack penalty'). Unfortunately, the direct implication is that this reward *penalizes* exploration. Over a finite set of training environments, we argue that this both encourages memorization and fails to teach the agent how to explore. We propose a reward called `ExploreTillSeen` where the agent is initially rewarded for exploring the environment and then for navigating to a target object as quickly as possible *after* seeing it.

3) Limited training data. Even with reduced sensors and a exploration-promoting reward, the fundamental challenge of data scarcity remains. Datasets for ObjectNav depend on large 3D scenes with high quality semantic annotations which are difficult to collect. We introduce Treasure Hunt Data Augmentation as a way to combat this. THDA inserts objects into existing 3D environments to generate synthetic ObjectNav episodes to augment the training set. This idea is illustrated in Fig. 1 with inserted objects mug, pliers, jug.

By addressing these causes of overfitting, we are able to train an agent that generalizes better to novel scenes.

Contributions. We summarize our contributions below:

– We significantly improve the performance of a simple mapless model-free RL baseline on the challenging Habitat ObjectNav benchmark – from 6.2% [25] to 21.2% success rate. We also demonstrate significant improvements on path efficiency – from 2.1% SPL to 7.7% SPL. Even without THDA, this already sets a new state of art on the task.

– We propose an effective `ExploreTillSeen` reward function for the ObjectNav task that combines exploration and distance to target rewards with the object identified reward.

– We introduce Treasure Hunt Data Augmentation (THDA) that is highly effective for pre-training the improved RL baseline, further improving the start of art from 21.2% success (7.7% SPL) to 27.4% success (11.1% SPL). All three of our ideas together result in a relative improvement in the state of art by +48% SPL and +90% Success. In fact, THDA shows competitive results with prior state of art even in a zero-shot setting, *i.e.* without *any training on the target dataset* used by the challenge.

2. Related work

Simulators and datasets for ObjectGoal navigation. The challenging task of object-goal navigation has received a lot of interest in the recent years. Photorealistic 3D indoor environments [29, 7, 18] are well suited for ObjectNav because they capture the distribution of layouts of real houses with a high-quality visual rendering, which might be needed for sim2real transfer [3]. We choose the Habitat-Challenge framework [3] to focus on multi-room navigation in multi-

floor photo-realistic scenes.

ObjectNav policy with explicit spatial representations. Some of the recent learning-based approaches for object-goal navigation construct and update an intermediate representation from observations which is then used for planning. This includes methods using a mapping module such as [8, 24] which use simultaneous localization and mapping (SLAM) to construct a map. Similarly, [13] learns a mapping strategy to construct a belief map of the world and use it for planning. More recently, [9, 26] use a semantic mapping module to learn semantic priors. In addition, other methods use topological representations [28, 30]. In [30], semantic priors are incorporated into a deep RL framework by using Graph Convolutional Networks. Whereas [28], learns a probabilistic graphical model to capture the semantic properties of houses. Furthermore, [5] stores previous joint encoded observations (including semantic features) in an unstructured memory. A transformer network is used on top of that memory for planning. In all the above methods, intermediate representations are used for planning. In this work, the learnt navigation policy is not constrained to a specific representation and does not rely on any motion or task planning. Instead, the agent learns to act directly from its egocentric observations. The resulting policy is faster and more generic, and does not suffer from errors accumulated from intermediate modules.

Augmentation by Inserting Objects. Prior work has examined object insertion as a data augmentation technique for instance segmentation in 2D images [10, 11] and in 3D object detection [22]. Likewise in this work, we insert objects into large 3D scenes in order to create new episodes for pretraining model-free reinforcement learning agents for semantic navigation.

3. Preliminaries: ObjectNav & Agents

We begin by detailing the ObjectNav task, agent specifications, and common model-free visual navigation agents.

3.1. ObjectNav Task Description

Objects as Navigation Goals. In ObjectNav [2, 3], an agent is spawned in a never-before-seen environment and tasked with finding an instance of a specified object class – akin to simple instructions like *'Find a chair'*.

We focus on the Habitat Challenge ObjectNav benchmark. This dataset provides ObjectNav episodes in large, photorealistic, simulated environments from the Matterport3D dataset [7]. These episodes consist of a start location for the agent, a target class, and ground truth locations for target object instances. Each episode requires navigating across multiple rooms (3-27 meter shortest-paths) as environments in the Matterport3D dataset are typically large. Rarely ever is the target object visible from the spawn location, with the average distance of the nearest target object from spawn being approximately 8m. The target object is almost surely always in a different room than the agent. As such, agents must explore the environment to spot a target object instance before invoking a 'stop' action.

Agent Specification. Following [3], our simulated agent is designed to match the height and radius of the Locobot robot (http://www.locobot.org/) with a simple action space of move forward 0.25m, turn right/left by 30 degrees, look up/down by 30 degrees, and stop. We dropped the look up/down actions from our policy, after observing that their exclusion accelerates training in our initial experiments.

Evaluating Success and Path Efficiency. An agent trajectory is considered successful if the agent invokes the 'stop' action within 1m of a target object instance and with direct line-of-sight to it. Note that the agent does not need to be looking at the object, merely that it is possible to do so from the terminal location [3]. Beyond success, agents are also evaluated based on their efficiency relative to an optimal oracle agent that moves along the shortest path to the nearest target object instance. This is captured via the Success weighted by inverse Path Length (SPL) metric [2]. Formalizing this for an episode, let L denote the oracle shortest-path length, P the agent's path length, and S be a binary indicator whether the agent succeeded. SPL is then defined as $S \cdot (L/\max(P, L))$ and is bounded in $[0, 1]$ where 1 (or 100%) reflects optimal performance. Both success and SPL are reported as averages over episodes in the dataset.

As ObjectNav requires a degree of search, achieving an SPL of 1 seems infeasible for both humans and agents without oracle information. Simply put, without knowing the location of the object a priori, moving towards it along a shortest path consistently is not possible over many episodes. In later sections, we will demonstrate that naively optimizing for this metric encourages overfitting during training.

3.2. Common Model-Free Navigation Agents

Generic agent architectures that directly map observations into low-level actions have shown near-perfect performance on navigating to relative point goals in indoor environments [27]. However, their application to ObjectNav has not shown similar results – with agents failing to generalize from training. These 'vanilla' agents form our starting point for analysis in this work and are described below.

Agent Architecture. We follow [27] and consider agents composed of two main components – a visual encoder and a recurrent policy network. For the vanilla model, the observation consists of an RGB-D frame from a forward-facing camera, the agent's position / heading in coordinates relative to its spawn configuration, and an encoding of the target object class. The policy network outputs a categorical distribution over the set of actions.

For RGB-D, the visual encoder is a modified ResNet-

50 [14] architecture with the number of output channels at every layer reduced by half. We use a first layer of `2x2-AvgPool` to reduce resolution (essentially performing low-pass filtering + downsampling) same as in [27]. In later experiments, we will also consider augmenting the input with semantic segmentation of the scene.

The recurrent policy network is parameterized by a 2-layer LSTM with a 512-dimensional hidden/cell state. It takes four inputs: the previous action, the object goal category embedding, the agent's position and heading relative to its initial pose, and the output of the visual encoder. The LSTM's output is used to produce a softmax distribution over the action space and an estimate of the value function for actor-critic style training – both are computed as linear functions of the current hidden state.

Training Procedure. We use PPO with Generalized Advantage Estimation [21]. We set the discount factor γ to 0.99 and the GAE parameter τ to 0.95. Each worker collects (up to) 64 frames of experience from 6 agents running in parallel (all in different environments) and then performs 4 epochs of PPO with 2 mini-batches per epoch. We use Adam [17] with a learning rate of 2.5×10^{-4} without normalized advantages. To parallelize training, we use Distributed Decentralized PPO (DD-PPO) [27] to train with 256 workers on 256 GPUs.

Reward. A natural first choice is to use the shaped navigation reward commonly used for PointGoal Navigation [20, 27] which optimizes path efficiency. This shaped navigation reward consists of a change in geodesic distance term ($\Delta_{\text{geo\_dist}}$) that encourages agents to make progress towards the goal and a large reward for successful episodes (r_{succ}). Specifically the reward at time t when the agent is in state s_t and executes action a_t is

$$r_t(s_t, a_t) = \begin{cases} S \cdot r_{\text{succ}} & \text{If } a_t \text{is stop} \\ -\Delta_{\text{geo\_dist}} + r_{\text{slack}} & \text{otherwise} \end{cases} \quad (1)$$

where $\Delta_{\text{geo\_dist}}$ is the change in geodesic distance to goal after executing action a_t in state s_t, S is a binary indicator of success, and r_{slack} is a slack penalty that encourage the agent to quickly find its goal. It is easy to see that optimizing this reward also optimizes SPL – an agent should successfully reach the goal in as few actions as possible. This reward works well for PointGoal Navigation and thus it has also been adapted to ObjectNav by calculating the geodesic distance with respect to the closest object [9, 20]. We refer to this reward as **Dense Path-Efficiency** throughout the paper, since it is a dense reward and encourages path efficiency.

4. Overfitting in ObjectNav

In this section we analyse the performance of the vanilla task-agnostic agent and training regime described above in ObjectNav. We find this paradigm performs poorly – corroborating the results demonstrated in Chaplot *et al*. [9] and the Habitat challenge leaderboard [25]. Digging deeper, we identify a specific problem – the agent achieves very high performance during training but fails to generalize.

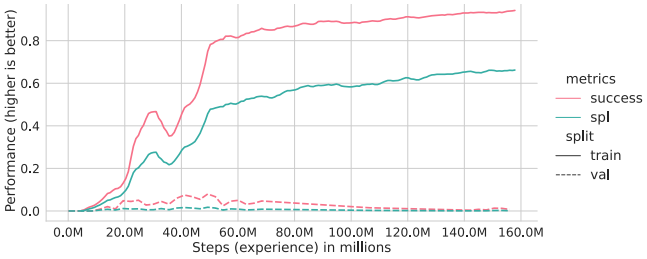

Figure 2: Training and validation metrics as a function of training experience for an ObjectNav policy with input of `RGB-D` showing strong overfitting behavior. Note the **significant** gap between train and validation performance.

We begin with the vanilla agent taking only RGB-D as visual input. This agent is trained with the dense path-efficiency reward and the training settings described above for 150 million steps. Figure 2 shows SPL and Success during training for both train and val splits. While training performance rises to high levels – reaching success rates of 94% – the performance on val scenes fails to rise – maxing out at 8%. With reduced encoder capacity from ResNet50 to ResNet18 it achieves even higher success 98% on train, while only 7.7% on val split.

This overfitting becomes even more evident when examining path efficiency. The agent reaches path efficiency (as measured by SPL) of around 66% on train – suggesting the agent has simply memorized training environments. To achieve such high path efficiency, we conjecture that the agent might be recognizing its location at spawn and then moving directly to a memorized object location of the specified class. We see examples of this behavior in Figure 3 where an agent spawns in locations where it cannot see the nearest target (*'toilet'*) many rooms away, but moves directly to the target anyway following a near-shortest path on episode (3). Notice the absolute lack of any search, backtracking, or switch-back behavior – this strongly suggests that the agent has memorized this path.

In the following sections, we will argue that this is a natural consequence of optimizing for the path-efficiency reward, and is further exacerbated by rich visual encoders and the small number of training environments.

5. Mid-Level Visual Representation

From this initial experiment, we confirm that `RGB-D`-equipped agents overfit significantly during training by memorizing the training episodes. Recent work in PointGoal navigation has shown that the richness of RGB observations can be a source of this sort of poor generalization in point-goal navigation tasks [20, 12] and it is common to use agents based only on depth images to avoid overfitting.

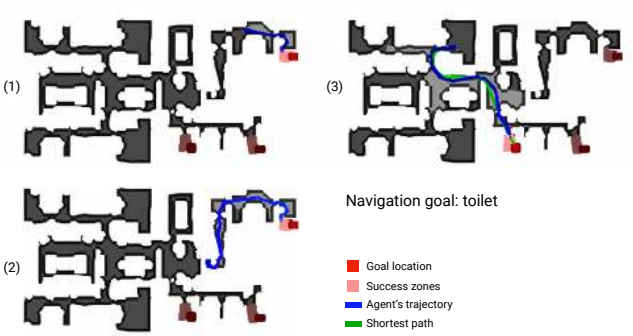

Figure 3: Example paths in **train** environments for the baseline RGBD agent. The agent is spawned multiple rooms from the goal object yet manages to move in near-shortest paths to the closest goal – providing strong evidence of memorization.

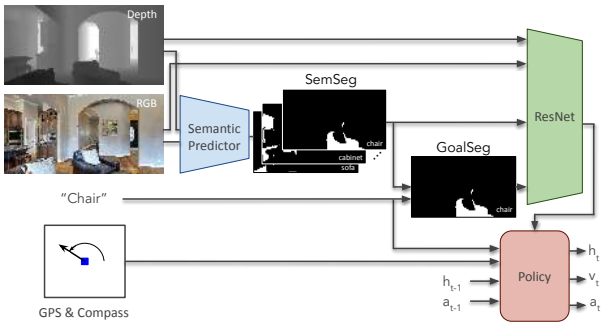

Figure 4: Architecture of a vanilla RL ObjectGoal Navigation Policy using the set of visual inputs `RGB-D` + `SemSeg` + `GoalSeg`, the target object and GPS + Compass. In all our experiments, ablations are made to the visual encoder by restricting the ResNet inputs to certain modalities.

However, in ObjectNav identifying target object instances through depth images alone is a challenging task. In our initial experiments with depth-only models, we found slow convergence on train and poor generalization.

As such, we consider removing RGB but augmenting our models with semantic segmentation masks produced by an intermediate model. We use a pre-trained semantic segmentation model (RedNet [16]) to predict semantic segmentation over the 21 goal categories. The agent is provided these masks by augmenting the depth mask with these additional 21 channels before encoding with the visual encoder. We denote this visual observation as `SemSeg`. Furthermore, we dedicate an additional channel to the semantic segmentation corresponding to the goal category (`GoalSeg`) – effectively duplicating one of the `SemSeg` channels – for a total of 23 input channels. We denote this model as `Depth +SemSeg +GoalSeg`. We train a `Depth +SemSeg +GoalSeg` agent with the dense path-efficiency reward described previously. The results of this model and our baseline `RGB-D` agent are shown in Tab. 1. We find our `Depth +SemSeg +GoalSeg` reduces overfitting somewhat – achieving a 10.2% success rate in validation (vs 8.0% for the `RGB-D` baseline).

Visual Input	Train				Val			
	SPL (%) ↑	SoftSPL (%) ↑	DTS (m) ↓	SR (%) ↑	SPL (%) ↑	SoftSPL (%) ↑	DTS (m) ↓	SR (%) ↑
`RGB-D`	66.2	67.3	0.171	94.2	1.8	13.4	6.9	8.0
`Depth +SemSeg +GoalSeg`	**68.7**	**69.7**	**0.149**	**95.7**	3.3	12.0	6.1	10.2

Table 1: Performance comparison of agents trained with different input modalities under the standard path-efficiency reward. The `Depth +SemSeg +GoalSeg` model shows to slightly reduce overfitting and will be considered for the remaining experiments.

We use this `Depth +SemSeg +GoalSeg` model as our base for the remaining experiments. As we will discuss in Sec. 7, this use of mid-level semantic features makes synthetic data augmentation straightforward by lowering the need for fully-realistic object insertion. We also note that this type of modular architecture separating semantic perception from control has been examined in the context of sim2real transfer for ObjectNav with positive results [9].

6. `ExploreTillSeen` Reward

As described in Sec. 3, the standard dense path-efficiency reward is maximized by an agent that navigates directly along a shortest path to the nearest target object instance. In new environments, this is infeasible and an agent must learn to explore in order to achieve high reward. As we demonstrated in Sec. 4 however, memorizing the environments is an alternative highly-rewarded behavior if operating in a small training set. For large models, memorizing all 61 environments in ObjectNav is certainly plausible; however, such an agent does not learn to explore and does not generalize. We note that given infinite environments during training, such a reward is not necessarily degenerate.

We would like an agent that learns to efficiently explore and then moves directly to discovered target object instances after they are first observed. To encourage this behavior, we devise an `ExploreTillSeen` reward for this task that explicitly breaks the agent trajectory into two subtasks – exploration/search and navigation to found/visible target objects. Initially the agent is rewarded for simply exploring the environment – given positive feedback for increasing the fraction of the environment it has observed. Once the agent sees an instance of the target class, the agents receives a significant positive reward and then the reward switches to a shaped path-efficiency promoting navigation reward.

Formalizing the `ExploreTillSeen` reward, let $\Delta_{\%\_\text{scene\_explored}}$ be the change in the fraction of environment area observed by the agent when taking action a_t in state s_t. We calculate this by projecting the agent's viewing frustum onto a 2D floor map of the environment at each time step and keeping track of the fraction of area

	Method	Train				Val			
		SPL (%) ↑	SoftSPL (%) ↑	DTS (m) ↓	Success (%) ↑	SPL (%) ↑	SoftSPL (%) ↑	DTS (m) ↓	Success (%) ↑
1	Dense Path-Efficiency	**68.7**	**69.7**	**0.149**	**95.7**	3.3	12.0	6.067	10.2
2	Sparse Success	0.0	0.0	6.988	0.0	0.0	0.0	7.843	0.0
3	Sparse Success + Exploration	48.0	49.9	1.925	78.9	3.4	**13.0**	6.561	10.8
4	`ExploreTillSeen`	58.5	59.3	1.752	85.0	**4.8**	12.2	**6.242**	**14.5**
5	`ExploreTillSeen` (`Depth` + `GoalSeg`)	47.8	50.3	1.559	79.2	**6.5**	**13.9**	6.243	**20.0**
6	`ExploreTillSeen` (`Depth` + GT `GoalSeg`)	74.7	74.8	0.285	97.2	20.0	24.9	6.776	40.7

Table 2: Comparison of different reward functions and input modalities. Rows (1-4) are `Depth` +`SemSeg` +`GoalSeg` models trained under different reward functions. The `Depth` +GT `GoalSeg` model uses ground truth segmentations at inference and acts as an upper bound on `Depth` + `SemSeg`. We find in these experiments that ablating the `SemSeg` from input modalities further reduces overfitting.

observed at least once. Further let $r_{\text{goal\_seen}}$ be a constant reward for the first observation of a target object instance. To define if the goal object was seen we check ground truth goal segmentation occupies $> 3\%$ of the frame. Formally:

$$r_t = r_{\text{slack}} + \begin{cases} r_{\text{expl}} \Delta_{\%\_\text{scene\_explored}} & \text{Until goal seen} \\ r_{\text{goal\_seen}} & \text{First time goal seen} \\ -\Delta_{\text{geo\_dist}} + S \cdot r_{\text{succ}} & \text{otherwise} \end{cases} \tag{2}$$

where r_{expl} is a scaling factor for the exploration reward and $\Delta_{\text{geo\_dist}}$ and r_{succ} are defined as in Eq. (1). In practice, we use values $r_{succ} = 10$, $r_{slack} = -0.01$, $r_{goal\_seen} = 3$, and $r_{expl} = 25$. Note that the reward switches to navigation once the agent sees the *first* instance of an object category, but we use all object instances in the environment when computing distance to goal. This is to handle cases like the agent navigating to an instance closer than the one observed first or if the instance it saw is not easily reachable.

We note that this reward too may result in a degenerate solution on a finite training set – memorizing optimal exploration paths that intentionally *do not* observe an object instance until some the slack reward overtakes the exploration reward. However, we find this solution is more difficult for policies to achieve in practice and that the learned behavior still encourages exploration followed by a switch to goal-driven navigation once a target object is observed.

6.1. Reward Experiments

We train our `Depth` + `SemSeg` + `GoalSeg` agent under four different reward schemes:

1) **Dense Path-Efficiency.** The standard dense shaped reward used for navigation tasks explained in Section 4.
2) **Sparse Success.** While potential-difference-based rewards are known to not change the optimal policy or the ordering of policies [20, 27], they may change the order we find policies and using a sparse reward may cause us to find suboptimal policies that generalize. We also consider using a sparse reward with only the success term.
3) **Exploration Only.** The dense shaped coverage exploration reward and success term from `ExploreTillSeen` reward without the go-to-goal term.
4) **`ExploreTillSeen`**. The reward proposed here where the agent explores the environment and then navigates to the object once it encounters an instance.

While [20, 27] showed that longer training horizons are beneficial we usually observe saturation of performance on val for ObjectNav before 150 million steps. Therefore, we train agents for 150 million steps and report train / val performance by selecting checkpoints with the largest SPL for each (i.e. each row of Tab. 2 shows two model checkpoints).

As in Sec. 4, we find that the Dense Path-Efficiency reward agent overfits, with near-perfect success on train while around 10% on val (row 1). The sparse success reward agent is unable to learn at all (row 2). Adding an exploration reward to the sparse reward (row 3) enables learning and perform similarly to the dense path-efficiency agent. Finally, our `ExploreTillSeen` reward significantly reduces the gap between training and validation – gaining around 4% success and 4.8 SPL. This amounts to a +42% higher SPL and Success compared to the dense path-efficiency and exploration rewards. While `ExploreTillSeen` leads to improvements, there remains a large gap between the best checkpoints for train (85% success) and validation (14% success) – indicating significant overfitting is still occurring.

6.2. Revisiting Visual Representations

GoalSeg. After completing this battery of experiments, we considered whether `SemSeg` may still provides sufficiently rich representations of the environment to encourage overfitting. In Tab. 2 (row 5) we ablate `SemSeg`, keeping only depth and `GoalSeg` for models trained with `ExploreTillSeen`. This agent observes the world via depth and a sparse semantic segmentation channel corresponding to the goal object category. We find this ablation improves performance significantly from 14.5% to 20% success and from 4.8 to 6.5 SPL (+35%).

Headroom Analysis of Semantic Segmentation. To inves-

tigate the importance of semantic segmentation prediction accuracy, we train a Depth + GoalSeg model using ground truth goal category segmentation at both train and inference. In the last row of Tab 2, we see significant (+160%) boost in performance for both SPL and Success. This suggests improvements in semantic segmentation prediction can drive future improvements of ObjectNav task.

7. Treasure Hunt Data Augmentation

Despite the improvements from ExploreTillSeen and the GoalSeg observation space, the fundamental problem remains – the training set is finite and partially memorizable by our agents. The natural response then is to either regularize the agent or to increase the number and diversity of training data. As effectively regularizing deep networks for RL is difficult [15], we opt to explore data augmentation.

Generating entirely new photorealistic 3D environments along with correct semantic category annotations is an open research area. Instead, we synthesize new ObjectNav episodes by inserting new objects into existing environments in a data augmentation scheme we refer to as Treasure Hunt Data Augmentation (THDA). As demonstrated in Figure 1, we expand the finite set of goals in ObjectNav by inserting 3D object models into the simulated training environments at random locations. These objects may be goals themselves (extending the goal space of ObjectNav during training), or simply serve to increase layout diversity.

We hypothesize that such data augmentation will make memorization of object positions in the training dataset more difficult and favor agents that explore the environment until the goal object is observed. However, THDA inserts objects not considered in ObjectNav and the policy may learn incorrect biases. As such, we use it as a pretraining stage before finetuning on the unagumented dataset.

Inserted Objects. We use the YCB dataset objects [4] as inserted objects. We characterize YCB objects into 7 categories based on synset intersections from Wordnet [19]: 'game equipment', 'foodstuff', 'hand tool', 'kitchenware', 'plaything', 'stationery', 'fruit'. While YCB objects differ substantially from goal categories in ObjectNav (scissors vs chairs) THDA focuses on the exploration and navigation portions of the task. In initial experiments, we observed inserting large objects such as a furniture can make part of the scene unreachable – complicating goal sampling.

Episode Generation. The objects are placed in free navigable locations to avoid situations where an object is hard to get near to after being spawned in the middle of the table or on a bed. To avoid the ObjectNav problem devolving into "find the YCB object", we sample and insert up to five YCB objects per episode – one of which will be selected as the goal class. To replicate conditions from the original dataset when there can be several goal objects in the agent's view (like chairs) we insert each of these sampled objects at up to 3 different locations. As result, we can insert from 1 to 15 additional objects per episode.

The agent spawn position is sampled 1-20 meters away from the closest goal. In initial experiments, we find that including some short episodes is crucial for policy's learning. After the position sampling procedure, we check navigability of areas around the goal objects from the agent's spawn position and re-sample the objects and agent positions until navigable. Additionally, objects are scaled up by 5 times to be closer to objects scale in Matterport3D scenes.

Relaxed Success Criteria. To simplify episode generation and reward computation, we relax the success criteria during training. Rather than precomputing positions where the object would be visible as in ObjectNav, we simply consider an episode successful if the agent stops within $1m$ Euclidean distance to the goal object's center. This success criteria is slightly different than the one from ObjectNav (defined in Sec. 3), as the agent does not need to be able to see the object at the final position. Considering the size of THDA inserted objects, this criteria is more strict in terms of distance, but less in terms of visibility. Training regime, model and hyper-parameters remains the same as in Sec. 6.

THDA with GoalSeg. THDA synergies well with GoalSeg as the agent's visual input is agnostic to the specific goal category and the number of goal categories. Further, we do not need to consider the effects that adding an object has on lighting (*i.e.* shadows and reflections due to the added object or the effect of the scene's lighting on the inserted object) beyond training the semantic segmentation network.

7.1. THDA Experiments

We experiment with adding THDA as a pretraining step and Tab. 3 shows our results which we discuss below.

We first train a Depth + GoalSeg model with ExploreTillSeen reward using THDA from scratch for 56 million steps until it reaches a plateau with 0.45 SPL and 54% Success on train. We evaluate the model on the original ObjectNav dataset and it reaches 0.038 SPL and 11% Success *without ever been trained on MP3D goal object categories*. We denote this model as Depth + GoalSeg + THDA (zero-shot) in Tab. 3. We note that this is competitive with the original RGBD and Depth +SemSeg +GoalSeg models trained on ObjectNav train that we considered in Sec. 5. Further, these results already outperform prior model-free agents on this task [25].

To further analyze this model, we also generate a validation split for THDA with the same goal object set and procedure but in validation ObjectNav scenes. We observe that for a model reaching 0.45 SPL on train, validation performance drops relatively little to 0.41 SPL. This supports the hypothesis that the THDA policy trained is not overfitting to training scenes. With THDA, we face a domain adaptation

Method	Val			
	SPL (%) ↑	SoftSPL (%) ↑	DTS (m) ↓	Success (%) ↑
Depth + GoalSeg	6.5	13.9	6.243	20.0
Depth + GoalSeg + THDA + fine-tune	**11.0**	**18.9**	**5.585**	**28.4**
Depth + GoalSeg + THDA (zero-shot)	3.8	13.0	6.194	11.4

Table 3: Experiment results on the validation split for Depth + GoalSeg models with/without THDA pretraining.

Method	Test			
	SPL (%) ↑	SoftSPL (%) ↑	DTS (m) ↓	Success (%) ↑
Depth + GoalSeg + THDA (Ours)	**11.1**	**18.6**	**7.782**	**27.4**
Depth + GoalSeg (Ours)	7.7	14.4	7.770	21.2
SRCB-robot-sudoer	7.5	16.8	9.578	14.4
SemExp [9]	7.1	14.5	8.818	17.9
Habitat Team (RGBD+DD-PPO)	2.1	14.7	9.316	6.2

Table 4: Comparison of our experiment results with the state-of-the-art on test-standard split from Habitat Challenge Leaderboard.

problem rather than overfitting.

Given the pretrained Depth + GoalSeg + THDA (zero-shot) model, we then finetune on the original ObjectNav dataset. Validation performance peaks within 11 million steps. We denote the resulting agent as Depth + GoalSeg + THDA + fine-tune in Tab. 3. This model reaches 0.11 SPL and 28.4% Success on validation split – a +69% SPL and +42% Success relative improvement compare to Depth + GoalSeg alone. Further, this result is a more than 4x improvement over baselines from [25]. These results suggest that THDA is an effective data augmentation scheme for ObjectNav. We speculate that it may be a fairly general way to reduce overfitting in embodied problems that results from finite training environments and is not specific to RL.

Comparison on ObjectNav leaderboard. In Tab. 4 we compare performance of our fine-tuned Depth + GoalSeg + THDA agent on the Habitat ObjectNav 2020 Challenge. We observe a +48% relative improvement for SPL metric and +90% for Success compared to best-performing prior methods. These works integrate map building strategies, semi-analytic planning, and pre-trained object detectors.

To re-emphsize this result, our model lacks any architectural elements or inductive biases towards mapping, builds no map, and does not execute any planning with external algorithms. That we find our agent outperforming prior work using these tools is a surprising result that demonstrates the potential for general-purpose architectures. However, developing appropriate training methods to avoid overfitting in low-data regimes remains a significant hurdle. One fair criticism of this entire line of work may be that inductive biases derived from human knowledge have 'merely' been shifted from architectural design to reward and training design. One mitigating perspective is that our work may be viewed as providing as a kind of 'existence proof' – answering the question of how far general-purpose models may be scaled. The answer appears to be 'surprisingly far'.

		mIoU	mRecall	mPrecision	Acc
train	All Objects	44.02	93.61	45.84	81.66
	Goal-Objects	37.13	92.25	39.23	81.96
	YCB	65.69	97.89	66.61	99.17
val	All Objects	27.38	53.24	32.04	69.02
	Goal-Objects	15.93	39.45	21.65	69.01
	YCB	63.36	96.55	64.71	98.59

Table 5: Semantic segmentation module [16] performance on the Matterport dataset on the combination of the goal target categories of the ObjNav Challenge and the newly inserted YCB categories.

7.2. Semantic Segmentation Predictor for THDA.

To produce SemSeg and GoalSeg inputs, we predict Semantic Segmentation using using RedNet [16]. RedNet is a network designed for the task of semantic segmentation of indoor scenes and its architecture structure has proven to be effective for parsing indoor environments [6]. We finetune RedNet using publicly available pre-trained weights (learned on the SUN-RGBD dataset [23]) on our dataset. The predictor is trained to predict the sets of goal objects categories from the ObjectNav Challenge plus the new goal objects introduced in THDA leading to a total number of 29 categories (including background). The training data is generated by rendering images from randomly sampled view points in the Matterport3D houses [7] with YCB inserted objects [4]. Tab. 5 reports similar overfitting behavior with a significant drop in mIoU between train and validation sets: $37.13 > 15.93$ for the Goal-Objects and $44.02 > 27.38$ for all objects. The same YCB objects are inserted in train and val so segmentation performances are similar across splits.

8. Conclusions

In this paper, we investigate if task-agnostic neural models can learn to navigate to goals specified as object names. We show the primary failure mode is overfitting – specifically, the model memorizing the locations of goal objects in the training data. We propose a series of modifications, use of mid-level visual representation, a reward function that initially rewards exploration and then switches to ObjectNav, and introduce Treasure Hunt Data Augmentation to increase the amount of training data. As result, we observe a +48% relative improvement for SPL and +90% for Success compared to previous state-of-the-art.

Acknowledgements. The Georgia Tech effort was supported in part by NSF, AFRL, DARPA, ONR YIPs, ARO PECASE, Amazon. EW is supported in part by an ARCS fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

References

[1] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[2] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[3] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*, 2020.

[4] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[5] Tommaso Campari, Paolo Eccher, Luciano Serafini, and Lamberto Ballan. Exploiting scene-specific features for object goal navigation. *arXiv preprint arXiv:2008.09403*, 2020.

[6] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic mapnet: Building allocentric semanticmaps and representations from egocentric views. *arXiv preprint arXiv:2010.01191*, 2020.

[7] 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. *arXiv preprint arXiv:1709.06158*, 2017. MatterPort3D dataset license available at: http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf.

[8] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*, 2020.

[9] Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33, 2020.

[10] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[11] Hao-Shu Fang, Jianhua Sun, Runzhong Wang, Minghao Gou, Yong-Lu Li, and Cewu Lu. Instaboost: Boosting instance segmentation via probability map guided copy-pasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[12] Daniel Gordon, Abhishek Kadian, Devi Parikh, Judy Hoffman, and Dhruv Batra. Splitnet: Sim2sim and task2task transfer for embodied visual navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1022–1031, 2019.

[13] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] M. Igl, K. Ciosek, Yingzhen Li, Sebastian Tschiatschek, C. Zhang, S. Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *NeurIPS*, 2019.

[16] Jindong Jiang, Lunan Zheng, Fei Luo, and Zhijun Zhang. Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation. *arXiv preprint arXiv:1806.01054*, 2018.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] 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. *arXiv preprint arXiv:1712.05474*, 2017.

[19] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: An On-line Lexical Database*. *International Journal of Lexicography*, 3(4):235–244, 12 1990.

[20] 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[21] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[22] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[24] Aleksey Staroverov, Dmitry A Yudin, Ilya Belkin, Vasily Adeshkin, Yaroslav K Solomentsev, and Aleksandr I Panov. Real-time object navigation with deep neural networks and hierarchical reinforcement learning. *IEEE Access*, 2020.

[25] Habitat Team. Habitat challenge, 2020. https://aihabitat.org/challenge/2020, 2020.

[26] Saim Wani, Shivansh Patel, Unnat Jain, Angel Chang, and Manolis Savva. Multion: Benchmarking semantic map memory using multi-object navigation. *Advances in Neural Information Processing Systems*, 33, 2020.

[27] 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. *International Conference on Learning Representations (ICLR)*, 2020.

[28] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Learning and planning with a semantic model. *arXiv preprint arXiv:1809.10842*, 2018.

[29] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[30] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018.