Auto-Navigator: Decoupled Neural Architecture Search for Visual Navigation

Tianqi Tang, Xin Yu, Xuanyi Dong, Yi Yang
ReLER Lab, AAII, University of Technology Sydney
Tang@stduent.uts.edu.au, Xin.Yu@uts.edu.au
Xuanyi.Dxy@gmail.com, Yi.Yang@uts.edu.au

Abstract

Existing visual navigation approaches leverage classification neural networks to extract global features from visual data for navigation. However, these networks are not originally designed for navigation tasks. Thus, the neural architectures might not be suitable to capture scene contents. Fortunately, neural architecture search (NAS) brings a hope to solve this problem. In this paper, we propose an Auto-Navigator to customize a specialized network for visual navigation. However, as navigation tasks mainly rely on reinforcement learning (RL) rewards in training, such weak supervision is insufficiently indicative for NAS to optimize visual perception network. Thus, we introduce imitation learning (IL) with optimal paths to optimize navigation policies while selecting an optimal architecture. As Auto-Navigator can obtain a direct supervision in every step, such guidance greatly facilitates architecture search. In particular, we initialize our Auto-Navigator with a learnable distribution over the search space of visual perception architecture, and then optimize the distribution with IL supervision. Afterwards, we employ an RL reward function to fine-tune our Auto-Navigator to improve the generalization ability of our model. Extensive experiments demonstrate that our Auto-Navigator outperforms baseline methods on Gibson and Matterport3D without significantly increasing network parameters.

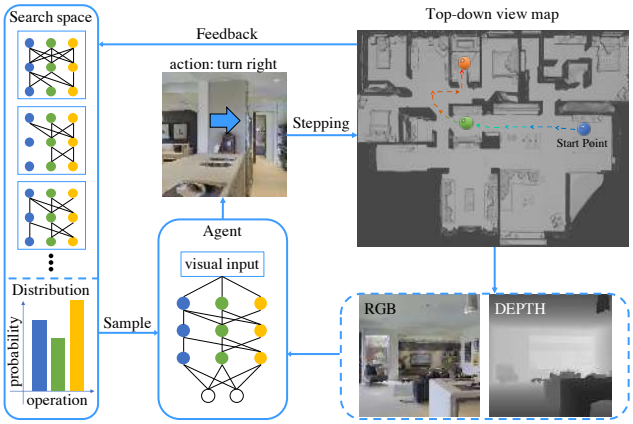


Figure 1: Illustration of our Auto-Navigator framework. We pre-define a search space of our visual perception network, where the candidate network blocks are parameterized by a probabilistic distribution. We optimize the parameterized distribution of the search space following two steps alternatively: (i) we first optimize the parameterized distribution of candidates under the supervision of IL and then sample a candidate from the distribution. (ii) this sampled candidate takes visual signals of the current state (green point) as input to predict an action (e.g., turn right) directing to the goal (orange point). Then, our navigation network is updated by minimizing the IL loss.

1. Introduction

Visual navigation aims to enable an autonomous agent to navigate through an environment based on sequentially observed images [14, 44, 41, 11, 46, 26]. An agent needs to understand the visual observation, memorize the previous trajectory, and deduce appropriate actions. According to the objective of visual navigation, we categorize navigation methods into three groups: (i) PointGoal, the agent is asked to navigate to a specific point. (ii) ObjectGoal, the agent searches a specific object in a scene. A target object is either specified by a language description or an image. (iii) AreaGoal, the agent is asked to move to a specific area, such as "foyer". In this paper, we mainly focus on PointGoal navigation tasks without losing generalization to other navigation tasks.

Previous visual navigation works leverage hand-crafted visual encoder architectures to extract global features and then feed the extracted features to a navigation policy network. However, these visual encoder architectures are originally designed for image classification but visual navigation. Even after fine-tuning, those networks may not be optimal to capture scene contents for visual navigation due to different scales and complexity of scenes. This poses a chal-

lenge to designing a deep visual perception network that is capable of extracting informative clues from observations. Due to the vast space of architecture choices in terms of the network depth, width and resolution, it is very difficult to manually decide an optimal perception network for navigation. Inspired by the advantages of network architecture search (NAS) techniques, we therefore intend to design a representative perception network for visual navigation in a data-driven fashion.

A straightforward solution is to apply existing neural architecture search (NAS) algorithms to a visual perception network. As RL rewards are mainly used to train a navigation policy network, we also firstly employ RL rewards as our supervision in searching our visual encoder architecture. However, we observed that such RL rewards are not suitable for NAS to search visual encoder architectures, and the navigation networks fail to converge in training. This is mainly because the changes of feature representations from our visual encoder significantly affect the internal states of the policy network, and it is hard to find a stable point where the perception network changes do not affect the internal states dramatically.

In this paper, we propose an Auto-Navigator to fully explore visual information from observations by optimizing a visual perception network architecture and introduce a newly developed multi-stage training strategy. We firstly define a parameterized distribution on the candidate architectures of our perception network and then introduce imitation learning (IL) as step-wise supervision for NAS to search an optimal architecture. This allows the supervision signal to impact on the visual network more directly without rolling through a recurrent neural networks based navigation policy network. The IL loss will be propagated to update the distribution of our visual encoder candidates. Then, we sample a candidate with respect to the parameterized distribution and then use the new architecture for navigation. IL loss in the next iteration will be used to optimize our navigation policy network. We conduct candidate selection and weight optimization in an alternating manner.

To improve the generalization ability of our Auto-Navigator, we employ RL rewards to fine-tune our navigation network in the second stage. To be specific, we select the optimal architecture with the highest probability from the parameterized distribution. Then, we only fine-tune the policy network via RL to avoid that our navigation policy might overfit to a specific scene.

Overall, our contributions can be summarized as follows:

- Our Auto-Navigator is *the first attempt* to automate the design for a visual navigation agent.
- We proposed an architecture searching schedule tailed to visual navigation by optimizing the parameterized architecture of a visual perception network and a policy network in an alternating fashion.
- Our method achieves superior performance in comparison to our baseline methods, thus demonstrating the effectiveness of our NAS to visual navigation tasks.

2. Related Work

Visual Navigation has a long history and is growing sensationally recently. For decades of research, a classical pipeline for navigation consists of building the map, locating an agent over the map via feature matching [24, 45, 38], and designing an optimal path based on the map. Mapping and locating are jointly viewed as the simultaneous localization and mapping (SLAM) problem [4, 10, 13]. Path planning has also been identified as an individual sub-task[18].

Recent learning-based navigation methods, in opposition to the classical pipeline, have received great attention. As representation learning and RL algorithms develop [14, 29, 15, 25, 46, 41], navigation methods have been boosted greatly. Several methods [14, 29] explore the semantic cues of environments via representation learning for navigation. Additionally, Mirowski et al. [27] employ an auxiliary depth prediction task to capture spatial geometric cues and Asynchronous Actor-Critic Agents (A3C) [28] to learn navigation policies. Beyond learning visual representations from RGB images, researchers also develop algorithms to learn scene structure information from RGB-D images [36, 29]. Du et al. [9] integrate object relationships and design a specialized memory-augmented tentative policy network for object navigation tasks.

Imitation learning allows an agent to take human expert trajectories or the shortest paths as supervision [29, 15]. Behavioral cloning (BC) [2] as a popular IL-based method has been widely adopted in navigation tasks to sample the shortest trajectory as ground-truth. It is used to initialize a network as a warm-up procedure, and then RL is employed to train a navigation policy network [46, 27, 26]. A self-adaptive visual navigation method [41] has been proposed to learn how to adapt to an unseen environment from interactions via meta-learning [12, 23, 22]. Ma et al. [25] present a system that allows an agent to learn when to backtrack. However, previous works pay little attention to the design of network architectures suitable for visual navigation tasks.

Neural Architecture Search aims to construct neural network architectures automatically. To improve heavily hand-crafted network design [17, 37, 16], many NAS algorithms have been proposed to maximize the validation accuracy [33, 32, 21, 6, 20, 7, 39]. The search strategy consists of random search, Bayesian optimization, evolutionary methods, RL, and gradient-based methods. Preceding works [21, 47, 48, 8, 3, 30, 42] focus on searching the architectures on proxy classification tasks and extend the found

CNN architecture for large scale image recognition challenges. Zoph et al. [47, 48] employ NAS to search neural architectures via RL rewards for image classification problems, while Real et al. [33] employ evolutionary algorithms to optimize neural architectures. Liu et al. [21] sample architectures from the parametric distribution and iteratively optimize weight parameters and the learnable distribution. Cai et al. [5] directly search architectures on large-scale datasets by random sampling one single path from candidate paths. Quan et al. [31] incorporate the structure information of input images to search a specifically designed reID architecture. Li et al. [19] develop a lightweight segmentation head via NAS with the resource-aware constraint. As the visual navigation task involves visual perception and navigation policy components, how to use NAS with RL rewards in this task still remains unknown.

3. Methodology

Preliminary. We formulate PointGoal navigation as a partially observable Markov decision process (POMDP). At the timestamp t, the policy network receives an observation o_t (e.g., RGB or RGB-D image) and outputs an action a_t. The agent takes this action, which causes the environment to transition to the next state s_{t+1}. Afterwards, the agent receives a reward r_t. During training the agent, we try to maximize the cumulative rewards over an episode.

The reward function is specified according the task. In this paper, we use the following reward r_t [34, 14]:

$$r_t = \begin{cases} \theta + d_{t-1} - d_t + \lambda, & \text{if reach the goal,} \\ d_{t-1} - d_t + \lambda, & \text{otherwise,} \end{cases} \tag{1}$$

where d_t is the geodesic distance to point target at timestamp t. θ is a constant reward activated when the agent reaches the target goal. λ is a time penalty. Let π denote a stochastic policy. We formulate the cumulative reward as $\eta(\pi) = \sum_{t=1}^{T} \gamma^{t-1} r_t$, where γ is the discount factor. Prevailing methods utilize proximal policy optimization (PPO) [35] to optimize the policy network.

3.1. Auto-Navigator Framework

Auto-Navigator framework, seen in Figure 1, is the first attempt to automate the design of navigation agent. Specifically, we (1) design the search space of the perception modules, (2) initialize a parametric distribution $\mathcal{A}$ over the search space, (3) alternately optimize the parameters of the distribution and the weights of all policy networks ω, (4) derive the final perception module α^* from the parametric distribution. We formulate this procedure as follows:

$$\alpha^* = \arg\min_{\alpha \in \mathcal{A}} \ L_{val}(\alpha, \omega_\alpha), \tag{2}$$

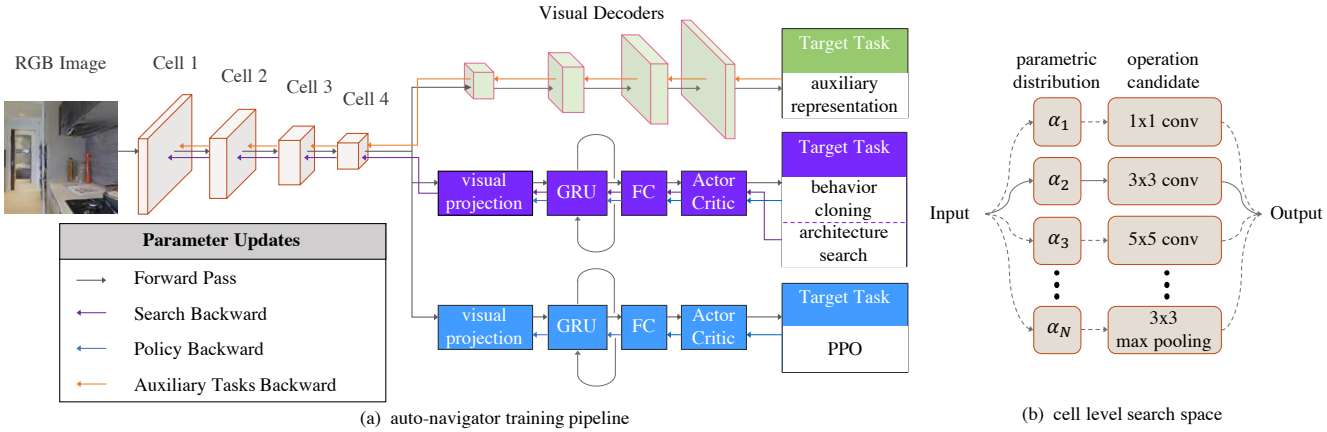
$$s.t. \ \omega_\alpha = \arg\min_{\omega} \ L_{train}(\alpha, \omega), \tag{3}$$

where $L_{train}(\cdot)$ is the objective function using the training set and $L_{val}(.)$ is the loss function with the validation set. Both training and validation set are from the original training set, and we split it for our searching algorithm. We update the parameters of the distribution over the search space ω_α with the validation split. The policy network is optimized with the training split.

We separate the agent into two concise parts: perception module f and policy decoder π. The perception module f extracts visual features $f(o_t)$, which encode the observation o_t. We initialize the agent with a learnable distribution over the search space of the perception module. The policy module π takes $f(o_t)$ as input and predicts the probability of each action a_t. Followed by SplitNet [14], we employ a one-layer GRU as our final policy network. To obtain better generalization in unseen environments, we optimize the neural architecture followed by Eq. (3) and employ a three-stage training pipeline. Stage$_1$: pre-training the perception module f via auxiliary tasks (see details in Sec. 3.2). Stage$_2$: pre-training the policy module π with fixed f using behavioral cloning. Stage$_3$: fine-tuning the policy module π with fixed f using PPO. We perform our searching methods when we update the policy module in the second stage. However, the agent consists of perception module and policy network , which is different from the CNN-based architectures in other vision tasks. Moreover, the objective functions are dynamic during the multi-stage training. These issues cause conventional NAS methods unsuitable in our task.

The classical gradient-based methods (e.g., DARTS [21]) have to face this problem of dynamic objectives for multi-stage training. Inspired by the idea of separating visual representation learning and navigation policy learning, we tailor our searching strategy via decoupling $\mathcal{A}$ and ω. Denote ω_f as the weights of the perception module, and ω_π as the weights of the policy module. We search the optimal perception module and optimize the policy alternately to solve the above issue. The perception representation ensembles feature selection to guarantee navigation performance although the objective functions are dynamically changed. We visualize our complete pipeline in Figure 2 and we update ω_f, $\mathcal{A}$ and ω_π independently. However, it is impractical to directly search the architecture via gradient-based methods due to high memory consumption for visual navigation tasks. During training ω_f, we only select one path from all candidate nodes in one neural block by uniformly sampling, and we update $\mathcal{A}$ through sampling two paths as if other competitive paths do not exist.

In the following sections, we first introduce the widely used visual representations in Sec. 3.2 to be self-contained and then describe the search algorithm as well as search space in Sec. 3.3.



(a) auto-navigator training pipeline (b) cell level search space

Figure 2: *Left:* The training pipeline of Auto-Navigator. We display four searchable cell structures as perception module and ignore pooling layers in this figure. Given the extracted features from the autoencoder, the policy network is optimized by the objective functions. For improving the generalization of the agent, we propose a three-stage training strategy. We first improve the visual representation via training with auxiliary tasks. Next, we search for the architectures of perception modules and train the policy network via behavioral cloning. In the third stage, we finetune the policy network via PPO. The forward pass and the gradient back-propagation can be represented as the grey lines and the orange lines. The blue lines are depicted for updating the policy network, and the purple arrows are depicted for optimizing the learnable distribution over the search space. *Right:* The search space we designed. α is the learnable distribution in our experiments.

3.2. Visual Representations

Representation learning has demonstrated the efficiency of the pre-training method and speeds up the convergence of training. We train one encoder-decoder network with supervised auxiliary tasks, which share the weights ω_f. This feature $f(o_t)$ is used as input for different visual decoders with auxiliary tasks and is also viewed as input for the navigation policy network.

To extract better geometric information, we leverage depth prediction, surface normals prediction, image reconstruction, egomotion prediction and feature prediction as auxiliary tasks. The respective decoders are denoted as $\mathcal{D}$, $\mathcal{S}$, $\mathcal{R}$, $\mathcal{E}$, $\mathcal{P}$. The corresponding objective functions are shown below:

$$\mathcal{L}_{\mathcal{D}} = \frac{1}{|V|} \sum_{V} \left\| \mathcal{D}(f(o_t)) - \bar{d}_t \right\|_1, \tag{4}$$

$$\mathcal{L}_{\mathcal{S}} = 1 - \sum_{V} \frac{\mathcal{S}(f(o_t)) \cdot \varphi_t}{\|\mathcal{S}(f(o_t))\|_2 * \|\varphi_t\|_2}, \tag{5}$$

$$\mathcal{L}_{\mathcal{R}} = \frac{1}{|V|} \sum_{V} \left\| \mathcal{R}(f(o_t)) - o_t \right\|_1, \tag{6}$$

where V stands for existing pixel points, and $|V|$ defines the total number of pixel points. $\bar{d}_t$, φ_t are corresponding ground truth. We apply l_1 loss for depth and reconstruction and cosine loss for surface normals.

$$\mathcal{L}_{\mathcal{E}} = -\sum_{a \in A} p(a_t = a) \log(\mathcal{E}(f(o_t), f(o_{t-1}))), \tag{7}$$

where a_t is the real action viewed as ground truth for the motion decoder, and we train the decoder with the cross-entropy loss.

$$\mathcal{L}_{\mathcal{P}} = 1 - \sum_{features} \frac{\mathcal{P}(f(o_{t-1}), a_t) \cdot f(o_t)}{\|\mathcal{P}(f(o_{t-1}), a_t)\|_2 * \|f(o_t)\|_2}, \tag{8}$$

where a_t is one-hot encoding of the action and $f(o_{t-1})$ is previous feature.

Our overall objective function can be formulated as follows:

$$\mathcal{L} = \lambda_{\mathcal{D}}\mathcal{L}_{\mathcal{D}} + \lambda_{\mathcal{S}}\mathcal{L}_{\mathcal{S}} + \lambda_{\mathcal{R}}\mathcal{L}_{\mathcal{R}} + \lambda_{\mathcal{E}}\mathcal{L}_{\mathcal{E}} + \lambda_{\mathcal{P}}\mathcal{L}_{\mathcal{P}}, \tag{9}$$

where $\lambda_{\mathcal{D}}$, $\lambda_{\mathcal{S}}$, $\lambda_{\mathcal{R}}$, $\lambda_{\mathcal{E}}$, $\lambda_{\mathcal{P}}$ stand for controllable weights of the corresponding loss.

Different from the classical neural network, we build a search space that contains all candidate nodes. GPU memory is limited if we directly train this network. Therefore, we randomly sample only one path for optimizing every iteration via uniformly sampling. We formulate the optimization for ω_f as

$$\omega_f = \arg\min \mathbb{E}_{i \sim \mathcal{U}}[\mathcal{L}(\alpha_i, \omega_f(\alpha_i))], \tag{10}$$

Algorithm 1: Auto-Navigator

Input : the whole original training set
the auxiliary training set
randomly initialized $\mathcal{A}$, ω_f and ω_π

Pre-train:

Update ω_f by optimizing Eq. (10) with the auxiliary training set.

Behavioral cloning:

Split the original training set into two disjoint sets $\mathbb{D}_T$ and $\mathbb{D}_V$

Fix the perception module weights ω_f **while** *not converged* **do**

1. Update $\mathcal{A}$ by optimizing Eq. (12)
2. Update ω_π by optimizing Eq. (13)

end

Derive the final agent α^* from $\mathcal{A}$

where $\mathcal{U}$ is a uniform distribution, and α_i represents the probability from the learnable distribution, which reflects the ranking of architecture.

As stated in SplitNet, representation learning is non-trivial for navigation tasks. It is even more challenging in our Auto-Navigator, because we need to learn the weights of all candidate agents over the search space. To reduce memory footprint, we only optimize the weights over the sampled agent at each iteration.

3.3. Path Sampling Search Algorithm

As illustrated in our Algorithm 1, the agent inherits the perception module weights ω_f from representation learning. The searching stage and training policy decoder are decoupled as two alternate steps to avoid joint optimization. The original training dataset is also separated into two parts: the training set $\mathbb{D}_T$ for training policy decoder and the validation set, $\mathbb{D}_V$ for searching architectures. To maintain navigation performance, we freeze the weights from the perception module during both the searching and the training stages.

During the searching stage, the perception module could be treated as one graph with N nodes, which are the sequence of N operations in a cell (e.g., convolution, max pooling, identity, $zero$). The parameters of the distribution $\mathcal{A}$ over the search space reflect the probabilities of the connection between these nodes. we introduce a learnable vector $\beta_i \in \mathcal{A}$, where i is the sequential index. For example, if β_i is equal to zero, it means that the operation $\bar{o}_i$ disconnects with the previous node. A natural solution is to relax the discrete choice of operation into a continuous search space by applying softmax to architecture vectors $\{\beta_i\}_{i=1}^N$, and the normalized vector values could be viewed as the probabilities of searching for different operations:

$$m(x) = \sum_{i=1}^{N} \frac{\exp \beta_i}{\sum_q \exp \beta_q} g_i(x) \quad s.t.\ g_i \in \mathcal{F}, \tag{11}$$

where the final output m is viewed as the weighted sum of different candidate features through various operations, x is the input and q is the index for different nodes.

However, the cost of memory is still unaffordable during searching. To alleviate this problem, we first sample two candidate paths according to the multinormal distribution that is calculated by softmax, as shown in Eq. (11), where we set $N = 2$. Benefiting from the sampling strategy, the searching procedure reduces memory footprint from N to 2. We ignore other nodes that are not selected. The optimization for the parameters of the distribution $\mathcal{A}$ process could be expressed as minimizing the following expectation:

$$\mathbb{E}_{\alpha \sim \mathcal{A}}[\mathcal{L}_{val}(\alpha, \omega_f, \omega_\pi)], \tag{12}$$

where $\mathcal{A}$ indicates the parametric distribution.

During training policy decoder alternately, we sample a single path by uniformly sampling while we freeze ω_f, which is identical with the sampling strategy for representation learning. The same strategy guarantees the adaption between different stages and reduces $N-1$ times GPU memory than the previous search approaches(e.g., DARTS [21]). Therefore, we can expand our search space with the consumption cost, which is the same as one classical convolutional architecture. This optimization could be formulated as minimizing the following expectation:

$$\mathbb{E}_{\alpha \sim \mathcal{U}}[\mathcal{L}_{train}(\alpha, \omega_f, \omega_\pi)], \tag{13}$$

where $\mathcal{U}$ is uniform distribution over all candidate agents.

The original training pipeline consists of multiple stages, where the objective dramatically changes. Behavioral cloning viewed the shortest path as ground truth, which formulates navigation into one supervised classification task. We utilize the reward based on the geodesic distance to the goal, as mentioned in Eq.(1) during PPO. In consideration of the policy decoder optimized with half of the whole training set, we concern about the agent would learn more bias about the environment via PPO. During the searching period, we propose behavioral cloning to update the architecture vectors. At the end of searching, we select the best operation with the highest probability:

$$op_i = \underset{g_i \in \mathcal{F}}{\arg\max}\, \beta_i, \tag{14}$$

where all $\{op_i\}$ consist the agent α.

A complete training process includes a pre-training stage with auxiliary tasks and two alternative searching stages via behavioral cloning. To achieve better performance during

searching, we learn the decoder with the split training set $\mathbb{D}_T$. After we discover the best architecture of the agent, we re-train it with a three-stage training protocol, which is referred to Sec. 3.1.

Our baseline neural network is one simple 7-layers neural architecture consists of convolution, group normalization, ELU, and max pooling. We define one cell-level search space that covers the primary network and repeats multiple times to construct the whole structure.

We apply two different search spaces. One contains five operations, called Auto-Navigator-S (Small):

- 1×1 conv
- 3×3 conv
- 5×5 conv
- 3×3 max pooling
- 7×7 conv

Another one, Auto-Navigator-L (Large), contains nine operations, where more prevalent layer types are added:

- 3×3 max pooling
- 1×1 conv
- 3×3 sepconv (expansion 3)
- 5×5 sepconv (expansion 3)
- 7×7 sepconv (expansion 3)
- 3×3 sepconv (expansion 6)
- 5×5 sepconv (expansion 6)
- 7×7 sepconv (expansion 6)
- identity

4. Experiments

We conduct experiments on the *Habitat* scene renderer [34] of *Matterport 3D* (referred to as *MP3D*) and *Gibson* to evaluate Auto-Navigator.

4.1. Experiment Setup

Datasets. We validate the effectiveness of our proposed method on three popular benchmarks. *Habitat* [34] is a new platform for training agents in different 3D simulated environments. Based on the platform, we evaluated our navigation models constructed automatically on two point-to-point navigation datasets. *Gibson* provides real-world perception for active robots. We follow the same training/val/test split strategy as the previous works [34]. To be specific, *Gibson* has 72 training scenes with 4.9 million episodes and 994 unseen episodes for testing. *MP3D* is a large-scale indoor scene dataset. There are 5 million training episodes and 495 episodes in the validation set. The validation set is unknown to a navigation policy model during training.

During moving the environment, an agent could access the static target point with a noiseless GPS sensor and a camera sensor. The action space includes moving forward by 0.25 meters, turning left by 10 degrees and turning right by 10 degrees. The discrete action "stop" is disregarded because the action "stop" tends to terminate an agent earlier to avoid further penality in training. An agent will stop when the agent reaches a position within 0.2 meters away from the target point or walks more than 500 steps during one episode.

Searching details. We utilize the structure of SplitNet with the hand-crafted backbone as our baseline. SplitNet consists of four convolutional layers with group normalization layers [43] and three pooling layers. We adopt the same policy model, which consists of a one-layer GRU and linear layers. There are two formulations for the search space: Auto-Navigator-S and Auto-Navigator-L. Auto-Navigator-S has four searched cells with five different nodes, which includes 625 architecture candidates. Auto-Navigator-L searches for a 7-layer structure, which has nine operations in a cell and 4.8×10^6 cell architectures. Both search spaces employ group normalization layers, and the number of groups is set to eight. For a fair comparison, we compare SplitNet [14] with its provided training strategy and network configuration with our navigation algorithm. SplitNet is also referred to as "baseline" in our results.

Training details. Before searching, we utilize the above objective function, as mentioned in Sec. 3.2 to optimize the perception network. The training dataset and ground-truth target points are provided as the same as SplitNet. We sample one candidate architecture every iteration and ignore other existing nodes. We input an image of size 256×256 pixels on Gibson and use an Adam optimizer with a learning rate of 0.0005. The whole training requires 200 epochs, and the batch size is set to 32.

During searching, the gradient will not back-propagate to the perception module. We introduce behavioral cloning to learn the architecture vectors and optimize the policy module. Similar to training SplitNet, we utilize Adam with a learning rate of 0.00025 and momentum $\beta = (0.5, 0.999)$. The number of training steps is 100M in our experiments. Habitat provides the baseline with 50M training steps, which are scaled to 100M in SplitNet. Their experimental results demonstrate using longer training steps is able to achieve better performance. Therefore, we apply the same training steps, and thus it is fair to compare our method with SplitNet. We select the highest probability nodes for our visual perception network.

To attain stronger generalization of our navigation network, we apply the PPO algorithm to our policy network initialized weights via behavioral cloning. The reward, as mentioned in Eq. (1), aims to decrease the geodesic distance between the target coordinate and the agent, which assists in avoiding over-fitting the supervised trajectory from behavioral cloning and increases the generalization in unseen test scenes. The training configuration is the same as the behavioral cloning stage. Additionally, we set a discount factor to 0.99 and the Generalized Advantage Estimation (GAE) parameter to 0.95. The weight decay in both stages is set to 0.001. There are 16 workers collecting 128 images in parallel and then we train our agent with four epochs with batch size 1.

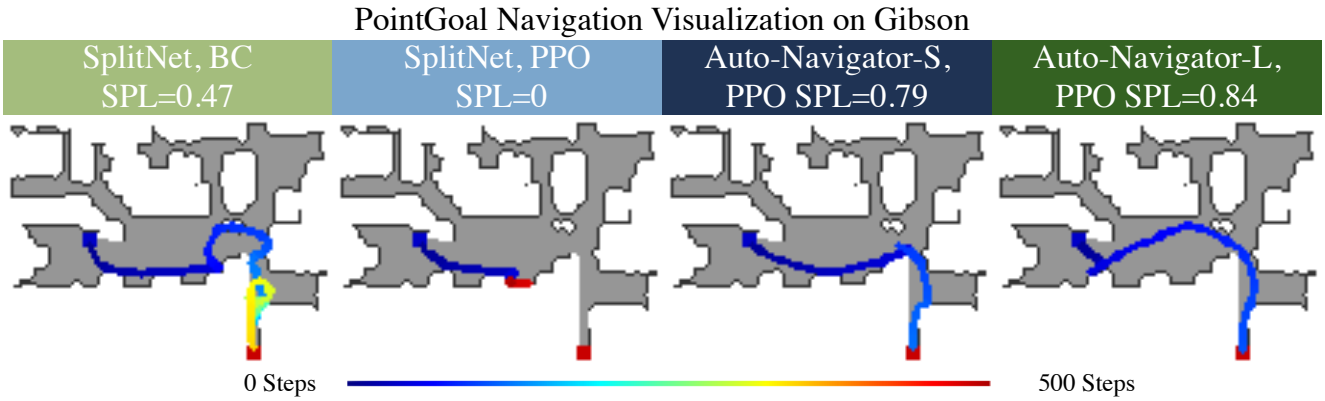


Figure 3: Comparisons for different agents on Gibson validation set. Visualization for the same episode, where the blue dot is the starting point and the red dot is the ending point. The line with many color variations reflects the number of steps in one episode.

4.2. Evaluation Metrics

Success rate is one primary evaluation metric for visual navigation. In one episode, it is considered as success when the agent stops within 0.2m geodesic distance from the endpoint. On the contrary, the episode fails after the agent walks 500 steps and is still far from the target point. We calculate the ratio of success in all validation episodes, called success rate (success).

SPL is the "Success weighted by Path Length" metric [1], defined as:

$$SPL = \frac{1}{\widetilde{N}} \sum_{i=1}^{\widetilde{N}} \widetilde{S}_i \frac{l_i}{max(p_i, l_i)}, \tag{15}$$

where $\widetilde{S}_i$ represents a binary success indicator for episode i, p_i is the path length for our agent, and l_i is the shortest distance between our current position and the target point. This metric estimates the efficient of a navigation agent, and if a trajectory is close to the optimal one the SPL will be close to 1.0.

4.3. Navigation Results

In this section, we report the experimental results about Gibson and MP3D. In Table 1, we compare both searched architectures with SplitNet. We adjust different training strategies, including behavioral cloning and PPO. As shown in the table, both architectures outperform SplitNet via BC or PPO. Our best model (Auto-Navigator-S) via PPO improves the performance of SPL by 5.57%. Another agent also outperforms the performance by 6.25%.

In Table 2, we experiment with different search spaces repeatedly. We show that the architectures from the same search space perform general enough on Gibson. For Auto-Navigator-S, the second running agent outperforms SPL by 0.25% but introduces more FLOPs and parameters. For Auto-Navigator-L, another agent outperforms the performance by 1.4% and reduce half of the FLOPs.

To better understand the generalization for our searched architecture, we plot mean SPL on the validation set with different training steps, shown as the top of Figure 4. Through this figure, we notice the searched architectures outperform our baseline by a wide margin. We also compare mean SPL with different starting geodesic distances at the bottom of Figure 4. We can view it more complicated with farther geodesic distance. All agents perform worse with faraway geodesic distance, but our agents perform a stronger generalization of complicated episodes.

We analyze the performance on MP3D in Table 3. Auto-Navigator-S achieves 52.74% SPL, which is surpassed by 51.70% SPL with 100 million training steps. We also report the result with other state-of-the-art modules. DD-PPO [40] utilizes RGB images and depth as input, whereas our approach only takes images as input.

Compared with other state-of-the-art models, as shown in Table 4, we achieve the best agent with the SPL of 76.35% and the success rate of 84.91% on the Gibson validation dataset. We notice that DD-PPO uses SE-ResNeXt101 as backbone and the training time is 250 times larger than ours.

4.4. Discussion

We notice one unusual situation that the success rate decreases, but SPL increases while the agent is optimized via PPO. It means that the agent performs a shorter path trajectory but fails in some episodes, which succeed in the behavioral cloning stage. The reward in PPO tends to decrease the geodesic distance. The agent may go round in the wrong room, which is close to the target point. Due to rare difficult episodes, the imbalance of the episodes may lead the agent

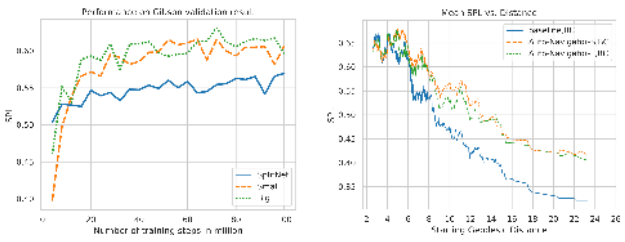


Figure 4: **Top:** Validation SPL during 100 million training steps with different architectures. **Bottom:** Mean SPL vs. starting geodesic distance on the Gibson dataset. We compare our approaches with SplitNet. We report mean SPL with different starting geodesic distances in unseen episodes. Both architectures perform better than our baseline, especially for difficult episodes

Method	BC	PPO	FLOPs	Params	SPL (%)	Success (%)
Auto-Navigator-S	✓		474.94M	0.39M	60.65	91.65
Auto-Navigator-L	✓		898.56M	0.59M	59.60	89.84
Auto-Navigator-S	✓	✓	474.94M	0.39M	75.67	85.81
Auto-Navigator-L	✓	✓	898.56M	0.59M	76.35	84.91
Random [14]			-	-	4.60	2.80
SplitNet [14]	✓		208.57M	0.25M	58.40	86.50
SplitNet [14]	✓	✓	208.57M	0.25M	70.10	85.50

Table 1: Gibson validation set results with different Auto-Navigator models. BC: behavioral cloning.

Method	FLOPs	Params	SPL (%)	Success (%)
Baseline	208.57M	0.25M	56.96	85.92
Auto-Navigator-S (run 1)	474.94M	0.39M	60.65	91.65
Auto-Navigator-S (run 2)	676.27M	0.63M	60.90	92.35
Auto-Navigator-S (run 3)	537.72M	1.02M	60.57	91.95
Auto-Navigator-S (run 4)	235.47M	0.23M	57.57	87.82
Auto-Navigator-L (run 1)	898.56M	0.59M	59.60	89.84
Auto-Navigator-L (run 2) †	368.63M	0.57M	61.01	91.85

Table 2: The effect of different search spaces on Gibson. All models are trained in *the same training strategy*. We analyze the effect of different architectures via behavioral cloning with 100 million steps. †: training with 64 million steps.

Method	FLOPs	Params	SPL (%)	Success (%)
Baseline	208.57M	0.25M	51.22	80.81
Auto-Navigator-S †	1112.47M	1.65M	54.30	85.05
Random [14]	-	-	1.10	1.60
SplitNet [14]	208.57M	0.25M	51.70	80.80

Table 3: MP3D validation set results. We experiment with Auto-Navigator-S and optimize the agent through behavioral cloning. †: training with 80 million steps.

to the local optimum.

Method	RGB	RGBD	SPL (%)	Success (%)
Habitat [34]	✓		46.00	64.00
Habitat [34]		✓	70.00	80.00
SplitNet [14]	✓		70.10	85.50
DD-PPO [40]		✓	96.90	-
Auto-Navigator-S	✓		75.67	85.81
Auto-Navigator-L	✓		76.35	84.91

Table 4: Comparisons with state-of-the-art navigation model on Gibson validation set. RGB: inputs only include RGB images. RGBD: using RGB images and depth maps.

During searching, we expect the larger search space could assist in the improvement of our performance. Unfortunately, both search spaces perform similar results on Gibson and MP3D. In Figure 4, we notice there exists the probability of improving the performance via enlarging training steps. Moreover, the curve for Auto-Navigator-L is not smooth enough due to the hyper-parameter setting. We could achieve better performance by expanding training steps and adjusting the configuration.

5. Conclusion

In this paper, we have proposed Auto-Navigator to automate the agent design for visual navigation. In order to solve the difficulty of training with weak supervision in RL, we introduce imitation learning to provide step-wise guidance as ground truth. Our derivations imply that a multi-stage training strategy is essential for improving the performance of our agent. We incorporate the multi-stage training strategy to our search algorithm in an alternating fashion. The induced searching method is both efficient and effective. Benefiting from our NAS method and the training strategy, our Auto-Navigator achieves superior performance on two widely-used benchmarks compared to our baseline methods while restricting our navigation model to be relatively small. This also makes our proposed navigation method more practical.

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:1807.06757*, 2018.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *RAS*, 2009.

[3] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks, 2018. ICLR.

[4] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *T-RO*, 2016.

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware, 2019. ICLR.

[6] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. AutoHAS: Efficient hyperparameter and architecture search. *arXiv preprint arXiv:2006.03656*, 2020.

[7] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network, 2019. ICCV.

[8] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours, 2019. CVPR.

[9] Heming Du, Xin Yu, and Liang Zheng. Learning object relation graph and tentative policy for visual navigation, 2020. ECCV.

[10] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *RAM*, 2006.

[11] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks, 2019. CVPR.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017. ICML.

[13] Emilio Garcia-Fidalgo and Alberto Ortiz. Vision-based topological mapping and localization methods: A survey. *Robot. Auton. Syst.*, 2015.

[14] Daniel Gordon, Abhishek Kadian, Devi Parikh, Judy Hoffman, and Dhruv Batra. Splitnet: Sim2sim and task2task transfer for embodied visual navigation, 2019. ICCV.

[15] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation, 2017. CVPR.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2016. CVPR.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks, 2012. NeurIPS.

[18] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[19] Peike Li, Xuanyi Dong, Xin Yu, and Yi Yang. When humans meet machines: Towards efficient segmentation networks, 2020. BMVC.

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search, 2018. ECCV.

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search, 2019. ICLR.

[22] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Prototype propagation networks (PPN) for weakly-supervised few-shot learning on category graph, 2019. IJCAI.

[23] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Learning to propagate for graph meta-learning, 2019. NeurIPS.

[24] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[25] Chih-Yao Ma, Zuxuan Wu, Ghassan AlRegib, Caiming Xiong, and Zsolt Kira. The regretful agent: Heuristic-aided navigation through progress estimation, 2019. CVPR.

[26] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map, 2018. NeurIPS.

[27] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments, 2017. ICLR.

[28] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. ICML.

[29] Arsalan Mousavian, Alexander Toshev, Marek Fišer, Jana Košecká, Ayzaan Wahid, and James Davidson. Visual representations for semantic target driven navigation, 2019. ICRA.

[30] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing, 2018. ICML.

[31] Ruijie Quan, Xuanyi Dong, Yu Wu, Linchao Zhu, and Yi Yang. Auto-ReID: Searching for a part-aware convnet for person re-identification, 2019. ICCV.

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search, 2019. AAAI.

[33] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers, 2017. ICML.

[34] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research, 2019. ICCV.

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[36] William B Shen, Danfei Xu, Yuke Zhu, Leonidas J Guibas, Li Fei-Fei, and Silvio Savarese. Situational fusion of visual representation for visual navigation, 2019. ICCV.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. ICLR.

[38] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. Sosnet: Second order similarity regularization for local descriptor learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11016–11025, 2019.

[39] Naiyan Wang, Shiming XIANG, Chunhong Pan, et al. You only search once: Single shot neural architecture search via direct sparse optimization. *TPAMI*, 2020.

[40] 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, 2020. ICLR.

[41] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning, 2019. CVPR.

[42] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, 2019. CVPR.

[43] Yuxin Wu and Kaiming He. Group normalization, 2018. ECCV.

[44] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Bayesian relational memory for semantic visual navigation, 2019. ICCV.

[45] Xin Yu, Yurun Tian, Fatih Porikli, Richard Hartley, Hongdong Li, Huub Heijnen, and Vassileios Balntas. Unsupervised extraction of local image descriptors via relative distance ranking loss. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[46] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning, 2017. ICRA.

[47] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning, 2017. ICLR.

[48] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition, 2018. CVPR.