Training and Deploying Deep Learning Models for Real-Time Pathfinding in Indoor Environments

Aser Atawya, Kellie Au, Francisco Morales Puente, Tommy Ryan, Ella Zhu, and Anthony J. Clark
Computer Science Department
Pomona College
Claremont, California, USA
anthony.clark@pomona.edu

***Abstract*—Pathfinding in dynamic, indoor environments is fundamental to the reliable, safe, and real-time navigation of autonomous systems. In this study, we present our research generating datasets and comparing deep learning architectures for real-time pathfinding in indoor environments. We use simulation for data collection, compare six architectures, and analyze real-time inference performance on an NVIDIA JetBot. Our work showcases an end-to-end pathfinding approach and highlights challenges to address in future research.**

***Index Terms*—robotics, simulation, learning, vision, edge computing**

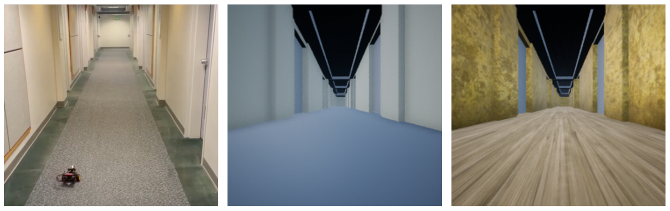

Fig. 1: (Left) An image of a hallway in Oldenborg Hall including an NVIDIA JetBot and (Middle/Right) simulated depictions of the environment.

I. Introduction

Autonomous robots are becoming increasingly prevalent, with applications ranging from self-driving vehicles to factory automation. Such robots must operate reliably and safely in highly uncertain environments. Our work concentrates on enhancing pathfinding models for indoor spaces and implementing them in real-time navigation tasks.

Specifically, we compare small and larger model architectures (in terms of the number of parameters) to bring autonomy to the edge, enabling real-time model inference and decision-making directly on the robot despite its limited computational abilities. So called “TinyML” (machine learning models running on devices with lower compute resources, such as microcontrollers and single-board computers) seeks to not only enhance the efficiency of robots by conserving electric power, but also expands the scope of autonomous robots to include resource-limited environments, ultimately contributing to the widespread adoption of such systems.

We chose the following architectures (the number of parameters for each is listed in parentheses): ResNet18 (11.7M) [1], ConvNextV2Base (88.7M) [2], ConvNextV2Atto (3.4M), EfficientNet (12.2M) [3], MobileNetV4 (3.8M) [4], and VisionTransformer (86.6M) [5]. These modern architectures include a mix of larger, traditional networks, and models specifically designed for low-resource computing environments. We trained models using fastai [6] and the Timm library [7]. Figure 1 shows both the real-world environment with an NVIDIA JetBot and the simulated environment used for data collection.

We collected three datasets, each with 100 000 images, using a simulation created with Unreal Engine 5. The simulated environment replicates a portion of Oldenborg Hall on Pomona College’s campus. Datasets differ in how a programmed agent navigates the hallways. We have three navigation behaviors: perfect, wandering, and teleporting (see Figure 2). Programmed agents have access to a full map of the environment, however, agent’s using trained models do not have access to this map. Each image is labeled with the signed angle between the agent’s current heading and the target location. Target locations were manually marked at the center of each overlapping hallway intersection. Labeling images with an angle enables us to train both classification- and regression based-models. Though, our prior research shows little different between these approaches [8]. The simulation also allows for all surface textures to be randomized and for random changes to internal and external lighting. Figure 1 shows two different configurations for the same hallway.

We trained three models for each combination of architecture and dataset for a total of 108 models (3 replicates, 6 architectures, 3 dataset collection agents, and 2 methods for texture randomization). Models were trained to classify input camera frames as one of three actions: move forward, rotate left, or rotate right. Models were evaluated on training metrics (i.e., classification accuracy and loss) using a validation dataset and behavior metrics (i.e., path progress, processing time, and reliability) by deploying models in simulation. Only the most promising models were evaluated on the JetBot.

In terms of the three datasets, the perfect dataset produced models with the highest accuracy (all models were greater than 99 % accurate) but the worst behaviors. Specifically, these models performed poorly during inference time on the simulated robot in that they frequently ran into walls and were unable to pathfind through the environment. Models trained on the teleporting dataset were less accurate (around 70 %) and

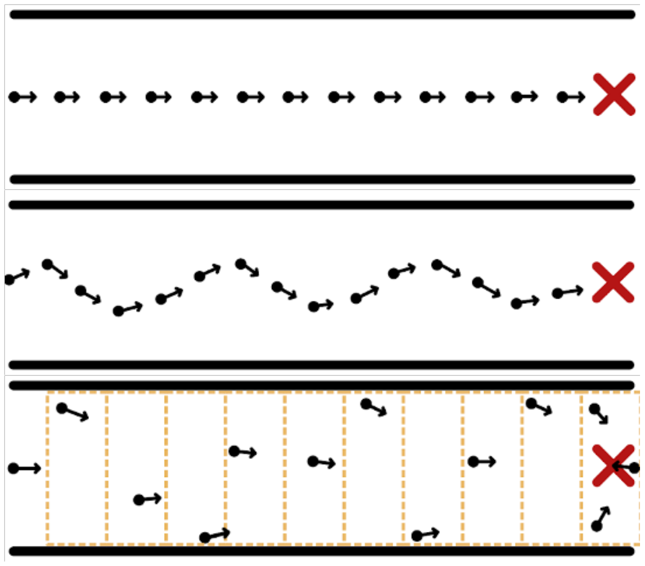

Fig. 2: Different data collection strategies (from left to right): perfect, wandering, and teleporting.

resulted in similarly poor behaviors. The wandering dataset, on the other hand, produced models with the best behaviors. Despite having lower accuracy on the validation dataset (around 90 %), these models were able to consistently navigate their way through the environment. Our current experiments suggest that randomizing textures reduces accuracy but improves the model's ability to generalize to new environments. Figure 3 shows a confusion matrix for the ConvNextV2Atto models. Data represented in the figure was collected while analyzing the models' behaviors in the simulated environment.

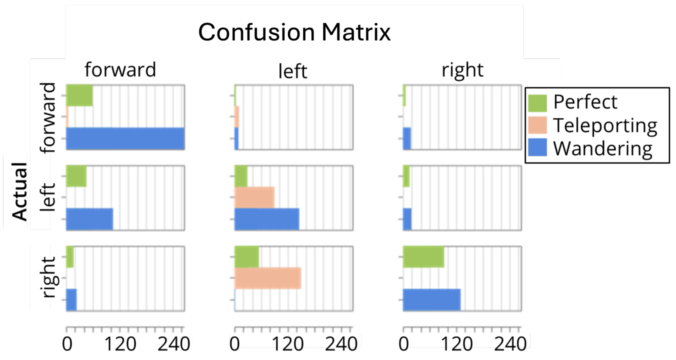

Fig. 3: A confusion matrix depicting for a ConvNextV2Atto model trained on randomized textures using the three different data collection agents.

As for the architectures, the ViT-based models consistently produced the most accurate validation results. However, the smaller architectures (i.e., ConvNextV2Atto and MobileNetV4) were only slightly behind in terms of accuracy and comparable in terms of behaviors. Moreover, the smaller architectures were significantly faster in terms of inference time per frame. Specifically, these models were able to process frames in about 0.2 s on average.

Building on these promising results, we proceeded to deploy the models in the real-world by implementing them on an Nvidia JetBot. Out JetBot is powered by the 4 GB version of the Jetson Nano Module. The robot has a wide angle camera with a 136° field of view; we calibrated the camera following standard procedures. We remotely communicated with the JetBot using remote procedure calls (RPCs). The robot, however, was responsible for autonomous navigation in which it captured images in real-time, processes the images, computed a trained models output, and then executed the most appropriate action.

Deploying our models in the real-world revealed the strengths and limitations of our approach. We observed trained models demonstrating promising performance in real-time navigation, successfully processing data and acting upon it for maneuvering. However, we also observed significant deviations in the robot's behavior based on manually configured parameters, such as the distance taken on a forward action and the amount of rotation resulting from a rotate left or rotate right action. Often, the robot became stuck in a cycle of rotating a few times in one direction and then back in the other, failing to make progress towards the target. These observations suggest the need to further fine-tune and optimize our data collection process to improve our overall performance across simulation and reality, ultimately bridging the gap between the two.

Moving forward, our focus will be on refining our data collection process. First, we will improve the fidelity of our environment to better match the actual Oldenborg setting. We will do so using a combination of better modeling techniques, photogrammetry, and neural-based scene generation (e.g., NeRF and gaussian splatting). Moreover, we will introduce noise into the collection process to enhance the model's flexibility when navigating complex environments where small details such as lighting, textures, and shadows can make a huge difference. Next, we will rely heavily on generalization techniques such as domain randomization and data augmentation. Finally, based on our experiments we recommend a combination of the wandering dataset collection pattern (with texture randomization) using the smaller ConvNextV2Atto and MobileNetV4 architectures.

Acknowledgment

The authors would like to thank members of the ARCS Lab, specifically Daisy Abbott, Anjali Nuggehalli, Gage Wyre, and Chau Vu.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[2] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie, "Convnext v2: Co-designing and scaling convnets with masked autoencoders," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 16 133–16 142.

[3] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[4] D. Qin, C. Leichner, M. Delakis, M. Fornoni, S. Luo, F. Yang, W. Wang, C. Banbury, C. Ye, B. Akin *et al.*, "Mobilenetv4-universal models for the mobile ecosystem," *arXiv preprint arXiv:2404.10518*, 2024.

[5] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, "How to train your vit? data, augmentation, and regularization in vision transformers," *CoRR*, vol. abs/2106.10270, 2021. [Online]. Available: https://arxiv.org/abs/2106.10270

[6] J. Howard and S. Gugger, "fastai: A layered API for deep learning," *Information*, vol. 11, no. 2, p. 108, 2020.

[7] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm," *CoRR*, vol. abs/2110.00476, 2021. [Online]. Available: https://arxiv.org/abs/2110.00476

[8] O. Chang, C. Marchese, J. Mejia, and A. J. Clark, "Investigating neural network architectures, techniques, and datasets for autonomous navigation in simulation," in *IEEE Symposium Series on Computational Intelligence*. IEEE.