Submitted on December 7, 2018 to State University of New York at Binghamton
Volume ??, Number ??,

Autonomous Exploration and Object Search for Mobile Robots Using Deep Learning

Jared Rosenberg and
Shriprasad Bhamare

Abstract One of the ongoing challenges for mobile robotics has been contextual awareness and scene understanding. This is not just an important problem in its own right, but is a necessary prerequisite for task planning and decision making. And since visual information provides the richest description of one's environment, developing methods to make sense of *RGB-D* images is very important. We present an efficient solution for mobile robots to autonomously explore an unknown environment in search of a hidden object. Our solution leverages the recent advances in computer vision, specifically Convolutional Neural Networks (CNNs), and introduces an "external memory unit" that stores abstract feature representations of scenes to ensure that the robot does not waste time searching in areas that it has already visited.

1 Introduction & Background

Exploration of an unknown environment is a fundamental problem in mobile robotics. The inability of a robot to successfully and efficiently accomplish this task is a hindrance to a diverse range of applications (rescue, cleaning, object search, many higher level reasoning problems, etc.). Autonomous exploration is essentially composed of two high level components: (1) basic obstacle avoidance, and (2) path planning. Traditional techniques, such as the use of Monte Carlo methods with laser scan data, use human crafted, geometric rules from the sensor data to update probabilistic models of the robot's location. The robot *repeatedly* traverses an area until it has a high degree of confidence in its structure. It then moves on to areas that it has not yet explored. This is in stark contrast to the way in which humans can effortlessly reason about their environment from visual information. With *one glimpse*, humans are able to contextually understand where they are and whether or not they had recently visited that location. In addition, changes in orientation, lighting conditions,

and objects in the scene have a trivial impact on the accuracy with which humans can accomplish this. Our project attempts to augment established autonomous navigation techniques by fusing convolutional feature layers into the robot's decision making.

1.1 Convolutional Neural Networks Convolutional Neural Networks (CNNs) are currently the state of the art in most computer vision tasks (majority of papers in CVPR). They were specifically designed to leverage certain properties about image data - mainly translation invariance, hierarchical composition, and autocorrelation among neighboring pixels - which allows them to efficiently and accurately disentangle factors of variation to solve the presented task. They work by successively applying transformations to the input space, essentially filtering out the information that is not relevant for use by downstream layers (i.e. if the last layer is a linear softmax/SVM, the successive transformations will reformat the data in such a way that the problem can be solved in a linear fashion). A corollary of this fact is that the last few layers of the network will hold increasingly abstract concepts that, for all intents and purposes, summarize or highlight *important* characteristics of the image. Again, what is considered *important* is strictly dependent on what task the network was asked to solve.

As long as the compute and memory requirements are satisfied within the problem scope (and obviously if one has access to either a pre-trained network or the available data to train or fine-tune a network), Convolutional Neural Networks are currently the sensible tool to use for extracting useful knowledge from visual data (2016 *Goodfellow*, 2016 *Karpathy*).

CNNs are used extensively in industry, ranging from self-driving cars to cancer detection (2018 *Serj*). There has also been a lot of recent research in the area of real-time scene understanding for mobile robotics. Low latency object detection architectures such as *Yolo / SSD / Faster-R-CNN*, and object instance segmentation architectures such as *Mask R-CNN*, are commonly used to make robots contextually aware of their environments (2018 *He*, 2017 *Huang*). So long are the days when humans tediously hand crafted rule based programs for robotic vision modules.

Our project emphasizes CNNs that were trained for the specific task of object detection. Object detection is the task of detecting instances of certain classes in images or videos. An object detection network is trained to both recognize which objects are contained in the input pixels, and a bounding box of exactly where each of those objects is located. Object detection design can really be broken down into two parts: (1) the base convolutional architecture (i.e. *ResNet*, *Inception*, etc.), and (2) the strategy used for region proposals. Until about 2014, object detection was too slow to be used in real time. Drastic improvements to the region proposals aspect have decreased the test time by three orders of magnitude since then, and latency for one test image has dropped to tens of milliseconds (2017 *Huang*). The plausibility of different object detection networks will be discussed further in the *Evaluation & Results* section.

1.2 Autonomous Exploration without using a Map Autonomous exploration of a Turtlebot without using a map is challenging to implement as it doesn't use a prebuilt map, given orientation, or pose.In order to implement Autonomous Navigation

of a Turtlebot without using a map, we tried to implement A* algorithm, Frontier Exploration, ASE exploration, and Octomap. A* algorithm, Frontier Exploration, and ASE exploration didn't give us accurate results as per the project problem definition. Using Octomap we successfully achieved results as per project problem definition.

The frontier-exploration package provides a costmap-2d layer plugin BoundedExploreLayer, and actionlib client/server nodes explore-client and explore-server. Layer BoundedExploreLayer can certainly be used for more complex exploration tasks, functionality is exposed through two Services: UpdatePolygonBoundary and GetNextFrontier. It did not give us efficient results in terms of time and exploration.

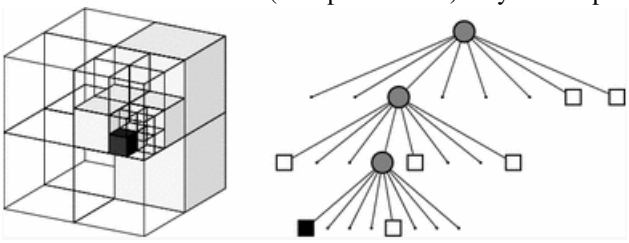
OctoMap is an efficient probabilistic 3D mapping Framework Based on Octrees.The OctoMap library implements a 3D occupancy grid mapping approach, providing data structures and mapping algorithms in C++ particularly suited for robotics.The map is able to model arbitrary environments without prior assumptions about it. The representation models occupied areas as well as free space. Unknown areas of the environment are implicitly encoded in the map. While the distinction between free and occupied space is essential for safe robot navigation, information about unknown areas is important, e.g., for autonomous exploration of an environment.

It is possible to add new information or sensor readings at any time. Modeling and updating is done in a probabilistic fashion. This accounts for sensor noise or measurements which result from dynamic changes in the environment, e.g., because of dynamic objects. Furthermore, multiple robots are able to contribute to the same map and a previously recorded map is extendable when new areas are explored. The extent of the map does not have to be known in advance. Instead, the map is dynamically expanded as needed. The map is multi-resolution so that, for instance, a high-level planner is able to use a coarse map, while a local planner may operate using a fine resolution. This also allows for efficient visualizations which scale from coarse overviews to detailed close-up views. The map is stored efficiently, both in memory and on disk. It is possible to generate compressed files for later usage or convenient exchange between robots even under bandwidth constraints.The map is a central component of any autonomous system because it is used during action planning and execution. For this reason, the map needs to be efficient with respect to access times but also with respect to memory consumption. From a practical point of view, memory consumption is often the major bottleneck in 3D mapping systems. Therefore, it is important that the model is compact in memory so that large environments can be mapped, a robot can keep the model in its main memory, and it can be transmitted efficiently between multiple robots.

In autonomous navigation tasks, a robot can plan collision-free paths only for those areas that have been covered by sensor measurements and detected to be free. Unmapped areas, in contrast, need to be avoided and for this reason the map has to represent such areas. Furthermore, the knowledge about unmapped areas is essential during exploration.When maps are created autonomously, the robot has to plan its actions so that measurements are taken in previously unmapped areas.

An octree is a hierarchical data structure for spatial subdivision in 3D (Meagher, 1982; Wilhelms and Van Gelder,1992). Each node in an octree represents the space contained in a cubic volume. This volume is recursively subdivided into eight subvolumes until a given minimum size is reached. The minimum size determines the resolution of the octree. Since an octree is a hierarchical data structure, the tree can

be cut at any level to obtain a coarser subdivision if the inner nodes are maintained accordingly. If a certain volume is measured as occupied, the corresponding node in the octree is initialized. Any uninitialized node could be free or unknown in this Boolean setting. To resolve this ambiguity, we explicitly represent free volumes in the tree. These are created in the area between the sensor and the measured end point, Areas that are not initialized implicitly model unknown space.Using occupancy states or discrete labels allows for compact representations of the octree: If all children of a node have the same state (occupied or free) they can be pruned.



In Octomap approach, sensor readings are integrated using occupancy grid mapping as introduced by Moravec and Elfes (1985). The probability P(n|z1:t) of a leaf node n to be occupied given the sensor measurements z1:t is estimated according to P(n|z1:t)

$$P(n \mid z_{1:t}) = \left[1 + \frac{1-P(n \mid z_t)}{P(n \mid z_t)} \frac{1-P(n \mid z_{1:t-1})}{P(n \mid z_{1:t-1})} \frac{P(n)}{1-P(n)}\right]^{-1}$$

This update formula depends on the current measurement zt, a prior probability P(n), and the previous estimate P(n|z1:t-1). The term P(n|zt) denotes the probability of cube n to be occupied given the measurement zt. This value is specific to the sensor that generated zt.The common assumption of a uniform prior probability leads to P(n)=0.5 and by using the log-odds notation can be rewritten as,

$$\mathrm{L}(n \mid z_{1:t}) = \mathrm{L}(n \mid z_{1:t-1}) + \mathrm{L}(n \mid z_t),$$

with

$$\mathrm{L}(n) = \log\left[\frac{P(n)}{1-P(n)}\right].$$

This formulation of the update rule allows for faster updates since multiplications are replaced by additions. In case of pre-computed sensor models, the logarithms do not have to be computed during the update step. Note that log-odds values can be converted into probabilities and vice versa and we therefore store this value for each voxel instead of the occupancy probability. It is worth noting that for certain configurations of the sensor model that are symmetric, i.e., nodes being updated as hits have the same weight as the ones updated as misses, this probability update has the same effect as counting hits and misses similar to Kelly et al. (2006).

When a 3D map is used for navigation, a threshold on the occupancy probability P(n|z1:t) is often applied. A voxel is considered to be occupied when the threshold is reached and is assumed to be free otherwise, thereby defining two discrete states. It is evident that to change the state of a voxel we need to integrate as many observations as have been integrated to define its current state. In other words, if a cube

was observed free for k times, then it has to be observed occupied at least k times before it is considered occupied according to the threshold (assuming that free and occupied measurements are equally likely in the sensor model). While this property is desirable in static environments, a mobile robot is often faced with temporary or permanent changes in the environment and the map has to adapt to these changes quickly. To ensure this adaptability, Yguel et al. (2007a) proposed a clamping update policy that defines an upper and lower bound on the occupancy estimate. Occupancy estimates are updated according to

$$\begin{aligned} &\mathrm{L}(n \mid z_{1:t}) \\ &\quad = \max(\min\left(\mathrm{L}\left(n \mid z_{1:t-1}\right) + \mathrm{L}\left(n \mid z_t\right), l_{\max}\right), l_{\min}), \end{aligned}$$

where lmin and lmax denote the lower and upper bound on the log-odds value. Intuitively, this modified update formula limits the number of updates that are needed to change the state of a voxel. Applying the clamping update policy in our approach leads to two advantages: we ensure that the confidence in the map remains bounded and as a consequence the model can adapt to changes in the environment quickly. Furthermore, we are able to compress neighboring voxels with pruning. This leads to a considerable reduction in the number of voxels that have to be maintained. The compression achieved with clamping is no longer completely lossless in terms of the full probabilities, since information close to zero and one is lost. In between the clamping thresholds, however, full probabilities are preserved.

Using probabilistic occupancy estimation, Octomap approach is able to include free and unknown areas and explore the environment without using any map.

2 Related Work

As already alluded to, object and scene recognition, and related visual tasks, have been a very active area of research in the mobile robotics community for the last few decades. The direction of research has somewhat seen a paradigm shift since 2012 however, which was the year when *AlexNet* destroyed its competition in the *LSVRC* (2017 *Huang*, 2015 *Ren*). Many techniques for understanding visual information have gradually started to adopt CNN based architectures. Visual based methods are used to estimate probability distributions for particle filtering, which is a powerful method for robot localization and visual tracking (2018 *Karkus*). CNNs have become very important in terrain estimation and obstacle avoidance (2018 *Zhang*). Autonomous vehicles use object detection networks to understand their environments, plan trajectories, and make decisions. A combination of CNN+RNN architectures have been explored for trajectory planning and robotic steering. An emerging line of research has also seen integration of algorithmic priors into deep learning architectures, where end-to-end systems have been trained that tie together multi-modal input features and various domain specific sub-modules (2018 *Karkus*).

The reinforcement learning community also experienced a revolution in 2013 when researchers from the English start-up Deepmind demonstrated that an agent could learn to play almost any Atari game using only raw pixels as input, and without any prior knowledge of the rules of the games. This has been followed up by many achievements, including the victory of their system *AlphaGo* against the world champion of *Go* in 2017 (2017 *Geron*). Although these successes have yet to translate to industrial applications in the real world, they have demonstrated an extremely

important concept. *If given enough data, an agent can learn superhuman ability in a reasoning and decision-making task*. There is currently an enormous amount of funding for reinforcement learning in the robotics community (2018 *Irpan*).

3 Contributions

- We show that augmenting autonomous exploration with convolutional feature vectors can lead to more efficient exploration of an unknown environment. Specifically, we implement an "external memory unit" that holds feature vectors (scene encodings) that are generated from various locations in the Turtlebot's environment. By comparing feature vectors generated from the robot's current pose to feature vectors that have already been stored in this memory unit, the robot can quickly determine whether or not it has previously explored the current location.
- We provide an easy-to-use ROS package that integrates Object Detection (Tensorflow Object Detection API) with autonomous exploration through an unknown environment.

4 Experiments and Methods

We set up a multi-room environment and placed the Turtlebot randomly in one of the rooms. We then placed a target object in another room (there were some constraints on the choice of the target object, mainly that it had to be one of the objects included in the training set for our pre-trained object detection network). The goal was for the Turtlebot to locate this object as quickly as possible.

This required the robot to autonomously explore its environment and reason about the likelihood of the object being in different locations. It generated control signals to move in directions where it believed to be most promising.

There are a number of methods that theoretically can be used to give the Turtlebot these 'abstract reasoning' capabilities (note that we understand that the robot does not truly have abstract reasoning capabilities in a human sense; what we mean here is that it will understand its context in a statistical manner). The following methods were all thoroughly considered: (1) Model the problem as a reinforcement learning problem, where the robot is the agent, the actions are control signals, the states are pixels from RGB-D images, and the reward is a -1 for each time step that the robot does not find the object. (2) Build an RNN (LSTM/GRU), where the inputs states at each time step are the RGB-D image feature vectors (after being passed through a CNN) and control signals from the previous time step, and the output predictions are control signals for the current time step. We also considered merging domain specific knowledge to help the model learn, such as the particle filter algorithm on laser scan data. (3) Combine the understanding from an object detection network with some human crafted heuristics, and store an abstract feature vector in an "external memory unit" for later retrieval and comparison to the robot's current contextual environment.

For all cases, we considered bootstrapping simulation environments for training data. If done, we would use feature level domain adaptation to force the model to extract discriminatory features that existed in both the simulated environment and real environment, but were not exclusive to either one (2017 *Bousmalis*).

For reasons that will be described below, the only feasible solution for our semester project was (3). If you don't mind a bit of anthropomorphism, think of this "external memory unit" as a form of long term memory. When humans move throughout an environment, we are able to store away abstract concepts of scenes in our minds. We are then able to retrieve and reason about these concepts at a later time, which ultimately leads to better decision making. We hoped to replicate this behavior in our solution.

This idea of using an "external memory unit" is a somewhat generalized form of a recurrent network. It more closely resembles the work done by Deepmind on Differential Neural Computers (2016 *Graves*), although on a much simpler scale. Think of it as a memory augmented neural network.

At first, we considered loading in two pre-trained CNNs during Turtlebot exploration; one for object detection purposes and the other for generation of these scene feature vectors. However, real-time requirements limited us to only be able to use one network. Therefore, we used an object detection network for both purposes, simply taking vectors from one of the higher layers to send to the memory unit.

When deciding on which network to use, we considered two characteristics: latency and accuracy. Our plan was to test a number of networks, starting with the one which contained the lowest latency, and stopping when we were comfortable with the accuracy that it provided. Truthfully, even the smallest and least accurate network that we tried still gave us very reasonable confidence in its localization and classification of objects in the scene. We decided to use *ssd mobilenet v1*, trained on the COCO Dataset (2015 *Lin*).

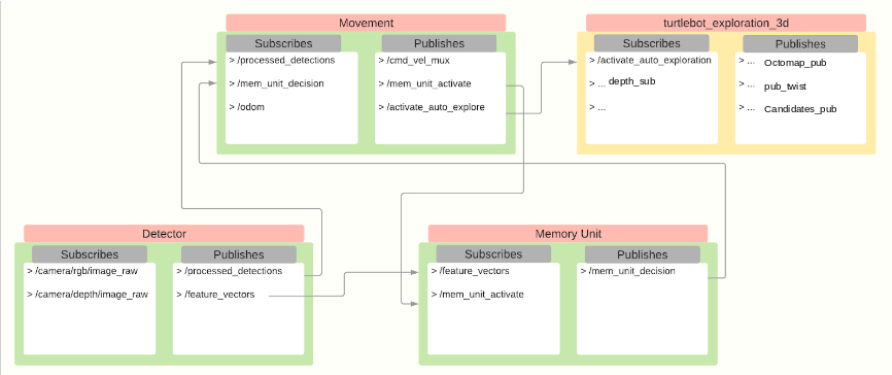
We intended to compare the time that it took the Turtlebot to find the target object, with and without the use of the memory unit. Our hope was that the memory unit would enable the Turtlebot to avoid retracing its steps in areas where the target object did not exist (and be able to do this with one glimpse of a scene location). To simplify our setup while still demonstrating our concept, we only activate the memory unit upon exiting and entering a room. When crossing a doorway, the Turtlebot generates 10 feature vectors of the room at slightly different angles. It then compares these feature vectors to those that exist in the memory unit. If a match is found, the Turtlebot can quickly determine that it should not explore this room again. These incoming feature vectors will then be stored in the memory unit for future reference.

Ideally, we would have liked to use some form of attention mechanism and sequential probability to implement our writing/reading of these feature vectors to the memory unit (2015 *Vaswani*). However, this was not able to be explored during the time allotted during the semester.

5 Application Architecture and Software Stack

Our application was implemented using both Python and C++ for ROS-Kinetic. It is composed of three nodes that were developed entirely by us (*Detector*, *Memory Unit*, *Movement*), and one community contributed package that was slightly modified to fit our needs (*turtlebot-exploration-3d*). Google's Tensorflow library was used for object detection.

A high level overview of our architecture is illustrated in the following diagram.



The *Detector* node is responsible for both object detection and the generation of scene feature vectors. This node subscribes to the '/camera/rgb/image-raw' and '/camera/depth/image-raw' topics. It first passes the RGB image through the pre-trained *ssd mobilenet v1* network and collects the confident predictions of objects and their locations in the field of view of the robot. It then aligns the depth image with this RGB image, and finds the approx distance of each object that is recognized. All of this information is packaged up in an easy to interpret format and published on the '/processed-detections' topic. At the same time, this node continually publishes each images feature encoding on the '/feature-vectors' topic.

The *Memory Unit* node is responsible for saving and comparing feature vectors. It subscribes to both the '/feature-vectors' topic that is published by the *Detector* node and the '/mem-unit-activate' topic that is published by the *Movement* node. When the *Memory Unit* is activated by the '/mem-unit-activate' topic, it begins its processing. It compares all incoming feature vectors to those already in the unit. If the Euclidean distance between the incoming feature vectors and any of those in its memory is below a certain threshold, then this Node publishes the String 'Already Explored' to the '/mem-unit-decision' topic. If none of the incoming feature vectors are close to those already in the memory bank, then the String 'Enter Room' is published to the '/mem-unit-decision' topic.

The *Movement* node is responsible for moving the Turtlebot in response to information generated by both the *Detector* and *Memory Unit* nodes. It subscribes to the following topics: '/processed-detections', '/mem-unit-decision', and '/odom'. When the target object is contained in the messages produced by the '/processed-detections' topic, the *Movement* node uses the location and depth information to move towards the object (does so by publishing to '/cmd-vel-mux/'). In addition, when a 'door way' is detected by the *Detection* node, the *Movement* node positions the Turtlebot to face into the room. However, before this node actually performs any movement, it needs to send a message to the *turtlebot-exploration-3d* node telling it to pause the autonomous navigation. It does this by sending the String 'SLEEP' to the topic '/activate-auto-explore'.

The *turtlebot-exploration-3d* controls autonomous navigation throughout an unknown environment. The majority of this node's functionality was not implemented by us. However, we did add additional subscribers and publishers that controlled correspondence with the *Movement* node.

6 Results & Evaluation

We successfully have an application that autonomously explores the environment, searches for a target object, and moves towards that object when it comes within the field of view of the Turtlebot. In addition, scene encodings are sent to the memory unit during exploration, and they are used to aid the Turtlebot in its decision making when deciding where to explore next. Unfortunately, at the time of this writing, we have not yet run a complete, timed, end-to-end trial of the memory augmented autonomous navigation on the Turtlebot. Although we put a lot of effort into getting the system to where it is, there is still one specific area that needs a little bit of work: positioning the Turtlebot accurately in the doorway to compare scene encodings from one room to the next. We chose to use this scenario of solely comparing the feature vectors in the doorway because there is a clear, easy notion of what the Turtlebot should do from that point: if the Turtlebot had already been in that room, turn around and go in another direction, while if the Turtlebot had not been in that room, enter the room and explore. If you think about a strategy where we would instead make scene comparisons in the memory unit every few seconds, this would get much more complicated. There would not always be a simple, generalizable, clear answer of what the Turtlebot should do even if it knows that it had been in that pose before, and there are many edge cases to consider. We wanted to try and stay away from as much human intervention as possible when guiding the Turtlebot, so devising a convoluted hand crafted solution to this scenario would be tedious and would somewhat miss the point of what we were trying to show. Further research on using LSTM networks to directly output control signals from raw image data should be explored for a scenario where we use a 'memory network' to continuously inject information into the robot's decision making.

With that being said, we do have strong evidence that our strategy would have offered improved performance. We manually placed the Turtlebot in the doorway of 3 different rooms, took 20 feature vectors while slowly rotating the Turtlebot in each of the doorways, and compared the average Euclidean distance of all pairs. The results are shown in the below table.

Average Euclidean Distance

	Room 1	Room 2	Room 3
Room 1	9.1	13.3	11.9
Room 2	13.3	9.5	12.7
Room 3	11.9	12.7	8.6

As can be seen, feature vectors taken from multiple angles of the same room have a much smaller Euclidean distance between them as compared with feature vectors taken from two different rooms (we tried multiple layers of *ssd mobilenet v1* to see which gave us the most convincing evidence). While this may seem trivial on the surface, it must be understood that to a computer, even a slight change in orientation

can produce a completely different representation of the raw image data (pixel values). To the Turtlebot, a 640x480 RGB-D image is viewed as a grid of 1,228,800 pixels. When we generate a feature vector from this image, we transform it into a 128 dimensional space. This feature vector is obviously capturing information that can differentiate between certain characteristics of the three rooms that we tested.

In the future, to make these scene encodings even more useful, we can employ a technique used heavily in facial recognition software – use a siamese network with a triple loss function to optimize the network for capturing semantic differences in indoor scenes. Simply, we would gather a large dataset of indoor scenes, label each image with the room or location that it was taken, and present the network with a pair of images from the same room and a pair of images from different rooms. The network would be trained to minimize the distance between vectors produced by images from the same room and maximize the distance between vectors produced by images from different rooms. Although they still showed promise, it must be remembered that the feature vectors that we actually used for this project were taken from a network that was pretrained specifically for object detection on the COCO dataset. While the higher levels of this network still provide abstract concepts such as shapes and objects in the image, it is not optimized to find the latent variables that differentiate one pose from another.

7 Group Member Participation

Both group members collaborated on all aspects of the project and really enjoyed working together.

References

[1] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, Wolfram Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *University of Freiburg, Baden-Wurttemberg, Germany*

[2] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard "Multimodal deep learning for robust rgb-d object recognition," *International Conference on. IEEE*, (2015) 681-687

[3] Jahanzaib Shabbir, Tarique Anwer, "A Survey of Deep Learning Techniques for Mobile Robot Applications," *CoRR*,abs/1803.07608

[4] Peter Karkus, David Hsu, Wee Sun Lee, " Particle Filter Networks: End-to-End Probabilistic Localization From Visual Observations,"*CoRR*, (2018) abs/1805.08975

[5] Serj M.F., Lavi B., Hoff G., Valls D.P, "A Deep Convolutional Neural Network for Lung Cancer Diagnostic," *CoRR*, abs/1804.08170

[6] Li W., Saeedi S., McCormac J., Clark R., Tzoumanikas D., Ye Q., Huang Y., Tang R., Leutenegger S., "InteriorNet: Mega-scale Multi-sensor Photo-realistic Indoor Scenes Dataset,"*BMVC*, (2018)

[7] Bousmalis K., Irpan A., Wohlhart P., Bai Y., Kelcey M., Kalakrishnan M., Downs L., Ibarz J. Pastor P., Konolige K., Levine S., Vanhoucke V. , "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping,"*IEEE International*

Conference on Robotics and Automation(ICRA),(2018)4243-4250

[8] Huang J., Rathod V., Sun C., Zhu M., Balan A.K., Fathi A., Fischer I., Wojna Z., Song Y., Guadarrama S., Murphy, K. , "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors,"*IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*,(2017)3296-3297

[9] Liu W., Anguelov D., Erhan D., Szegedy C., Reed S.E., Fu C., Berg A.C., "SSD: Single Shot MultiBox Detector,"*ECCV*,(2016)

[10] Ren S., He K., Girshick R.B., Sun J., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,"*IEEE Transactions on Pattern Analysis and Machine Intelligence* ,39 (2015) 1137-1149

[11] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I., "Attention Is All You Need,"*NIPS* ,(2017)

[12] Graves A., Wayne G., Reynolds M., Harley T., Danihelka I., Grabska-Barwinska A., Colmenarejo S.G., Grefenstette E., Ramalho T., Agapiou J., Badia A.P., Hermann K.M., Zwols Y., Ostrovski, G., Cain A., King H., Summerfield C., Blunsom P., Kavukcuoglu K., Hassabis D., "Hybrid computing using a neural network with dynamic external memory,"*Nature* ,538 (2017) 471-476

Rosenberg
Department of Computer Science
State University of New York at Binghamton
Binghamton NY 13905
USA
jrosen46@binghamton.edu

Bhamare
Department of Computer Science
State University of New York at Binghamton
Binghamton NY 13905
USA
sbhamar1@binghamton.edu