Using Deep Convolutional Neural Networks to Abstract Obstacle Avoidance for Indoor Environments

Mohammad O. Khan
Department of Computer Science
Connecticut College
New London, CT, USA
mkhan4@conncoll.edu

Gary B. Parker
Department of Computer Science
Connecticut College
New London, CT, USA
parker@conncoll.edu

***Abstract*—In this paper, an approach to learning an obstacle avoidance program for an autonomous robot is presented. A deep learning network, which matches one that was successfully used in the past for a classification task, was replicated and used to classify ten categories in the CIFAR10 dataset. This trained network was then altered by replacing the final fully-connected feed-forward network with a new one that was initiated with random weights. Using a new database made up of images labeled with the actions taken by an operator as he remotely drove the robot, the network learned the proper action for each image. In previous work, we reported that this network operating on the actual robot successfully moved through the desired path in the training environment while avoiding obstacles. Now we have expanded this work by showing that the obstacle avoidance control program is generalized enough that it was successful when tested in three environments not seen during training.**

Keywords—mobile robotics, obstacle avoidance, deep learning, artificial neural networks, indoor, TurtleBot

I. INTRODUCTION

Learning systems for autonomous robots that allow them to adapt to new environments by helping them move through spaces while avoiding obstacles would help reduce development time and increase the efficacy of the robots. In this paper, the learning system is used to learn an obstacle avoidance system, which will make up one part of an overall control system for the robot. As the robot operates in its environment performing the desired operation through the task control system, the obstacle avoidance system overrides other control commands being made in order to avoid collisions. For this research, the task control system moves the robot forward at a constant speed, while the obstacle avoidance system keeps the robot from colliding with any obstacles in the path.

Deep learning, which involves the use of massive artificial neural network architectures, is a reasonable approach for robots equipped with cameras. One of the main functions assigned to deep learning models is image processing. Some of these architectures are capable of outperforming humans in tasks like classifying objects – differentiating one object type from another, such as dog versus wolf. In addition to identifying specific types of animals / objects, which is a skill that even many humans struggle with, deep learning systems can identify, with some level of certainty, which objects exist within an image. In this paper, we present previous work in which we used deep learning to develop a TurtleBot type robot that can autonomous drive within a tight classroom / laboratory setting based strictly on images. In addition, we present new work that shows that not only is the robot able to successfully and autonomously drive without hitting obstacles within the original room, but it is also able to avoid obstacles in other environments such as: the hallways of a building, a completely different room with a different set up (different design of chairs, wall colors, etc.), and a blocked off carpeted area with wooden blocks it has never seen before. Additionally, the robot abstracted the concept of being able to navigate around human legs.

Much research has been done using TurtleBot robots. Some of this research has considered applications of TurtleBot as it interacts with humans. Hotze [1] used the TurtleBot platform to develop a robotic framework for identifying anomalies based on the detection of unconscious humans. Mannequins were used in substitution of humans to test the system and the robot approached fallen mannequins from different positions in a mapped environment. These "humans" were detected based on temperature, breath, and face detection. Significant additions to the TurtleBot platform were made for this task. An Arduino, two Adafruit motor shields, a robot arm, a webcam, a temperature sensor, and a breath sensor were attached to the body of the TurtleBot. In other human interaction research, which was appropriate for the TurtleBot, but used another platform, Correa et al. [2] used thermal images to identify humans along with raw visual images of faces. The approach used was a nearest neighbor classifier that utilized similarity measurements like histogram intersection. The robot was used to detect particular humans in a waiting room. Gritti et al. [3] developed a Kinect camera based approach to detecting and tracking humans. Detecting humans is not a trivial task for a robot close to the ground. Either the camera needs to be tilted upward or a camera needs to be mounted on a rod. Gritti et al. focused on identifying "leg-like" objects that protruded from the ground. This is a natural approach to take considering that the TurtleBot is a short robot with a limited view from the ground. They developed a statistical classifier to differentiate between legs and other objects protruding from the ground plane. There has been strong interest in using the TurtleBot platform for obstacle detection and avoidance. Boucher used the Point Cloud Library and depth information along with plane detection algorithms to build methods of obstacle avoidance [4]. High

curvature edge detection was used to locate boundaries between the ground and objects that rest on the ground.

Object recognition is an important task that has been tackled by the Machine Learning community. Deep learning has outperformed many other algorithms in this arena. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) has attracted and pushed many researchers to develop advanced algorithms. This challenge includes 1000 different categories of images that are to be classified into categories. The training set by itself is about 1.2 million images, with 150,000 testing images used to benchmark during training and 50,000 images for validation, which are used to test the system after training is complete [5].

A very influential paper in regards to deep learning was published in 2012 by Krizhevsky, Sutskever, and Hinton [6]. This neural network had 60 million parameters and 650,000 neurons, which were part of five convolutional layers alongside pooling layers and three fully connected layers including a final output layer of 1000 outputs. At the time, they achieved a top-5 classification (of the 1000 classes) error rate of only 15.3% compared to a much higher second-place error rate of 26.2% (top-5 means that one of the system's top five predictions to identify the object was correct). This paper contributed to the discussion of the importance of depth in neural networks by noting that removal of a single hidden layer dropped the top-1 classification error rate by 2% (top-1 means that the system's top prediction to identify the object was correct).

In 2014, Szegedy et al. [7] entered the ILSVRC challenge with a 22 layer deep network that won the competition with 12 times fewer parameters than Krizhevsky's deep network. The team obtained an impressive 6.66% error rate for top-5 classification. In 2015, He, Zhang, Ren, and Sun of Microsoft Research surpassed the previous team with a 19 layer deep neural network for the task and obtained an accuracy of 4.94% for top-5 classification [8]. This was a landmark accomplishment as it was the first to beat the human level performance of 5.1% for the ImageNet dataset.

The most relevant deep learning research for our work is that of Alex Krizhevsky where he worked on the CIFAR10 dataset from the Canadian Institute for Advanced Research [9]. Prior to this, tiny images on the scale of 32 x 32 were not easily labeled for classification tasks in regards to algorithms like deep learning. The CIFAR10 dataset includes 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The classes are set up in a way to be mutually exclusive. For example, automobile and truck are completely different categories. Krizhevsky worked on this dataset in 2009 [10] and developed different deep neural network models in 2010 to run training with the dataset. At the time, he obtained the highest accuracy using this dataset as his best model classified objects correctly with a success rate of 78.9% [11]. Since then, Mishkin and Matas have obtained 94.16% accuracy on the CIFAR10 dataset [12] and Springenberg et al. have obtained 95.59% accuracy [13]. The current best performance is by Graham with an accuracy of 96.53% using max pooling [14].

Researchers have considered the use of deep learning for the purpose of obstacle avoidance using the TurtleBot platform. Tai, Li, and Liu used depth images as the only input into the deep network for training purposes [15]. They discretized control commands with outputs such as: "go-straightforward", "turning-half-right", "turning-full-right", etc. The depth image is from a Kinect camera with dimensions of 640 x 480. This image was downsampled to 160 x 120. Three stages of processing were completed where the layering is ordered as such: convolution, activation, pooling. The first convolution layer uses 32 convolution kernels, each of size 5 x 5. The final layer includes a fully-connected layer with outputs for each discretized movement decision. In all trials, the robot never collided with obstacles, and the accuracy obtained after training in relation to the testing set was 80.2%. Their network was trained on only 1104 depth images. This hints at the idea that maybe many edge cases are missing or that this dataset is specific to only one environment. The environment used in this dataset seems fairly straightforward – meaning that the only "obstacles" seem to be walls or pillars. Tai and Liu produced another paper related to the previous paper [16]. Instead of a real-world environment, this was tested in a simulated environment provided by the TurtleBot platform, called Gazebo. Different types of corridor environments were tested and learned. A reinforcement learning technique called Q-learning was paired with the power of deep learning.

The research reported in this paper describes an approach to learning an obstacle avoidance system by training and leveraging a generalized image classification deep learning network and repurposing it for obstacle avoidance. Specifically, the final fully-connected feed-forward network layer is altered to form a new one. The new network layer is initiated with random weights and then the entire learning system is trained on a new database. The new database of images was derived from a training process involving a robot operator driving the robot and labeled each snapshot of the drive with a movement option. The network now operates as an obstacle avoidance system, which takes proper action in the trained space. In addition, it has sufficiently abstracted the concept of an obstacle so that it can avoid obstacles in new and distinct environments with obstacles never encountered in the training environment.

Our approach differs from the two Tai et al. papers and that of Boucher's obstacle avoidance work. Research like Boucher's does not consider higher level learning, but instead builds upon advanced expert systems, the likes of which can detect differentials in the ground plane. By focusing on deep learning, our research allows a pattern based learning approach that is more general and which does not need to be explicitly programmed. The logic structure for obstacle avoidance can be fairly complex depending on the variability in the environment. While Tai et al. used deep learning, their dataset was limited with just over 1100 images. We built our own dataset to have over 30,000 images, increasing the size of the effective dataset by about 28 times. The environment for our research is much more complex than just the flat surfaces of walls and columns. In addition, for our research, we developed a dataset that was based on raw RGB images, opening the door to further research with cameras that do not have depth. Moreover, the sizes of the images used in this work were dramatically smaller, which also opens up the door for faster training and a speed up in forward propagation. Lastly, the results of this research are solely tested in the real world -- a simulated environment is not used. In

addition, our learned controller was able to avoid obstacles in environments other than the training environment.

II. TurtleBot Platform

While the robot (which can be seen in Fig. 2) used for this research was not marketed as one that is based on the TurtleBot framework, its functionality is essentially equivalent to that of the TurtleBot platform. The research robot was marketed as a "Deep Learning Robot" from the robotics company, Autonomous AI [17]. The robot includes an Asus Xtion Pro 3D Depth Camera, a microphone embedded in the camera, and a speaker. A Kobuki mobile base allows it to rotate and move in any direction on the ground plane. Most importantly, it is equipped with an Nvidia Tegra TK1, which allows it to carry out deep learning computations on a GPU instead of having to resort to long wait times for training with a CPU. This is its main differentiator from a regular TurtleBot. While the Tegra TK1 was marketed as the world's most powerful mobile processor, it only has 2GB of memory. This is problematic for training very deep networks and holding too many parameters in memory causes the robot to crash. While training, the robot is unstable because of this limited memory so running multiple programs at the same time has to be avoided.

This robot is virtually a computer in itself, and it allows us to treat it as such as it is very compatible with Ubuntu 14.04. Wi-Fi and Bluetooth give us the flexibility of setting up network connections and connecting devices for communication, which we may consider in the future. The TurtleBot framework works hand in hand with the Robot Operating System (ROS). Google TensorFlow, Torch, Theano, and Caffe are all compatable deep learning frameworks. CUDA and cuDNN are available for implementing deep learning on GPUs and for speeding up that computation.

III. Deep Learning

A. Artificial Neural Networks

A deep learning network is a type of Artificial Neural Network (ANN). The typical ANN has interconnections of single neurons linked together by weighted connections through which information is transferred. The idea is to create a biologically inspired control / learning system.

These neurons are stacked in multiple layers where different layers of neurons serve different functions. There is an input, which goes to the first layer of neurons and the output of these neurons becomes the input for other neurons in a subsequent layer. In fully connected models, every neuron in the previous layer is connected to every neuron in the next layer. A threshold value (θ) symbolically represents the activation potential of a biological neuron. A weighted sum of all the inputs is compared to this value to determine an output after it is fed into an activation function, which in our research is the sigmoid function.

B. Deep Learning Network

The ANN mode of architecture is excellent for many applications, but has limitations when dealing with large images, which was one of the main motivations for the development of deep learning. Instead of sending all input values from layer to layer, deep networks take regions or subsamples of inputs. Instead of sending all pixels in the entire image as inputs, different neurons will only take regions of the image as inputs. Essentially, full connectivity is reduced to local connectivity. In some ways, this model is more biologically inspired than ANNs because it better models the visual cortex. Neurons in the visual cortex are activated by stimuli in a specific location based manner. This means that neurons that are linked as direct neighbors share the saturation from an impulse [18].

1) Convolution: This method of subsampling used in deep learning is called convolution. The local receptive fields can be seen as small windows, with a predefined number of panes, that slide over the image. These panes help determine what features under the window we want to extract, and over time these features are better refined. The weighted windows are commonly called kernels. The kernel defines how the image gets subsampled. Matrix operations use the weights of the kernel and the values of the image to produce the parts of a newer, smaller image by placing the product of the operation in a correlated region in the new image. Depending on the type of kernel, different features of the image may be highlighted, which leads to different results, such as blurring and sharpening. In this way, networks can develop identification of complex patterns in datasets just by applying kernel filters. Deep learning networks are able to develop these kernels through training -- supervision is from a loss function in the output layer determining how close the network's prediction was to the actual value of the image. Through training, these kernels become more fine-grained to reduce the loss function's output.

2) Pooling: Deep networks may include many different types of layers. A general strategy is to follow a convolution layer with a pooling layer. The convolution layer is responsible for learning the lower level features of an image, such as edges, and the pooling layer seeks to detect higher-level features, such as corners where two edges meet. Pooling is also good for building translational invariance, where the system can detect an object even if it moves to a different part of the image in a subsequent frame. In this way, dominant features persist deeper into the network. In addition, pooling can help in dramatically reducing the size of an image by down sampling a rectangular region of pixels in one of three ways: max pooling (choose the maximum pixel value), min pooling (choose the minimum pixel value), and Average pooling (choose the average pixel value). Reducing the size of the image significantly cuts down on the time for processing an image. The convolution layer passes convolution windows over the image to produce new images that are smaller. The number of images (each accompanied by a convolution kernel signifying the weights) produced can be specified by the programmer. Pooling is then applied to each one of these new images. Convolution and pooling are the most prominent deep learning layers, but there were other types of layers that we used in this research.

3) Rectified Linear Unit: The Rectified Linear Unit (RLU) layer involves returning a numerical input if it is positive, otherwise multiplying the input by -1. This effectively eliminates negative inputs and boosts computation time over other functions such as the sigmoid activation function since no exponentiation and computations are required. Alex Krizhevsky et al. were able to accelerate convergence in their

training by a factor of six times in relation to the sigmoid activation function using this function [6].

4) Local Response Normalization: The Local Response Normalization layer imitates biological lateral inhibition, which is a situation in which excited neurons have the capability and often tendency of subduing neighbor neurons [19]. This results in an amplification when there is a differential in neuron excitement. These layers allow neuron's with large activation values to be much more influential than other neurons, which results in the survival of significant features from one layer into deeper layers of the network.

5) Fully Connected Layer: The fully connected layer that we used is like that of a layer in a regular ANN. This was used as the final layer of the network so the outputs of the neurons in this layer were the actual outputs of the network. These outputs are compared to the desired outputs to compute the loss with the learning dependent on gradient descent updates.

IV. Deep Learning in the Learning Environment

The problem addressed was that of training a deep neural network to control an autonomous vehicle driving in a tight, chaotic room/lab environment. To test the functionality and success of the program, the performance of the robot was checked to see if it could autonomously follow an approximately rectangular path in a tight environment without colliding into obstacles.

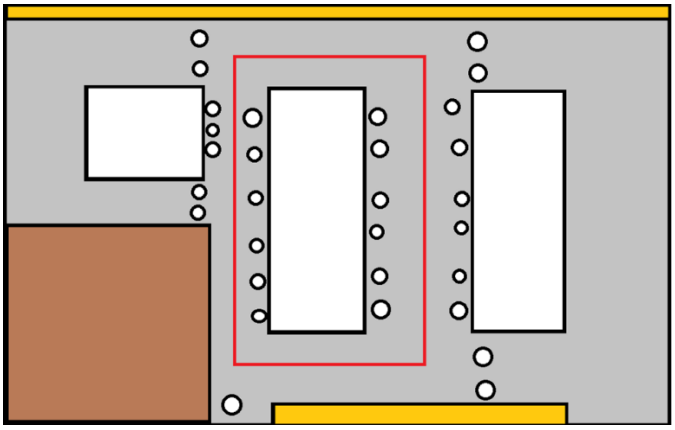

Fig. 1. A visual of the environment with lab tables, chairs, and cabinets. Images are provided below to help understand this environment even more.

A. Environment

The training/testing environment (drawing in Fig. 1) was set up in a robotics lab. The white circles with dark borders represent chairs. The radii of the chairs are larger than the circles show because the feet of the chairs extend further such that there is no gap for the robot to move in between neighboring chairs (in most cases). The white rectangles with dark borders are lab tables, which are solid sides on the ends, but not along the long sides. They are tall enough for the robot to be able to drive through and for this reason, chairs were placed in those locations. The dark brown rectangle is a boxed off area in the lab that is used for other experiments, but in this research acts as borders that the robot would need to avoid hitting. The golden rectangles denote cabinets, which the robot must also avoid. The red rectangle in the middle of the figure shows the general path that the robot must follow to avoid chairs, tables, and other obstacles. During training and tests we ran the robot on this path in both the clockwise and counterclockwise directions.

Fig. 2 shows some photographs of areas in the environment. The intent was to create a complex testing area by the placement of the obstacles. One can see from Fig. 2 that the gaps outside of the desired path were closed with the adjustable round chairs, which each have 5 rounded legs and a circular stump. The chair heights can be adjusted and the orientation can change 360 degrees for both the base and the actual seating. The chairs were chosen as the main objects of interest because, due to the legs, there is a good amount of gap area in between the objects. This allows for complexity in defining what an obstacle is and what an obstacle is not. The deep learning system cannot simply learn to follow the color of the carpet because even the gaps reveal the carpet. There was also a dynamic nature to the environment since students often moved the chairs each night. While the chairs might be in the same relative location, the orientations were completely different each time. This added to the complexity of the problem since it is not easy for a pattern to be developed when the orientation of objects keeps changing.

Although we wanted sufficient complexity to ensure this was a difficult task, there were two situations in the environment that would be impossible for the robot to solve. In the first, if there is enough of a gap between two chairs the robot may make the decision to go straight instead of turning away from the chairs. Therefore, the chairs were placed close enough so that the gap was smaller than the robot. In the second, if the robot is heading directly toward a cabinet and close enough so that it only sees the cabinet, it won't know which way to turn. Even for a human with limited peripheral vision, it would be impossible to know which direction to turn. To solve this issue, areas with cabinets included an open cabinet that swiveled to a direction the robot was supposed to avoid. These changes established rough guidelines as to the correct path for the robot, plus since the contents of the cabinets added a variety of different items for the visual system to process, these changes added complexity to the environment.

Fig. 2. The images above demonstrate various obstacle avoidance scenarios in the training environment.

Chairs, cabinets, and tables were not the only obstacles to avoid. A good amount of the dataset included the borders of a colony space environment and a few images in the dataset included small cardboard boxes. It was important to include obstacles like this in order to confirm that the concept of obstacle

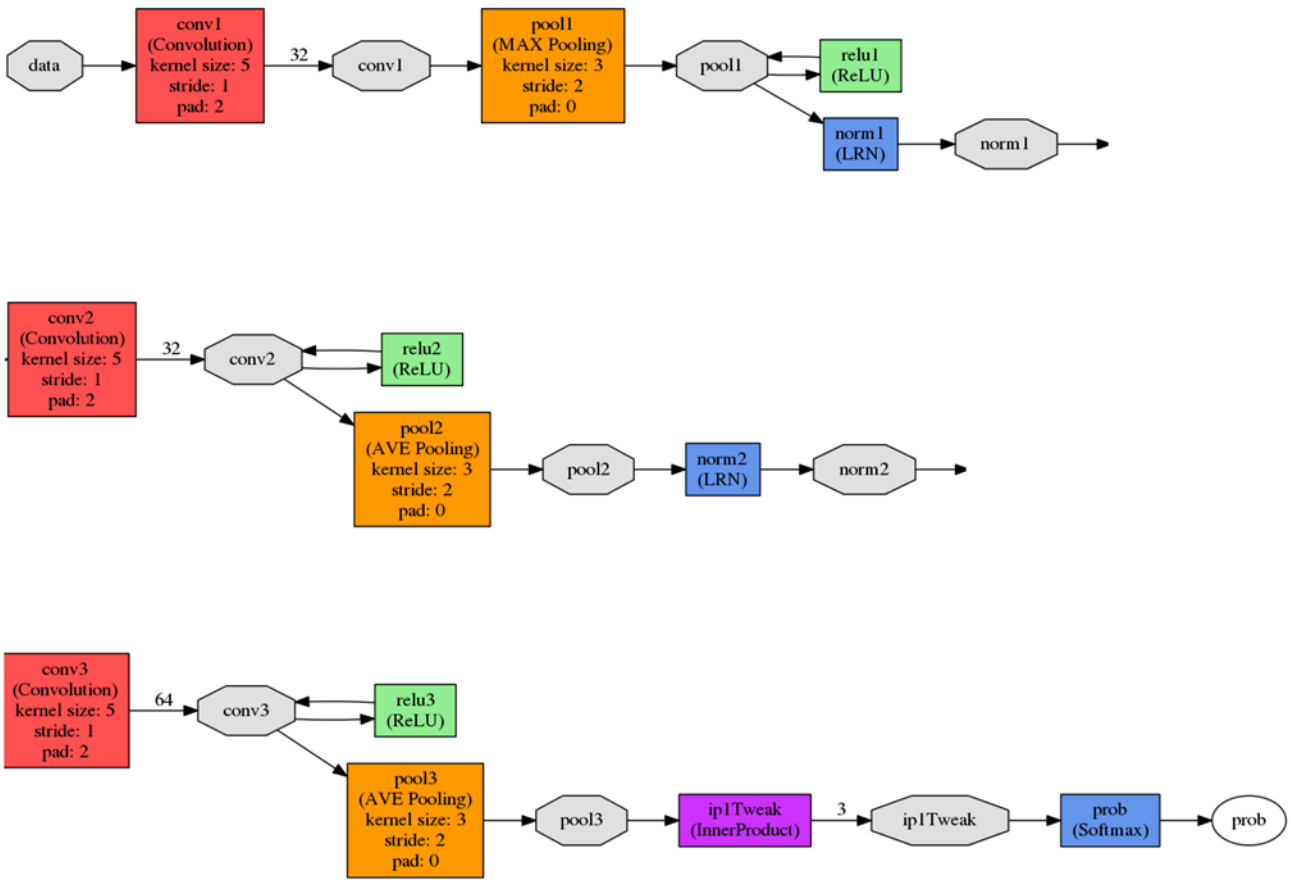

Fig. 3. The final architecture for the deep network. This is inspired by the architecture for solving the CIFAR10 dataset.

avoidance was being abstracted instead of the robot only avoiding black colored objects (the black chairs). It is also significant to note that students used the lab throughout the day and night, so conditions of the carpet changed while the dataset was being developed. For example, coins and shreds of paper were found at different locations on the path. We decided not to remove some of these items while building the dataset because it only added to the diversity in what we might consider edge cases.

A. Dataset Collection:

To collect data for learning, an operator remotely drove the robot along its path avoiding obstacles while the robot's camera took photographs. These images were each labeled with the corresponding action taken by the operator in the position where the photos were taken. Every time the user hit a directional key, the image was saved along with its label. The TurtleBot was driven around the lab following the path in both directions. To increase the diversity of the dataset, different starting points were chosen and hard scenarios such as being close to walls were considered. Overall, 30,754 images were collected. By default the images from the camera were 640 x 480. These were later downsized to 64 x 64 for our deep learning network.

B. The Deep Neural Network Architecture

To learn the controller for our robot we replicated the deep network architecture used by Alex Krizhevsky to solve the CIFAR10 dataset. We trained this network with the original CIFAR10 dataset and obtained about 74% accuracy. We then used this trained network, with some modifications to learn our dataset. The final network used for our research is shown in Fig. 3. It is split into three lines to ease the visualization. We can see that there are 3 iterations of the layer combinations of convolution, pooling, and normalization. The layer "ip1Tweak" is labeled as such because the final layer of Krizhevsky's network was removed and replaced with an inner product, or also considered as fully connected, layer that only had 3 outputs (his had 10). This is signified by the value 3 above the ip1Tweak layer in the visual. The 3 outputs correspond to the decision making of the TurtleBot in terms of autonomous driving directions. The original network included 32 convolution kernels for the first convolution, 32 convolution kernels for the second convolution, and 64 convolution kernels for the last one. We can also see how each convolution layer is immediately followed by a pooling layer. Every convolution layer also includes a rectified linear unit attached to it. Local response normalization also appears to be an effective addition to this network, as it augments the outputs of 2 of the 3 pooling layers.

The dataset was split for the final network: 7,689 images for testing and 23,065 images for training. This is based on a 75% training split of the entire dataset. The hyperparameters were defined as:
- testing iterations (how many forward passes the test will carry out): 100
- batch size (this is for batch gradient descent – notice that batch size * testing iterations will cover the entire testing dataset): 77
- testing interval (testing will be carried out every x training iterations): 150
- base learning rate: .001
- momentum .9
- weight decay: .004
- learning rate policy: fixed
- maximum iterations: 15,000

B. Results Using Deep Learning for Autonomous Driving

After training with the CIFAR10 dataset and altering the network, we trained using approximately 30,000 images. It took the network about 200 iterations to get to 84% accuracy and around 2000 iterations to achieve an accuracy of about 90%. It was able to obtain an accuracy of about 92% after 15,000 iterations. To test the results in the environment, ten different runs were completed where the robot was reversed after a completion of a lap in order to complete the lap in both directions [20]. The robot did very well in these tests, although rarely, it lightly grazed against the leg of a chair or a cardboard box. However, this did not change the trajectory of the robot and it was still able to complete its course. For this reason, we did not consider these rare occurrences to be major collision events.

V. Testing in New Environments

To fully understand the extent of the learning it was important that we tested the robot in the real world. The robot was first tested in the original environment where the dataset was collected and used to train the controller. This was a fair assessment to make, and a true "validation dataset" of a sorts, since the robot had not seen these exact orientations as every run was different. To take this testing a step further, we explored testing the robot in three different environments.

The first new environment test was to see if the robot would be successful in avoiding bumping into walls in a hallway (Fig. 4 top photos). For the most part the robot was able to turn away from walls in the hallway. This is especially surprising since there were very few images in the original dataset that had white walls to turn away from. Perhaps the robot is instead turning away from the dark brown bottom lining of the white walls as that color is close to the color of the chairs. The robot also successfully managed to turn away from doors in the hallway. However, a head-on approach to a door was a problem and the robot had trouble navigating its way out of that situation. This makes sense as we can liken this to the issue with head on approaches of cabinets in the original training environment. Some diagonal approaches are also problematic. In some cases, the robot would not make enough of a turn and did not see the edge of a wood lining for a door and would bump into it. We speculate that this could be mitigated by tweaking the turning angle so that smaller turns are made.

Fig. 4. New environments not seen by the robot during training: first row: hall; second row: colony space; third row: other lab with lower chairs/tables; fourth row: person.

The second new environment that was tested was the colony space in the laboratory (Fig. 4 second row photos). This is an 8 x 8 foot area where wooden blocks of 1 x 1 foot dimension can be rearranged to create paths. The robot was generally successful in avoiding these blocks, but only if the blocks were doubly stacked on each other or if the blocks were placed horizontally. One hypothesis for this behavior is that most of the avoidance of similar colored and textured objects from the original dataset had objects that were large (cabinets) or high enough for the robot to detect ahead of time (border of colony space).

The third environment that the robot was introduced into was a differently configured room (Fig. 4 third row photos). This room does not have cabinets like the original room and the tables and chairs are different. The robot successfully avoided all chairs and tables. It was also able to go under tables and avoid objects there, where lighting conditions were much darker than in any image in the dataset. The fact that the robot was able to avoid chairs with different designs in the new room is yet another confirmation that the deep neural network has abstracted the concept of obstacle avoidance. Perhaps what is more important for the network is the concept of going to an open area, which might focus on the color and texture of the carpet, since the carpets in the two rooms are very similar.

Another interesting scenario presented to the robot was that of obstacle avoidance with humans, or specifically human legs (Fig. 4 bottom row photos). Recall that Gritti et al. approached a similar problem scenario or at least to the point of detecting legs. Although nowhere in our original dataset were jeans or human legs introduced, the robot was able to avoid these obstacles as well. One hypothesis is that the network correlates dark objects with obstacles and the individual in the experiment wore dark pants. However, this is very hard to verify since the inner workings of the network do not easily reveal that information.

VI. Conclusions

The approach of fine-tuning Krizhevsky's network that solved the CIFAR10 dataset was greatly successful. The robot effectively avoided obstacles in the original room where the dataset was collected and it also avoided colliding into other obstacles that were not part of the dataset. This was demonstrated by placing the robot in three new environments. This implies that the learning system has abstracted the idea of obstacle avoidance and that the network did not overfit in its training on the original dataset. In other words, the deep network did not solely focus on specific chairs, tables, and cabinets as the only obstacles to avoid. In future work, we plan to use this same approach to learn control programs for other robots in completely new environments. For example, in an outdoor environment, this approach can learn to stay on the path in addition to obstacle avoidance. It could also learn to take the correct action, such as stopping in a dynamic environment with no place to go, such as when surrounded by people, and start moving again when the area is sufficiently clear.

References

[1] W. Hotze, "Robotic First Aid: Using a mobile robot to localise and visualise points of interest for first aid." Master's Thesis. Embedded and Intelligent Systems, Hamstad University, 2016.

[2] M. Correa, G. Hermosilla, R. Verschae, & J. Ruiz-del-Solar, J., "Human Detection and Identification by Robots Using Thermal and Visual Information in Domestic Environments." Journal of Intelligent Robot Systems, 2012, 66:223–243. DOI 10.1007/s10846-011-9612-2

[3] A. Gritti, O. Tarabini, J. Guzzi, G. Caro, V. Cagliotti, L. Gambardella, & A. Giusti, "Kinect-based people detection and tracking from small-footprint ground robot." International Conference on Intelligent Robots and Systems (IROS), 2014.

[4] S. Boucher, "Obstacle Detection and Avoidance Using TurtleBot Platform and XBox Kinect." Research Assistantship Report. Department of Computer Science, Rochester Institute of Technology, 2012.

[5] O. Russakovsky, J. Deng, H. Su, et al., "ImageNet Large Scale Visual Recognition Challenge." International Journal of Computer Vision, 2015, 115: 211. doi:10.1007/s11263-015-0816-y

[6] A. Krizhevsky, I. Sutskever, & G. Hinton, "ImageNet classification with deep convolutional neural networks", Neural Information Processing Systems (NIPS), 2012.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions.", 2014, CoRR, abs/1409.4842

[8] K. He, X. Zhang, S. Ren, & J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In International Conference on Computer Vision, 2015.

[9] A. Krizhevsky, CIFAR10 Dataset Project Page. https://www.cs.toronto.edu/~kriz/cifar.html

[10] A. Krizhevsky, "Learning multiple layers of features from Tiny Images", Master's thesis, Dept of Computer Science, University of Toronto, 2009.

[11] A. Krizhevsky, "Convolutional Deep Belief Networks on CIFAR-10." Unpublished manuscript, 2010.

[12] D. Mishkin, & J. Matas, "All you need is a good init", International Conference on Learning Representations, 2016.

[13] J. Springenberg, A. Dosovitskiy, T. Brox, & M. Riedmiller, "Striving for simplicity: The all convolutional net." 2014, arXiv:1412.6806

[14] B. Graham, "Fractional max-pooling." CoRR, 2014, abs/1412.6071. http://arxiv.org/abs/1412.6071.

[15] L. Tai, S. Li, & M. Liu, "A Deep-Network Solution Towords Model-Less Obstacle Avoidence", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016.

[16] L. Tai & M. Liu, "A robot exploration strategy based on q-learning network", IEEE International Conference on Real-time Computing and Robotics (RCAR), 2016.

[17] Deep Learning Robot. Produced and sold by Autonomous AI. https://www.autonomous.ai/deep-learning-robot

[18] A. Ng, et al. "Unsupervised Feature Learning and Deep Learning (UFLDL)", online tutorial and notes. Computer Science Department, Stanford University. http://ufldl.stanford.edu/tutorial

[19] P. Joshi, "Perpetual enigma: perennial fascination with all things tech", 2016, Online blog. https://prateekvjoshi.com/2016/04/05/what-is-local-response-normalization-in-convolutional-neural-networks/

[20] M. Khan & G. Parker, "Deep Convolutional Neural Network Processing of Images for Obstacle Avoidance", Studies in Computational Intelligence, vol 922. Springer, Cham., 2021.