3-2021

Indoor Navigation Using Convolutional Neural Networks and Floor Plans

Ricky D. Anderson

INDOOR NAVIGATION USING CONVOLUTIONAL NEURAL NETWORKS AND FLOOR PLANS

THESIS

Ricky D. Anderson, First Lieutenant, USAF

AFIT-ENG-MS-21-M-006

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-006

INDOOR NAVIGATION USING CONVOLUTIONAL NEURAL NETWORKS AND FLOOR PLANS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Ricky D. Anderson, B.S.C.S.

First Lieutenant, USAF

March 25, 2021

AFIT-ENG-MS-21-M-006

INDOOR NAVIGATION USING CONVOLUTIONAL NEURAL NETWORKS AND FLOOR PLANS

THESIS

Ricky D. Anderson, B.S.C.S.
First Lieutenant, USAF

Committee Membership:

Joseph A. Curro, Ph.D
Chair

Scott L. Nykl, Ph.D
Member

Robert C. Leishman, Ph.D
Member

AFIT-ENG-MS-21-M-006

Abstract

The Global Positioning System (GPS) is the primary solution for outdoor navigation. However, the signals from these satellites are blocked in indoor environments leading to the need for alternative indoor solutions. The goal of this thesis is to evaluate a new indoor navigation technique by incorporating floor plans along with monocular camera images into a Convolutional Neural Network (CNN) as a potential means for identifying camera position. Building floor plans are widely available and provide potential information for localizing within the building. This work sets out to determine if a CNN can learn the architectural features of a floor plan and use that information to determine a location.

In this work, a simulated indoor data set is created and used to train two CNNs. A classification CNN, which breaks up the floor plan into 100 discrete bins and achieved 76.1% top 5 accuracy on test data. Also, a regression CNN which achieved a distance error of 25.4 meters or less between the truth and predicted position on 80% of the test data. The models are further improved by combining them with a filter solution. The best performing classification CNN is evaluated on real world data captured via a TurtleBot 3, demonstrating the potential for this solution to be useful to real world Air Force indoor localization problems.

Acknowledgements

I would like to thank my advisor for all the guidance and encouragement that made this work possible. I would also like to thank my committee members and ANT center staff for their feedback and support throughout this research.

Ricky D. Anderson

Table of Contents

Page

List of Figures

List of Tables

INDOOR NAVIGATION USING CONVOLUTIONAL NEURAL NETWORKS AND FLOOR PLANS

I. Introduction

1.1 Problem Background

Navigation is fundamental in many aspects of our lives. Global Navigation Satellite System (GNSS)s make this possible in an outdoor environment, however signals from these satellites are not receivable in indoor environments. Therefore, other solutions need to be presented to make this possible and convenient. Indoor navigation technology is very important to many consumers such as security forces and first responders because it can provide them the means to handle dangerous situations more easily. Many techniques have been developed to provide indoor navigation solutions. Pedestrain Dead Reckoning (PDR) is one of the most common used technologies for smartphone-based pedestrian indoor navigation [1]. This technique estimates the users' location based on step detection, step length and heading via sensors on the smart phone. However, this approach can start to become inaccurate over time due errors in the step length, heading and step count. Other solutions involve using the buildings wireless transmitters to triangulate a users' position. Common techniques include Wireless Fidelity (WiFi) fingerprinting [2] and Bluetooth Low Energy (BLE) [3]. However, these techniques can suffer from signal attenuation and require the devices to be available and maintained, which may not always be the case. Vision-based navigation encompasses many different techniques such as Simultaneous Localization And Mapping (SLAM) [4] and Visual Odometry (VO) [5] and require the use of

optical sensors.

With current advancements in the field of deep learning, it has been shown that Artifical Neural Networks (ANN) can be used for navigation purposes [6], [7], [8]. A Convolutional Neural Network (CNN) is a specific type of ANN that is suited for analyzing and extracting features from images. Features of an indoor building could include doors, windows and hallways and this idea can be exploited for navigation purposes. The Air Force provides Geographic Information System (GIS) map data for most bases. With the GIS map data, a floor plan for a particular building on base can be extracted which depicts the aforementioned features. By matching architectural features in an image to its corresponding floor plan, a users location inside the building may be ascertained.

1.2 Research Objectives

The goal of this research is to evaluate the viability of a CNN incorporating floor plans along with monocular camera images as a potential means for indoor navigation. The research will attempt to answer weather or not a CNN can learn the features of a building floor plan and fuse that information with an image taken from inside the building to predict a users' location. To train a CNN for this kind of task, a large number of unique floor plans will need to be analyzed. This data set will need to be generated in a simulated environment due to time constraints. However, small amount of real world data will also be collected and used to further analyze the CNNs viability. The CNN metrics will be analyzed and discussed to determine if this concept can be a potential indoor navigation solution. If so, the solution could give Air Force first responders potentially safer and faster response times to crises as well as provide the Air Force with an additional use for GIS map data.

1.3 Assumptions

This work is not attempting to beat the location accuracy of the current state-of-the-art navigation solutions. It is trying to determine the viability of a new technique for indoor localization by using a multi-input CNN to fuse information from an image and a floor plan. Additionally, the simulated buildings were designed simple in nature. This could make the generalization to real data less accurate. Also, in order to reduce the computational burden of training a CNN with the large number of training samples generated, the input image sizes are defined to be $100 \times 100 \times 3$ and $101 \times 101 \times 3$ for building images and floor plan images respectively. Images larger than this would need to be formatted to this size. Finally, in terms of floor plan positions, the locations are defined by a two-dimensional Cartesian coordinate system beginning at the bottom left corner of the floor plan image.

1.4 Document Overview

The remainder of this thesis is organized as follows: Chapter II provides the relevant background information needed to understand the content of this thesis. Chapter III discusses the processes that were used to produce and process the data. CNN architectures and network training are discussed. Also, particle filter implementations are explained. Chapter IV analyzes the results of the different experiments conducted. Finally, Chapter V provides a summary of the work that was done as well as future improvements to the techniques developed.

II. Background and Literature Review

This chapter provides background information necessary to understand the methodology in Chapter III. Topics covered include Machine Learning, Artifical Neural Networks (ANN), Convolutional Neural Network (CNN), U-Nets and Particle Filters. Furthermore, additional information will be presented on alternative indoor navigation techniques.

2.1 Machine Learning

Machine learning is a subset of the field of Artificial Intelligence (AI) and can be broadly defined as the study of designing algorithms that use experience to improve performance and make accurate decisions. The experience comes from exposure to training data. The main objective of machine learning is to be able to generalize to previously unseen data which comes from having learned meaningful patterns in the training data. Common machine learning tasks include classification, regression, ranking, clustering and manifold learning [9].

2.2 Artificial Neural Networks

ANNs are a powerful machine learning model loosely inspired by the human brain. They are a collection of interconnected neurons organized into layers. The first layer is the input layer. It consumes the raw data, processes it and passes information on to the next layer as output. A common type of layer in a network is a dense layer, or fully-connected layer. This layer will take as input, the outputs of all the neurons in the previous layer. Information flows through the network until it reaches the output layer which makes a prediction. When information is only passed as the output of one layer to the input of another it is considered a feed-forward neural network [10].

Layers in between the input and output layers are called hidden layers. Typically, when a network contains multiple hidden layers they are refereed to as deep neural networks [11]. A simple example of a basic neural network and deep neural network are shown in Figure 1.

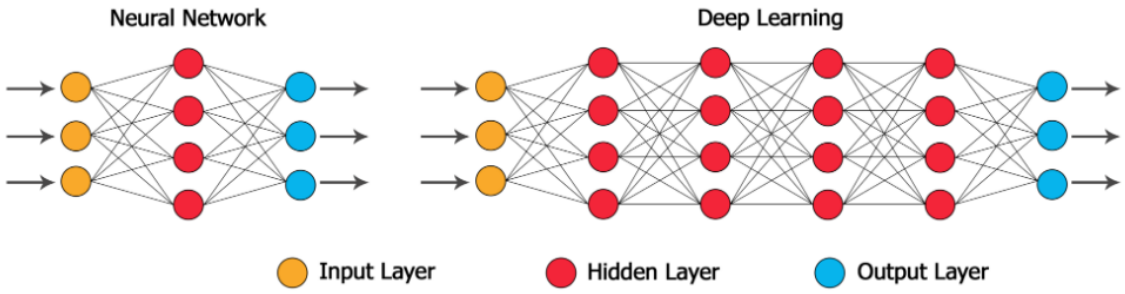

Figure 1: The figure on the left depicts a simple neural network with one hidden dense layer. The figure on the right depicts a deep neural network. Notice that this network has multiple hidden layers in between the input and output layers.

2.2.1 Weights and Bias

Each layer of a neural network is made up of one or more neurons. A neuron can have one or more inputs and one output. The inputs to the neuron are multiplied by their respective weights and then summed. A bias term is added to the summation and the result is fed into an activation function. The bias term serves to apply an affine transformation to the summation and can be thought of as how easy it is for a neuron to fire [10, 12]. This concept is illustrated in Figure 2. The output of the activation function is the output of the neuron. The weights and bias of a neuron are the parameters that need to be adjusted during training to make the network more accurate.

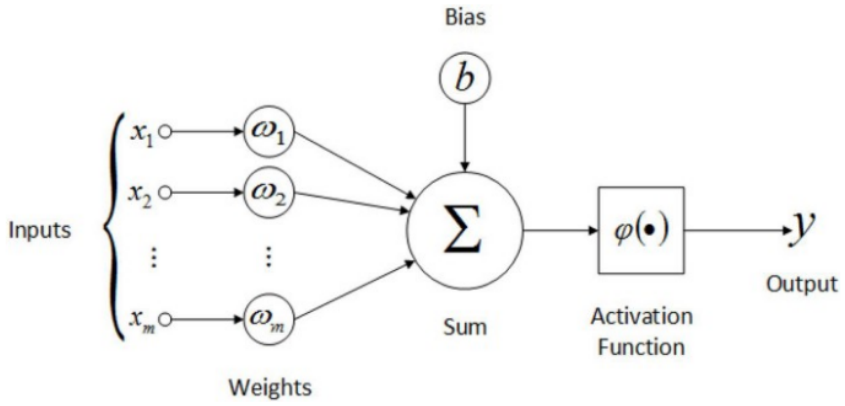

Figure 2: Structure of artificial neuron: the inputs to the neuron are multiplied by their respective weights and summed. A bias is added to the summation and the output is passed through an activation function.

2.2.2 Activation Function

The purpose of the activation function is to limit the range of possible outputs of a neuron and add non-linearity to the network which helps it learn complex patterns in the data. There are a number of different activation functions that can be used. This research focused on Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh) and softmax activation functions [13], [14], [15]. ReLU is defined as $f(x) = max(0, x)$. It is one of the most widely used activation functions due to its fast computation and ability to alleviate the vanishing gradient problem. ReLU can experience a problem sometimes during training in which some of the gradients will die, causing a neuron to never activate again. This is known as the dying ReLU problem and can be fixed by using a smaller learning rate during training [16]. ReLU is depicted graphically in Figure 3.

The tanh function is given by $f(x) = (\frac{e^x - e^{-x}}{e^x + e^{-x}})$. Its range is between -1 and 1 and its main advantage is that it can help with the back-propagation process by providing zero centered output [16]. The tanh function is depicted graphically in Figure 4.

The softmax function is defined as $f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$. The softmax function takes

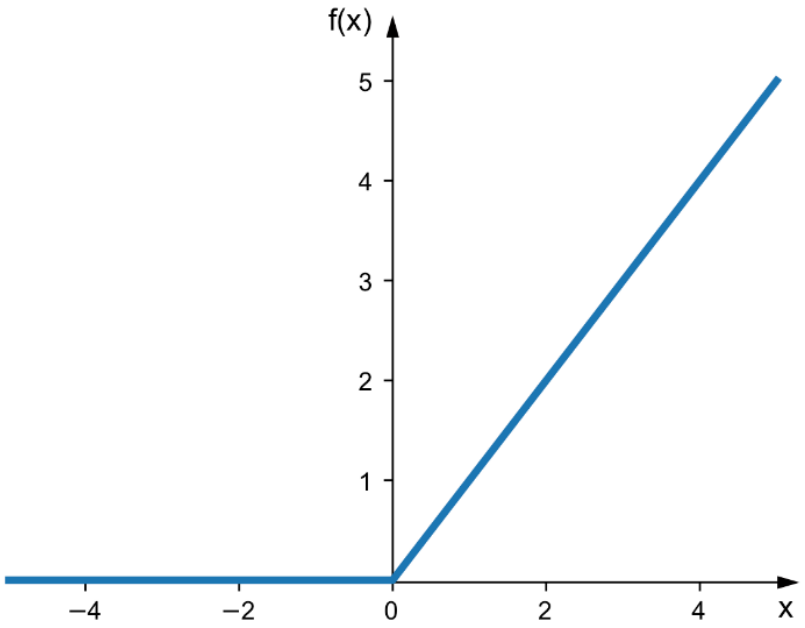

Figure 3: Graphical representation of the ReLu function. This function is represented mathematically by the expression $f(x) = max(0, x)$

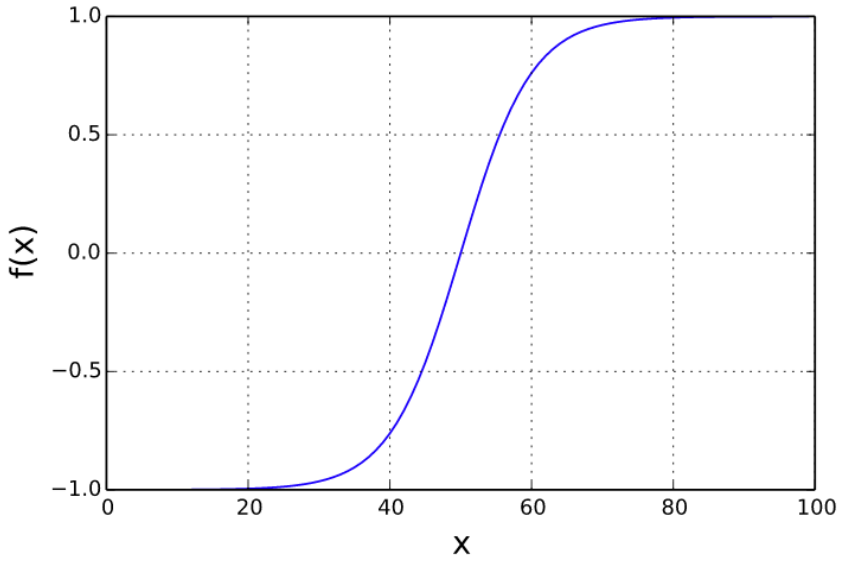

Figure 4: Graphical representation of the tanh function. This function is represented mathematically by the expression $f(x) = (\frac{e^x - e^{-x}}{e^x + e^{-x}})$

a vector of real numbers and converts them into a probability distribution. Each output is in the range 0 to 1 and the sum of all the probabilities is equal to 1. It is usually used as the last layer activation for a classification problem [16]. The softmax

function is depicted graphically in Figure 5.

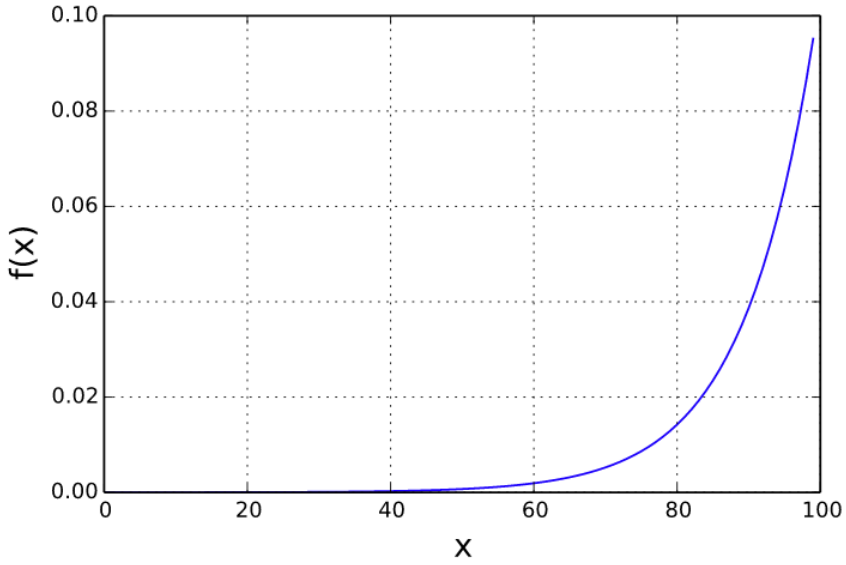

Figure 5: Graphical representation of the softmax function. This function is represented mathematically by the expression $f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$

2.2.3 Objective Function

The objective function is what needs to be minimized or maximized during training of a neural network. When it needs to be minimized it is referred to as the loss function [17]. The loss function computes the error of the network by comparing the predicted output with the expected output. The error is used by the optimizer to update the weights and biases of the network in order to improve the error. There are a variety of loss functions that can be used and should be chosen to match the problem type. For a regression problem the common choice is Mean Squared Error (MSE) defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1}$$

MSE computes the square difference between the networks prediction $\hat{y}$ and the

expected output y, squares it and averages it out of all the training samples N. MSE gets smaller the closer the predicted output is to the expected output.

For a problem involving multi-class classification, where the output is a vector of scalars representing a valid probability distribution over all the classes, the Categorical Cross-Entropy (CCE) loss function is typically used. To calculate the CCE loss for one training sample the following equation would be used [10]:

$$CCE = -\sum_{i=1}^{X} y_i \cdot \log \hat{y}_i \tag{2}$$

where X is the total number of classification classes, y_i is the i-th scalar value in the truth output and $\hat{y}_i$ is the i-th predicted scalar value. The negative sign makes sure that the loss gets smaller as the probability distributions get closer together. This loss would be summed together with all other training samples in the batch to calculate the overall loss of the batch.

2.2.4 Training

The weights and biases of an ANN are learned during the training process which involves the use of an optimizer. There are a number of available optimizers for example Root Mean Square Propagation (RMSprop), Adaptive Gradient Algorithm (Adagrad) and Adaptive Moment Estimation (Adam) to name a few [18]. The optimizer utilizes a form of Gradient Descent to minimize the error of the loss function. There are three variants of Gradient Descent; Batch Gradient Descent, Stochastic Gradient Descent, and Mini-batch Gradient Descent [18]. These variations differ in the amount of training samples that are used to compute the gradient. Batch Gradient Descent computes the gradient for the entire training set whereas Stochastic Gradient Descent computes the gradient for a each training sample. The preferred algorithm is Mini-batch Gradient Descent which involves computing the gradient for

a small batch of training samples typically ranging in size from 50 to 256 [18]. The gradient is a partial derivative of the loss function with respect to each learnable parameters [19], and is found via a process called Back Propagation (BP) in which the derivative chain rule is applied to all the weights while remembering previously calculated values [20]. Once the gradient is computed, the optimizer updates all the weights in the negative direction of the gradient in order to minimize the error of the network. An arbitrary learning rate can be multiplied by the gradient to control how large the weight change will be [20]. When all the training samples have had a chance to pass through the network and update the weights, one epoch has been completed.

Another important concept of training is weight initialization. Poor initial weights can lead to vanishing or exploding gradients. In these cases, the gradients are either too small or too large to flow backwards beneficially. If the gradients are to small, weights in the initial layers may receive small weight updates which will lead to slower training or even stop the network from learning at all. If the gradients are too large, due to large initial weights, the network will spend the majority of its time trying to decrease these weights. Both problems lead to the network struggling to converge to the optimal solution along with longer training times [21]. Common initialization strategies include Glorot Normal and Glorot Uniform. Glorot Normal is the one used in this thesis and works by initializing weights with values from a random uniform distribution as shown in (3) [22].

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \tag{3}$$

Where n_j represents the number of incoming connections to the layer and n_{j+1} represents the number of outgoing connections from that layer.

2.2.5 Training, Validation and Test Sets

Training a network also involves splitting the available data into three different sets. The training set is the data that the network will learn from and use to update weight values. The validation set is used to evaluate the network configuration. This data does not impact the weights, but instead is used as a metric to tune the hyperparameters of the network. Finally, after the network has been fully trained it is evaluated on test data. Some common schemes for spitting the data up are hold-out validation, k-fold validation and iterated k-fold validation with shuffling [11].

2.2.6 Overfitting and Regularization

The goal of training an ANN is that it generalizes to new data that it hasn't seen before. This goal is hindered in the event of overfitting. A network that is overfitting has modeled the training data too well, meaning it is learning patterns that are specific to the training data only and irrelevant when it comes to new data [23]. Regularization techniques combat the overfitting problem. Some of these techniques include adding weight regularization, adding dropout, batch normalization and data augmentation [11].

Weight regularization is the process of penalizing the network for having large weights by adding a cost value to the loss function. There are two popular kinds of costs, L1 regularization and L2 regularization. L1 regularization is when the cost is proportional to the absolute value of the weight coefficients and L2 regularization is when the cost is proportional to the square of the value of the weight coefficients. The penalty is controlled by a hyperparameter which determines the amount to penalize the network. This hyperparamter can be sent to a value between 0.0 meaning no penalty, and 1.0 meaning full penalty.

Dropout is a common regularization technique and is applied to a layer. It works

by randomly setting weights in the layer to zero, which basically means that the neuron is deactivated. This deactivation of neurons is done per batch. The number of neurons selected for dropout is specified by the dropout rate and is typically between 0.2 and 0.5. Dropout effectively introduces noise into the network and forces it to be more robust.

Batch normalization is a type of layer that can be added to the network which normalizes the input of the following layer. The batch normalization layer does this by storing two parameters, the current mean and standard deviation of the batch. These parameters are learned as part of the training process. Not only does this layer help with overfitting, but also improves the gradient flow through the network allowing for deeper networks [19].

Data augmentation is very common and easy to perform on image data. It consists of creating more training data for the network to learn from by augmenting the existing training data. This is done by applying different kinds of transformations to the images such as rotations, translations and shearing. Not only does this increase the size of the training data but also provides examples of different aspects of the same features which helps the network to generalize [11].

The main idea with regularization techniques is that they help a network be resistant to memorizing the noise in the training data which forces it to learn more about the underlying concepts present in the training data [10].

2.3 Convolutional Neural Networks

CNNs are a sub branch of deep ANNs. CNNs are specific for handling data that has a grid-like topology, such as images. CNNs work on multiple channels and are designed to be able to learn spatial hierarchies of patterns in data. CNNs have become the main approach for solving complex image classification and object detection tasks

[24]. A CNN is usually made up of three kinds of layers; convolution, pooling and fully connected. Convolution and pooling layers perform feature extraction by generating a desired number of feature maps. Feature maps represent extracted characteristics found in the data. The output of the feature maps are fed into a fully connected layer typically for classification purposes [19].

The convolutional layer is responsible for feature extraction. It does this using the convolution operation. The convolution operation works by sliding a convolution kernel across every location in the input data and doing a dot product between the two matrices. The output is fed into an activation function and the result is a pixel on the feature map, which is repeated for each slide position to make a full feature map. The convolution operation is illustrated in Figure 6. Two important parameters for this operation are the kernel size and number of kernels to use. Common kernel sizes are 3×3, 5×5 or 7×7. The number and type of kernels used will determine the number of feature maps created, each of which will represent a different characteristic of the input data [19]. Feature maps allow a CNN to learn small local patterns in the data and are translation invariant. By chaining convolutions together a CNN can learn and recognize increasingly complex patterns [11]. The weights of the kernels are what need to be refined during the training process. Additional common hyperparameters of the convolution layer include stride and padding. Stride determines how far the convolution kernel moves between steps. A stride larger than 1 will downsize the resulting feature map. Padding is the process of adding additional rows and columns of zeros to the input data so that the convolution kernel is able to capture the edges of the data. The resulting feature map will be the same size as the kernel [19].

A pooling layer is used to summarize the the major features detected in the data. The pooling layer is applied to all feature maps and creates a new set of feature maps which are usually smaller in size. The smaller size reduces the number of learnable

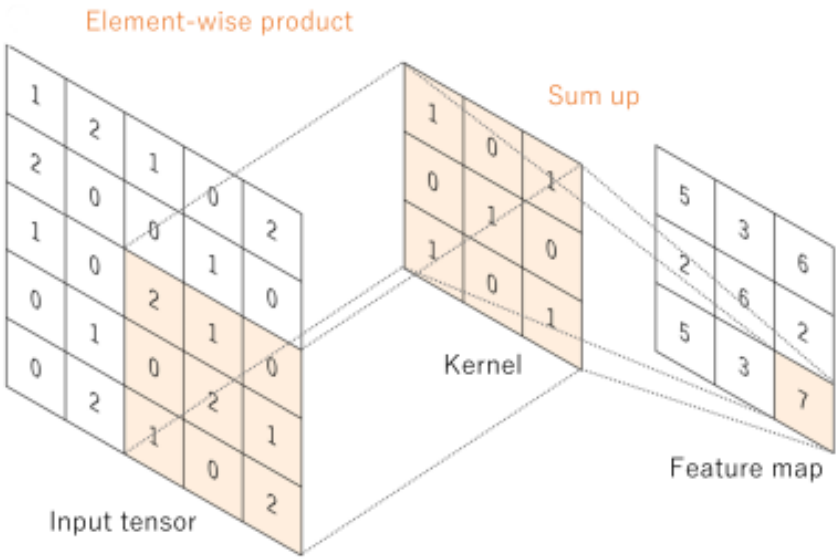

Figure 6: Convolution operation: 3x3 convolution kernel with no padding and a stride of 1 applied to input tensor and generating the resulting feature map.

parameters in the network, leading to an increase in computational efficiency. The pooling layer is also what provides the network with translation invariance for features found in the data. The pooling layer involves choosing a type and size of pooling operation. There are two common types of pooling operations, max pooling and average pooling [19]. Max pooling creates a new feature map from the maximum values of each patch specified by the pool size, whereas average pooling takes the average value of each pool size patch and uses that in the new feature map [25]. The max pooling and average pooling operations are illustrated in Figure 7.

After the last convolution or pooling layer, the final features maps are typically flattened into a 1D vector and then connected to a dense layer as described in 2.2. From there the dense layer can easily map to the needed number of final outputs for the given problem [19].

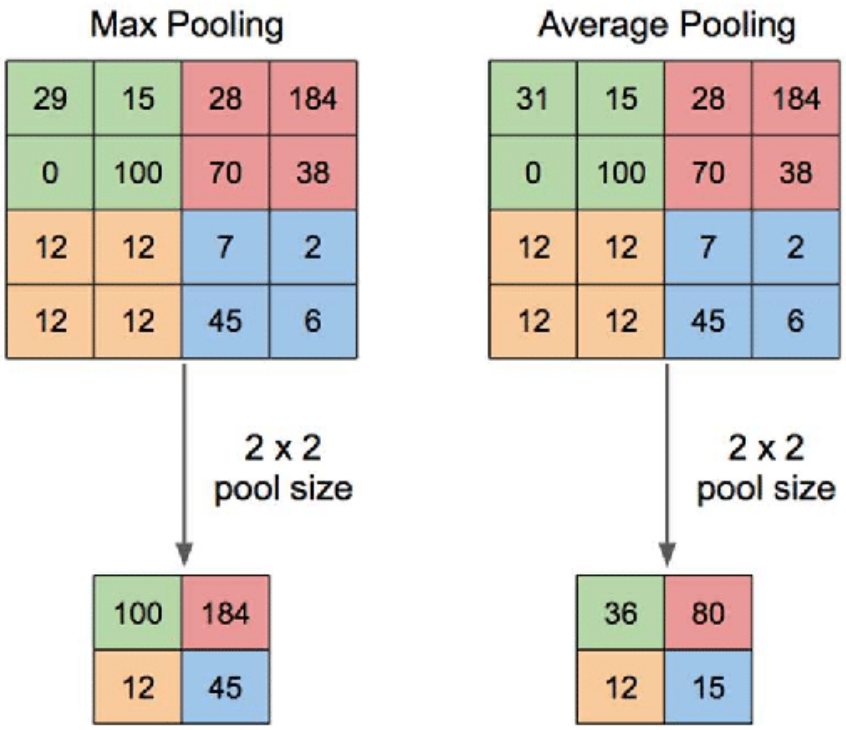

Figure 7: The max pooling operations is shown to the left and the average pooling operation is shown to the right. The pooling filter size is 2×2 with a stride length of 2. The resulting feature map is reduced by a factor of four after the operation.

2.4 U-Nets

U-nets are a sub branch of CNNs with a particular type of architecture. The design was originally developed by Ronneberge et al. for biomedical image segmentation [26]. A key concept of u-nets is that they can provide not only image object detection but also localization of the feature in the image. The architecture consists of two paths; the contracting path and the expansive path. The contracting path follows a typical CNN architecture using stacks of convolutions and max pooling operations that reduces the image size and generates a number of feature maps to identify what's in the image. The symmetric expansive path uses transpose convolution to upsample the size of the image, decrease the number of feature maps and concatenate the feature maps of the correspondingly level in the contracting path. This maintains the

localization ability of the network. The final layer uses 1×1 convolution to match the output feature maps to the desired number of classes for classification. This architecture is shown in Figure 8.

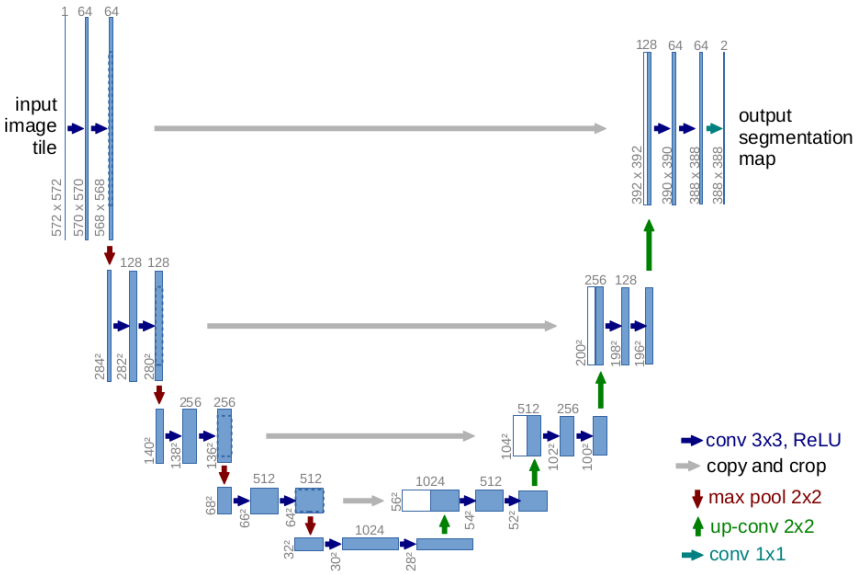

Figure 8: Example u-net architecture: 32x32 pixels at the lowest resolution. The number of feature maps is shown on top of each box. The bottom left value denotes image size. White boxes represent copied features. The arrows denote the different operations.

2.5 Particle Filter

Particle filters are a popular solution for localization problems [27], [28], [29]. Particle filters use Sequential Monte Carlo (SMC) methods to estimate non-Gaussian, non-linear, Markovian state space [20]. The basic idea is to estimate the posterior probability of some state of a system given the distribution of a number of of samples, or particles, in state space. A particle represents a possible state in the system and has an associated weight. The weight represents the probability of that particle in relation to the other particles in the system. In a dynamic system, the state equation can be described as:

$$x_k = f(x_{k-1}, v_{k-1}) \tag{4}$$

In this equation, f represents the state transfer equation, x_k represents the state of the system at time step k, and v_k represents the system noise at time step k. The observation equation can be expressed as:

$$z_k = h(x_k, w_k) \tag{5}$$

In this equation, h is the observation function, x_k is the state of the system at time step k, and w_k is the observed noise at time step k. Given these equations, a basic particle filter would start off by initializing N particles according to the initial state $p(x_0)$. The initial state is assumed to be known. The weights of all the particles would initially be set to $1/N$. Next, for each particle at time step $k-1$, calculate the next state of the particle at time step k via the state transition equation $x_k = f(x_{k-1}, v_{k-1})$ or the state transition probability $p(x_k|x_{k-1})$. Next, update the weight of the particle to be $w_k = p(z_k|x_k)$ according to the observed value z_k at time step k. The weight should then be normalized. After that, resampling of the particles should occur. Resampling is the process of replacing particles that have small weights with ones that have larger weights, or in other words, replace particles that are bad predictions with ones that are more likely. Resampling helps particle filters avoid the degeneracy problem, in which the majority of the particles have been assigned negligible weights [30]. Resampling does not necessarily need to be done every iteration. A way to determine if resampling is required is to compute the effective sample size N_{eff}. This can be done via the equation:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} w_i^2} \tag{6}$$

Where N is the number of particles and w is the weight of the particle. If N_{eff} falls below a predefined threshold, then resampling should occur [31]. After resampling the particles, the current estimated state of the system can be calculated by computing the weighted average mean of all the particles in the system [32].

2.6 Localization with ANNs

The problem of indoor localization has many kinds of solution possibilities with various degrees of accuracy. Some of these techniques include Pedestrain Dead Reckoning (PDR) [33], Wi-Fi [34], Bluetooth [35] and Radio-Frequency Identification (RFID) [36] based methods to name a few. Image-based localization is another technique that incorporates the use of images to determine position. The authors in [37] developed a relocalization system called PoseNet. This system is designed to regress the 6-DoF camera pose from a single RGB image using a CNN and can operate both indoors and outdoors. The authors utilized a technique called Structure from Motion (SfM) to generate the truth labels for their training data. This was done by applying the SfM technique on videos captures of the locations and regressing camera poses from the images. The authors used transfer learning techniques with the GoogLeNet [38] architecture to train their 23 layer CNN. The GoogLeNet architecture was modified to perform regression instead of classification and the authors were able to achieve results within two meters and five degrees of the truth labels. Another image based solution is called ICPS-net [39]. This architecture uses a tandem of two CNNs to determine camera position and quaternion information for a given image. The authors took images of an indoor building and separated them into different scenes found in that building. The first CNN was based on the EfficientNet [40] architecture and was trained as scene classifier. The role of this CNN was to determine which scene an image belonged to and then its output was fed into the second CNN. The second

CNN was based on the MobileNet [41] architecture and was trained as the position regressor. This CNN was used to determine the XYZ position and quaternion information of the image in the scene it was belonging to. The authors were able to attain accuracy of 98.099% on test data.

2.7 Indoor Localization with Maps

The before mentioned techniques did not attempt to incorporate widely available building floor plans into the localization prediction. A floor plan is a top down scale drawing of a building that depicts architectural features. Features could include walls, hallways, rooms, doors and windows. Knowing the layout of these types of features can allow a person to figure out where they are on a floor plan. Furthermore, signage such as bathroom and exit signs will be present in a building which can provide information about landmarks that are nearby, again allowing for a person to determine where they are in a building. Floor plans can provide useful information that can aid with various types of indoor localization techniques, such as particle filters. With a particle filter, a user's position can be modeled by a set of particles. When incorporating a floor plan, particles can be penalized for violating map constraints such as particles crossing over wall segments leading to a more accurate prediction. This technique can be seen in [42, 43, 31]. In [44] the authors developed a robot localization system that estimates a robots pose on a floor plan. The authors estimate the vanishing lines in the monocular image using the algorithm defined in [45] and then overlay that information back onto the original image. The authors are then able to predict room layout edges using a CNN based upon the AdapNet++ architecture. Next they used the harris corner detector implementation of OpenCV to extract room information from a floor plan. The authors used a particle filter to match the extracted edges to the given floor plan. This solution achieved an average linear and

angular root-mean-square error of (223 ± 126) mm and $(2.3 \pm 2.0)^\circ$ respectively.

III. Methodology

The objective of this research is to determine if a Convolutional Neural Network (CNN) could learn the architectural features of a floor plan and use that information to make a prediction based off what it sees in an image of the inside of that building. In order to test this, two models were created and evaluated. A discrete model, which predicted the grid locations on the floor plan where an image could have been taken from, and a regression model, which predicted the XY location of the camera. This chapter discusses the techniques used to generate and process the needed data. It also discusses the model architectures and training. Lastly, discrete and continuous particle filter implementations are defined.

3.1 Simulated Data Collection

For a CNN to be accurate, extensive training data needs to be accessible. For this problem, the training data needed to be generated for different floor plans. The simulated data was generated using the AftrBurner engine, a cross-platform visualization engine developed as the successor to the STEAMiE educational game engine [46]. The engine provides the ability to create virtual 3D worlds complete with complex lighting and texturing, as well as modules for creating and working with virtual cameras among many other things. To provide a CNN enough examples to learn from, building architectures were procedural generated using a Binary Space Partitioning algorithm and inserted into the virtual environment. The building was partitioned until a ratio of 40% hallway area to building area was achieved. The hallway width and wall height were defined to be 3 meters. The size of the building was set to 100 x 100 meters and only single story buildings were created. Randomness was added to the building in the form or adding doors and windows to arbitrary locations within

the building. Data collection points were identified by traversing the hallways at a distance of 1.5 meters from the wall. The distance between each point in the sequence is 1 meter. Room interiors were not used for data collection points. This concept is illustrated in Figure 9.

Once the building and the first texture were rendered, the camera was placed at each data collection point. For each point, 8 images were taken by rotating the camera 45deg. The image size was 100×100 pixels. This image size was chosen to prevent having to perform a prepossessing step later to make the image size compatible with the models. The images were labeled with the XYZ location of the camera as well as the normalized XYZ unit vector defining the look direction of the camera. Once all points had been visited and the images saved to disk, the next building texture was applied and the process was repeated. In total there were three different building textures used. These textures are shown in Figure 10.

Once all textures had been completed the generated floor plan was saved to disk. The floor plan image size was $101x101$ pixels. Black lines represent wall segments,

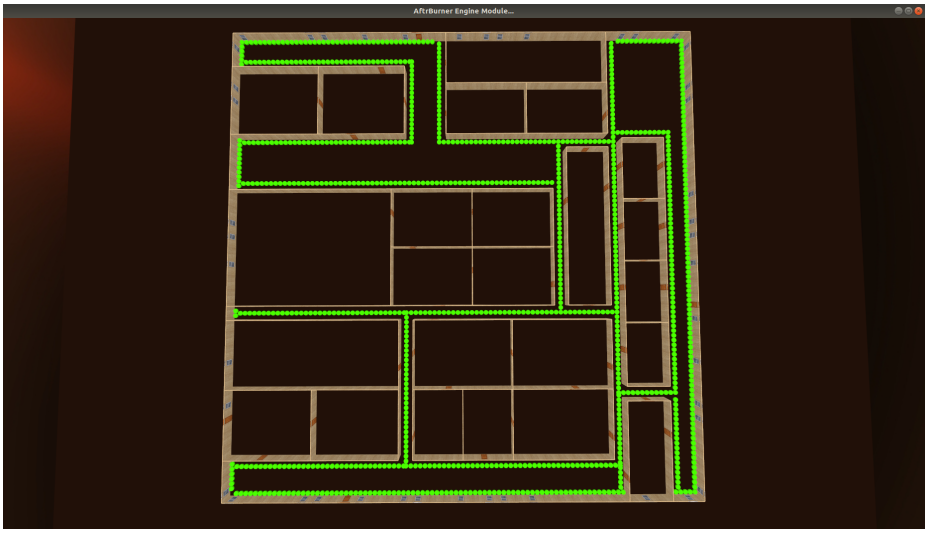

Figure 9: Placement of camera locations inside the building: Each green sphere depicts a location where the camera would capture images from.

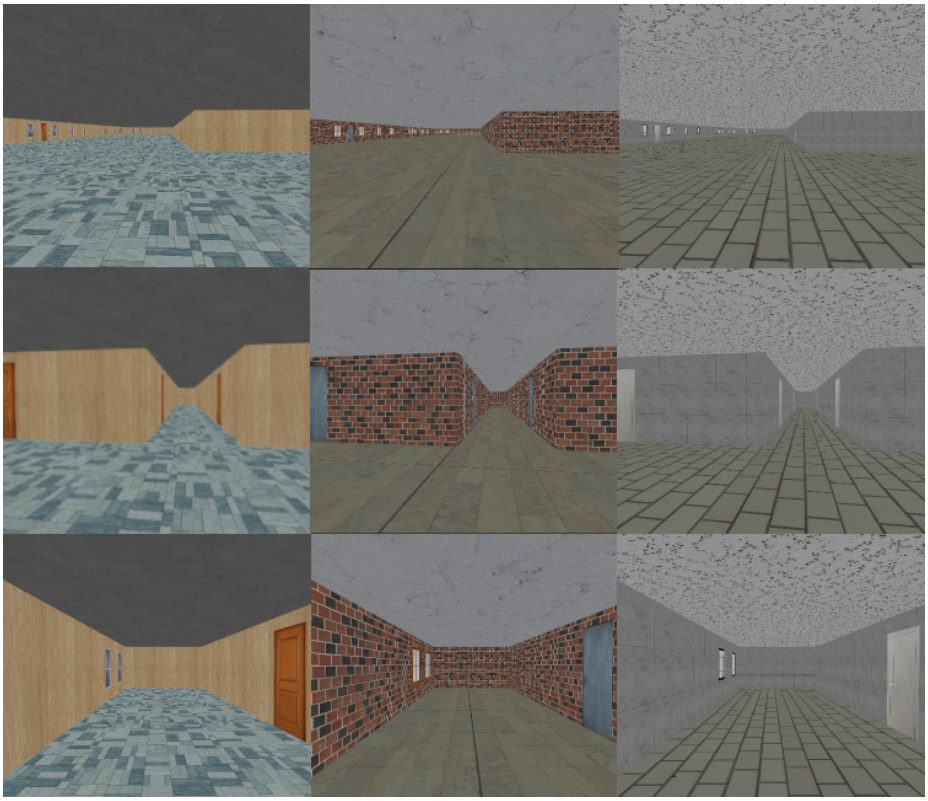

Figure 10: Building textures used for the simulated data. Each row depicts the same location in a building with the three different textures applied.

blank spaces in the wall segments represent doors and grey segments along the exterior wall represented windows. An example floor plan is shown in Figure 11. In total 13 million interior pictures were generated which consisted of 1000 different floor plans. Each floor plan, along with its corresponding imagery, was stored in its own directory. This process took approximately three days to complete.

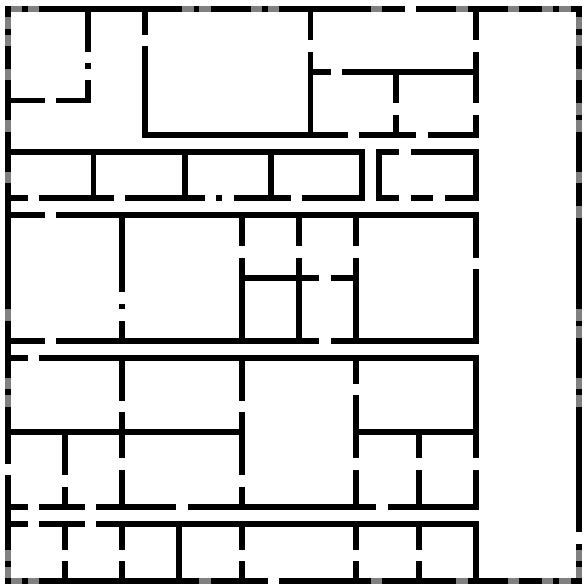

Figure 11: Example generated floor plan: 101x101 pixels. Wall segments are represented by black lines. Blank spaces in the wall segments represent doors. Grey segments along the exterior walls represent windows.

3.2 Real Data Collection

In order to collect real data, a TurtleBot 3 Waffle Pi equipped with a ZED Mini stereo camera, D435I RealSense RGB-D camera, wheel odometry and 2D Light Detection and Ranging (LIDAR) was used. An image of the TurtleBot 3 is shown in Figure 12.

The robot gathers navigation data and images of the environment, and then using the Real-Time Appearance-Based Mapping (RTAB-Map) [47] module in Robot Operating System (ROS), creates a map database of image features. The images from this database were extracted and named with the XYZ and RPY pose information determined by the wheel odometry. The 3rd floor of building 640 at the Air Force Institute of Technology (AFIT) was chosen as a data collection site. The path that the robot collected data from is shown by the blue lines in Figure 13.

In total, 446 images were collected. The size of the images captured by the TurtleBot 3 were 1280×720. These images were resized to 100×100 to make them

Figure 12: TurtleBot 3 used to collect real data imagery.

compatible with the models. Finally, an image of the floor plan was taken with a Samsung Galaxy Note 8. The size of that image was 4032×1960 and was then resized to 101×101 to make it compatible with the models. Example imagery captured from

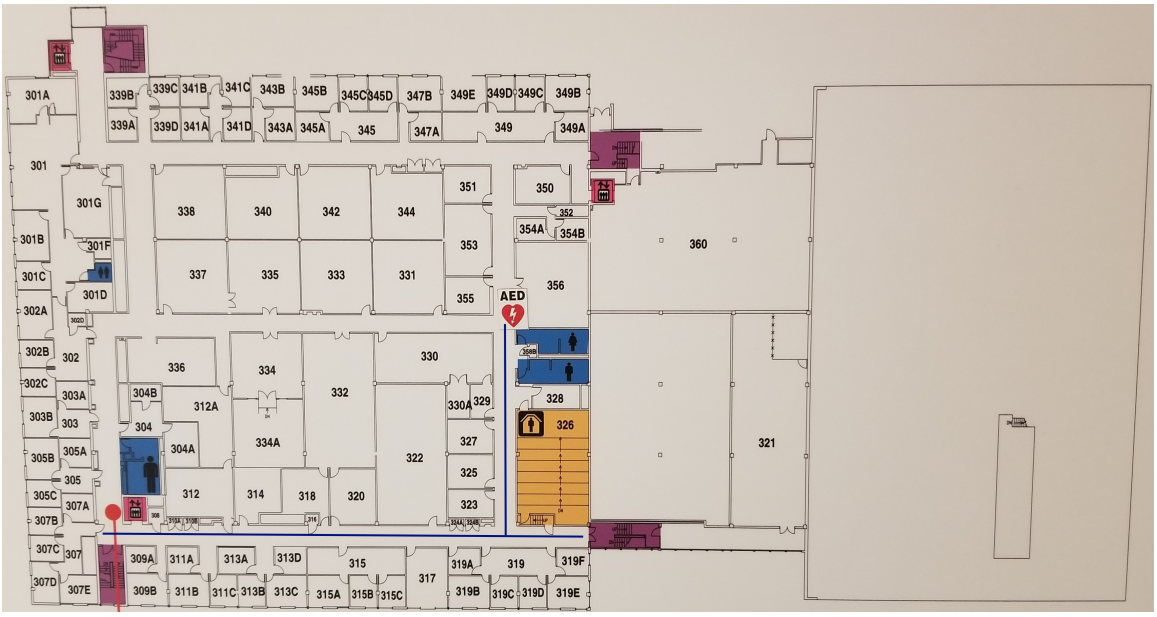

Figure 13: Floor plan of third floor of building 640 at AFIT. The blue line represents the path that the TurtleBot 3 collected imagery from.

the TurtleBot 3 is shown in Figure 14.

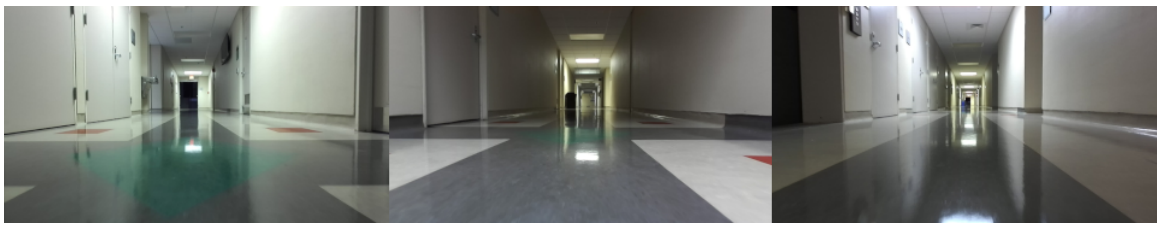

Figure 14: Example imagery captured from the TurtleBot 3.

3.3 Data Processing

After data creation, the data needed to be processed into the correct format for the given model. Due to the large amount of data generated and for the model to train on the data efficiently, the decision was made to convert all of the data into TFRecords. TFRecords are a binary file format that make reading from disk during training more efficient which in turn reduces the training time. TFRecords are composed up of TensorFlow example objects which are created from TensorFlow features. When initially creating the data, all images pertaining to the same floor plan were saved into the same directory along with the floor plan image. To create the corresponding TFRecords, each floor plan directory was processed by reading in and storing all image file names from that directory. The file names were shuffled and then each image was processed as follows: The RGB pixel values of the image were normalized to values between 0 and 1. Data augmentation was performed to create three new images for every original image. The first transformation randomly rotated the image by an angle between -10 and 10 degrees. the second transformation applied a random vertical translation between -10 and 10 pixels. The final transformation applied Gaussian-distributed additive noise to the image. The XYZ position as well as the XYZ look direction was extracted from the image file name. To work with the classification model, the floor plan was divided into a grid of 100 blocks. Each

grid block spanned a 10 × 10 pixel region of the floor plan. The XY position was converted into a single scalar value representing the grid location on the floor plan via Equation 3.

$$g = \left(\left\lfloor \frac{y}{10} \right\rfloor \times 10\right) + \left\lfloor \frac{x}{10} \right\rfloor \tag{7}$$

This scalar was then converted into a binary class matrix of length 100. This process is known as one-hot encoding and is a common technique when working with categorical data. The matrix will have a 1 at the index corresponding to the scalar value and zeros in all other indexes. At this point the data was ready to be written to disk in TFRecord format. To do this, a feature object for each image was created which contained the image, the corresponding floor plan and the binary class matrix representing its position. For the regression model, instead of the grid position the original XY position was stored in the feature. A TensorFlow example was created from each feature and then written to the corresponding TFRecord file on disk. Each TFRecord file contained a batch of 400 examples. This was done to take advantage of parallel disk I/O features of TensorFlow data sets. This research used hold-out validation. 90% of the floor plan directories were used as training data and the remaining 10% were used as validation data. Additional building floor plans were later generated for testing purposes.

3.4 Model Architecture

The models in this research consisted of a two input CNN based upon the U-Net architecture described in Section 2.4. These models were built with version 2.3 of the TensorFlow machine learning platform utilizing the Keras functional Application Programming Interface (API). The programming language used was Python version 3.6.9. The left side of the model pertained to the image of the indoor environment.

The input tensor to this side had a shape of $(100, 100, 3)$. Each layer used Rectified Linear Unit (ReLU) activation and same padding. The kernel weights used Glorot normal initialization and a kernel size of 3. The right side of the model pertained to the floor plan image. The shape was $(101, 101, 3)$. This side of the model used the same hyperparameters as the left side with the exception of the layer activations. For this side, Hyperbolic Tangent (tanh) activations were used due to the limited possible pixel colors of the floor plan images. For each side of the model the U-Net architecture was utilized via three stacks of three convolution layers. Each stack was followed by a max pooling layer that down-sampled the image to a final size of 25×25 pixels. The filter sizes of for each convolution block was 16, 32 and 64. The combination of up-sampling layers with a kernel size of 2 and concatenation layers were used to form a final shape of $(100, 100, 32)$. At this point the output from each side of the model was concatenated along the feature map axis, forming a shape of $(100, 100, 64)$. Next, 1D convolution with ReLU activations was performed to blend the feature maps while maintaining the spatial information. Then the output was shrunk to a size of $(10, 10, 1)$ via two more max pooling layers with ReLU activations and kernel sizes of 5 and 3 respectively and a convolution layer with 1 filter, kernel size of 1 and ReLU activation. Finally, a flatten layer was applied and the output was fed into a dense layer of 100 neurons. For the classification model, this layer used softmax activation and the output pertained to the possible grid locations on the floor plan that the image could have been taken from. For the regression model, the final dense layer of 100 neurons was modified to use ReLU activation and another dense layer with two neurons was appended afterwards which used linear activation. The output of this model pertained to the XY location of the floor plan where the image could have been taken from.

To increase generalization and fight overfitting, dropout layers with a rate of

0.1 were placed after each max pooling and concatenation layers. Also, two batch normalization layers were added after the final two convolution layers. The total number of weights in the classification model was 278,969. The total number of weights in the regression model was and 279,171. Both models were trained using the RMSProp optimizer with a learning rate of 0.0001. The loss function for the classification model was categorical crossentropy and the metric of the top 5 was calculated. For the regression model, the loss function was set to Mean Squared Error (MSE) and the metric used was root-mean-square error (RMSE). Simplified versions of these architectures are shown in Figure 15 and Figure 16. The complete model details can be found in Appendix A.

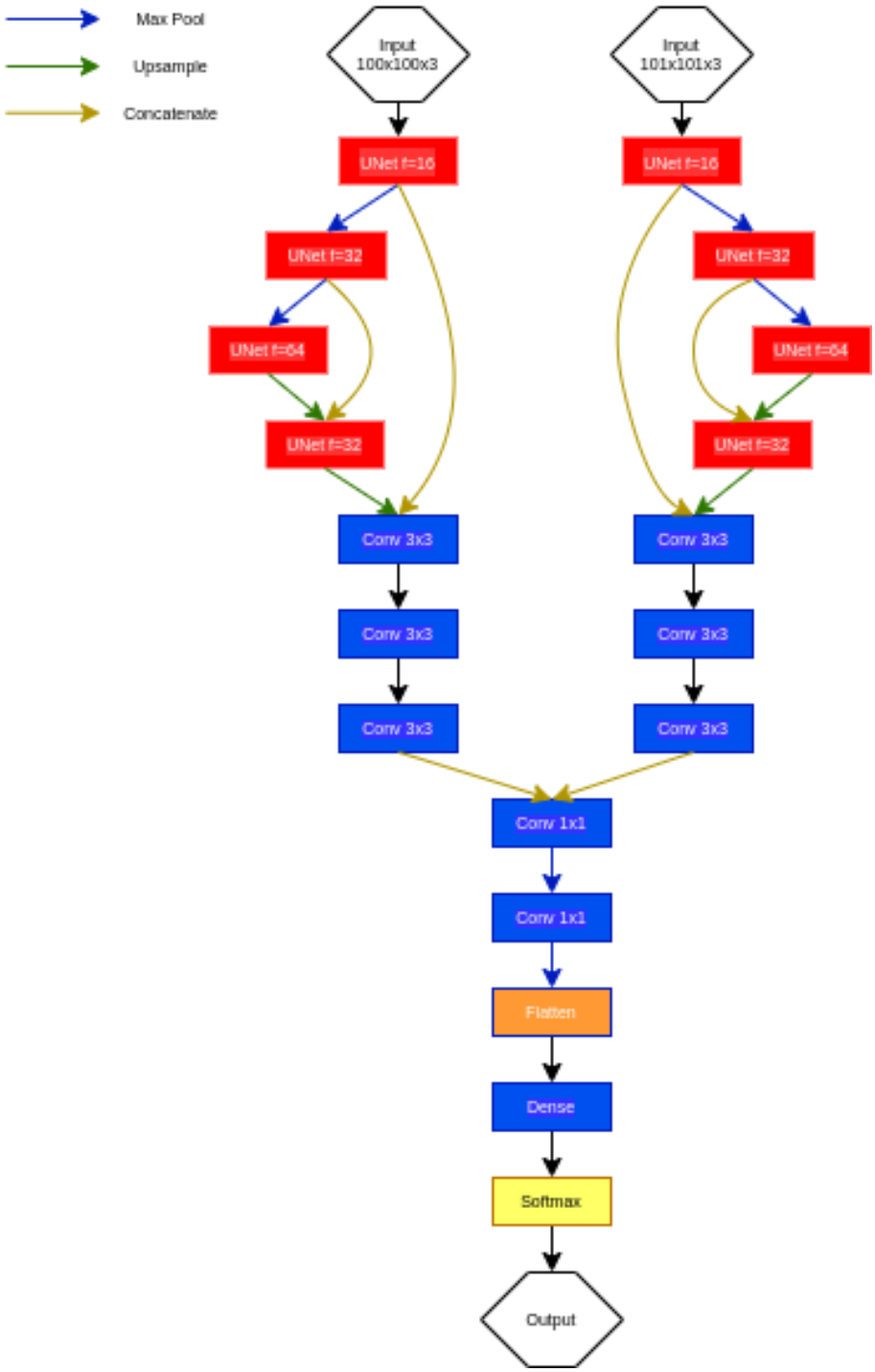

Figure 15: Simplified Classification Model. The model mimicked a U-Net architecture. The red UNet blocks correspond to three convolution layers with the beginning filter size specified by f. The kernel size was 3x3. The blue convolution blocks represent a convolution layer with the filter size specified in the block. The output of this model was 100 units corresponding to individual floor plan block probabilities.

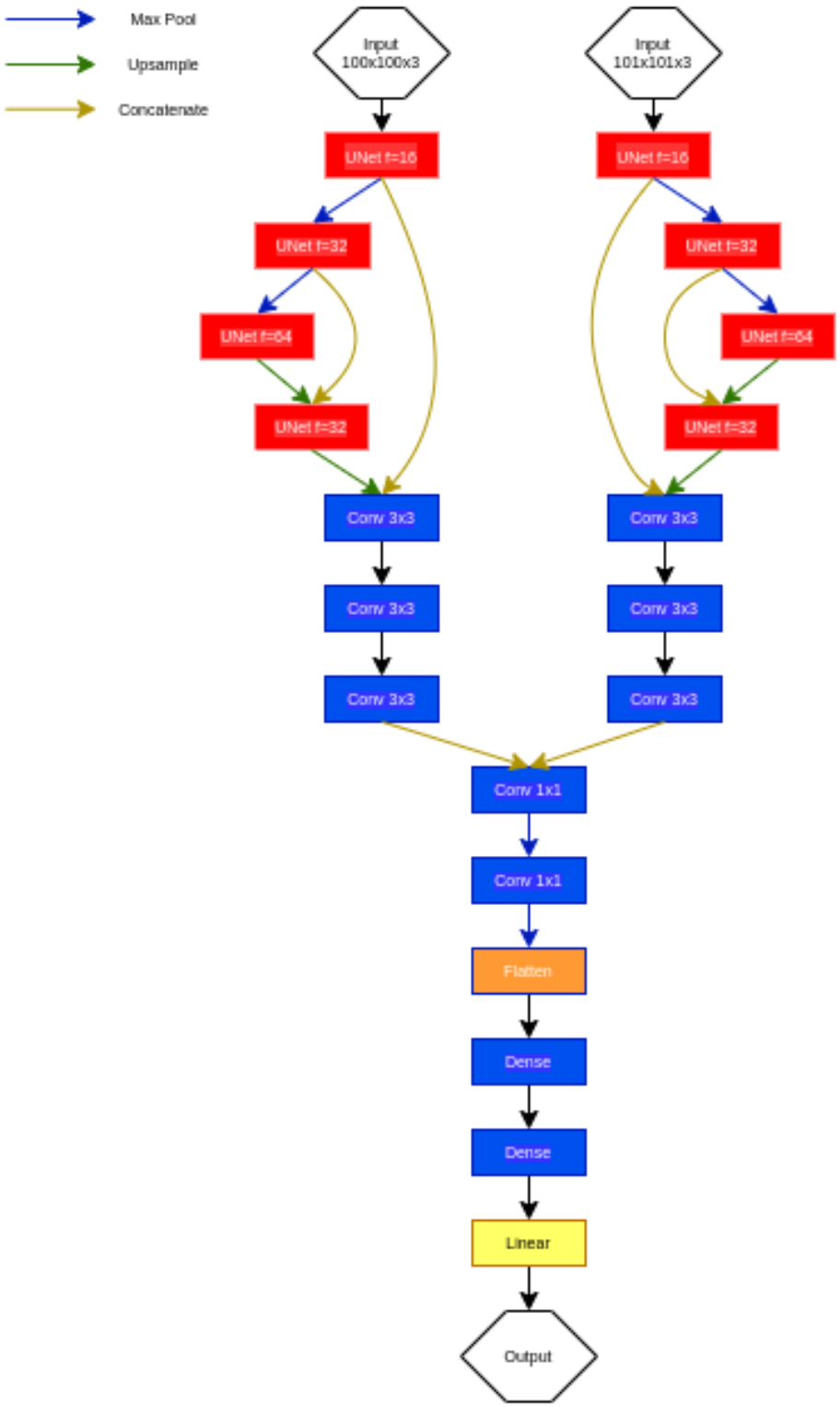

Figure 16: Simplified Regression Model. The model mimicked a U-Net architecture. The red UNet blocks correspond to three convolution layers with the beginning filter size specified by f. The kernel size was 3x3. The blue convolution blocks represent a convolution layer with the filter size specified in the block. The output of this model was 2 units corresponding to the XY position prediction.

3.5 Model Training

Due to the large data set, extensive Graphics Processing Unit (GPU) resources were needed to train these models. The Navigational Hyperspectral Learning (NHL) cluster located at AFIT provided the needed processing power. The models were trained on a GPU rack mounted server running Ubuntu version 18.04 with an AMD EPYC 7H12 with 256 cores, 1519 Gigabytes (Gb) of Random-access memory (RAM), four Nvidia Quadro RTX 6000s each with 24 Gb of RAM, and an Nvidia V100 with 16 Gb of RAM. The models were trained for 100 epochs with batch sizes of 32. The learning rate was configured to be reduced by a factor of 0.1 if the validation loss did not improve after ten epochs. Also, in order to find the best solution possible, model check pointing was used. For the classification CNN, the model with the highest validation categorical accuracy was saved. For the regressions CNN, the model with the lowest validation RMSE was saved.

3.6 Particle Filters

To improve the accuracy of both CNNs, two different particle filters were developed. The problem with the CNNs is that they do not take in to account any previous information. They simply makes a prediction based off the features in the image and try to correlate that with the floor plan image. This could lead to inaccurate predictions because there may be multiple feasible locations on the floor plan that match what is seen in the image. The goal of a particle filters is to estimate a new state based off previous states and measurements. For the classification CNN, this concept was applied to the grid of blocks, with the idea being that a new location prediction should be close to the last predicted block. For the regression CNN, a continuous particle filter was designed that incorporated wall constraints from the floor plan.

3.6.1 Discrete Particle Filter

A simple discrete particle filter was developed to be used in conjunction with the classification CNN. A 1D array of length 100 is used to represent the the particle filter predictions for the grid locations. The initial location is known and that position is given 100% probability. Next, the cells adjacent to the current location are computed. 5% probability is added to each of those cells and the cell corresponding to the current location has its probability reduced by 0.05 $\times$ the number of adjacent cells. In the particle filter sense, this could be thought of as the dynamics update that disperses the particles from their current location. The image predictions from the classification CNN are then multiplied by the particle filter array, which fuses together the model predictions and previous possible positions. This array is normalized and the highest probability cell becomes the new prediction for that image. The resulting array of probabilities is then used for the next prediction. This process is repeated until all the images are processed.

3.6.2 Continuous Particle Filter

A continuous particle filter was developed to try and improve the accuracy of the regression CNN. In this particle filter, each particle represented a possible XY location of an image with a corresponding weight. The weight represented the probability that the particle was the true position. To start off, 1,000 particles were initialized with a uniform position that was $\pm$ 20 meters from the known starting location. The weight for each particle was initialized to 1/1000. The particle filter would run for each image prediction by the regression CNN. This involved iterating over all the particles to predict their next states, update their weights, resample and compute the combined particle filter prediction.

To compute the next state of a particle, the particle position was moved 2.5 meters

in a random direction. Next, the weight of the particle was updated. The new weight was calculated by taking the reciprocal of the Euclidean distance between the XY position predicted by the regression CNN and the newly computed particle position. This allowed for particles that were closer to the prediction to be assigned a higher weight. Also, to take advantage of the provided floor plan, wall constraints were incorporated. If a particle tried to move over a wall segment on the floor plan, then the weight of that particle was reduced by a factor of 100. The new weight of the particle was then multiplied by the old weight and saved. After all particles had been processed, the weights were normalized. Next the effective sample size N_{eff} was computed to determine if resampling should occur. If the computed value was less than $N * 0.75$, then systematic resampling was applied to the particles. The value $N * 0.75$ was set as a predefined threshold and was discovered through trial and error as the best value for this particular problem [43]. Lastly, the weighted average mean of all the particles was computed and used as the particle filter prediction for that image. This process was repeated for each prediction by the regression CNN.

IV. Results and Analysis

This chapter analyzes the training results of the models described in 3.5. Next, the models are evaluated on the test data set and their performance is discussed. In order to improve the accuracy of the Convolutional Neural Network (CNN)s, discrete and continuous particle filters were added to create a more complete navigation solution. The results on five simulated floor plan data collections are evaluated with those combined solutions. Finally, the classification CNN is evaluated with real data that was collected as described in 3.2.

4.1 Training Results

Initially, this research attempted to solve this problem with a typical double input CNN architecture. However these models did not yield accurate location predictions. Therefore, the decision was made to apply the U-Net architecture described in 2.4. The idea was that the U-Net architecture would allow for better localization by identifying where on the floor plan particular features existed, as opposed to considering the floor plan as a whole. In this research, two CNNs were designed for this problem. They were trained as described in 3.5. The results of that training is described below.

At first, the localization problem was broken down into a more simple representation by breaking the floor plan down into 100 discrete blocks, which represented a set of XY positions. For this representation of the problem, the classification CNN was used to predict the blocks that it believed the image to have been taken from. The model was trained for 100 epochs with Categorical Cross-Entropy (CCE) as the metric. Recall that CCE will assign a probability to each block to show the models confidence in that block as the possible truth location. On training data, the model attained 77.3% categorical accuracy. It achieved a categorical accuracy of 76.5% on

validation data. The overall training loss was reduced to a value of 1.99, whereas the validation loss was reduced to 1.97. These results are shown in Figures 17 and 18. There are two noticeable jumps in the training and validation losses that occurred at epoch 53 and 65. The model was configured to reduce the learning rate in the event that the validation loss stagnates after ten epochs. Those jumps are related to the learning rate being adjusted by a factor of 0.1 at those epochs. After the learning rate was adjusted, the validation loss continued to decline.

The second CNN trained for the purposes of regression. The task for this model was to predict the XY position for an image relative to the given floor plan. This model was trained for 100 epochs and achieved a validation root-mean-square error (RMSE) of 13.57. On training data, this model achieved an RMSE of 13.39. The training loss was reduced to 280.43, whereas the validation loss was reduced to 277.82. These results are shown in Figures 19 and 20. Table 1 shows the captured metrics for both models.

	Classification CNN	**Regression CNN**
Optimizer	RMSprop	RMSprop
Loss Function	Categorical Crossentropy	Mean Squared Error (MSE)
Learning Rate	$1e^{-4}$	$1e^{-4}$
Training Loss	1.99	280.43
Validation Loss	1.97	277.82
Training CCE	77.3%	NA
Validation CCE	76.5%	NA
Training RMSE	NA	13.39
Validation RMSE	NA	13.57

Table 1: Summary of the captured metrics for the classification CNN and the regression CNN.

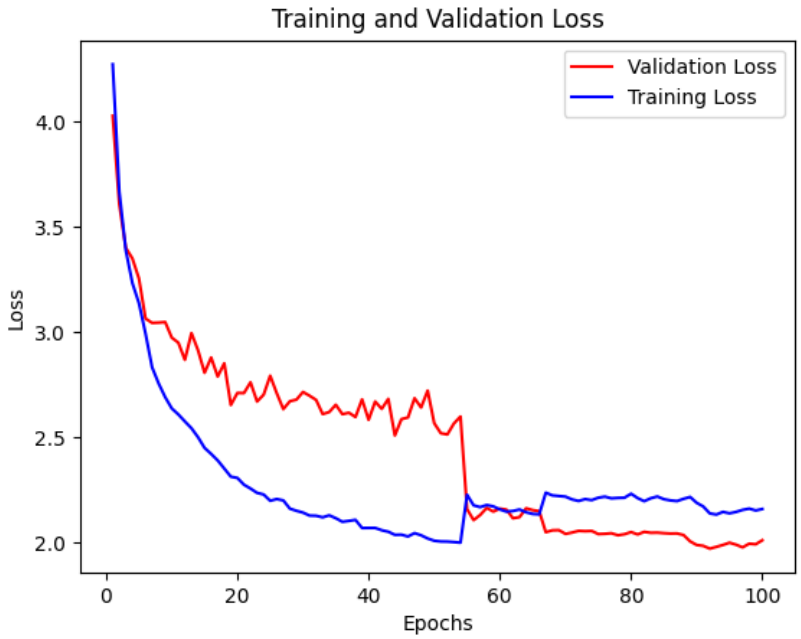

Figure 17: Training and validation loss plot for the classification CNN over 100 epochs.

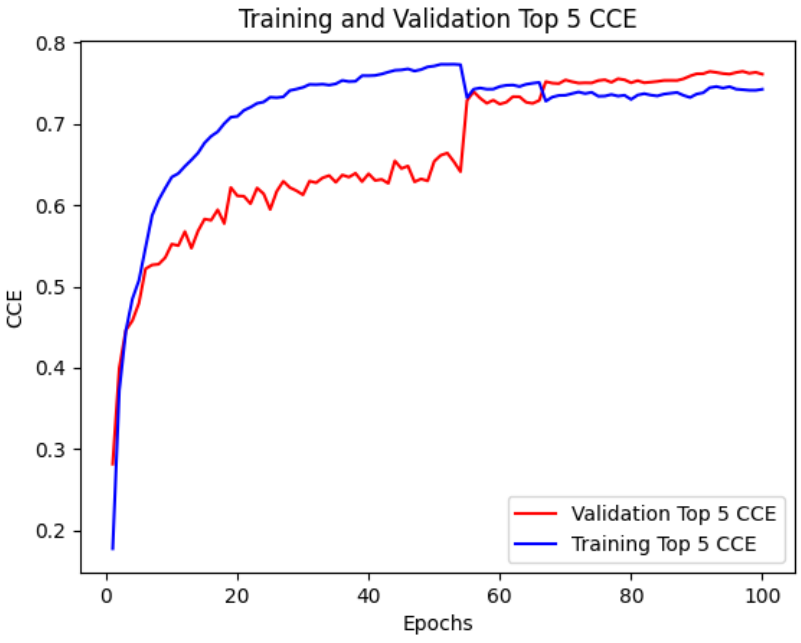

Figure 18: Training and validation CCE plot for the classification CNN over 100 epochs.

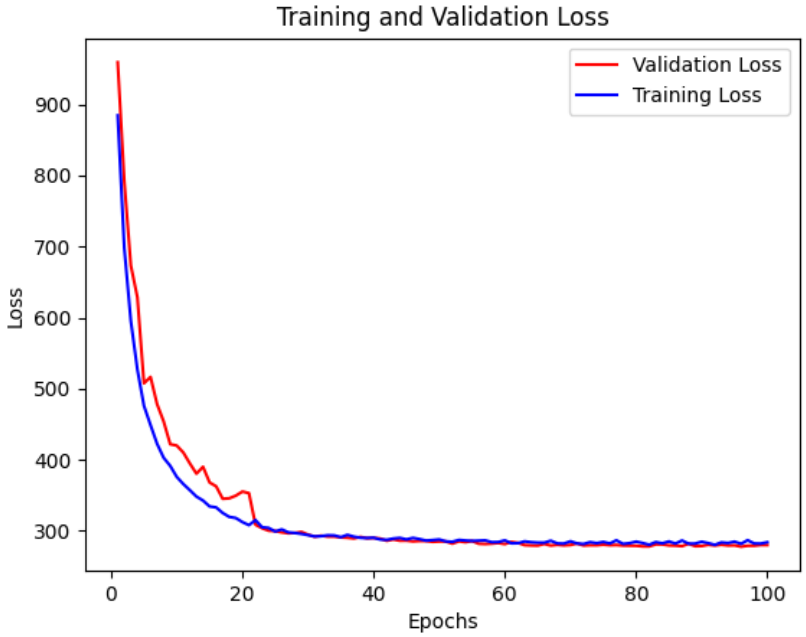

Figure 19: Training and validation loss plot for the regression CNN over 100 epochs.

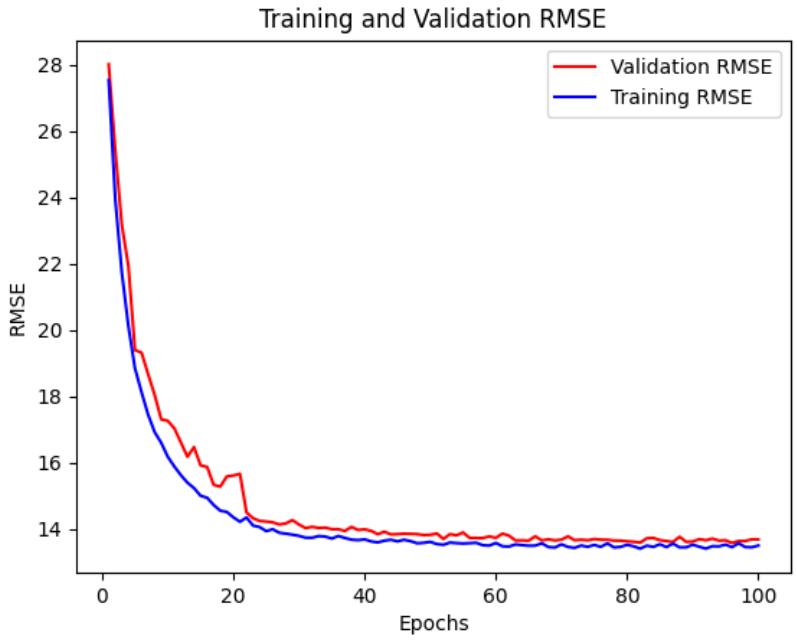

Figure 20: Training and validation RMSE plot for the regression CNN over 100 epochs.

4.2 Discrete Simulated Data Results

4.2.1 Classification Model Results

The performance of these models were evaluated on a new simulated test data set generated via the AftrBurner engine. This test set contained 4,614 test samples. For the classification CNN, a heat map was generated that showed the top 5 blocks that the model predicted for the given image. Redder shading of the block indicates a higher probability of the models confidence in that prediction, whereas bluer shading indicates less confidence. Figures 21 and 22 show examples of this. In these figures, a blue rectangle around the block represents the true location.

In Figure 21, the models top choice is the true location. In this situation the model seems to be keying off of tight hallways that lack doors and windows in the image and selecting locations that might match that. The image was taken from the vantage point showing the corner of a room which may have been the key to getting this prediction right.

In Figure 22, the true location of the image was not in the top 5 choices predicted by the model, however all the predictions were along the same hallway. In this image, windows are visible and the model would need to select locations that are close to exterior walls.

The other heat maps are omitted for brevity, but the total results of the 4,614 predictions are shown in Figure 23. In 28% of the test images, the first choice by the model was the true location of the image. The top 5 accuracy for these images was 76.1% which is inline with the model training results discussed in 4.1.

A distance metric was also computed for the 4,614 images. The distance metric was computed by summing the weighted average position predictions for each image and then computing the average predicted block. Then, assuming the discrete estimate is the center of the block, the distance between the truth block and predicted

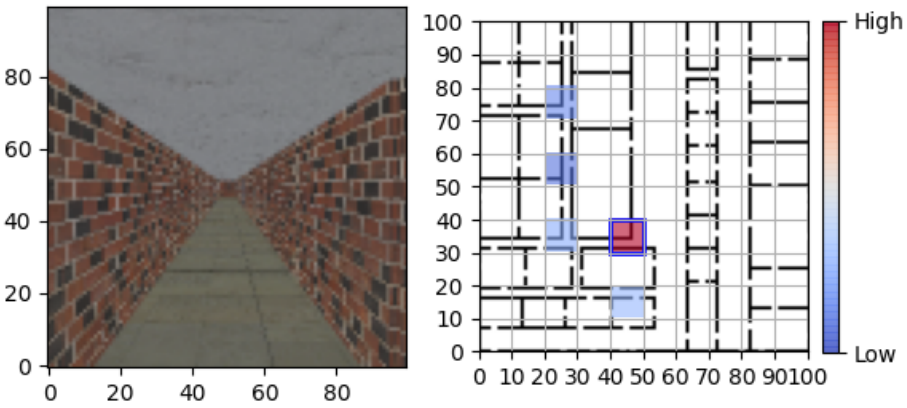

Figure 21: Heat map showing a correct prediction for the test image shown to the left.

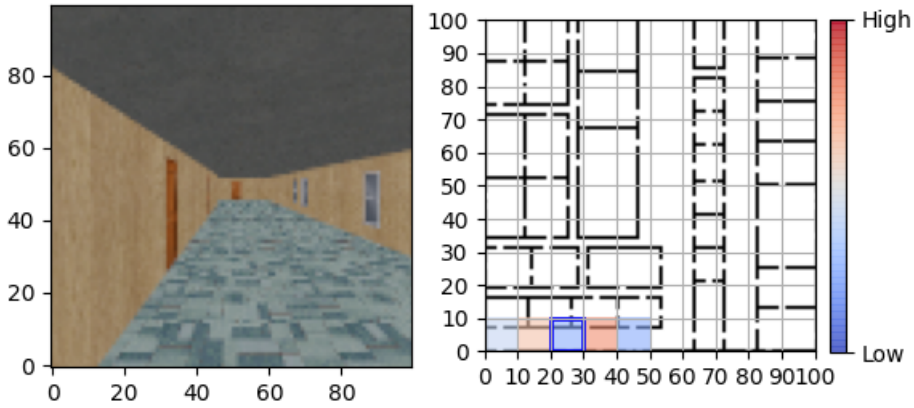

Figure 22: Heat map for the image on the left. In this case the top choice by the model was not the true location of the image.

block is computed. For these 4,614 images, the average position error was 31.58 meters off of the truth position with a standard deviation of 21.1 meters. The median position error was 30 meters.

4.2.2 Classification Model Results with New Floor Plan Representations

To further test the robustness of the classification CNN, an experiment was conducted to see how the model responded when the color of the walls on the floor plan

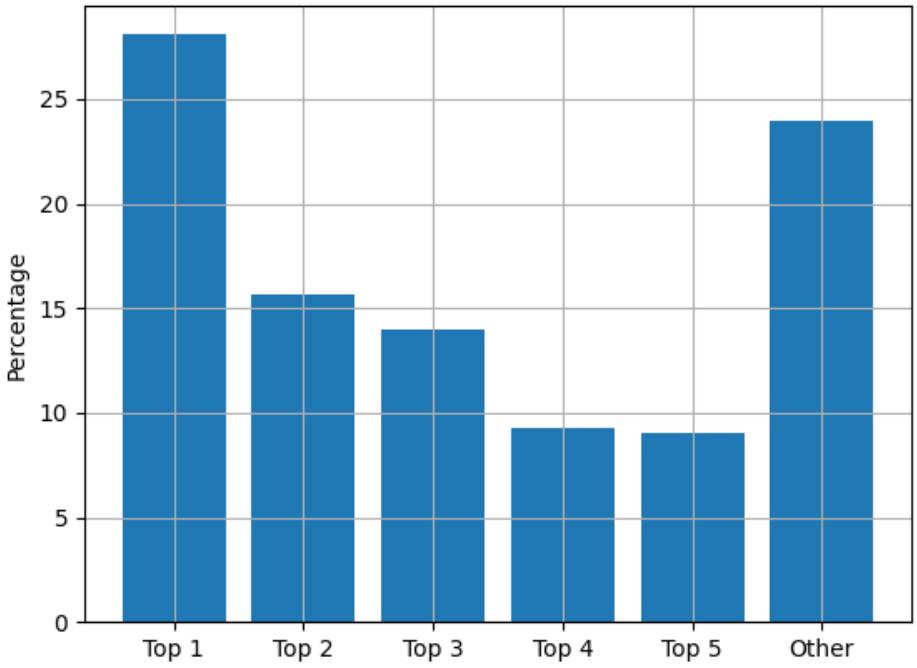

Figure 23: Bar plot depicting the number of correct predictions for each class on the 4,614 test data samples.

were changed to something other than black. Using the same test samples as before, the classification CNN was evaluated with red, green and blue representations of the original black floor plan. The results of that experiment are shown in Figure 24. For the most part the model wasn't bothered by the fact that the walls were represented by a different color. However, the floor plan that used red walls had the worst top 5 categorical accuracy which was 73.6%. The original black floor plan achieved a top 5 categorical accuracy of 76.1%. When the model was evaluated with the green floor plan, the average distance error was the highest at 32.06 meters, compared to the original black floor plan distance error which was 31.58 meters.

4.2.3 Classification Model Results with Floor Plan Rotations

This research also tested how the classification CNN performed when the floor plan image was rotated left or right by some degree. For the real data collection

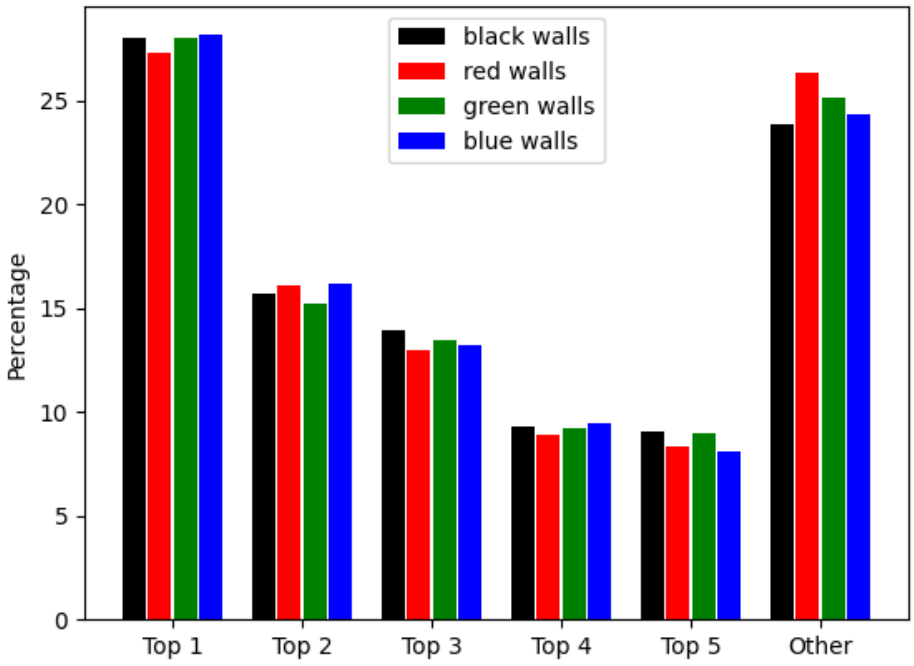

Figure 24: Bar plot depicting the number of correct predictions for each class on the 4,614 test data samples for each color floor plan.

portion of this research, the floor plan images were captured by hand with a cell phone as described in 3.2. In those cases, the floor plan image would most likely have some rotation to it. The classification CNN was evaluated four more times with the same test samples as before, except the floor plan images were rotated. The floor plan rotations tested were $-10°$, $-5°$, $5°$ and $10°$. Table 2 shows the results. In all cases where the floor plan was rotated, the top 5 categorical accuracy was worse than when it wasn't rotated. Furthermore, the larger the rotation, the worse the top 5 accuracy became. Also, the average distance error between the truth and predicted position was increased when compared to the original floor plan that was not rotated.

Rotation (Degrees)	% in Top 5	Avg Distance Error (m)
-10	59.65%	32.43
-5	66.67%	33.85
0	76.1%	31.58
5	64.65%	34.65
10	62.25%	33.64

Table 2: Summary of the results when floor plan image was rotated for the classification CNN.

4.3 Continuous Simulated Data Results

Next, the regression CNN was evaluated on the same 4,614 test samples. Predicting the XY position was a more difficult problem than the classification CNN was tasked to solve. This is due to the bimodal nature of the floor plan problem. Consider an image of a hallway. In the simulated data, multiple hallways could appear similar. The classification CNN would be able to assign higher probabilities to blocks in different hallways, which could still lead to an accurate top 5 categorical accuracy. However, with the regression CNN, when having to decide about similar hallway representations, it would have to predict the average of the positions. This would lead to it picking an XY position somewhere in between the predicted hallways, which would yield an inaccurate XY position prediction. Figure 25 shows the cumulative position error for the 4,614 test samples. The maximum distance error was 83.8 meters. 80% of the test samples had a position error of 25.4 meters or less.

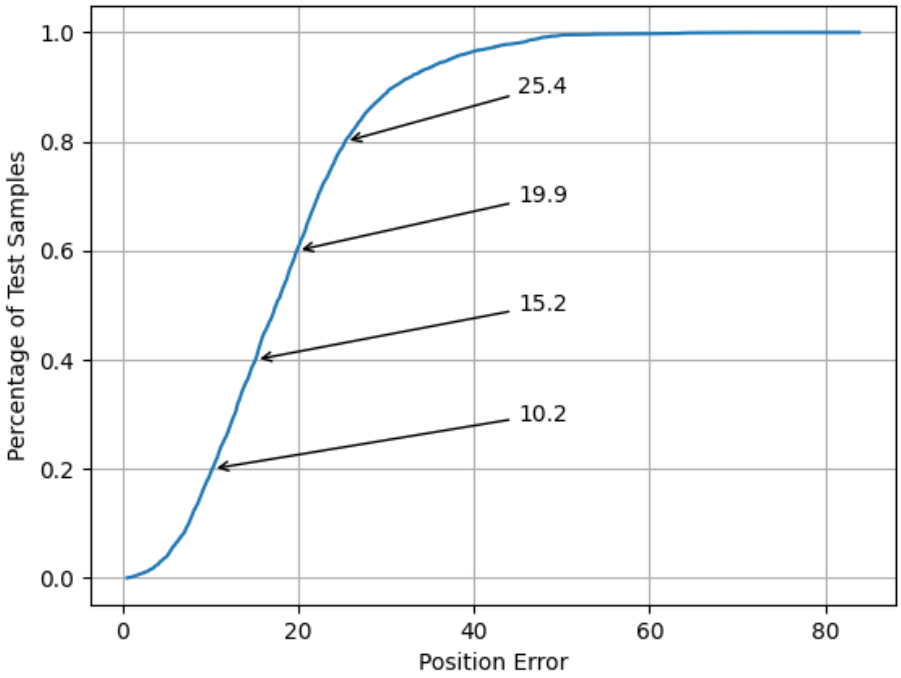

Figure 25: Cumulative Distribution Function plot showing the position error over the 4,614 test data samples.

4.4 Discrete Particle Filter Results

The discrete particle filter implementation discussed in 3.6.1 was evaluated five new simulated floor plans generated with the AftrBurner engine. All five floor plans along with the data collection paths are show in Appendix A. To test this with the simulated data, a sequence of images were selected from each floor plan directory data to simulate walking through that building. The total number of test images selected was 728. First, the classification CNN evaluated the images alone. Next, the discrete particle filter was added and the same images were evaluated again. The results are compared in Tables 3 and 4.

In all five experiments, the top 5 categorical accuracy was improved with the addition of the discrete particle filter. In experiment 2 the top 5 categorical accuracy was improved by over 11%. In this experiment, a large portion of the walked path was through tight hallways on the floor plan. The classification CNN did not perform

Experiment	% Top 1	% Top 2	% Top 3	% Top 4	% Top 5	% Other
Exp 1 no filter	41.53	33.05	11.02	7.63	3.39	3.39
Exp 1 with filter	45.76	42.37	11.02	0.0	0.85	0.0
Exp 2 no filter	27.05	17.07	10.37	7.93	9.76	26.83
Exp 2 with filter	40.24	14.63	15.24	6.71	7.93	15.24
Exp 3 no filter	52.41	16.55	9.66	6.90	2.76	11.72
Exp 3 with filter	47.59	20.0	8.28	6.21	6.90	11.03
Exp 4 no filter	56.12	20.14	2.88	2.88	1.44	16.55
Exp 4 with filter	64.75	7.91	8.63	1.44	1.44	15.83
Exp 5 no filter	59.26	16.05	4.32	3.70	6.79	9.88
Exp 5 with filter	72.22	21.60	3.70	1.85	0.0	0.62

Table 3: Summary comparison of the top 5 results for the classification CNN with and without the discrete particle filter over the five different floor plans.

as well on these types of floor plans as compared to floor plans that contained larger open areas. This could be due to hallways appearing similar in different areas of the floor plan. However, with the addition of the discrete particle filter, predictions that were too far away from a previous prediction would have had its probability reduced. This would help the system eliminate incorrect predictions and ultimately improve the overall navigation solution. The average distance error of the predictions were also compared. Table 4 shows these results. In all five experiments the average distance error between the truth and predicted positions were reduced when the discrete particle filter was utilized. Most notable in experiment 1 where the average distance error was reduced by 29.9 meters.

To better visualize the discrete particle filter experiments, Figure 27 shows the heat map results of experiment 5. In that figure, the green dots represent the true location of the sequence of images that made up the walked path. The different colors represent the aggregated probability of all the test images. Red colors indicate high probability locations, whereas blue colors indicate low probability locations. With the addition of the particle filter, many of the predictions that were far off from the walked path were eliminated. Also, the confidence in the images near the walked path

Classification CNN		
Experiment	**Average Distance Error (m)**	**Standard Deviation (m)**
Experiment 1 without filter	35.13	19.32
Experiment 1 with filter	5.23	9.13
Experiment 2 without filter	40.07	25.24
Experiment 2 with filter	25.20	27.28
Experiment 3 without filter	30.25	17.94
Experiment 3 with filter	14.63	17.40
Experiment 4 without filter	21.07	23.37
Experiment 4 with filter	15.72	20.58
Experiment 5 without filter	32.23	21.89
Experiment 5 with filter	11.17	16.81

Table 4: Summary comparison of average distance error and standard deviations for the classification CNN with and without the discrete particle filter over the five different floor plans.

were increased.

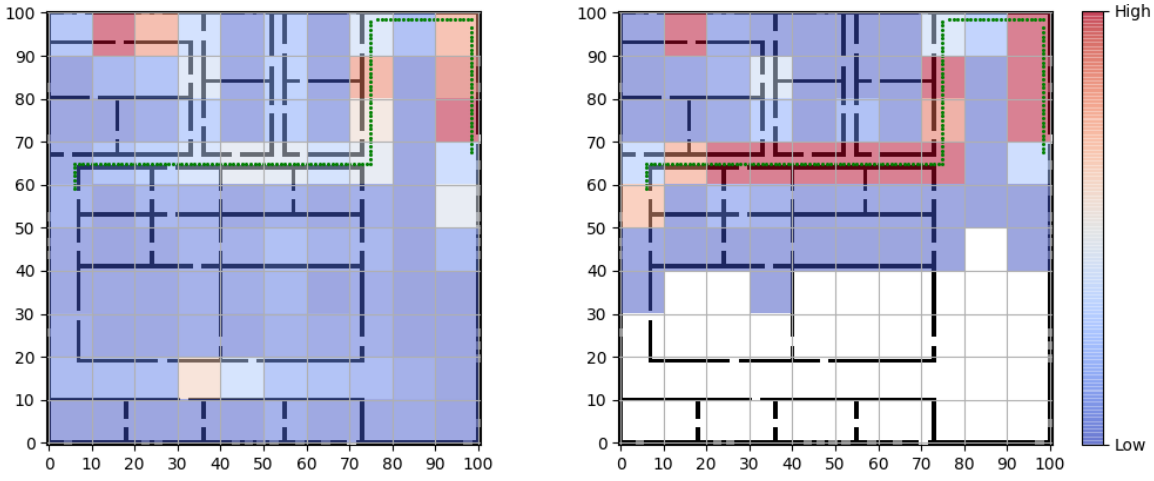

Figure 26: The heat map to the left shows the aggregated predictions of all test image by the classification CNN without the discrete particle filter for experiment 5. The heat map to the right shows the results when the particle filter is added. The truth positions of the walked path are shown by the green dots. With the addition of the particle filter the confidence in the walked path is increased, while also eliminating the number of low probability predictions. The white blocks on the bottom portion of the right heat map had a probability lower than 0.001 and were not shaded.

4.5 Continuous Particle Filter Results

The continuous particle filter described in 3.6.2 was developed to be used in conjunction with the regression CNN. The continuous particle filter was evaluated using the same five generated floor plans and test images that discrete particle filter used. First, the regression CNN evaluated the images alone. Next, the continuous particle filter was added and the images were evaluated again. A comparison of these results are shown in Table 5. In all five experiments, 80% of the test samples had a decrease in the average distance error when the continuous particle filter was included. Also, the maximum distance error was reduced in every experiment. In experiment 4, the maximum distance error was reduced by 66%. In that experiment, the predictions became inaccurate in the middle portion of the of the run. However, given the movement of the particles being at 2.5 meters, the average of the particle cloud was able to adjust for the bad predictions and reduce the maximum error for those predictions.

To illustrate how the continuous particle filter was used refer to Figure 27. This image shows the first nine particle clouds of experiment 5 overlaid on top of the floor

Average Distance Error (m) for % of Test Samples					
Experiment	**20%**	**40%**	**60%**	**80%**	**Max Error**
Experiment 1 without filter	2.8	5.9	9.0	12.9	33.7
Experiment 1 with filter	2.5	5.6	9.1	12.3	24.8
Experiment 2 without filter	22.4	27.9	32.2	37.3	76.9
Experiment 2 with filter	21.7	25.9	30.9	33.9	41.1
Experiment 3 without filter	11.6	17.4	20.9	24.6	36.7
Experiment 3 with filter	11.4	17.4	20.5	23.8	30.7
Experiment 4 without filter	4.6	7.0	11.6	21.9	75.9
Experiment 4 with filter	4.9	6.8	9.1	13.5	24.6
Experiment 5 without filter	4.1	7.2	12.8	18.5	94.8
Experiment 5 with filter	3.3	6.3	9.7	14.8	59.0

Table 5: Summary of the distance error for the five floor plan experiments conducted. The table shows the performance of the model alone and then with the addition of the continuous particle filter.

plan. The particles are represented by the red dots. Initially, they are all positioned around the known starting truth position. The blue dot represents the XY position prediction by the regression CNN for the given image. The black dot represents the truth position for that image. For each image, the particles begin to converge towards the predicted position. In the floor plan at the bottom right hand corner, it can be seen how the weighted average position of the particles are closer to the truth position than the prediction by the regression CNN alone. Thus, improving the initial measurement.

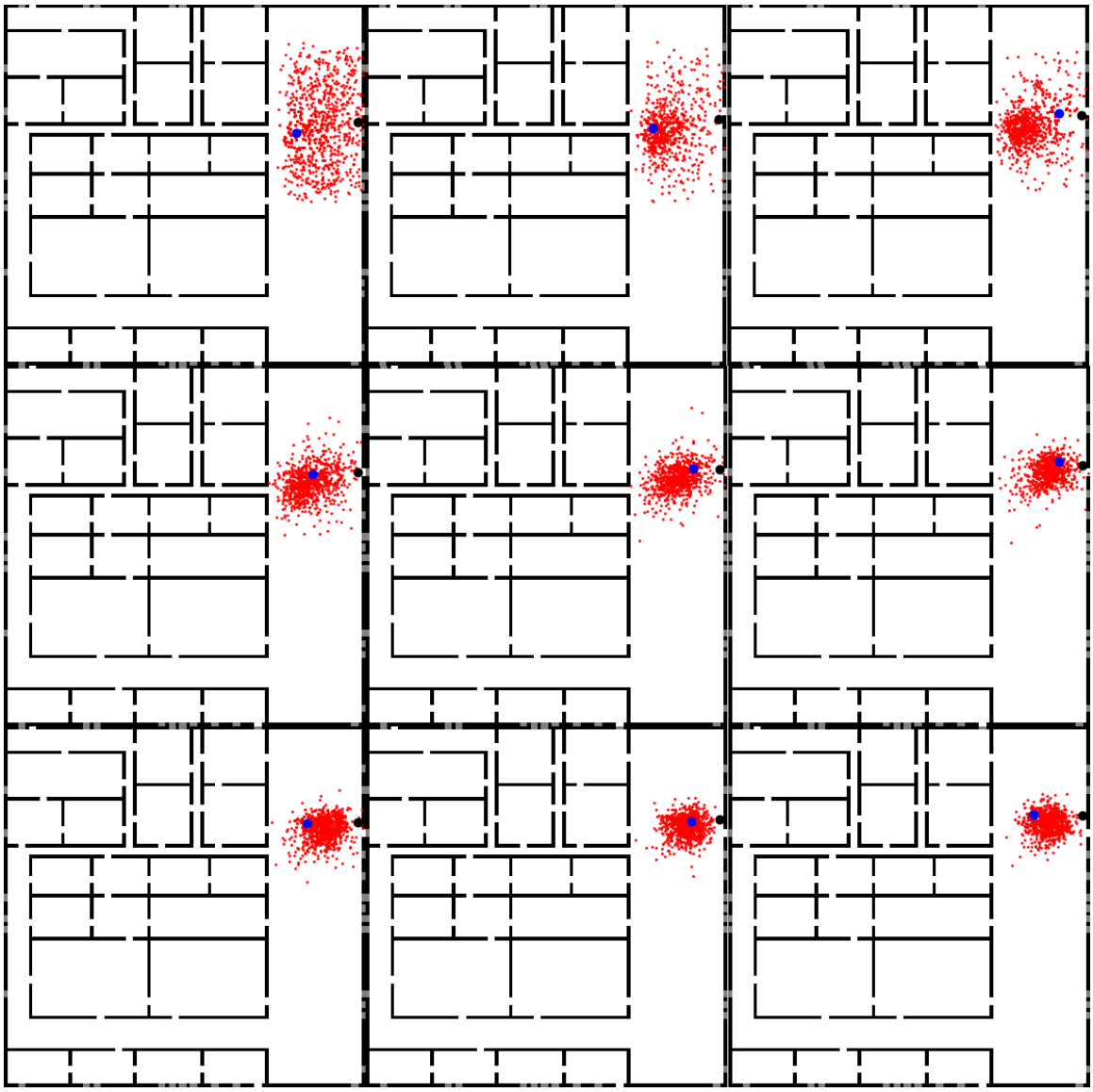

Figure 27: Starting from the top left and moving across, this image shows the convergence of the particles towards the predicted position for the first 9 test images. The black dot represents the truth position. The blue dot represents the position prediction by the regression CNN. The particles are represented by the red dots.

4.6 Comparing the Discrete and Continuous Particle Filters on Simulated Data

Table 6 shows the average distance errors for both the classification CNN and regression CNN with their respective particle filters for the five particle experiments.

In each experiment, the combined particle filter solution reduced the average distance error of the initial raw predictions by their respective models. The regression CNN alone had lower error metrics than the classification CNN in all experiments. However, the regression CNN combined with the continuous particle filter did not have a lower average error than the discrete particle filter in experiments 1, 2 and 3. The continuous particle filter included wall constraints from the floor plans, which made it difficult for particles to cross over wall segments. However, it was possible for particles to cross a wall segment and then be selected for resampling. When this happened, it was observed that the particle cloud could become trapped inside of a room for some time due to being bounded by the same wall constraints. The majority of the particles would remain there until the dynamics and resampling allowed the particles to escape. This could cause the particle cloud to begin to lag behind the predictions of the regression CNN, leading to less accurate position predictions. An example of this concept is shown in Figure 28.

Average Distance Error (m)		
Experiment	**Classification CNN Discrete PF**	**Regression CNN Continuous PF**
Experiment 1 without filter	35.13	8.95
Experiment 1 with filter	5.23	8.05
Experiment 2 without filter	40.07	31.95
Experiment 2 with filter	25.20	27.92
Experiment 3 without filter	30.25	18.53
Experiment 3 with filter	14.63	17.96
Experiment 4 without filter	21.07	13.94
Experiment 4 with filter	15.72	8.47
Experiment 5 without filter	32.23	15.2
Experiment 5 with filter	11.17	10.37

Table 6: Comparison of average distance errors for the classification CNN and regression CNN with and without their respective particle filters over the five different floor plans.

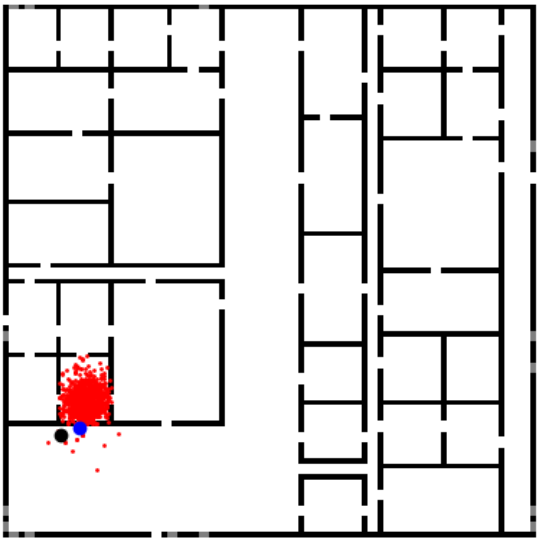

Figure 28: This figure shows an example of when particles became trapped inside of a room on the floor plan. The particles were initially draw into the room due to an inaccurate prediction by the regression CNN, and due to the wall constraints of the floor plan, remained there until the right happenstance of dynamics and resampling occurred for them to escape.

4.7 Real Data Results

The performance of the classification CNN was evaluated on real data. This data was collected with the TurtleBot 3 as described in 3.2. Initially, the classification CNN did not perform well with real data, achieving 4.8% top 5 accuracy. However, given the fact that the model was not trained with imagery of this detail this was expected. Another point to mention is that the images taken from the TurtleBot 3 were from a lower perspective than the ones taken in the simulated environment.

In order to improve the results, modifications were made to the the data set and the model was retrained in the same manner as before. A small amount of real training data was added to the data set. This data was procured by mapping a portion of the first floor of building 644 at the Air Force Institute of Technology (AFIT) with

the TurtleBot 3. After using the data augmentation techniques as described in 3.3, a total of 2,304 real data images were generated for training. Another observation that was made after the first real data experiment, was the difference in lighting that appeared in the images. To incorporate that into training, 20% of the original simulated data set was altered to include darkened images. This augmentation was done with the ImageEnhance module of Pillow, a Python imaging library. After retraining the classification CNN with the new data set, the module was evaluated again on the real data collected from building 640. The top 5 accuracy was improved from 4.8% to 10.7%. This demonstrates the ability of this solution to transition to real world data, if enough real world training data can be ascertained.

Another item that was believed to be an issue during this experiment was the quality of the floor plan after being resized to 101 $\times$ 101. Given its initial high resolution and due to the image interpolation process, the resulting floor plan was quite blurry which can have an effect on the models ability to interpret the floor plan. The floor plan also contained different colors and symbols for doors and windows than the simulated floor plans had, which would also be confusing for the model as it had not been trained to identify these features. To verify this, the a new floor plan of building 640 was generated to resemble the style of floor plans that the model was trained with. This floor plan is shown in Figure 29. The new data set along with this newly created floor plan yielded a top 5 accuracy of 16.6%, an improvement of almost 6% over the experiment that used the true floor plan. As with the real images, the model was not trained with these types of floor plans and demonstrates the need to develop a data set that includes the type of data that would be seen in the real world.

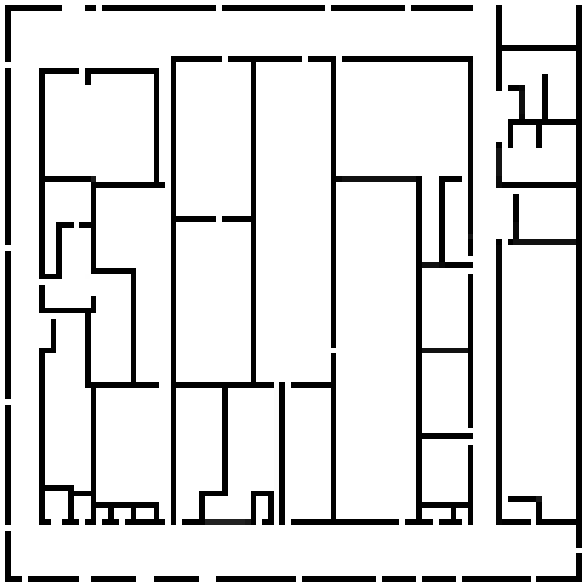

Figure 29: A portion of the floor plan for the third floor of building 640 recreated in the same style as the simulated floor plans.

V. Conclusions

The purpose of this research was to explore the viability of a Convolutional Neural Network (CNN) incorporating floor plans along with monocular camera images as a potential means for indoor navigation. A simulated training data set was created that modeled 1,000 building floor plans. Two CNN models were developed that attempted to localize an image on the given floor plan. The first CNN performed classification and achieved 76.1% top 5 categorical accuracy on test data. The second CNN was a regression model and achieved a max distance error of 25.4 meters or less on 80% of test data. This solution is unique compared to other navigation solutions, in that it doesn't require the use of inertial measurement unit (IMU) sensors or wireless transmitting devices. Nor does it require mapping a building ahead of time in order to make accurate position predictions. However, the floor plan of the building that is being navigated is required.

To improve the results of the models and to demonstrate the potential to incorporate them with other navigation solutions, a discrete and a continuous particle filter were both implemented. Through the course of five experiments, both particle filter implementations were able to reduce the average distance error between the truth and predicted positions of the original models. Finally, the classification CNN was evaluated with real data. Initially, the model only achieved a top 5 accuracy of 4.8%. After retraining the model with a small of amount of real imagery, the model was able to achieve 10.7% top 5 accuracy on the test data. While these are not great results, the amount of real data needed to improve these metrics could not be procured at the given time. However, the results from the real data experiment do indicate that the model could achieve similar results as the simulated data if given enough real data to learn from.

5.1 Future Work

This research showed that it is possible for a CNN to learn features of a building floor plan and incorporate them in a localization prediction. However, this work was not comprehensive and additional work can be done to improve the results. Possible improvements are as follows:

- Acquire a more real data to train with. This would involve imaging a large number of buildings in order for the models to generalize to the different building architectures and their associated floor plans. The location of the captured image would also need to be known relative to the floor plan.

- If real data is not able to be collected, then improving the existing simulated data set could also improve the results. This can be done by generating more types of building architectures other than rectangular shapes. Also, specifying different symbols for features such as doors and windows could help with generalization. Furthermore, depicting additional features inside of the simulated buildings such as exit signs and bath room signs could provide additional ways to determine an image location against the associated floor plan. Finally, using more realistic textures for the indoor environment could help the model generalize better to real data.

- A better process for downsizing the real data floor plan image to the needed input size of 101×101 needs to be explored. Possible solutions could be capturing the floor plan at a lower resolution and then downsizing or utilizing a better sharpening filter to increase the precision of lines in the floor plan image.

- Utilizing hyper-parameter optimization algorithms to further improve the learning process.

- Improve the continuous particle filter implementation. This research incorporated floor plans into the continuous particle filter by penalizing particles that crossed over wall segments on the floor plan. However, this lead to issues were particles could become trapped inside of rooms. A possible improvement would be to not allow particles to cross wall segments at all. Instead have the particles bounce of the walls and remain in the hallways.

Appendix A. Additional Results

Table 7: Classification Model Hyperparameters.

Layer Name (Type)	Size	Activation	Output Shape	Connected To
input_2 (InputLayer)			(None, 100, 100, 3)	
input_1 (InputLayer)			(None, 101, 101, 3)	
conv2d_15 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	input_2
conv2d (Conv2D)	3x3	tanh	(None, 101, 101, 16)	input_1
conv2d_16 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	conv2d_15
conv2d_1 (Conv2D)	3x3	tanh	(None, 101, 101, 16)	conv2d
conv2d_17 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	conv2d_16
conv2d_2 (Conv2D)	3x3	tanh	(None, 101, 101, 16)	conv2d_1
max_pooling2d_2 (MaxPooling2D)	2x2		(None, 50, 50, 16)	conv2d_17
max_pooling2d (MaxPooling2D)	2x2		(None, 50, 50, 16)	conv2d_2
dropout_4 (Dropout)			(None, 50, 50, 16)	max_pooling2d_2
dropout (Dropout)			(None, 50, 50, 16)	max_pooling2d
conv2d_18 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	dropout_4
conv2d_3 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	dropout
conv2d_19 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	conv2d_18
conv2d_4 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	conv2d_3
conv2d_20 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	conv2d_19
conv2d_5 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	conv2d_4
max_pooling2d_3 (MaxPooling2D)	2x2		(None, 25, 25, 32)	conv2d_20
max_pooling2d_1 (MaxPooling2D)	2x2		(None, 25, 25, 32)	conv2d_5
dropout_5 (Dropout)			(None, 25, 25, 32)	max_pooling2d_3

dropout_1 (Dropout)			(None, 25, 25, 32)	max_pooling2d_1
conv2d_21 (Conv2D)	3x3	ReLu	(None, 25, 25, 64)	dropout_5
conv2d_6 (Conv2D)	3x3	tanh	(None, 25, 25, 64)	dropout_1
conv2d_22 (Conv2D)	3x3	ReLu	(None, 25, 25, 48)	conv2d_21
conv2d_7 (Conv2D)	3x3	tanh	(None, 25, 25, 48)	conv2d_6
conv2d_23 (Conv2D)	3x3	ReLu	(None, 25, 25, 32)	conv2d_22
conv2d_8 (Conv2D)	3x3	tanh	(None, 25, 25, 32)	conv2d_7
up_sampling2d_2 (UpSampling2D)	2x2		(None, 50, 50, 32)	conv2d_23
up_sampling2d (UpSampling2D)	2x2		(None, 50, 50, 32)	conv2d_8
concatenate_1 (Concatenate)			(None, 50, 50, 64)	conv2d_20 up_sampling2d_2
concatenate (Concatenate)			(None, 50, 50, 64)	conv2d_5 up_sampling2d
dropout_6 (Dropout)			(None, 50, 50, 64)	concatenate_1
dropout_2 (Dropout)			(None, 50, 50, 64)	concatenate
conv2d_24 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	dropout_6
conv2d_9 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	dropout_2
conv2d_25 (Conv2D)	3x3	ReLu	(None, 50, 50, 24)	conv2d_24
conv2d_10 (Conv2D)	3x3	tanh	(None, 50, 50, 24)	conv2d_9
conv2d_26 (Conv2D)	3x3	ReLu	(None, 50, 50, 16)	conv2d_9
conv2d_11 (Conv2D)	3x3	tanh	(None, 50, 50, 16)	conv2d_25
up_sampling2d_3 (UpSampling2D)	2x2		(None, 100, 100, 16)	conv2d_26
up_sampling2d_1 (UpSampling2D)	2x2		(None, 100, 100, 16)	conv2d_11
concatenate_2 (Concatenate)			(None, 100, 100, 32)	conv2d_17 up_sampling2d_3

concatenate_3 (Concatenate)			(None, 100, 100, 32)	conv2d_2 up_sampling2d_1
dropout_7 (Dropout)			(None, 100, 100, 32)	concatenate_2
dropout_3 (Dropout)			(None, 100, 100, 32)	concatenate_3
conv2d_27 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	dropout_7
conv2d_12 (Conv2D)	3x3	tanh	(None, 100, 100, 16)	dropout_3
conv2d_28 (Conv2D)	3x3	ReLu	(None, 100, 100, 24)	conv2d_27
conv2d_13 (Conv2D)	3x3	tanh	(None, 100, 100, 24)	conv2d_12
conv2d_29 (Conv2D)	3x3	ReLu	(None, 100, 100, 32)	conv2d_28
conv2d_14 (Conv2D)	3x3	tanh	(None, 100, 100, 32)	conv2d_13
concatenate_4 (Concatenate)			(None, 100, 100, 64)	conv2d_29 conv2d_14
dropout_8 (Dropout)			(None, 100, 100, 64)	concatenate_4
conv2d_30 (Conv2D)	1x1	ReLu	(None, 100, 100, 64)	dropout_8
bn (BatchNormalization)			(None, 100, 100, 64)	conv2d_30
max_pooling2d_4 (MaxPooling2D)	5x5		(None, 20, 20, 64)	bn
conv2d_31 (Conv2D)	1x1	ReLu	(None, 20, 20, 1)	max_pooling2d_4
bn_1 (BatchNormalization)			(None, 20, 20, 1)	conv2d_31
max_pooling2d_5 (MaxPooling2D)	2x2		(None, 10, 10, 1)	bn_1
flatten (Flatten)			(None, 100)	max_pooling2d_5
dense (Dense)	100	softmax	(None, 100)	flatten

Table 8: Regression Model Hyperparameters.

Layer Name (Type)	**Size**	**Activation**	**Output Shape**	**Connected To**
input_2 (InputLayer)			(None, 100, 100, 3)	
input_1 (InputLayer)			(None, 101, 101, 3)	
conv2d_15 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	input_2
conv2d (Conv2D)	3x3	tanh	(None, 101, 101, 16)	input_1
conv2d_16 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	conv2d_15
conv2d_1 (Conv2D)	3x3	tanh	(None, 101, 101, 16)	conv2d
conv2d_17 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	conv2d_16
conv2d_2 (Conv2D)	3x3	tanh	(None, 101, 101, 16)	conv2d_1
max_pooling2d_2 (MaxPooling2D)	2x2		(None, 50, 50, 16)	conv2d_17
max_pooling2d (MaxPooling2D)	2x2		(None, 50, 50, 16)	conv2d_2
dropout_4 (Dropout)			(None, 50, 50, 16)	max_pooling2d_2
dropout (Dropout)			(None, 50, 50, 16)	max_pooling2d
conv2d_18 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	dropout_4
conv2d_3 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	dropout
conv2d_19 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	conv2d_18
conv2d_4 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	conv2d_3
conv2d_20 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	conv2d_19
conv2d_5 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	conv2d_4
max_pooling2d_3 (MaxPooling2D)	2x2		(None, 25, 25, 32)	conv2d_20
max_pooling2d_1 (MaxPooling2D)	2x2		(None, 25, 25, 32)	conv2d_5
dropout_5 (Dropout)			(None, 25, 25, 32)	max_pooling2d_3
dropout_1 (Dropout)			(None, 25, 25, 32)	max_pooling2d_1
conv2d_21 (Conv2D)	3x3	ReLu	(None, 25, 25, 64)	dropout_5

conv2d_6 (Conv2D)	3x3	tanh	(None, 25, 25, 64)	dropout_1
conv2d_22 (Conv2D)	3x3	ReLu	(None, 25, 25, 48)	conv2d_21
conv2d_7 (Conv2D)	3x3	tanh	(None, 25, 25, 48)	conv2d_6
conv2d_23 (Conv2D)	3x3	ReLu	(None, 25, 25, 32)	conv2d_22
conv2d_8 (Conv2D)	3x3	tanh	(None, 25, 25, 32)	conv2d_7
up_sampling2d_2 (UpSampling2D)	2x2		(None, 50, 50, 32)	conv2d_23
up_sampling2d (UpSampling2D)	2x2		(None, 50, 50, 32)	conv2d_8
concatenate_1 (Concatenate)			(None, 50, 50, 64)	conv2d_20 up_sampling2d_2
concatenate (Concatenate)			(None, 50, 50, 64)	conv2d_5 up_sampling2d
dropout_6 (Dropout)			(None, 50, 50, 64)	concatenate_1
dropout_2 (Dropout)			(None, 50, 50, 64)	concatenate
conv2d_24 (Conv2D)	3x3	ReLu	(None, 50, 50, 32)	dropout_6
conv2d_9 (Conv2D)	3x3	tanh	(None, 50, 50, 32)	dropout_2
conv2d_25 (Conv2D)	3x3	ReLu	(None, 50, 50, 24)	conv2d_24
conv2d_10 (Conv2D)	3x3	tanh	(None, 50, 50, 24)	conv2d_9
conv2d_26 (Conv2D)	3x3	ReLu	(None, 50, 50, 16)	conv2d_9
conv2d_11 (Conv2D)	3x3	tanh	(None, 50, 50, 16)	conv2d_25
up_sampling2d_3 (UpSampling2D)	2x2		(None, 100, 100, 16)	conv2d_26
up_sampling2d_1 (UpSampling2D)	2x2		(None, 100, 100, 16)	conv2d_11
concatenate_2 (Concatenate)			(None, 100, 100, 32)	conv2d_17 up_sampling2d_3
concatenate_3 (Concatenate)			(None, 100, 100, 32)	conv2d_2 up_sampling2d_1

dropout_7 (Dropout)			(None, 100, 100, 32)	concatenate_2
dropout_3 (Dropout)			(None, 100, 100, 32)	concatenate_3
conv2d_27 (Conv2D)	3x3	ReLu	(None, 100, 100, 16)	dropout_7
conv2d_12 (Conv2D)	3x3	tanh	(None, 100, 100, 16)	dropout_3
conv2d_28 (Conv2D)	3x3	ReLu	(None, 100, 100, 24)	conv2d_27
conv2d_13 (Conv2D)	3x3	tanh	(None, 100, 100, 24)	conv2d_12
conv2d_29 (Conv2D)	3x3	ReLu	(None, 100, 100, 32)	conv2d_28
conv2d_14 (Conv2D)	3x3	tanh	(None, 100, 100, 32)	conv2d_13
concatenate_4 (Concatenate)			(None, 100, 100, 64)	conv2d_29 conv2d_14
dropout_8 (Dropout)			(None, 100, 100, 64)	concatenate_4
conv2d_30 (Conv2D)	1x1	ReLu	(None, 100, 100, 64)	dropout_8
bn (BatchNormalization)			(None, 100, 100, 64)	conv2d_30
max_pooling2d_4 (MaxPooling2D)	5x5		(None, 20, 20, 64)	bn
conv2d_31 (Conv2D)	1x1	ReLu	(None, 20, 20, 1)	max_pooling2d_4
bn_1 (BatchNormalization)			(None, 20, 20, 1)	conv2d_31
max_pooling2d_5 (MaxPooling2D)	2x2		(None, 10, 10, 1)	bn_1
flatten (Flatten)			(None, 100)	max_pooling2d_5
dense (Dense)	100	ReLu	(None, 100)	flatten
dense_1 (Dense)	2	Linear	(None, 2)	dense

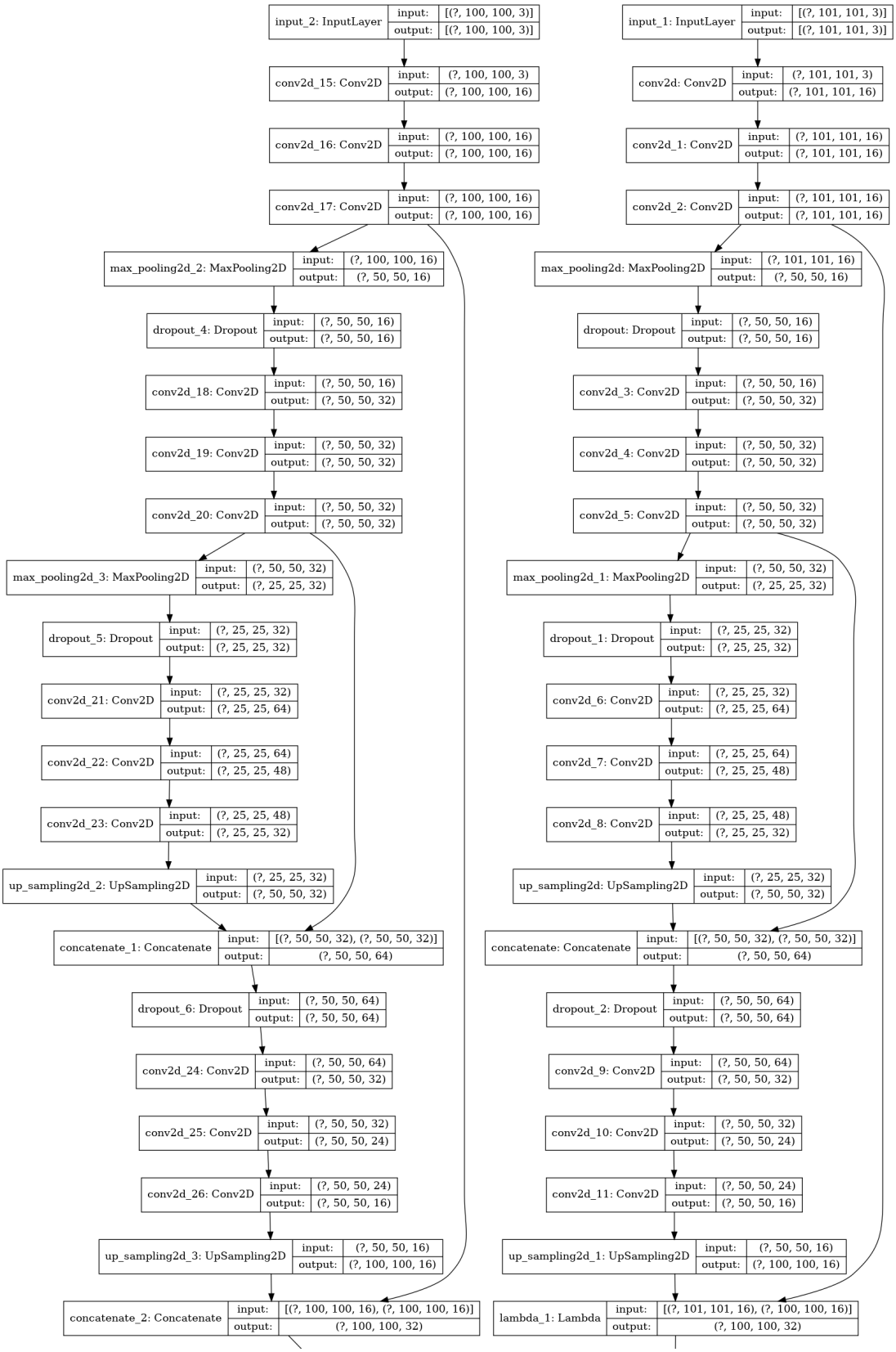

Figure 30: Top Layers of Classification Model

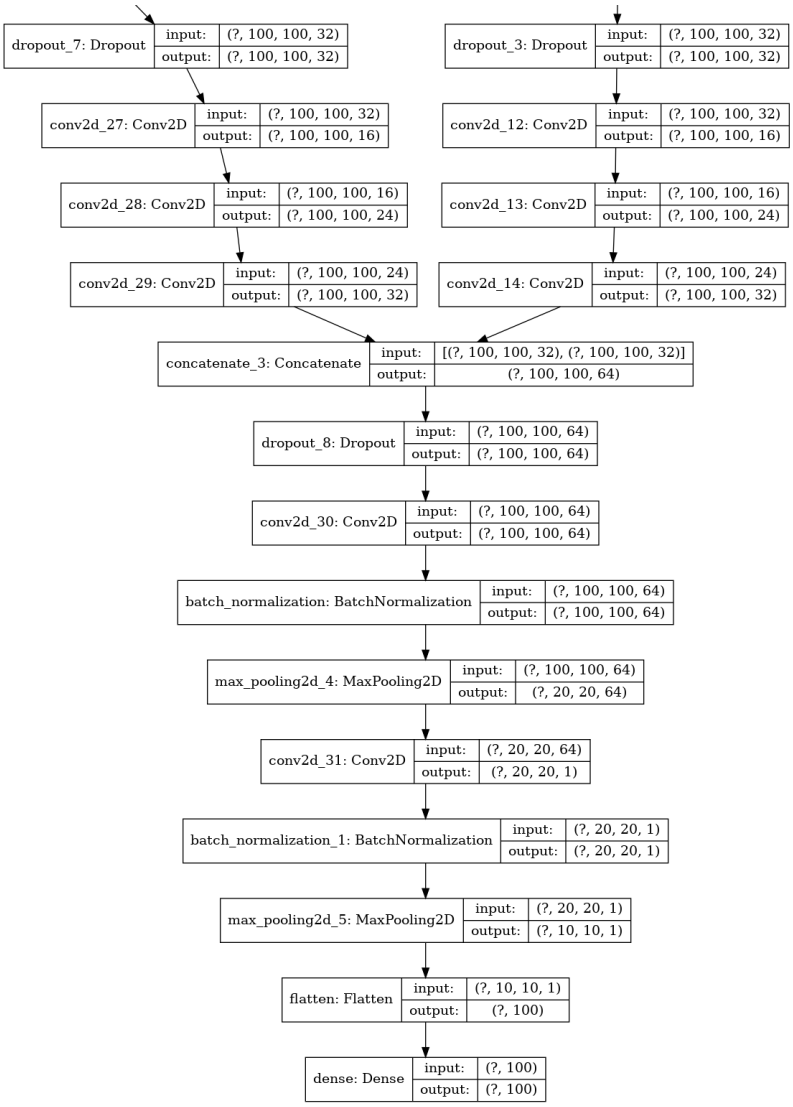

Figure 31: Bottom Layers of Classification Model

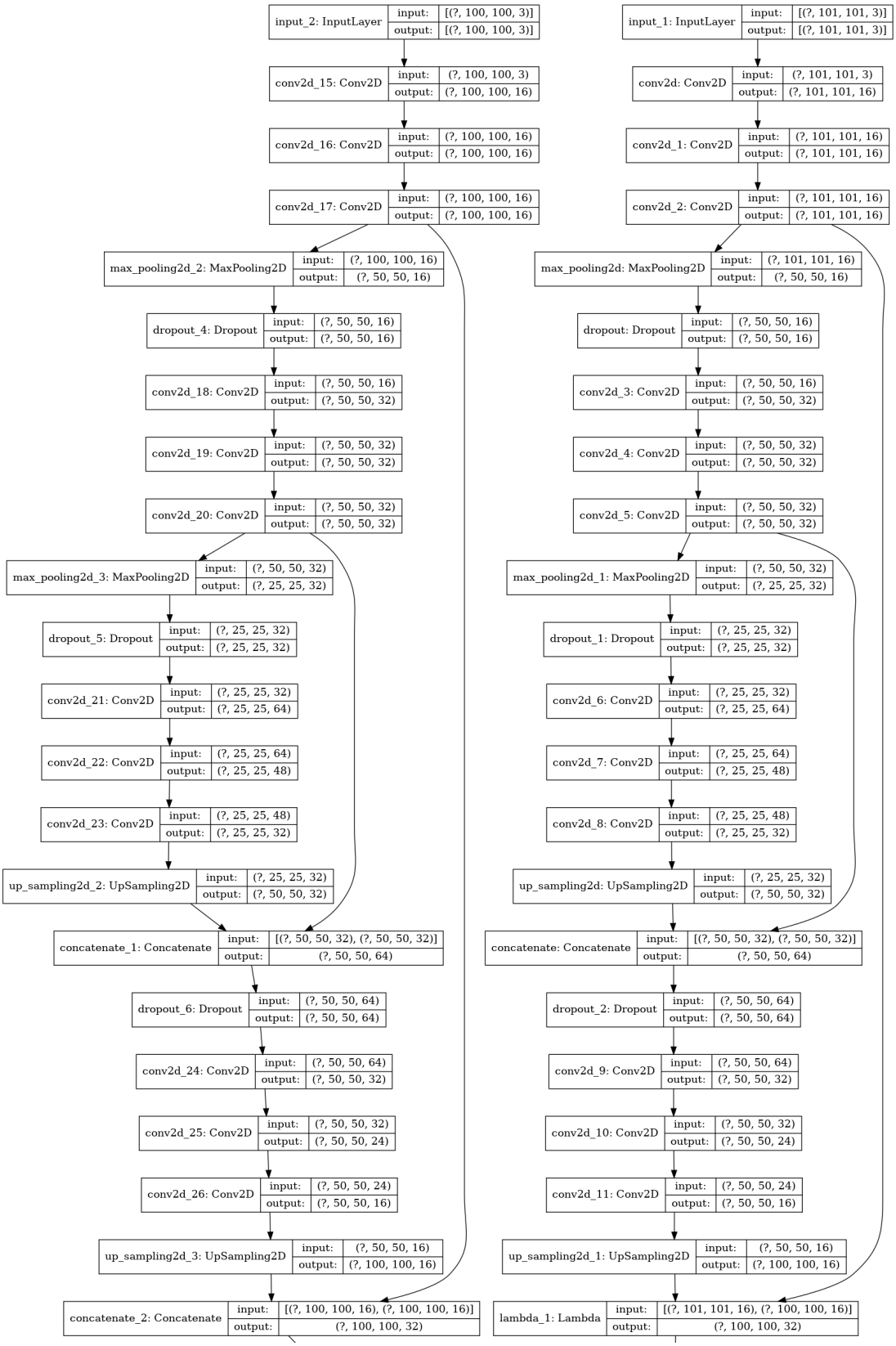

Figure 32: Top Layers of Regression Model

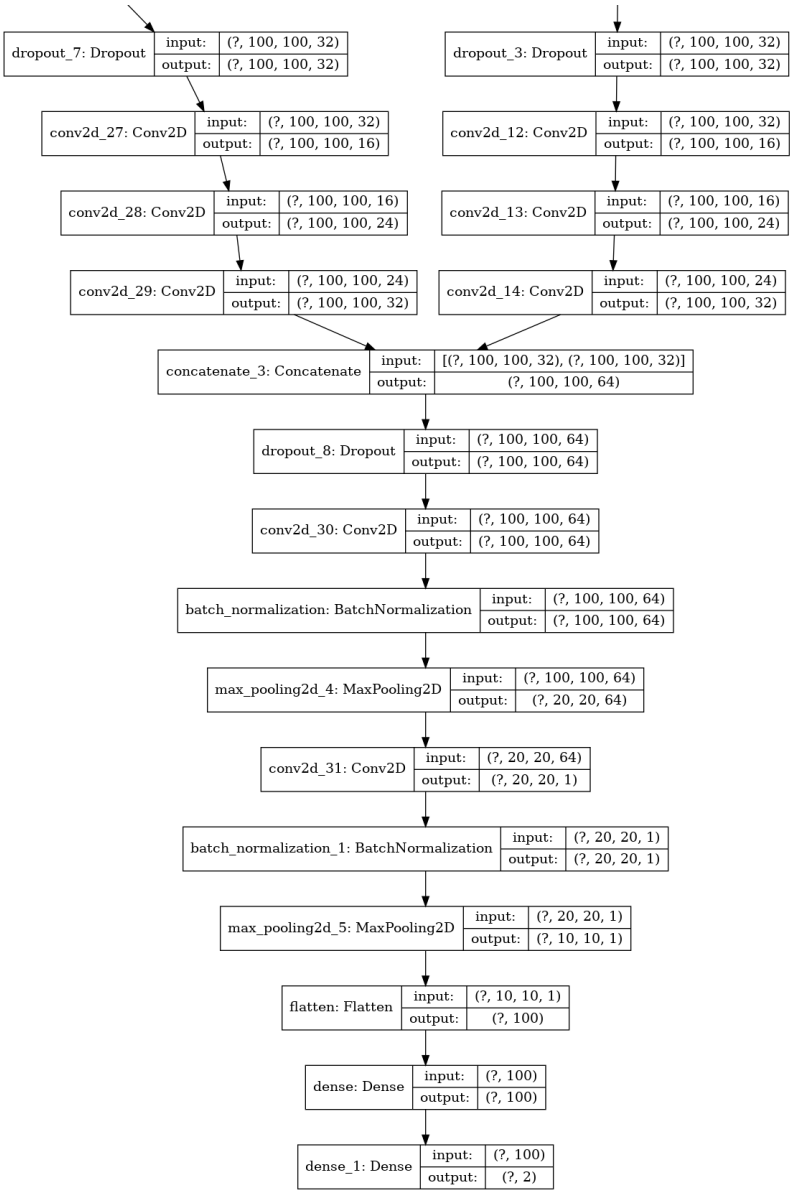

Figure 33: Bottom Layers of Regression Model

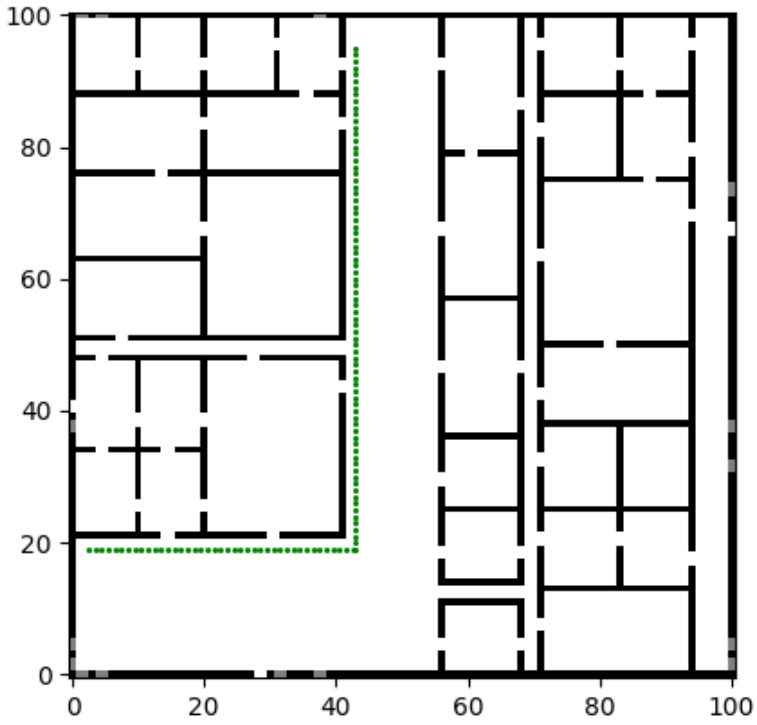

Figure 34: The floor plan that was used for experiment 1. The green dots show the true locations of the images for the walked path.

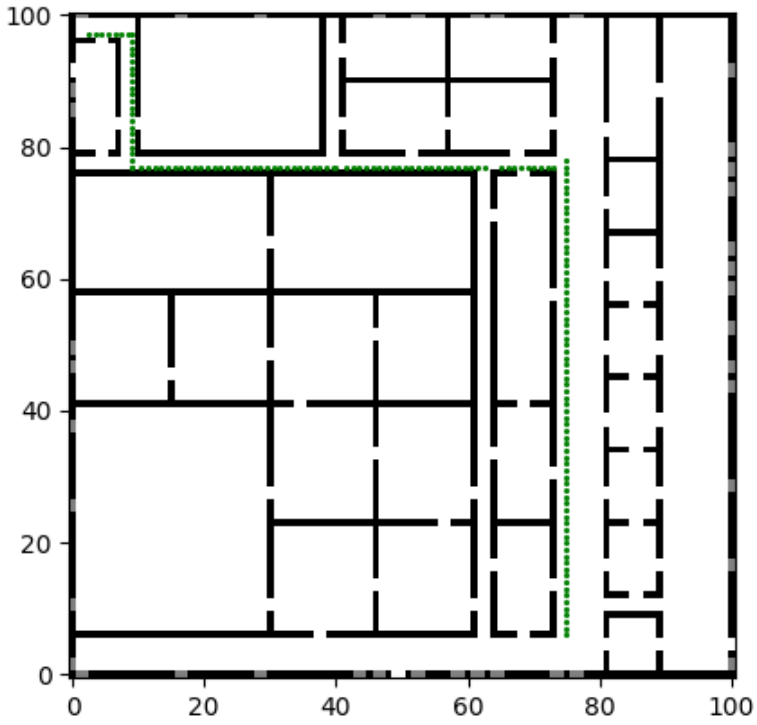

Figure 35: The floor plan that was used for experiment 2. The green dots show the true locations of the images for the walked path.

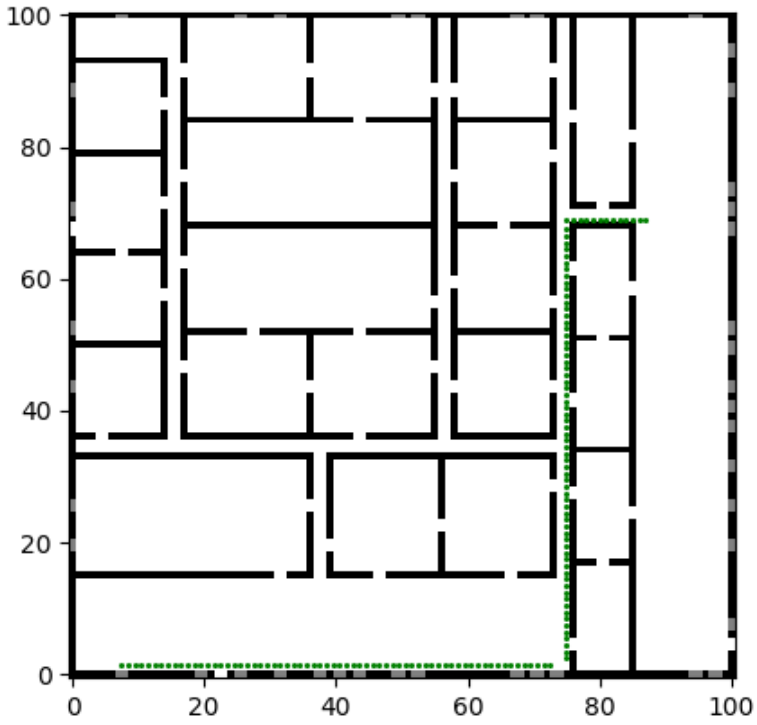

Figure 36: The floor plan that was used for experiment 3. The green dots show the true locations of the images for the walked path.

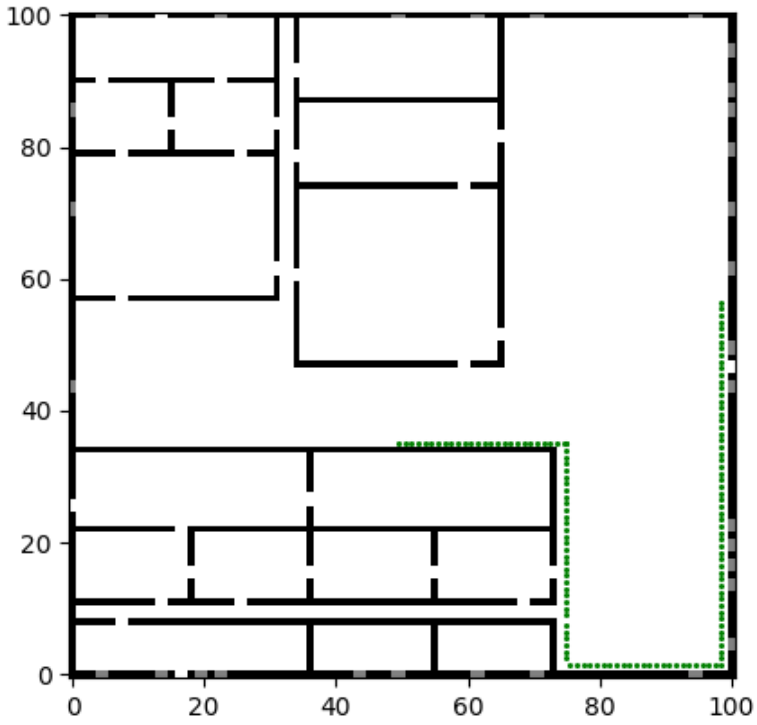

Figure 37: The floor plan that was used for experiment 24. The green dots show the true locations of the images for the walked path.

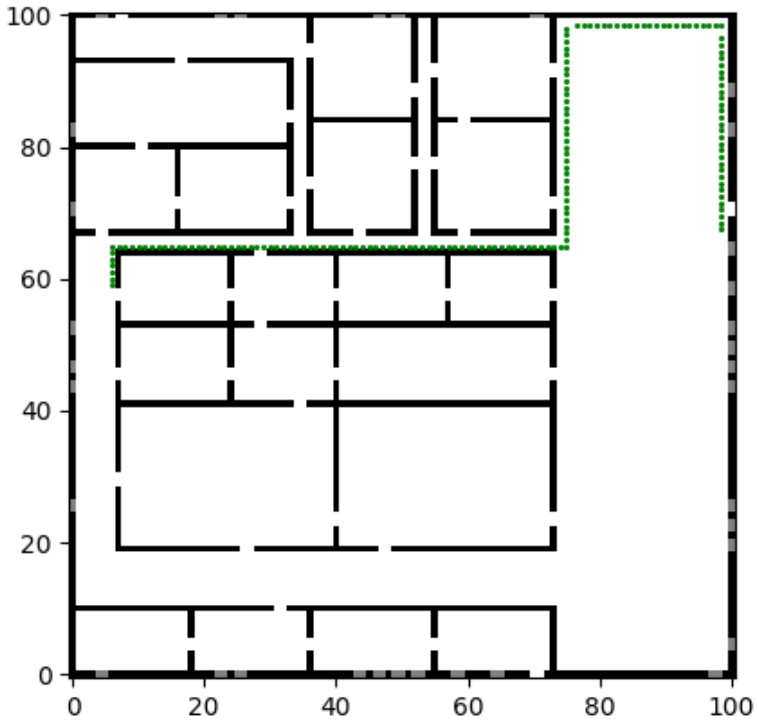

Figure 38: The floor plan that was used for experiment 5. The green dots show the true locations of the images for the walked path.

Bibliography

1. H. Ju, S. Y. Park, and C. G. Park. A smartphone-based pedestrian dead reckoning system with multiple virtual tracking for indoor navigation. *IEEE Sensors Journal*, 18(16):6756–6764, 2018.

2. Z. Hu, G. Huang, Y. Hu, and Z. Yang. Wi-vi fingerprint: Wifi and vision integrated fingerprint for smartphone-based indoor self-localization. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4402–4406, 2017.

3. X. Hou and T. Arslan. Monte carlo localization algorithm for indoor positioning using bluetooth low energy devices. In *2017 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–6, 2017.

4. Christian Röger and Sabine Timpf. Indoor mapping for human navigation – a low-cost slam solution. $GI_Forum, 1 : 152 - -161, 072018$.

5. Sebastian Hilsenbeck, Andreas Möller, Robert Huitl, Georg Schroth, Matthias Kranz, and Eckehard Steinbach. Scale-preserving long-term visual odometry for indoor navigation. pages 1 –10, 11 2012.

6. J. Liao, K. Chiang, H. Chang, and Y. Li. Artificial neural networks aided image localization for pedestrian dead reckoning for indoor navigation applications. In *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, pages 1–10, 2018.

7. Ayush Mittal, Saideep Tiku, and Sudeep Pasricha. Adapting convolutional neural networks for indoor localization with smart mobile devices. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, GLSVLSI '18, page 117–122, New York, NY, USA, 2018. Association for Computing Machinery.

8. Song Xu, Wusheng Chou, and Hongyi Dong. A robust indoor localization system integrating visual localization aided by CNN-based image retrieval with Monte Carlo localization. *Sensors (Switzerland)*, 19(2), jan 2019.

9. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edition, 2018.

10. Michael A. Nielsen. Neural networks and deep learning, 2018.

11. François Chollet. *Deep Learning with Python*. Manning, November 2017.

12. Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

13. Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. cite arxiv:1803.08375Comment: 7 pages, 11 figures, 9 tables.

14. Mahesh Chandra. A novel method for scalable vlsi implementation of hyperbolic tangent function, 2020.

15. Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

16. Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. nov 2018.

17. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

18. Sebastian Ruder. An overview of gradient descent optimization algorithms. sep 2016.

19. Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology, aug 2018.

20. Joseph A. Curro II. Navigation with Artificial Neural Networks. pages 1–211, 2018.

21. Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

22. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

23. Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, 02 2019.

24. Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

25. S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

26. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, pages 234–241. Springer International Publishing, Cham, 2015.

27. Y. Xiao, Y. Ou, and W. Feng. Localization of indoor robot based on particle filter with ekf proposal distribution. In *2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 568–571, 2017.

28. S. Katwe and N. Iyer. Map-based particle filter for localization: Autonomous vehicle. In *2020 International Conference on Computational Performance Evaluation (ComPE)*, pages 303–304, 2020.

29. Jinxia Yu, Yongli Tang, Zixing Cai, and Zhuohua Duan. Monte carlo localization for mobile robot with the improvement of particle filter. In *2008 7th World Congress on Intelligent Control and Automation*, pages 3910–3914, 2008.

30. P. M. Djuric, J. H. Kotecha, Jianqui Zhang, Yufei Huang, T. Ghirmai, M. F. Bugallo, and J. Miguez. Particle filtering. *IEEE Signal Processing Magazine*, 20(5):19–38, 2003.

31. Pavel Davidson, Jussi Collin, and Jarmo Takala. Application of particle filters for indoor positioning using floor plans. pages 1 – 4, 11 2010.

32. X. Du, X. Liao, Z. Gao, and Y. Fan. An enhanced particle filter algorithm with map information for indoor positioning system. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

33. Wonho Kang and Youngnam Han. SmartPDR: Smartphone-based pedestrian dead reckoning for indoor localization. *IEEE Sensors Journal*, 15(5):2906–2916, 2015.

34. Jian Wang, Andong Hu, Chunyan Liu, and Xin Li. A floor-map-aided WiFi/pseudo-odometry integration algorithm for an indoor positioning system. *Sensors (Switzerland)*, 15(4):7096–7124, 2015.

35. M. Altini, D. Brunelli, E. Farella, and L. Benini. Bluetooth indoor localization with multiple neural networks. In *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*, pages 295–300, 2010.

36. T. Sanpechuda and L. Kovavisaruch. A review of rfid localization: Applications and techniques. In *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, volume 2, pages 769–772, 2008.

37. Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. pages 2938–2946, 2015.

38. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

39. Ali Ghofrani, Rahil Mahdian Toroghi, and Sayed Mojtaba Tabatabaie. ICPS-net : An End-to-End RGB-based Indoor Camera Positioning System using deep convolutional neural networks. 2019.

40. Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

41. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets:

Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

42. Qinglin Tian, Zoran Salcic, Kevin Wang, and Yun Pan. A hybrid indoor localization and navigation system with map matching for pedestrians using smartphones. *Sensors*, 15:30759–30783, 12 2015.

43. X. Du, X. Liao, Zhenzhen Gao, and Ye Fan. An enhanced particle filter algorithm with map information for indoor positioning system. *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

44. Federico Boniardi, Abhinav Valada, Rohit Mohan, Tim Caselitz, and Wolfram Burgard. Robot localization in floor plans using a room layout edge extraction network, 2019.

45. Varsha Hedau, Derek Hoiem, and David A. Forsyth. Recovering the spatial layout of cluttered rooms. In *ICCV*, pages 1849–1856. IEEE Computer Society, 2009.

46. Scott Nykl, Chad Mourning, Mitchell Leitch, David Chelberg, Teresa Franklin, and Chang Liu. An overview of the steamie educational game engine. In *2008 38th Annual Frontiers in Education Conference*, pages F3B–21. IEEE, 2008.

47. M. Labbé. Rtab-map as an open-source lidar and visual slam library for large-scale and long-term online operation. 2018.

Acronyms

Adagrad Adaptive Gradient Algorithm. 9

Adam Adaptive Moment Estimation. 9

AFIT Air Force Institute of Technology. 24, 25, 32, 51

AI Artificial Intelligence. 4

ANN Artifical Neural Networks. vi, 2, 4, 9, 12, 18

API Application Programming Interface. 27

BLE Bluetooth Low Energy. 1

BP Back Propagation. 10

CCE Categorical Cross-Entropy. 9, 35, 37

CNN Convolutional Neural Network. iv, x, 2, 3, 4, 12, 13, 15, 18, 19, 21, 27, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 54, 55, 1

Gb Gigabytes. 32

GIS Geographic Information System. 2

GNSS Global Navigation Satellite System. 1

GPS Global Positioning System. iv

GPU Graphics Processing Unit. 32

IMU inertial measurement unit. 54

LIDAR Light Detection and Ranging. 24

MSE Mean Squared Error. 8, 9, 29, 36

NHL Navigational Hyperspectral Learning. 32

PDR Pedestrain Dead Reckoning. 1, 18

RAM Random-access memory. 32

ReLU Rectified Linear Unit. 6, 28

RFID Radio-Frequency Identification. 18

RMSE root-mean-square error. viii, 29, 32, 36, 38

RMSprop Root Mean Square Propagation. 9

ROS Robot Operating System. 24

RTAB-Map Real-Time Appearance-Based Mapping. 24

SfM Structure from Motion. 18

SLAM Simultaneous Localization And Mapping. 1

SMC Sequential Monte Carlo. 16

tanh Hyperbolic Tangent. 6, 28

VO Visual Odometry. 1

WiFi Wireless Fidelity. 1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE

Indoor Navigation Using Convolutional Neural Networks and Floor Plans

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Ricky D. Anderson

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-006

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISA - Information Management Technologies Branch
26 Electronics Parkway Rome, NY 13441-4514
COMM 315-313-5302
Email: james.metzler@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RISA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The goal of this thesis is to evaluate a new indoor navigation technique by incorporating floor plans along with monocular camera images into a CNN as a potential means for identifying camera position. Building floor plans are widely available and provide potential information for localizing within the building. This work sets out to determine if a CNN can learn the architectural features of a floor plan and use that information to determine a location. In this work, a simulated indoor data set is created and used to train two CNNs. A classification CNN, which breaks up the floor plan into 100 discrete bins and achieved 76.1% top 5 accuracy on test data. Also, a regression CNN which achieved a distance error of 25.4 meters or less between the truth and predicted position on 80% of the test data. The models are further improved by combining them with a filter solution. The best performing classification CNN is evaluated on real world data captured via a TurtleBot 3, demonstrating the potential for this solution to be useful to real world Air Force indoor localization problems.

15. SUBJECT TERMS

Machine Learning, Convolutional Neural Networks, Indoor Localization, Particle Filtering

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Major Joseph A. Curro, AFIT/ENG
U	U	U	UU	80	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, ext 4620; joseph.curro@afit.edu