POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Computer Science and Engineering

A data-driven approach for early stopping
in autonomous robot exploration based on convolutional neural networks

Advisor: Prof. Francesco AMIGONI

Co-advisor: Dr. Matteo LUPERTO

Master thesis of:

LI Mingju Matr. 10574864/898045
LIN Chang Matr. 10597034/894651

Academic Year 2018 - 2019

Politecnico di Milano:
www.polimi.it

Scuola di Ingegneria Industriale e dell'Informazione:
www.ingindinf.polimi.it

Acknowledgements

This thesis is finished with the help of many people. First we would like to thank our project supervisor, Prof.Francesco Amigoni, for giving us this opportunity to work on this thesis project, and our project advisor, Dr. Matteo Luperto for his continuous guidance, support and motivation throughout the project.

We would also thank our families and our friends. They provided us a lot of support and encouragement throughout our academic career.

Sommario

In questo lavoro di tesi, consideriamo le attività di esplorazione eseguite da robot mobili autonomi in ambienti interni. Durante l'esplorazione, il robot opera una sequenza di decisioni su dove andare e su quando fermarsi. In questa tesi, presentiamo un approccio di deep learning che permette al robot se fermarsi. Il nostro modello è stato sviluppato per terminare l'esplorazione quando le informazioni raccolte nella mappa sono sufficienti per ricostruire una mappa accurata, invece di terminare l'esplorazione quando ogni area dell'ambiente è stata effettivamente osservata. Le attività sperimentali, condotte in diversi ambienti in simulazione, mostrano che il nostro approccio è in grado di terminare l'esplorazione in modo tempestivo e di risparmiare una notevole quantità di tempo.

Parole chiave: AlexNet, ROS Simulation, Exploration Early Stopping, CNN, Deep Learning

Abstract

We consider exploration tasks performed by an autonomous robot in indoor environments. During the exploration, the robot goes through a sequence of decisions on where to go and whether to stop. In this thesis, we present a deep learning approach that could decide whether to stop. Our deep learning model is developed to terminate the exploration when the map information is enough to reconstruct and output an accurate map, instead of terminating the exploration when every area of the environment has been actually observed. Experimental activities, performed in several different environments, show that our approach is able to terminate the exploration timely, and to save a significant amount of time.

Keywords: AlexNet, ROS Simulation, Exploration Early Stopping, CNN, Deep Learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Exploration by means of autonomous mobile robot is a task that incrementally builds maps of initially unknown, or partially known, environments [38]. At each stage of exploration, the robot decides whether to terminate the exploration process by using some criterion, and if not, decides the next location (often on a frontier between known and unknown space in the current map) to move to according to an exploration strategy [13]. When the selected location is reached by the robot, the map is updated according to the new knowledge of the environment and the process would re-start.

In this paper, we concentrate on the exploration of indoor environments. In previous work [11], the robot terminates the exploration when it has fully observed all the environment. We present a method to terminate the exploration process earlier, which could decide whether the current map is good enough to stop the exploration task because the unobserved parts can be reliably predicted.

To be more specific, we consider a mobile robot exploring an initially unknown indoor environment in order to build its 2D grid map. At each stage of the exploration, the layout of the known part of the environment could be extracted from the current partial map and the layout of the unknown part of the environment could be predicted by the method of [24]. The layout is an abstract geometrical representation of the rooms features and shapes [22]. The shape of partially observed rooms is predicted considering that different parts of the building share some common features. Different rooms could be related by the fact that they share the same shapes, and could be symmetric with each other.

The main original contribution of this thesis is that we came up with a method, which is data-driven, that could terminate the exploration process earlier with respect to cover all area until no frontiers left to explored. The stopping criterion in [11] could be regarded as a baseline stopping criterion. This baseline stopping criterion is relatively conservative in the terminating exploration. So, our early

stopping criterion is developed to terminate the exploration earlier, and output a map, with a cost of potentially losing some accuracy. Experimental activities show that our approach is able to halve the time spent in the exploration in many different environments. Though there are some environments in which our model could not terminate the exploration early, this problem could be fixed by adding more environments in the training set.

The thesis is organized as follows. In the next chapter we will discuss the state of the art in robot exploration. In Chapter 3, we will formulate the problem we address in the thesis. In Chapter 4, several solutions will be proposed to solve the problem formulated, and in Chapter 5, details of implementation will be given. Chapter 6 includes the online and offine tests. At last, in Chapter 7, the conclusion will be made and future works will be discussed.

Chapter 2

Start of the Art

In this chapter, we would like to introduce the background knowledge and state of art technology related to our research. We are presenting an overview of other approaches that have been proposed in the literature for our problem, plus a set of techniques and methods that are relevant to the task of solving the problem of autonomous exploration of indoor environments.

2.1 Exploration

2.1.1 Definition

Exploration is the process in which an autonomous robot incrementally explores an initially unknown area [38]. The robot uses multiple sensors, including laser range sensor, collision detection sensors, to discover information of the current exploring map and use these information to make the decision of the next position to move. We assume that the robot has no prior knowledge of the map and all the information relative to the environment should be extracted from current observed map.

There are two main families of approaches which are mainly used in robot exploration.

The first one is frontier-based approach. The frontier is defined as the boundary between known and unknown portion of the environment, as shown in Figure 2.1. In this approach, the robot selects the next target location, which is a frontier, from the group of boundaries between known and unknown portions of environments [45].

The second is information-based, which moves the robot to the most informative locations. We select frontier-based approach since it naturally address the discovery of space for the problem of map building we consider [34].

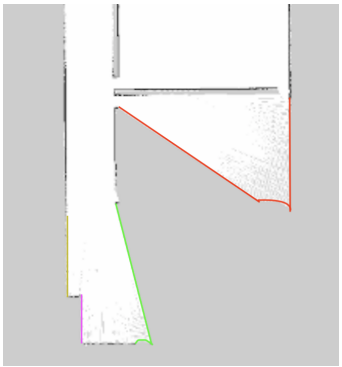


Figure 2.1: This is an example of frontiers in a partially observed map. The lines in different colors are all frontiers.

Typically, at each stage of the exploration process, a robot selects the next best location according to an exploration strategy [13]. There are multiple exploration strategies which are commonly adopted in robot exploration. Usually most of these strategies are greedy due to the inherent online nature of exploration problem [39]. In a greedy strategy, the robot always selects the best solution of the current stage. Strategies can be defined by a single criterion or by combined criteria, to guide the next action of the robot. For example, a robot can always select the closest frontier for the reason of cost reduction, or select the frontier which is widest among the set of frontiers, since the widest frontier has intuitively the highest value for exploration. A better approach is to consider multiple criteria together, instead of a single one. For instance, [13] chooses the next frontier by both considering the distance from the robot current position and the expected information gain. This is also the approach adopted by our method.

2.1.2 Exploration Procedure

In our work, an exploration process [11] can be defined as following. A single robot is equipped with a laser range scanner with given field of view and range, which is able to build the map M^E from the environment E. The map is not guaranteed to be the same as the environment, due to noise and error of the laser scanner. The robot starts to explore an initially unknown environment by frontier-based approach following the process below:

1. The robot observes a portion of E from its current position using its laser range scanner. Then it updates its partial observed map by integrating this portion.

2. The robot filters and generates the set of frontiers as the candidate set.

3. According to an exploration strategy, the robot selects the next best frontier candidate.

4. The robot moves to the selected frontier and start from the first step again.

The robot will repeat above steps until there is no frontier left. At this point, map M^E could represents all free space of E.

2.1.3 Information Gain And Distance

In order to apply a combined criteria for selecting the next best candidate, a common approach as [13] and [3] is to define an utility function. This function, given a frontier as input, outputs its utility value which represents how much it is worth if the frontier is selected as the best next one. The authors consider two criterias as a combined one. The first one is the distance utility, while the second one is the information gain of the frontier. The equation is shown below:

$$u(p) = \alpha * d(p) + (1 - \alpha) * i(p) \tag{2.1}$$

In the above equation, $d(p)$ is the distance utility which is calculated as:

$$d(p) = (D_{max} - D(p, p_r))/D_{max} \tag{2.2}$$

where $D(p, p_r)$ is the distance between the robot and the center of the frontier, and D_{max} represents the maximum value of distance of all frontiers in the candidate set. $i(p)$ is the information gain utility value, which is calculated as:

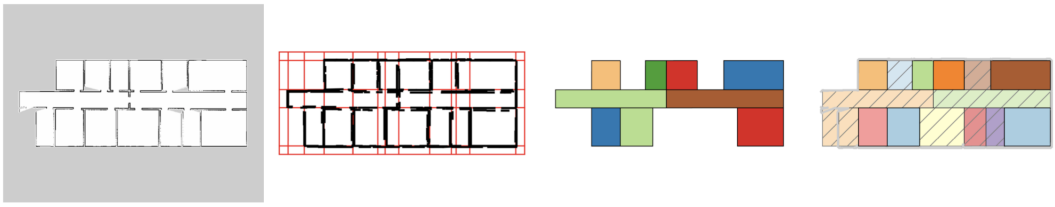
$$i(p) = I(p)/I_{max} \tag{2.3}$$

where $I(p)$ is the estimated unexplored area of the frontier, which can be obtained in the process of layout reconstruction (proposed in Section 2.2) and I_{max} is the maximum value of $I(p)$ for all frontiers in the candidate set.

To combine these two criteria, a weighted sum needs to be computed, to represent the best balance between closeness and expected new area. By setting α to 1, the strategy becomes a closest-frontier strategy. As well, by setting α to 0, the strategy focuses only on expected information gain. Values of α can be selected by doing multiple test experiments.

2.2 Layout Reconstruction

To reconstruct the layout L of the environment starting from its partial map M, we use the method presented in [23] and [24]. We provide here a brief summary



(a) Partial grid map. (b) Representative lines and faces. (c) Layout of fully observed rooms. (d) Reconstructed layout $\mathcal{L}$.

Figure 2.2: Here we show an example of the procedure of layout reconstruction [11].

of the algorithms using a running example as shown in Figure 2.2. Please refer to the original papers for full details.

The algorithm starts from a grid map M of the environment, like the one in Figure 2.2(a). A grid map is a map composed of a matrix of grids, in which each value represents the probability of the grid being occupied. From M a set of edges can be extracted, which are used to identify walls. Each wall is associated to a representative line by clustering. The representative line identifies the direction of the associated aligned walls. It means that all walls associated with the same representative line, even if they are in different portions of the environment, have the same direction. Representative lines can define a segmentation of the environment. Examples of representative lines can be seen, in red, in Figure 2.2(b). By this definition, after clustering all walls to a set of representative lines, a set of faces can be obtained by the intersection of representative lines (exactly four representative lines if the environment is rectilinear).

A face is the smallest unit representing potentially a room or part of a room, and a face could be one of the following three types:

- Fully observed, which means the area has been completely observed in M.
- Partial observed, which means the area has been partially observed in M.
- Unknown, if no point of the area has been observed in M.

After this step, we cluster all faces into clusters of faces, in which each represents a room. There are two different types of rooms:

- Fully observed room. This kind of room is composed of fully observed faces.
- Partially observed room. This kind of room is composed of both fully observed faces and partially observed faces.

Starting from fully observed rooms, in order to obtain a representation of the rooms, we follow the rules that fully observed room consists only of fully observed

faces and between any two faces there should not be any wall in M [26]. By traversing the whole set of fully observed faces and merging them, we are able to obtain a set of fully observed rooms. An example of fully observed rooms identified from the partial grid map of Figure 2.2(a) is in Figure 2.2(c).

Afterwards, the best combination of fully and partially observed faces could be obtained, to generate the set of partially observed rooms by means of representative lines. The idea is to utilize some common structures or symmetries in floor plan layout architecture. We assume that the unknown environments we explore all share some common regularities. For example, if one side of the room is bounded by a corridor, then the other side of the room has a high probability to share the same wall with the neighbor room along the same corridor.

Practically, we usually start from the partially observed faces F, which contains at least a frontier. We iteratively consider all adjacent faces, including fully observed, partially observed, and unknown faces). Starting from F, we will also consider all adjacent faces of those faces and so on, up to a maximum number of hops from F (2 hops in our experiments). Afterwards, we generate a set of combinations of faces (they must be connected), of which each one could be a potential room. For each combination of faces F', we try to calculate a utility value representing the probability of being an actual room. Here we introduce three criterias forming our objective function $\varphi(F \cup F')$.

The first criteria is an intuition that the partial observed room (which is the combination of faces $F \cup F'$) should have similar shape with respect to the rooms which are fully observed. To be more detailed, a combination would be better if an outer edge of the faces in this combination is longer and aligned to the representative lines. The second criteria follows the fact that commonly a room in a floor plan architecture usually has a simple shape (for example, rectangular) but not a complex shape (for example, concave polygon) and not a complex shape (for example, concave polygon). By means of this, we can compare the polygon formed by the combination of faces $F \cup F'$ with that of its convex hull. The third criteria prefers the shapes which are delimited by a smaller number of walls.

Finally, the set of faces $F \cup F'\{*\}$ that maximizes $F \cup F'$ is then associated to the partially observed room. Then in this case, the polygon representing the layout reconstruction of rooms is obtained by merging faces $F \cup F'\{*\}$. As a result, we obtain a layout $L = r_1, r_2, ..$ reconstructed which is completely composed of fully observed rooms and partially observed rooms.

2.3 Backward Coverage And Forward Coverage

After the process of reconstruction, a correct layout should have the following two characteristics:

- All rooms in the ground truth (the actual layout of the environment) should be in the layout.
- All rooms in the layout should also be in the ground truth.
- The shape of each reconstructed room should be the same as that of the corresponding room in the ground truth.

Following the approach of [23] and [4], we introduce two measurements to compare the reconstructed layout $\mathcal{L}$ and ground truth G_t visually and numerically.

We introduce two mapping function between rooms of $\mathcal{L}$ and G_t, which are forward accuracy and backward accuracy. Forward accuracy represents how well the reconstructed layout is described by ground truth, while backward accuracy represents how well ground truth is described by the reconstructed layout. The mapping relationship can be shown as:

$$FC : r \in \mathcal{L} \mapsto r^{'} \in G_t \quad BC : r^{'} \in G_t \mapsto r \in \mathcal{L} \tag{2.4}$$

For each room $r \in \mathcal{L}$, forward coverage finds the room $r^{'} \in G_t$ which has the maximum overlap area with r. As for backward coverage, for each room $r^{'} \in G_t$, it finds the room $r \in \mathcal{L}$ having maximum overlap area with $r^{'}$. These two mapping functions can be used to calculate two accuray measurements, which are called forward accuracy (A_{FC}) and backward accuracy (A_{BC}). These two values can be calculated by the following equations:

$$A_{FC} = \frac{\sum_{r \in \mathcal{L}} area(r \cap FC(r))}{\sum_{r \in \mathcal{L}} area(r)} A_{BC} = \frac{\sum_{r^{'} \in G_t} area(BC(r^{'}) \cap r^{'})}{\sum_{r^{'} \in G_t} area(r^{'})} \tag{2.5}$$

where $area()$ is the function for calculating the area of a polygon, and the overlap between room $r \in \mathcal{L}$ and room $r^{'} \in \mathcal{L}$ is defined as area $(r \cap r^{'})$.

This accuracy measurement is able to measure the similarity between two layouts, especially between the reconstructed layout from the partially observed map and ground truth, due to the fact that a room is the best unit to describe and construct a floor plan layout in our problem.

At the early stage of exploration, the partial map only covers part of the ground truth, with a relatively high A_{FC} and a relatively low A_{BC}. That's because for a room r (fully or partially observed) in the layout built from the partial map, we are always able to find the real room $r^{'}$ in ground truth which almost (due to noise

and error) completely covers the area of room r. And this room apparently has the maximum overlap area. On the contrary, for a room $r^{'}$ in ground truth which is still unexplored in the partial map, it's apparently impossible for find a overlap room r in the partial map. That's the reason why we will obtain a high A_{FC} and low A_{BC}.

When the partial map is almost fully explored, we will obtain a high A_{FC} and high A_{BC} if the method correctly reconstructs the layout from the partial map.

2.4 Convolutional Neural Network

Nowadays, with the devolpment of machine learning and deep learning, convolutional neural networks are commonly used in many domains, both scientifically and industrially. Image classification and object recognition are two of these domains, while autonomous exploration is considered in this thesis. In this section, we will first describe the basic theory of convolutional neural networks, then propose they can be used in some application domains, especially autonomous exploration of unknown environments.

2.4.1 Introduction

Convolutional neural networks (CNNs) are a type of feedforward neural network with convolutional computation and deep structure. It is one of the most representative algorithms of deep learning [17] [15]. Convolutional neural networks have the ability of representation learning, which means instead of learning predesigned features, CNNs could discover features and learn these features, and are able to shift-invariant classify the input information according to their hierarchical structure. Therefore, it is also called "Shift-Invariant Artificial".

The study of convolutional neural networks began in 1980s and 1990s. The time-delay network and LeNet-5 were the earliest convolutional neural networks [21]. After the 21st century, with the introduction of deep learning theory and the improvement of numerical computing device, convolutional neural network has been rapidly developed and applied to computer vision, natural language processing and many other fields.

The convolutional neural networks can perform both supervised learning and unsupervised learning. The kernel parameter sharing in hidden layers and the sparseness of the inter-layer connection guarantee that convolutional neural network is able to extract features from grid-like topology in a small computational cost. Therefore, convolutional neural network can learn from pixels or audio with a stable effect and have no additional requirements for feature engineering [17] [15].

2.4.2 Theory Support

Basic Structure

A convolutional neural network consists of multiple convolutional layers and one or more fully connected layers at the top (corresponding to a classical neural network), and also includes associated weights and pooling layers. This structure enables the convolutional neural to take advantage of the two-dimensional structure of the input data. By this feature, convolutional neural network gives better results on image and speech recognition than other deep learning structures. Figure 2.3 shows a standard convolutional neural network structure:

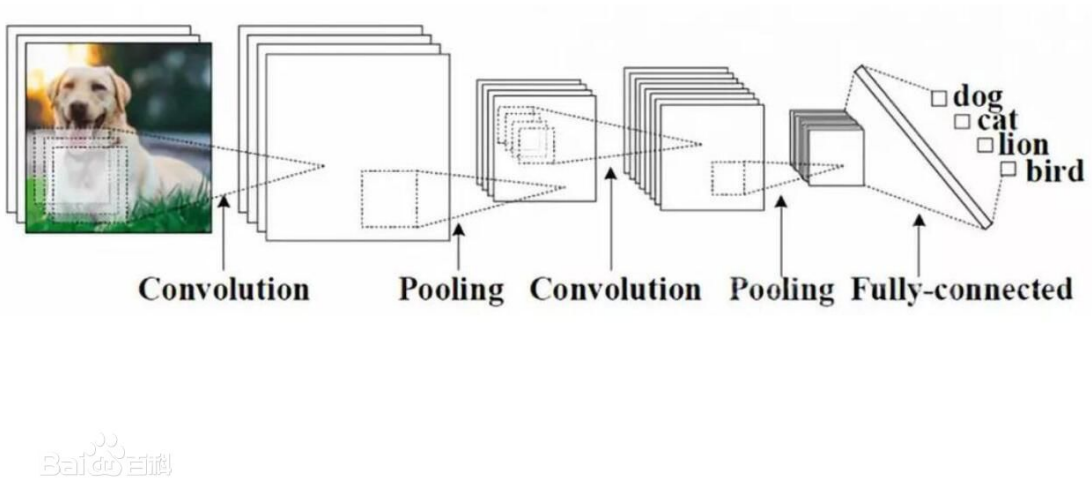


Figure 2.3: This is a very simple structure of a convolutional neural network [2].

Input Layer

The input layer of a convolutional neural network can process multi-dimensional data. Commonly, the input layer of a one-dimensional convolutional neural network receives a one or two-dimensional array, in which a one-dimensional array is usually sampled by time or spectrum, while a two-dimensional array may contain multiple channels. A two-dimensional convolutional neural network receives a two or three-dimensional array, and so on [29]. Since convolutional neural networks are widely used in the field of computer vision, many studies presuppose three-dimensional input data, which are the width, length and the color channels of the pictures.

Similar to other neural network algorithms, the input features of the convolutional neural network need to be normalized since we use gradient descent as the learning algorithm. Specifically, before we input the data into our convolutional neural network, we need to normalize the input data in the dimension of channel, time, or frequency. For example in computer vision, the original pixel values could be normalized to a specific interval. By normalization, we are able to improve the learning efficiency and performance of our convolutional network [29].

Convolutional Layer

The hidden layer of the convolutional neural network includes three common constructions, which are convolutional layer, pooling layer, and fully connected layer.

The function of the convolutional layer is to perform feature extraction on the input data, and contains multiple convolutional kernels. Each element in the convolutional kernel corresponds to a weight coefficient and a bias vector, similar to the neuron in feedforward neural network. Each neuron in the convolutional layer is connected to a plurality of neurons in a region of the previous layer. The size of the region depends on the size of the convolutional kernel and is called "receptive field" in the literature [15]. Its meaning can be analogous to the receptive field of visual cortical cells in animals. When the convolution kernel is working, it will regularly scan the input features, multiply the input features by matrix elements in the receptive field, and add the bias [17]:

$$
\begin{aligned}
&Z^{l+1}(i,j) = [Z^l \otimes w^l](i,j) + b = \sum_{k=1}^{K_l}\sum_{x=1}^{f}\sum_{y=1}^{f} Z_k^l(s_0 i + x, s_0 j + y) w_k^{l+1}(x,y)] + b \\
&(i,j) \in \{0,1,...,L_{l+1}\} \quad L_{l+1} = \frac{L_l + 2p - f}{s_0} + 1
\end{aligned}
\tag{2.6}
$$

The above equation describes how convolutional kernel works in each convolutional layer. The summation in the equation is equivalent to solving a cross-correlation. b is the bias. Z^l and Z^{l+1} represents the input and output of the $(l+1)$ convolutional layer, which is also known as the feature map. L_{l+1} is size of Z_{l+1} since here we assume that the width and the height of the feature map are the same. $Z(i,j)$ corresponds to each pixel in the feature map. K is the number of channels of the feature map. f, s_0, and p are the parameters of convolutional layer, corresponding to convolutional kernel size, convolution step (stride), and padding. These three values are hyperparameters of the convolutional neural network, together determining the size of the output feature map of convolutional layer [17].

Kernel size can be specified as any value smaller than the input image size. The larger the convolutional kernel, the more complex the input features extracted from the input image. The convolution step (stride) defines how many pixels the convolutional kernel shifts at each step. In other words, stride controls how the filter convolves around the input volume. For example, when stride is set to 1, the convolutional kernel sweeps the elements of the feature map one by one. If set to n, the convolutional kernel skips $(n-1)$ pixels and shifts to the nth one [9].

As it can be known from the cross-correlation calculation of the convolutional kernel, the size of feature map will gradually decrease with the stacking of the convolutional layers. For example, consider a 16×16 input image. After convolved by a 5×5 convolutional kernel with one stride at each step and no padding, a 12×12 feature map is output. Padding is a method of increasing the size of a feature map before is passes through the convolutional kernel to offset the effect of dimensional shrinkage in the calculation. A common padding is padding with zero and repetition padding by boundary value.

The convolutional layer also contains the activation function which helps to express complex feature, represented as following [17]:

$$A^l_{i,j,k} = f(Z^l_{i,j,k}) \tag{2.7}$$

where $Z^l_{i,j,k}$ is the output of each convolutional layer and $A^l_{i,j,k}$ represents the value after activation.

Like other deep learning algorithm, convolutional neural network usually uses Rectified Linear Unit (ReLU) as activation function. Some other variants like ReLU includes Leaky ReLU (LReLU), parametric ReLU (PReLU), randomized ReLU (RReLU), and so on [15]. Activation function is commonly used after the convolutional kernel, while some algorithms using preactivation techniques place the activation function before the convolution kernel.

Pooling Layer

After feature extraction in the convolutional layer, the output feature map is passed to the pooling layer for feature selection and information filtering. The pooling layer contains a predefined pooling function, which replaces the result of previous convolutional layers. The way pooling layer selects pooling area is similar to convolutional kernel, controlled by the pool size, step size (stride), and padding [17].

Fully Connected Layer

The fully connected layer in a convolutional neural network is equivalent to the hidden layer in a traditional feedforward neural network. The fully connected layer is usually build at the last part of the hidden layers in convolutional neural network, and only passes signals to other fully connected layers. The feature map looses the three-dimensional structure in the fully connected layer, and is expanded into a vector and passed to the next layer through the activation function.

In some convolutional neural networks, the function of fully connected layer can be partially replaced by global average pooling [35]. The global average pooling will average all the values of each channel in the feature map. For example, considering a $7 \times 7 \times 256$ feature map, the global average pooling will return a 256 vector, where each element is a 7×7 average pooling with stride equals to 7 and no padding.

Output Layer

The upstream of the output layer in the convolutional neural network is usually a fully connected layer, so its structure and working principle are the same as those in the traditional feedforward neural network. For image classification problems, the output layer uses a logic function or a normalized exponential function (softmax function) to output the classification label. In object detection problems, the output can be designed as the center coordinate, size, and classification of the output object. In semantic image segmentation, the output layer directly outputs the classification result for each pixel [29].

2.4.3 Applications

Image Classification

For a long time, convolutional neural network has been one of the core algorithms in the field of image recognition, and has a stable performance when learning a large amount of data [10]. For general large-scale image classification problems, convolutional neural network can be used to construct hierarchical classifiers [33]. It can also be used in fine-grained recognition to extract discriminant features of images for other classifiers to learn [40]. For the latter, features can be extracted artificially from different parts of the image [5], or by the convolutional neural network through unsupervised learning [18].

For text detection and text recognition, a convolutional neural network is used to determine whether an input image contains characters and to clip valid character segments from the image [46]. For example, the convolutional neural network

using multiple normalized exponential functions is used for the identification of the numbers in Google Street View Image [14]. Also in recent research, a convolutional neural network combined with a recurrent neural network (CNN-RNN) can extract character features and process sequence labelling [16].

Object Recognition

Convolutional neural network can be used for object recognition through three types of method : sliding window, selective search, and YOLO (You Only Look Once) [15]. The sliding window was used for gesture recognition [30]. But, due to the large amount of calculation, it has been eliminated by the latter two. Selective search corresponds to region-based convolutional neural network. It first determines whether a window may contain a target object through general steps, and further inputs into a complex identifier [12]. The YOLO algorithm defines object recognition as a regression problem for the probability of occurrence of each target in a segmentation box in the image, and uses the same convolutional neural network to output the probability, the center coordinates, and the size of the frame [32]. Object recognition based on convolutional neural networks has been applied to autonomous driving [25] and real-time traffic monitoring systems [20].

Autonomous Exploration

Convolutional neural networks can also used in the field of autonomous exploration of unknown environments. In recent research, [8] proposed the application of convolution neural networks in predicting human trajectory in environment. The authors use the model to predict average occupancy maps of walking humans even in environments where no human trajectory data are available. With such a predictive model, a robot can use this information to find nondisturbing waiting positions, avoid crowded areas, or clean heavily frequented areas more often. This approach transfers from simulation to real world data, and generalizes better to new maps than five baselines and surpasses the performance of the baseline models even when they are applied directly to the test set.

[6] proposed another application of convolutional neural networks in autonomous exploration of unknown environments by predicting future observations. The author trains a convolutional model using a database of building blueprints, which exploits the inherent structure of buildings. The model guides the robot to find the exit location of buildings. Comparing to traditional image processing apporaches of extracting features through histogram of gradients (HOG) and training a support vector machine (SVM), this method reduces the total exploration time to find the exit location by 36%.

Both these two methods take advantage of the ability of feature extraction and abstraction of convolutional neural network acquired. In domains of autonomous exploration, most of the input images are maps obtained by the robot in a specific environment. Comparing to image classification and object recognition, these data are simpler (usually with only one channel), and the structure of lines and corners is much more obvious, which means the convolutional model is able to extract enough features for prediction even without a large amount of training data.

2.5 Summary

In this chapter, we presented an introduction of the state of the art of exploration and deep learning in autonomous exploration. First of all, we briefly introduced how exploration problem is defined and the general solution we used for solving an exploration problem. Secondly, we detailed the methods we used for reconstructing the layout of the partial map. Afterwards, we described an intuitive method to measure the quality of the reconstructed layout of a partial map. Finally, we illustrated what a convolutional neural network is and how the model could be used in multiple application fields, including autonomous exploration.

Chapter 3

Problem Formulation

Chapter 2 introduces the state of art methods in robot exploration, while in this chapter we will focus on early stopping criterion in the exploration process, and fomulate the problem we address in the thesis. The robot with layout reconstruction information gain exploration strategy introduced above could works stably in many cases. However, in many cases, we want the robot to be fast in exploration.

Faster exploration speed could help us build a map in a shorter time. Sometimes, like in search, the time spent on building a map could be more important than some single details of the map. Even in some domestic tasks, we want our robot to be fast, knowing the fact that sometimes this means it maybe reckless in a way.

3.1 Motivations

In this section, we would like to introduce the motivation of what problem we found in this process of exploration with more details and statistics.

Though in the previous part of thesis, "early stopping criterion" / "early stopping criteria" have been mentioned for several times, it is necessary to provide a formal definition of this terminology. Generally speaking, the exploration process consists of exploration and *stopping condition*, which will give an answer for the question, "whether the agent should be stopped". Stopping criterion notifies the robot whether the map has been fully explored and it should terminate the exploration process. Method in [11] implements the criterion that "stops when no effective frontier exists" as a stoppoing criterion. *Early stopping criterion*, based on stopping criterion, is a method to stop before it has detected every area in the environment, when it could provide a correct reconstructed map, based on its current exploration progress.

From the thesis of [11], the robot stops exploration when it has already observed the whole environment space. Consider the fact that in our exploration process, the robot always selects the frontier with the highest utility value 2.1.3, and usually

with a relatively high information gain 2.1.3. Therefore, the robot is able to quickly increment the knowledge of its map. However, it will also leave small scattered frontiers across different rooms, as a consequence of their small information gain. Then in the final stage of exploration (usually when the total area explored reaches 80%), the robot needs to reach all the remaining small frontiers.

The time cost for reaching all those frontiers is particularly high. They are usually located in rooms that are far away from each other, representing small gaps like corners, with a low information gain. Therefore, we can avoid the exploration in these small frontiers if we are able to introduce a mechanism for the early stopping of exploration.

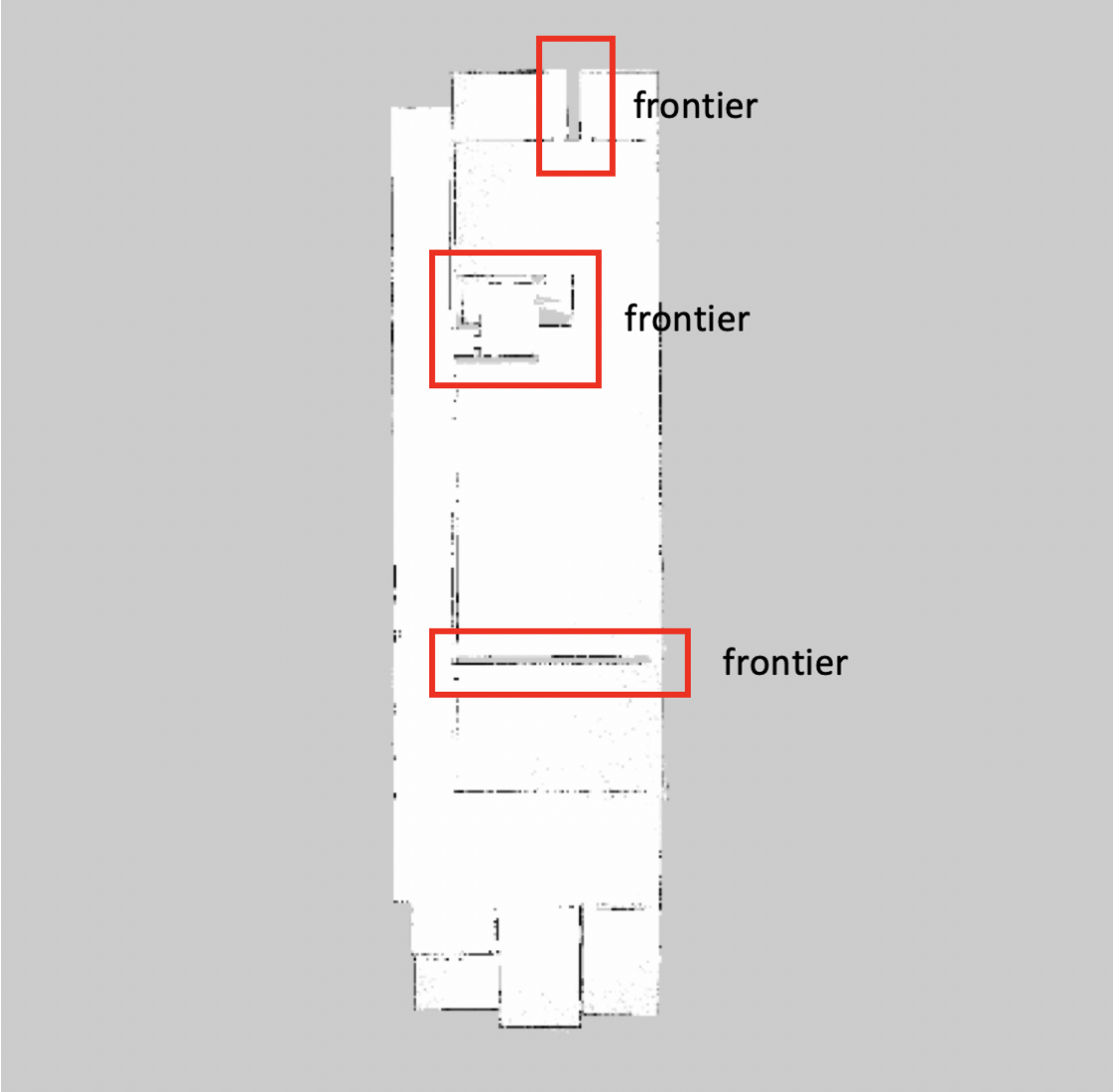


Figure 3.1: An example when the robot failed to stop timely.

Figure 3.1 shows the state of one partially explored map. It is not hard for us to conclude that this map is well explored, and we could stop and output a complete

map. Nevertheless, our robot, with its current stopping criterion, detects some tiny frontiers, and regards those frontiers as potential next destinations.

3.1.1 Current Early Stopping Criterion

In current exploration strategy, a baseline stopping criterion, that "stops when no frontier exists", has been implemented in the system. Figure 3.2 represents the workflow and relationships between different modules in our exploration.

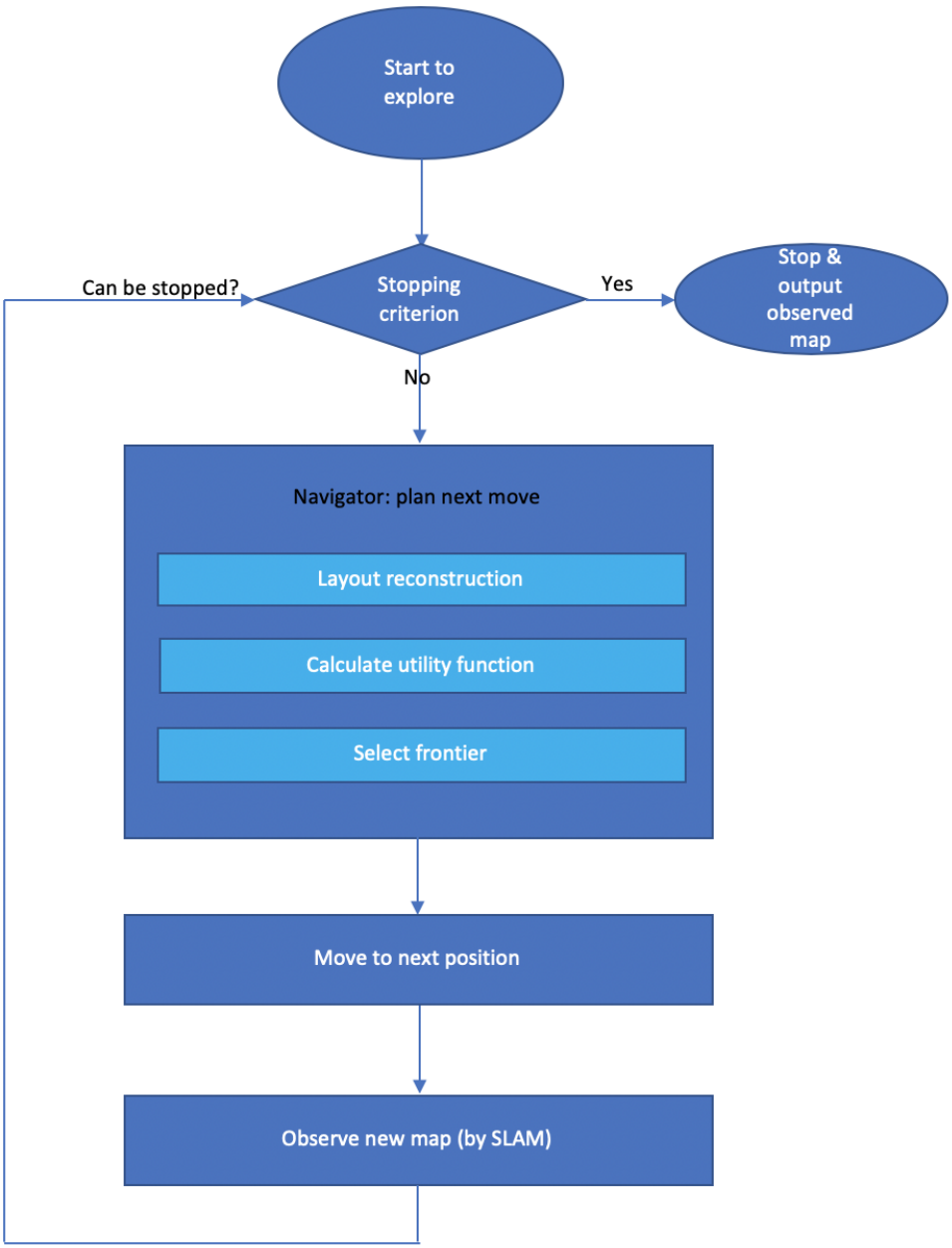


Figure 3.2: The workflow of robot exploration with a baseline stopping criterion.

Popular baseline stopping criteria include the following techniques.

* Stop when there is no frontiers in the map

  In every moment in the grid map, as shown in Figure 3.3, three kinds of pixels could be observed, free space, obstacle, unknown space. We call the set of continuous pixel between free space and unknown space "frontier". And here in this stopping criterion, we only stop when there is no frontier left in the diagram. First, we have to filter noise, which is represented by tiny frontiers. After filtering, we analysis whether there is any effective frontier left, which could lead the agent to a new room or space. If the answer is positive, keep on exploring, else, stop exploring.

  This is a popular and basic stopping criterion applied in todays applications due to the reason that it is easy to be implemented. It is robust, because actually how it works is that "it never gives up any oppotunities". However, this also lead to the problem that the exploration process always takes longer than needed.

* Stop when the update of the map is small

  If an effective exploration is being executed, the map should be updated continuously, and we expect our map covers increasingly more area. Hence we could use this property to stop the exploration. Assuming map area at time stamp t1 is Map^{t1}, and our map area at time stamp t2 is Map^{t2}. The area increase between two timestamps is

  $$\Delta Map = Map^{t1} - Map^{t2}$$

  We could set a threshold for this ΔMap. If the ΔMap is zero for a long time, it is reasonable to believe the map is well explored and we could stop.

  Of course, more details are needed to make this method work. The most significant thing is that we must assure that our navigator module works normally. And another thing we must take into consideration is that it would take a relative long time for a robot, moving from one position to another position, especially when the structure of the building is very complex.

  During the moving process showed in Figure 3.4, the robot may detect no map area increase for a relative long time, but we still could expect that some new area could be found by our robot from the target position.

Compared with current stopping criterion, a new stopping criterion, "stop when the prediction area is small" could be introduced. This stopping method estimates

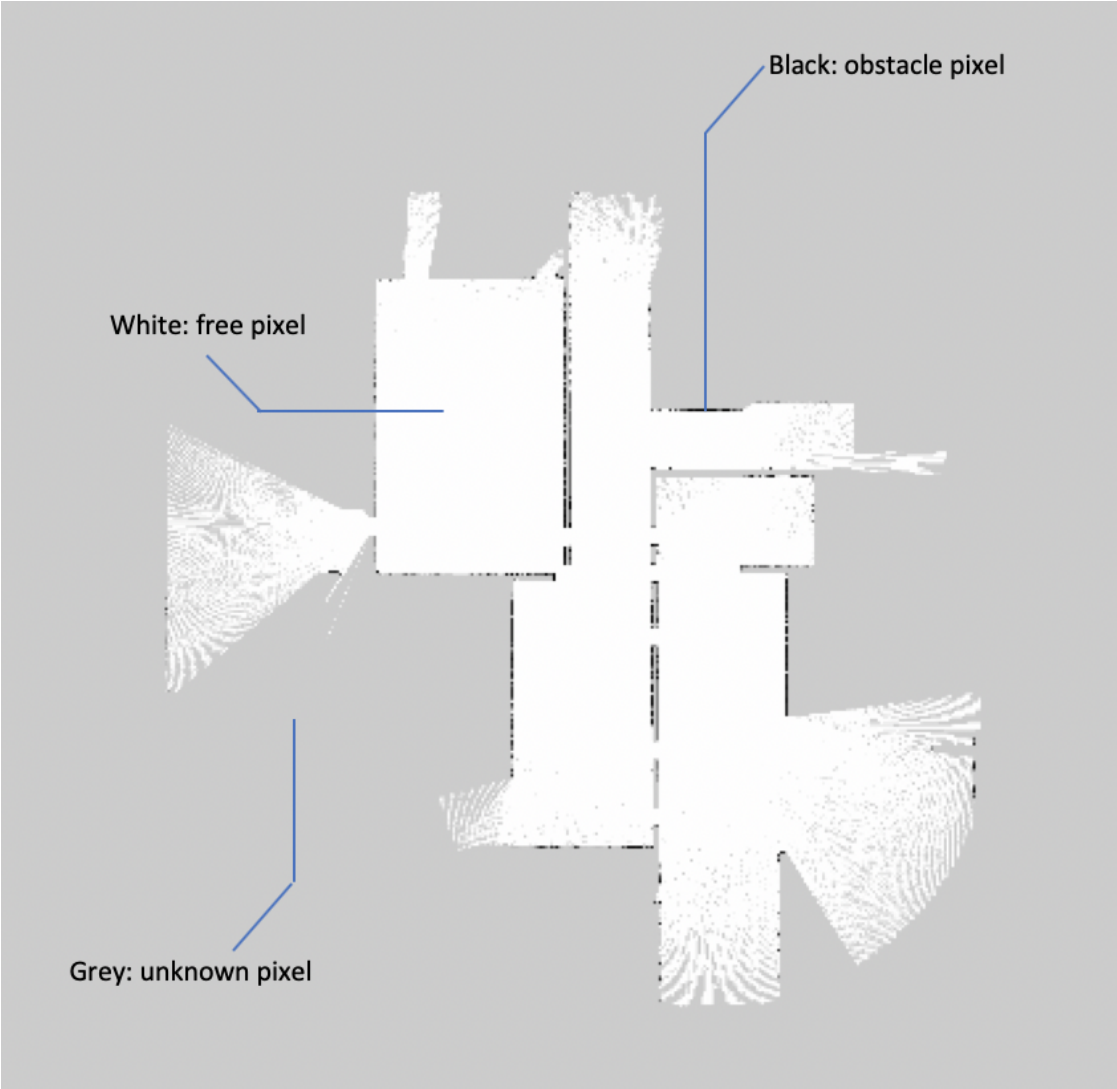


Figure 3.3: The observed map in relative early stage

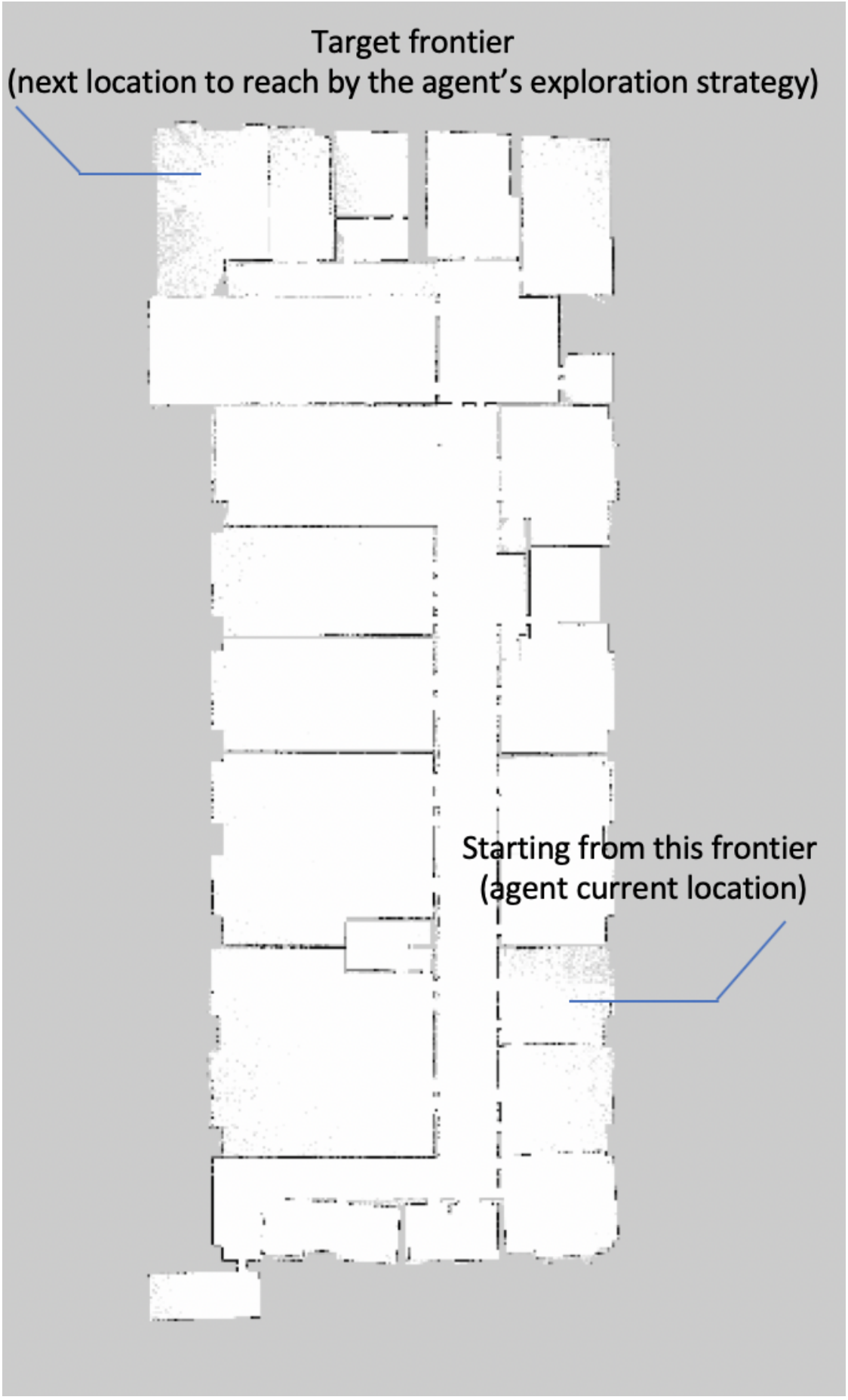


Figure 3.4: One scenario when the criterion "stop when the update of the map is small" does not fit

the amount of unexplored area, and stops exploration when it's lower than a predefined threshold. By using our layout reconstruction algorithm, we are able to predict the missing part of the partially observed rooms, and automatically fill these small gaps without actually explorating them. Therefore, we could introduce a criteria of early stopping based on the reconstructed layout. By the work of [24], if the unexplored area for all candidate frontiers could be predicted, and the total amount of such area is less than 1 m^2 or of another threshold, exploration would be terminated. We can assume that the robot can finish the exploration process and discard all remaining frontiers since we are able to predict the area beyond them from the corresponding reconstructed layout.

However, this mechanism has its limitations which are hard to avoid in practical situations. The first limitation is that it's hard to decide the threshold for early stopping, due to the fact that we have no prior knowledge about the unknown environment we explore. The environment could be large, like school or hotel. But it can also be just a combination of four rooms, being the house of a family. Therefore, no matter how we choose a static threshold, there will always be some cases which are out of the scope. The second limitation lies in the use of unexplored amount of area as a measurement. To decide whether the robot can stop exploration or not, using area as a single measurement is sometimes not enough. Therefore, as will be proposed in later chapters, we introduce a new data-driven method to predict the correct timestamp for early stopping, taking advantages of convolutional neural networks.

3.2 Current Time-area Diagram And Analysis

In this section, we perform some analysis and visualization on the data we obtained from using the stopping criterion in [11] (stop when no frontier exists). The information shown in Figure 3.5 is extracted from environment 41-1.

In Figure 3.5, we could see that the robot spends almost 50% time to cover the last 10% part of the map. Here we select the map at about 3600 seconds, and below are the diagram of observed map (at timestamp 3600 seconds) vs ground truth, and the diagram of predicted map vs ground truth.

On the left of Figure 3.6 is the observed map at 3600 seconds, on the right of Figure 3.6 is the ground truth. We could see that at this timestamp, we have already covered most of the areas in the ground truth. Though it is possible that in the right bottom corner of the map still exists a corridor, or a door to another room or area, we can reliably predict the shape of the map, and stop the exploration process with some degree of confidence.

On the left of Figure 3.7 is the prediction map at 3600 seconds, and on the

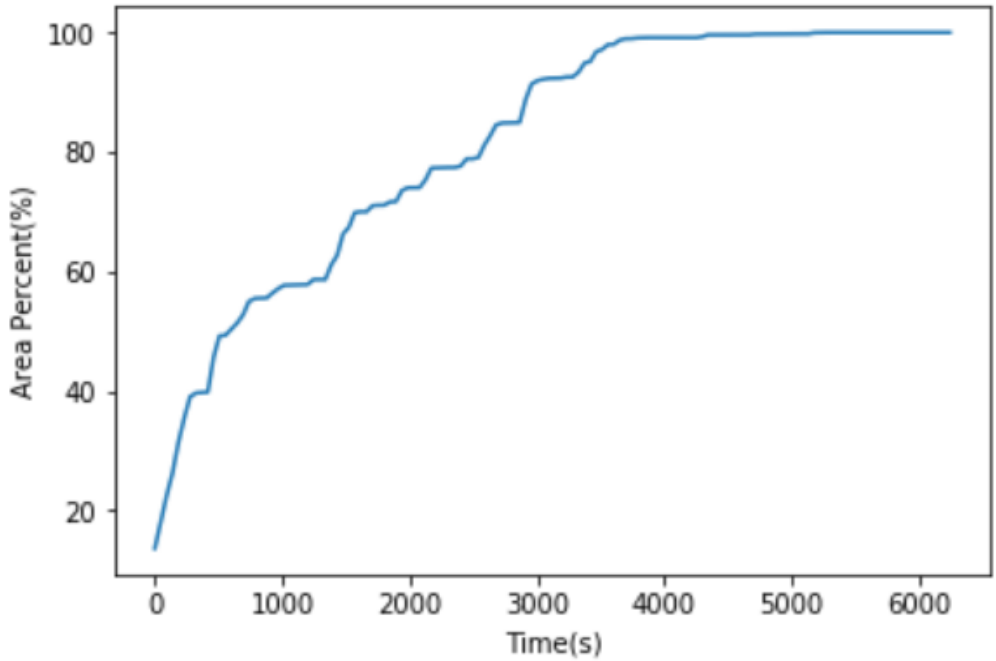


Figure 3.5: The relationship of the exploration time and map coverage percentage [11]

Figure 3.6: Observed map, compared with the ground truth map

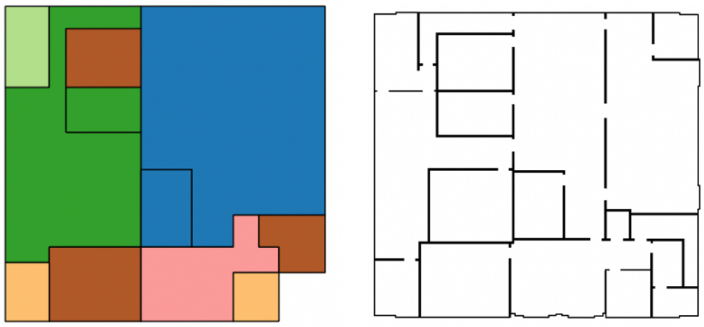


Figure 3.7: Reconstructed map, compared with the ground truth map

right of Figure 3.7 is the ground truth. In our layout reconstruct method, we use different colors to represent different rooms. Here it could be seen that there is only one small corner missed in this prediction.

In the balance of time efficiency and map accuracy, we have to choose which one is prefered. Sometimes we could tolerate that, just like above situation, one small corner is missed, but sometimes we could not allow that, and we aim for our robot not to miss anything.

Of course the predicted map is not a 100% equal to the ground truth. Besides the small room on the right bottom corner, the big blue room on the right side of the predicted map is not accurate. This is because these colored rooms are obtained by the clustering algorithms. By adopting different values of parameters, we could get different clustering results. So this could not be regarded as a false prediction. However, still we need some good methods to combine the predicted map with the observed map to output a final map as result of exploration. But this would not be discussed in this thesis.

Being able to stop early is an important ability for the robot. Since it could send a signal to the user or administrator that they could trust the current map. We will start from this idea to find a solution for early stopping in exploration of indoor environments.

3.3 Problem Formulation

From the analysis above, it is quite easy to find that in our current exploration strategy, the autonomous agent keeps on exploring the environment even when it

could do some prediction based on the layout reconstruction method and output an accurate map compared with the ground truth.

If we could come up with a method that could effectively tell whether the robot could stop and do a prediction, some time could be saved. This could be a very great improvements in applications of autonomous exploring agents.

So now the problem could be formulated as building a model, given the information of the map and exploration process, that could output a conclusion either "we could stop exploration, and output the reconstructed layout as the exploration result" or "further exploration is needed to generate an accurate map layout reconstruction" (just as the yellow modules in Figure 3.8).

To be more specific, as shown in Figure 3.9, the problem to be solved is how to predict "a given map is a well-observed map" is TRUE or FALSE. Here this "well-observed map" means a map that is almost completely explored, and for those areas we have not observed so far, our current layout reconstruction could provide a relative accurate prediction, and an accurate predicted map could be reconstructed and output.

The information we mentioned about the map and the exploration process includes but is not restricted to: the shape of the map, the scale of the map, the relationship between one single pixel and real area it represents in the environment, the exploration time. With this information, we could use some data driven methods to build a model to do a prediction.

3.3.1 Assumptions

First, we have to set some assumptions that could help us in our research. For most of the assumptions made here, they are not 100% accurate, but we could expect them to be correct in most cases. By doing so we could simplify our problem. These assumptions include:

1. The simulation with ROS[31] could simulate the real exploration process

   In this thesis, most of our analysis are based on the simulation result performed on ROS. Hence we should trust this ROS simulation process, the simulation process showed in our ROS system could represent the robot action in the live environments. This includes the computing ability of the agent in live environment should be better than what we simulate in ROS. And the execution of the planning trajectory, layout reconstruction, and SLAM in the real environment should be executed just like how it is done in ROS.

2. The sensor information simulated is same with live environment

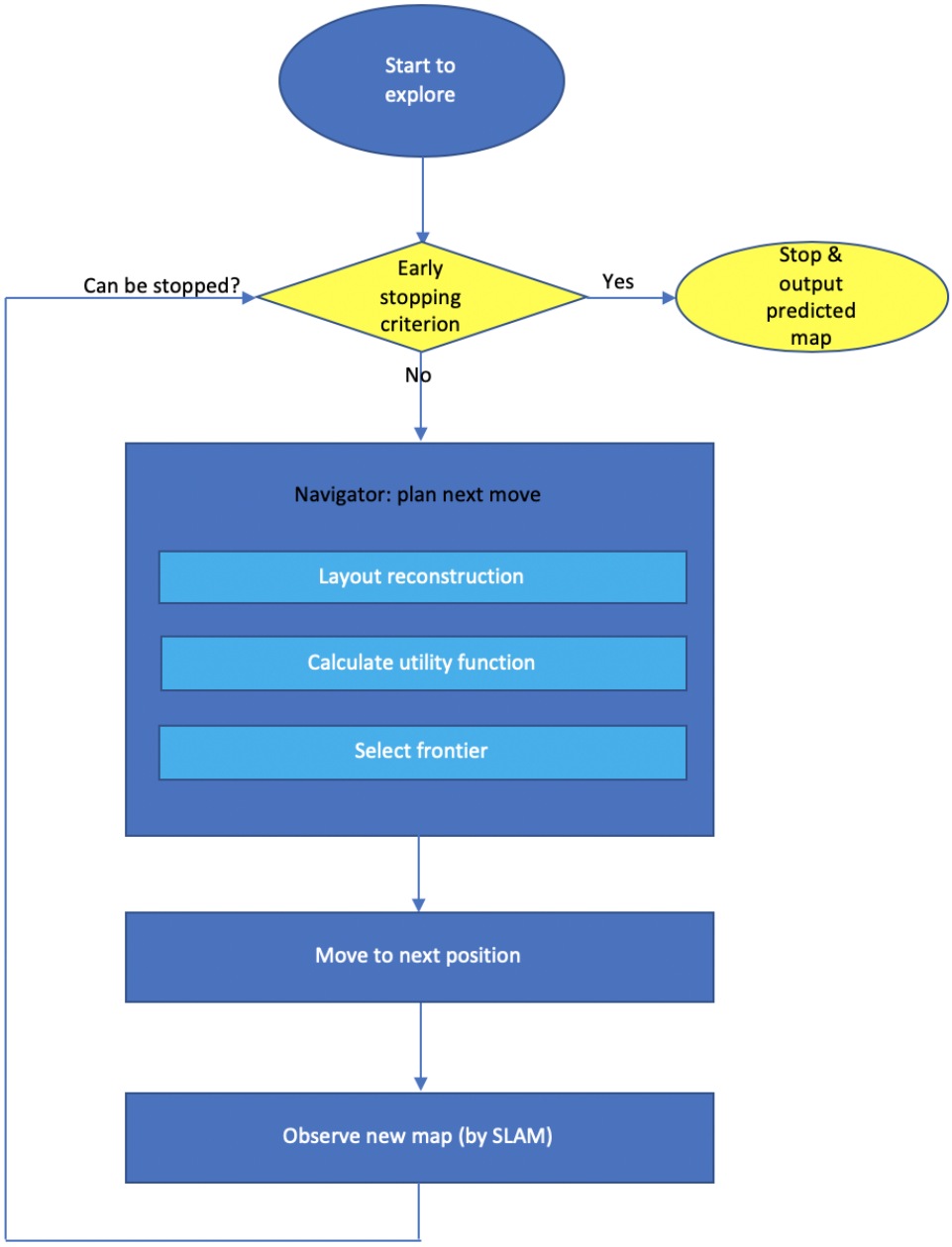


Figure 3.8: The updated workflow

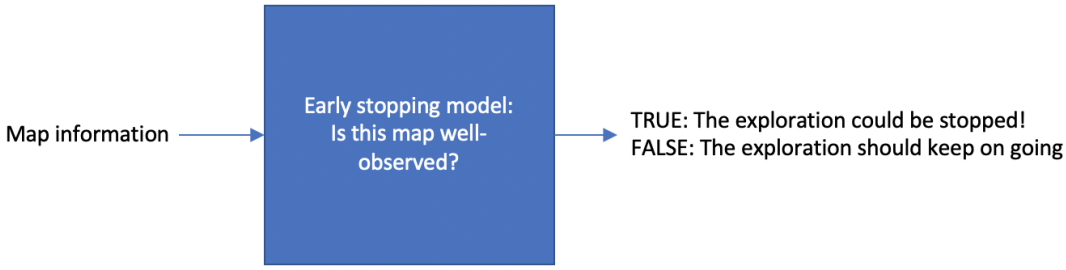


Figure 3.9: The simplified model or our target early stopping module

For simulation, we choose a 180 degree laser-sensor in ROS. However there are several cases we could expect during collecting sensor information. The first one is that solid obstacles could be detected in the sensor. However, in another situation, the sensor may detected nothing in its reception range. And in this cases, it would have a shape like a fan-shape area during simulation, and we assume this shape is same as the real environments.

3. We always consider indoor environments

   In our research, we only considerindoor environments. Any other environment like forests and underground caves are not taken into consideration, because these natural architecture are very different from the artificial building. All of our training and testing data (100 environments) are obtained from the indoor buildings datasets.

4. The floor is clean

   We always assume that the environment has a clean floor, or the robot could regard those obstacles as noise when dealing with them. For all the environments we are dealing with, they have the same elevation.

We could proposed several solution for this formulated problem in Table 3.1. The factors that could potentially help us decide the status of the map are listed with different granularities. Explanation is attached for each factors.

We divide all these factors into three granularities:

Granualrity	Factor Name	Explanation
Global	Time	Is there a time boundary that we could say a map is well-observed (after the exploration taking t seconds, we could conclude a map well-observed).
Map	Map Shape	Is there any image analysis technology we could apply on the map (given the map shape, do the prediction).
Map	Map Area	Is there any pattern on the area of the map (when the area of the map is greater than m m^2, we could say that map is well observed).
Frontier	Frontier Number	Is there any rule on the number of frontiers (when the number of frontiers is less than n, we could conclude a map is well observed).
Frontier	Frontier Size	Is there are any rule of the frontier size (if the frontier is smaller than a threshold fs, we could say that no further exploration is needed on this map).
Frontier	Frontier Shape	Is there are any image analysis technology we could apply on the frontier shape (given a specific frontier shape, we could conclude this frontier no longer needs any further exploration).

Table 3.1: Analysis on factor exploration time, exploration area.

* Global: It means these factors are considered globally during the exploration process (exploration time, ...).

* Map: It means these factors are directly related to the maps (map area, map complexity, map shape, ...).

* Frontier: It means these factors are directly related to each frontier in the maps (frontier size, frontier numbers, frontier shape, ...).

Here we do some further analysis on these factors in Table 3.2, Table 3.3, Table 3.4 and Table 3.5.

Exploration Time, Map Area Covered	
Pros	Easy to measure.
Cons	For different maps and robots, even different exploration speeds, the time spent on a single map could be really different, it could hardly represent the progress of our current exploration. And for the map area it is the same as well.
Conclusion	We could use time and map area as a complement criterion in the early stopping criterion. However, no direct relationship between these factors and exploration status is expected in real situation.

Table 3.2: Analysis on factor exploration time, exploration area.

Frontier Number, Frontier Size	
Pros	Easy to do statistics and find the rules.
Cons	The number of frontiers is not a very representative factor for the map, because in real situations (in most situations), there are not too many frontiers, but maybe there is just one frontier that leads to another new area and rooms.
Conclusion	Could not be chosen as one of the main factors to decide.

Table 3.3: Analysis on factor of current frontier numbers, frontier sizes

To summarize above information, we could reach this conclusion. The early stopping criterion is closely related to many factors and information collected during the exploration process. However, from a practical point of view, the best direction to start from is to start from map shape analysis with data driven techniques.

We would like to further clarify our goal and solution with details in the next section.

Frontier Shape	
Pros	Enough data could be obtained, since for each map we could sample about three or four frontiers.
Cons	The data is enough, but we have to decide what area should be cropped out, because for the frontier what is important is not their shape, but the shape of obstacles (those solid boundaries obtained by sensors) around them. And another problem is that it is not easy to label these frontiers, for three or four frontiers we found in the map, only one of them could be the reason why the agent still needs to explore. So the labelling is not easy to be implemented, we need some exquisite method to do this labelling.
Conclusion	This is a promising method we could explore.

Table 3.4: Analysis on factor of frontier shape

Map Shape	
Pros	The most direct and global information for the map status, could be very effectively labelled and checked.
Cons	We hold just 100 environments, even if we have about 30 to 40 runs for each environments, 100 is a small number when it comes to learn general rules. Data enhancement methods should be applied. And new environments should be added to our current training set.
Conclusion	This is the best direction we should go along.

Table 3.5: Analysis on factor of observed map shapes

3.4 Goal

With above analysis, our goal is to build a model to classify the statement "current map is well-observed" is true or false. The input of the model is the map shape and the output is a binary classification of true or false.

We could abstract this problem into a binary classification task on images. And for these images, we know there are some special features, compared with common images:

* Single channel Our images are grayscale single channel images, could be represented by a 2D arrays.

* Simple values For the elements appearing in the 2D array, there are only 3 unique values, meaning free pixel, unknown pixel, obstacle pixel.

It would be a good idea to keep these properties even during processing input images, because these simple properties could help us to be more robust against the noise and information loss happening in data processing.

And besides the map shape, other information listed above could be added into the system as a complement for the model, as shown in Figure 3.10. Since they could be related to the map complexity and map scale, they are all good measurements for this kinds of abstract values.

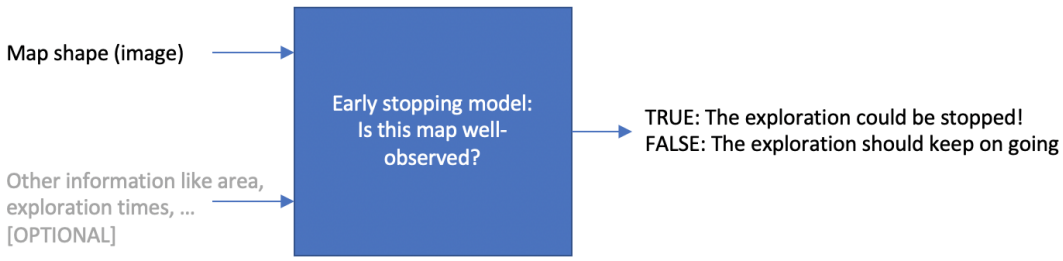


Figure 3.10: The goal of our current module to implement.

Now to simplify the model we are dealing with, we just consider the model with one input, that is the map shape. While still it is possible to attach other information on this image input (input the model with a map shape of .png/.pgm format and another file recording all the context for the map with .xml/.yml formats). Another idea is that we could try to re-encode our map file with the

context information, like setting the color to be lighter if the map is big, and setting the color to be darker if the map is small. Last but not least, we could integrate these context value in our fully connected layer if we are dealing a convolutional neural network + fully connected layers deep learning models. Figure 3.11 is a common structure of a normal CNN+FCL model.

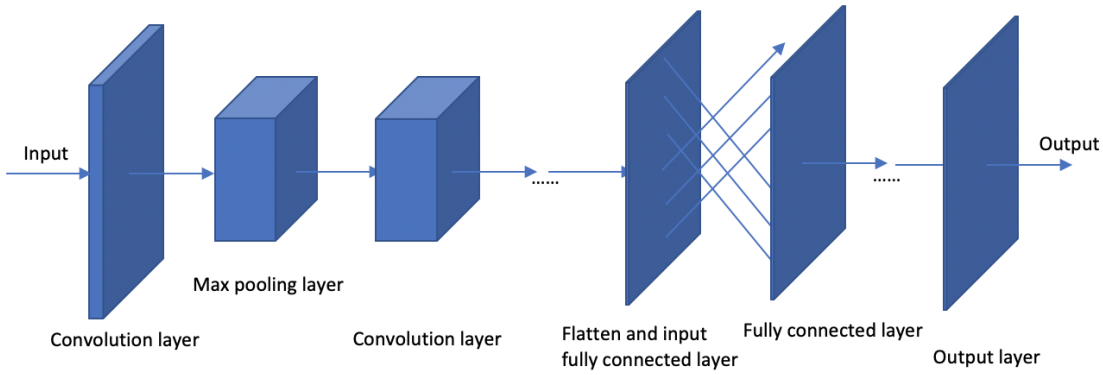


Figure 3.11: The structure of a common CNN network.

And we could integrate those complement information in the flatten input of fully connected layer, because before the fully connected layer, the result of the convolutional layer must by flattened to one array. Hence these complement information could be of help just as shown in Figure 3.12.

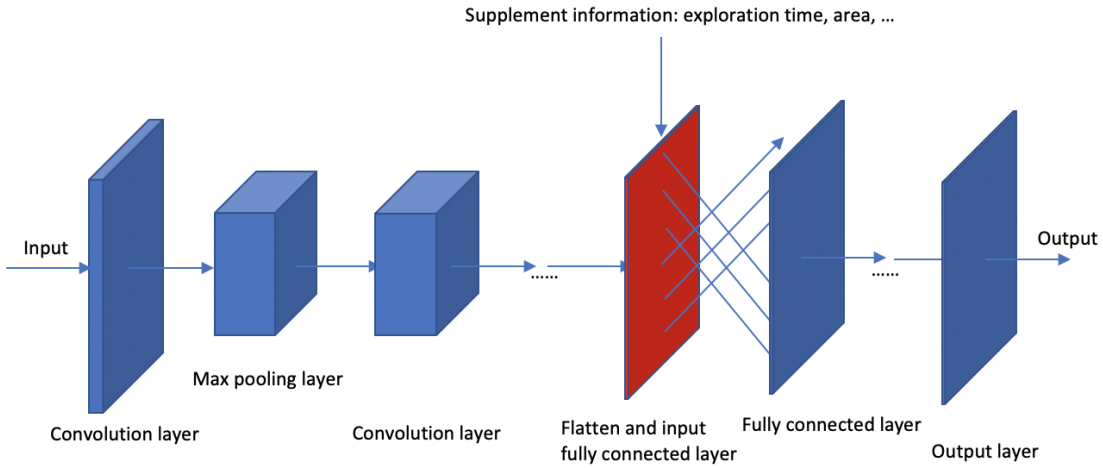


Figure 3.12: The structure of a CNN network with complemented information.

The use of this complement network given as a future work could potentially

improve the model performance based on our current model performance, but this will not be discussed in our thesis.

3.4.1 Summary

In this section we would summarize all the points mentioned in this chapter.

The problem we are dealing with is that we want to add an effective early stopping decision system into our current autonomous exploration agent. Hopefully it will inherit the advantages of the "stop when no frontier exists", which means it should be robust, and would not give false prediction when the map still needs exploration. But, at the same time, it could stop earlier.

This new model could decrease the time spent on those areas for which we could easily predict their shape, and speed up the whole exploration process.

The best way to realize such goal is to apply data-driven techniques, building a model, which could classify input map to be “well observed” or not.

Chapter 4

Proposed Solution

To reach the goal mentioned above, we have several ideas to realize the model. In this chapter, we will discuss all the possible solutions we offered, their advantages, and their disadvantages. And in the later part of this chapter, we will pick out one of the most practical methods (CNN), and gives more details about why we could use it to solve this problem, and how we could do that.

4.1 General Overview

To build a model which could solve the above problem, we have these ideas.

* Static and manually designed rules in Table 4.1

Static and Manually Designed Rules	
How	We could go through all the map information we have trying to summarize the features, and design a set of rules. To do so, we could follow this procedure * Use image analysis method to capture special features in the map (corners/frontiers/open areas). * Do statistics on these features and the maps to find the relationships between them. * Design a set of rules (if no fan shape appears in the image, assert true/elif the number of corners>n, assert false / ...).
Pros	* Readable, reasonable and understandable rules. * Could be modified and edited easily. * Easy programming, and implemented with ROS system.
Cons	* Not easy to be designed, quite hard to find the potential rules. * Hard to be implemented for the rooms with different scales. * Hard to be implemented for the rooms with peculiar shapes.
Conclusion	NOT suitable

Table 4.1: Analysis on static rules model

* Map Encoding techiniques in Table 4.2

Map Encoding Techiniques	
How	The core idea of this method is to abstract the structure of the map, and then learn its structure. It could be done using following procedure. * Abstract the map into an undirected graph. For each room, corridor, extract them into a node in the graph, for each door, connection, extract them into a undirected edge. * (Manually) label these graphs. * Learn and generate the possible patterns and rules for these graphs.
Pros	* Easy to visualize. * Very light-weight model.
Cons	* Undirected graphs are hard to compare (could lead to a NP problem). * No mature models to learn these kinds of structures and tasks.
Conclusion	NOT suitable

Table 4.2: Analysis on map encoding model

* Deep learning models with image analysis method in Table 4.3

Deep Learning Techiniques	
How	We could build deep learning models with Convolutional Neural Networks + Fully Connected Layers structures. To do so, we have to follow this procedure. * Data extration and preprocessing. * (Manually) label training data. * Define loss function. * Define CNN structure. * Feed the data to our model and training. * Parameter fine tuning. * Validation. * Offline test. * Online test.
Pros	* Mature method, a lot of previous research results and sucessful industrial applications could be found. * Good performance in image (binary) classification task. * Easy to be implemented and integrated. * Fit well in peculiar shape buildings. * Does not influensed by the building scale. * Robust to noise.
Cons	* Training takes time. * Limited data (we only have 100 environments), and the result may be not representative enough.
Conclusion	Suitable

Table 4.3: Analysis on deep learning (CNN) model

4.2 AlexNet

In chapter 2, we have already introduced convolutional neural networks, and their applications in real world. In the area of advanced image classification, they are the best performance tools we have so far.

Nevertheless, convolutional neural networks are a great class of deep learning models. In the family of convolutional neural networks, GoogleNet, PReLUNet, Vggs and SqueezeNet are some of the most advanced CNN structures with the core idea of convolutional layer to extract features, and fully connected layer to execute the classifications. They have reach very high accuracy in the ImageNet image classification task. To achieve our goal, we select AlexNet to carry out this task.

4.2.1 AlexNet Defination, And History

AlexNet [19] is known as one of the most mature, famous, and influential convolutional networks. We could almost say that the victory of AlexNet on ImageNet in 2012 started the era of deep learning. Later advanced CNNs also inherit some parts of AlexNet characteristics.

The most well-known AlexNet has following parameters in Table 4.4:

Number of convolutional layers	5
Number of fully connected layers	3
Depth	8
Number of parameters	60M
Number of neurons	650k
Number of classification	1000
Batch normalization	None

Table 4.4: Parameters of classical AlexNet structure

The general structrues of AlexNet could be represented as Figure 4.1.

As a epoch-making deep learning structure in 2012, AlexNet has these characteristics which helped it winning the ImageNet in 2012.

1. Data Augmentation

   Horizontal flipping, random cropping, object shifting, color transformation, lighting transformation, contrast transformation, etc. These effectively added more data to the training set and could keep the model away from overfitting.

2. Dropout

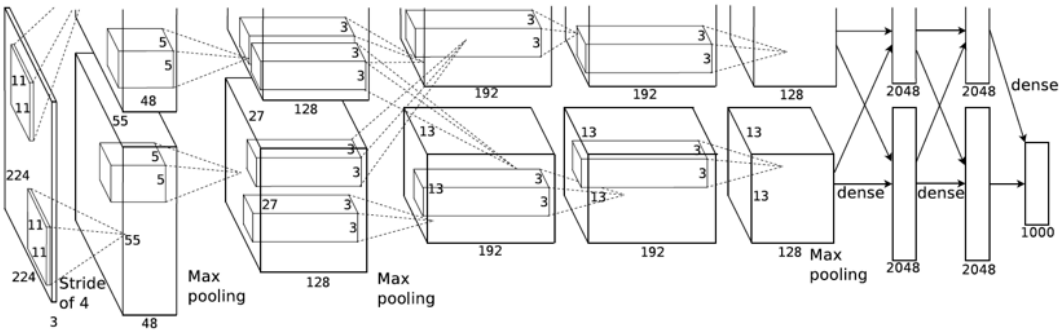


Figure 4.1: The current structure of AlexNet [19]

In all kinds of deep learning methods, the overfitting is an inevitable problem we are going to face in the model training. Take AlexNet as an example, we have 60M of parameters. And for the training set, our data is relatively limited compared with such a huge system. So it is very important to solve this problem.

Dropout is one of the method applied by AlexNet. The core idea of dropout could be summarized in Figure 4.2: when we are trying to perform forward propagation, we make a neuron stop working with probability p. This could significantly enhance the generalization ability of our model, because it means that it will not strongly depend on some local characteristics.

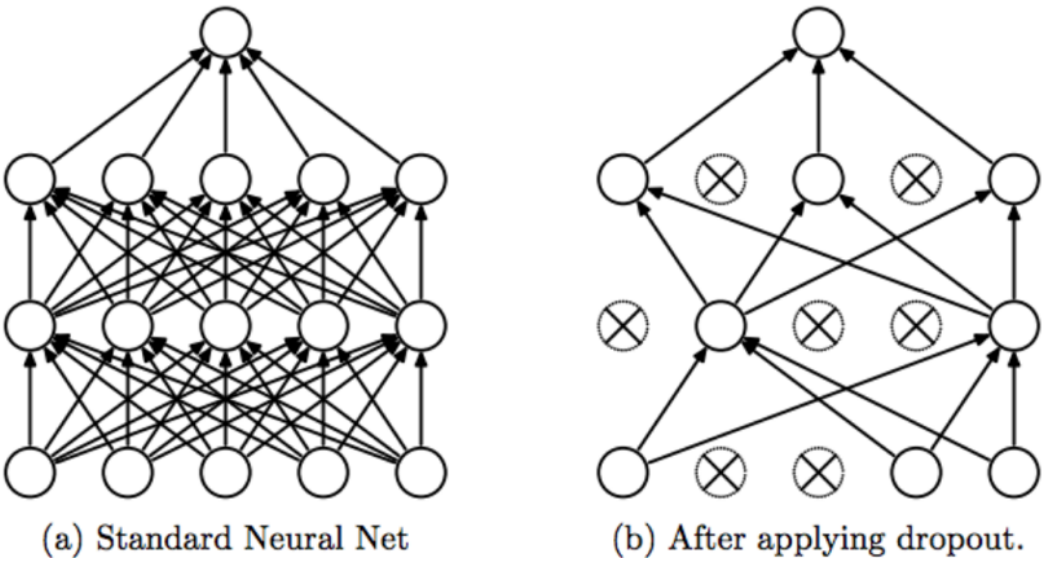


Figure 4.2: The structure of a CNN with/witout dropout [42]

3. ReLu Activation Function

   Activation function is designed to keep the non-linear properties of the network [27]. Before AlexNet, tanh activation function and sigmoid activation

function were the most widely accepted activation functions. In AlexNet, authors applied ReLU function[1] to replace T/S activation function. For the traditional T/S activation function, two problems we have to face is that gradient vanishing and the high cost of exponential computation. However, both problems could be solved by ReLU function. It not only solves the problem of gradient vanishing in the positive interval, but also brings a faster computing speed to the training process.

4. Local Response Normalization

   In AlexNet, one controversial method it applied is local response normalization. This could be regarded as simulation of the suppressing function of biological neural networks. With this method, the response with higher value could suppress the local response with lower value. This method is also regarded as a step to be taken to strengthen the generalization of the neural network.

5. Overlapping Pooling

   Instead of the simple pooling in which each pooling window has no intersection with others, AlexNet applied overlapping pooling. Different pooling windows have some parts overlapped. This could be represented mathematically by strides $<$ windowSize. By this method (and with every other setting, parameters and network structures unmodified), the performance on the test data have improved by 0.4% and 0.3% [19].

6. Multi-GPU Cooperate together to train

   It is quite common that the power of a single GPU is not enough to support the training of a big model with a satisfying speed. So AlexNet is trained by introducing several GPUs together, and then integrating the training result together to get the final model. This kind of distributed computation is very valuable. Though now our personal computers have become more powerful, and in this thesis, we use centralized computation, we still need to mention this idea and know the importance of it.

So in our research, to solve the problem formulated in chapter 3, we would apply a AlexNet for this task. In the research, we will develop the AlexNet from its classical structure, inheriting the characteristics mentioned above, and fine tuning it so that it could fit this specific situation.

4.2.2 Basic Configuration Of AlexNet

In this section, we are going to start from the configuration showed in Table 4.5 and Table 4.6 to build our AlexNet model

NAME	EXPLANATION
Input layer	
Input	227*227*1 single channel cropped and enhanced observed map
Convolutional layer 1	
Number of Convolutional Kernel in Layer 1	96
Stride of Convolutional Kernel in Layer 1	4*4
Size of Convolutional Kernel in Layer 1	11*11
Output of Convolutional Layer 1	55*55*96
Pooling layer 1	
Stride of Pooling Layer 1	3*3
Size of Pooling Layer 1	2*2
Output of Pooling Layer 1	27*27*96
Convolutional layer 2	
Number of Convolutional Kernel in Layer 2	256
Stride of Convolutional Kernel in Layer 2	1*1
Size of Convolutional Kernel in Layer 2	5*5
Output of Convolutional Layer 2	27*27*256
Pooling layer 2	
Stride of Pooling Layer 2	2*2
Size of Pooling Layer 2	3*3
Output of Pooling Layer 2	13*13*256
Convolutional layer 3	
Number of Convolutional Kernel in Layer 3	384
Stride of Convolutional Kernel in Layer 3	1*1
Size of Convolutional Kernel in Layer 3	3*3
Output of Convolutional Layer 3	13*13*384
Convolutional layer 4	
Number of Convolutional Kernel in Layer 4	384
Stride of Convolutional Kernel in Layer 4	1*1
Size of Convolutional Kernel in Layer 4	3*3
Output of Convolutional Layer 4	13*13*384
Convolutional layer 5	
Number of Convolutional Kernel in Layer 5	256
Stride of Convolutional Kernel in Layer 5	1*1
Size of Convolutional Kernel in Layer 5	3*3
Output of Convolutional Layer 5	13*13*256
Pooling layer 3	
Stride of Pooling Layer 3	2*2
Size of Pooling Layer 3	3*3
Output of Pooling Layer 3	6*6*256

Table 4.5: Parameter and details of our AlexNet (Part 1)

Flatten layer	
Output of Flatten Layer	9216 (=6*6*256)
Fully connected layer 1	
Number of Neurons on fully connected Layer 1	4096
Dropout layer 1	
Dropout Probability on Dropout Layer 1	0.5
Fully connected layer 2	
Number of Neurons on fully connected Layer 2	4096
Dropout layer 2	
Dropout Probability on Dropout Layer 2	0.5
Fully connected layer 3 (Output layer)	
Output classes	2

Table 4.6: Parameter and details of our AlexNet (Part 2)

And those above parameter could be summarized into this structure of our AlexNet in Figure 4.3.

4.2.3 Differences Between AlexNet Tasks

One of the most important things we have to take into consideration is that we have to take into account our particularity in this task. So we compare our task/goal with those tasks that CNN has proved its capability in previous work. These differences between our task and previous image classification task could be found in Table 4.7.

Our task	Previous task [19]
Single channel grey scale picture	3 Channel RGB colorful picture
Original size 4000*4000	Original size about 1200*900
Input 227*227*1	Input 227*227*3
Output classification categories 2	Output classification categories1000
Training data about 2800 samples from 2 categories	Training data about 150000 samples from 1000 categories

Table 4.7: Differences between our binary classfication task and previous general image classification task of AlexNet

It is quite obvious that our task is far more simpler than the previous image classification tasks. However, it is our very limited training set that is the most tricky problem that we should solve. From AlexNet very outstanding performance

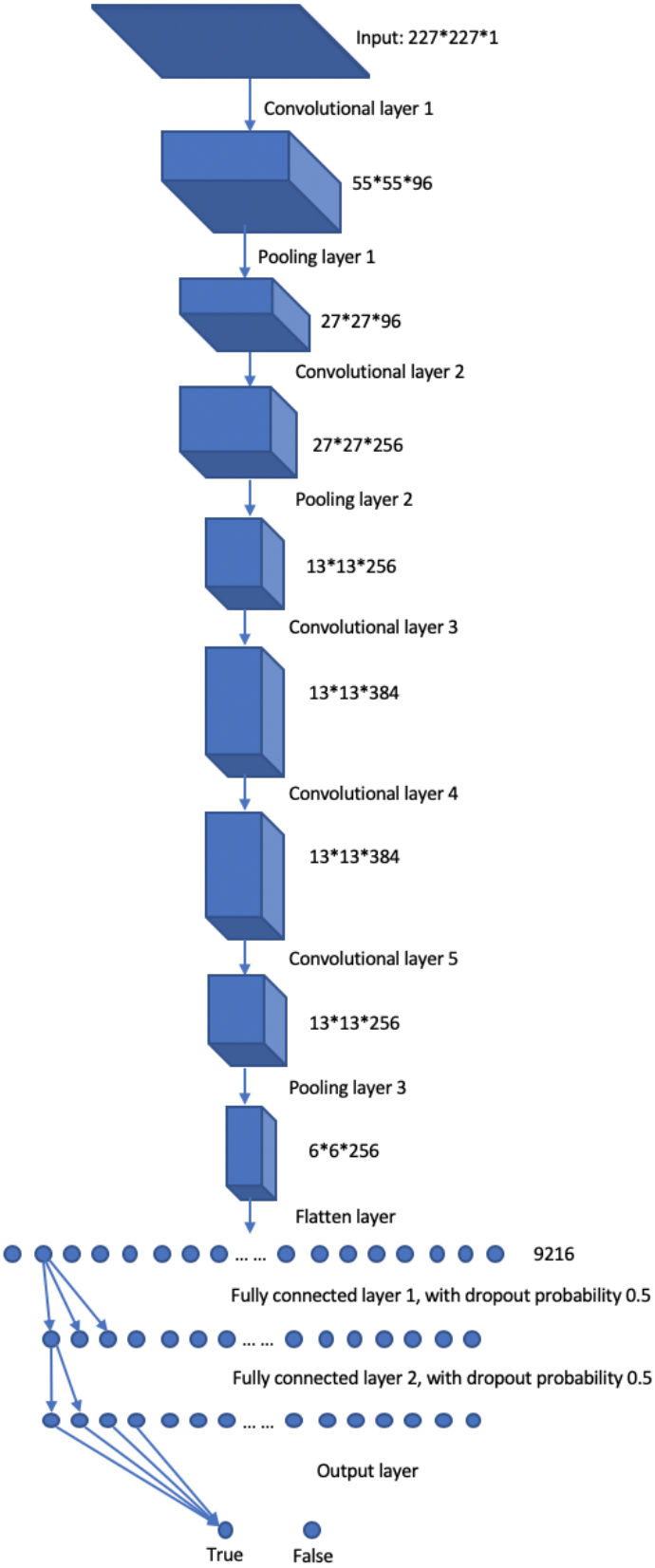


Figure 4.3: Structures of our AlexNet.

in Image Net tasks (it achieved an error of 15.3%, more than 10.8 percentage points lower than that of the runner up), we could say that AlexNet is a very suitable solution for that task. However, it also means our task, which is far more simpler than Image Net task, could increase the risk of overfitting.

To solve this problem, we propose following solutions to prevent our model from overfitting. More details would be given in chapter 5 on model developments.

* Early Stopping

  We save the parameters of the model in each step of training, always keeping the best performance model on validation set, and stop before overfitting.

* Batch Normalization

  We don't have to care about the learning rate, dropout, localization response normalization and other parameters. It could greatly save our efforts on parameter fine tuning, which also decrease the effect of overfitting on those parameters randomly set manually.

* Data Augmentation

  The best solution to solve overfitting is to get enough data, a possible solution is to generate data by randomly cropping and rotation. Distortion may be not a good idea, since indoor designed environments have very regular pattern and distortion may destroy intern structures, so we will not execute this.

4.3 Summary

In this chapter, we discussed our solution for our special image binary classification task. Since convolutional neural network is a mature and steady model, which has proved itself to work in many industrial tasks, we are interested in discovering whether it could fit in our specific task.

In next chapter, we will show the details of our model, and how we integrate different simulation techniques, CNN model, and different training methods together.

Chapter 5

Implementation

In this chapter, we introduce the details of the implementation and integration of our system.

5.1 ROS Architecture

The Robot Operating System (ROS) [31] is a set of software libraries and tools that help you build robot applications. With embedded state-of-the-art algorithms, and with powerful developer tools, it offers highly integrated and complete solutions for all kinds of scenarios in robot developments.

So in our research, we use ROS as a middleware, with a ROS-based framework. The version and details of our experimental environment are listed here:

* Ubuntu 16.04
* ROS Kinetic
* Python 2.7
* Jupyter notebook

We divide our work of how to implement our model into two parts, as shown in Table 5.1. And in the rest of this chapter, we will discuss about the details of the implementation of our system.

5.1.1 Package And Rqt_graph

All ROS software is organized into packages. A ROS package is a coherent collection of files, generally including both executables and supporting files, that serves a specific purpose. Our current analysis is based on the previous research of [23], [24].

PART 1	PART 2
* Data extraction from ROS bag files * Convolution neural network structure design * Fuzzy rules * Decision trees * Data visualization * Dateset creation * Model training * Model testing * Parameter fine tuning * Test result analysing	* Data pre-processing * Map cropping, resize, and processing * BC and FC labelling * Model training * Model offline testing * Model online testing * Model selection * Model freezing and saving * ROS integration * Model real application testing
MINGJU LI is in charge of this part	CHANG LIN is in charge of this part

Table 5.1: The responsibility during the project

So besides those packages which we could find as standard libraries in ROS, we use also:

* blueprint_exploration package: this package implements information processing and decides the next step to exploration.

* floorplan_analyzer package: this package would do map layout reconstruction.

* navigation_2d-master package: this package provides a node for higher level navigation of a mobile robot in a planar environment.

* partialmap_navigator package: this package provides sensor information processing and navigation in the partial map exploration.

Different packages could communicate with each other by ROS topic, acting as publishers-subscribers.

Node is the function unit in ROS [44], which could be regarded as one branch of the package. Messages in ROS are organized into named topics. A node that wants to share information will publish messages on the appropriate topic or topics; a node that wants to receive information will subscribe to the topic or topics that it's interested in. The ROS master takes care of ensuring that publishers and subscribers can find each other; the messages themselves are sent directly from publisher to subscriber.

And a ROS tool, known as rqt_graph is widely used to monitoring the status of ROS. Figure 5.1 is the rqt_graph of our current project.

Here each oval means a node , and each square means a topic. The lines mean the relationships between different objects. We describe the most interesting part, which is the part of navigator and floor plan analyzer in Figure 5.2.

The core part of these processes happen between node */Navigator*, which would provide agent navigation in the environment, and */FloorplanAnalyzer* which would give map information processing, and the topics related are topic */analyzer*, which is about the information of time and current map, and topic */analyzerResult* which is about the information of analyzer result.

The message type of these topics are listed in Table 5.2:

The current stop criterion could be visualized. The */Navigator* sends current collected map information, integrates it into a map and then forward the map to */analyzer* topic periodically. The interval between two messages is about 5 seconds. The */FloorplanAnalyzer* node is subscribing to this topic, and collects the map information broadcasted on this topic.

From the official document [41], this *nav_msgs/OccupancyGrid* represents a 2D grid map, in which each cell represents the probability of occupancy. std_-msgs/Header is a */std_msgs*, which represents the name and timestamp of the

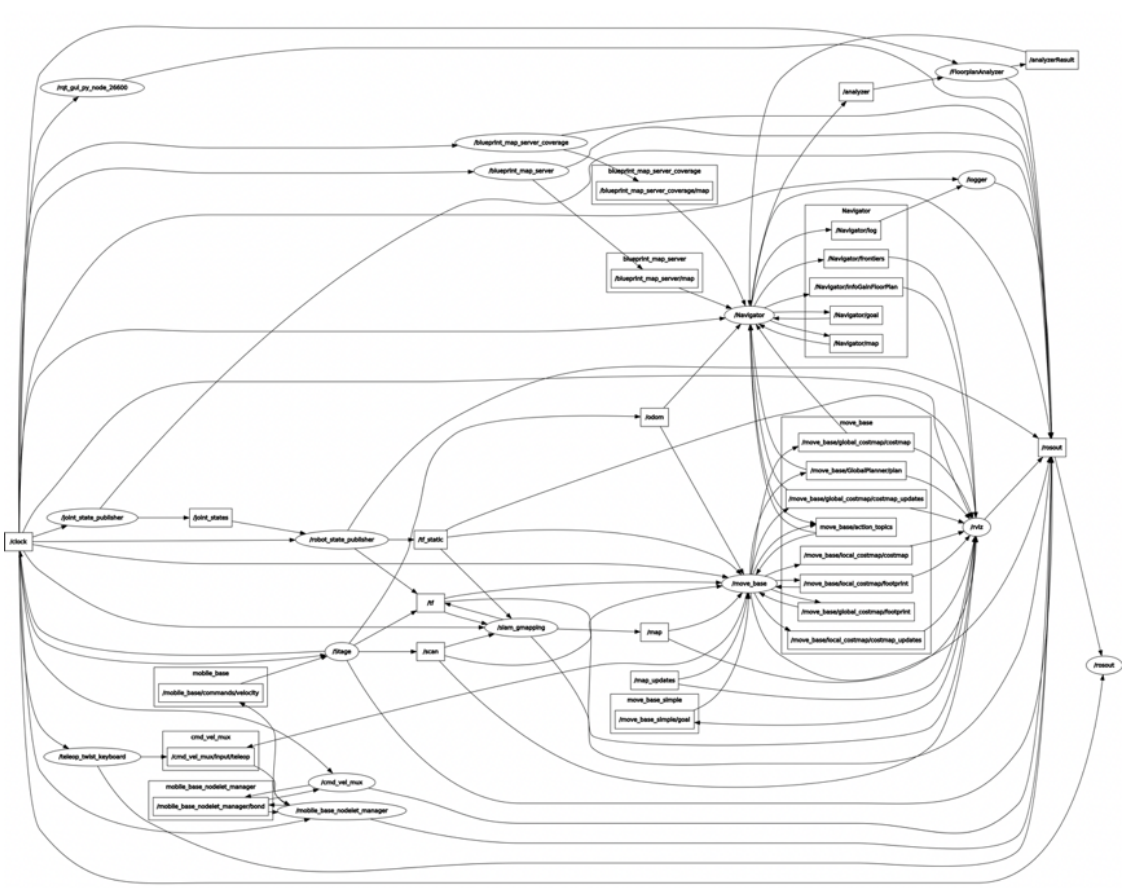


Figure 5.1: The topics and nodes in our current ROS project)

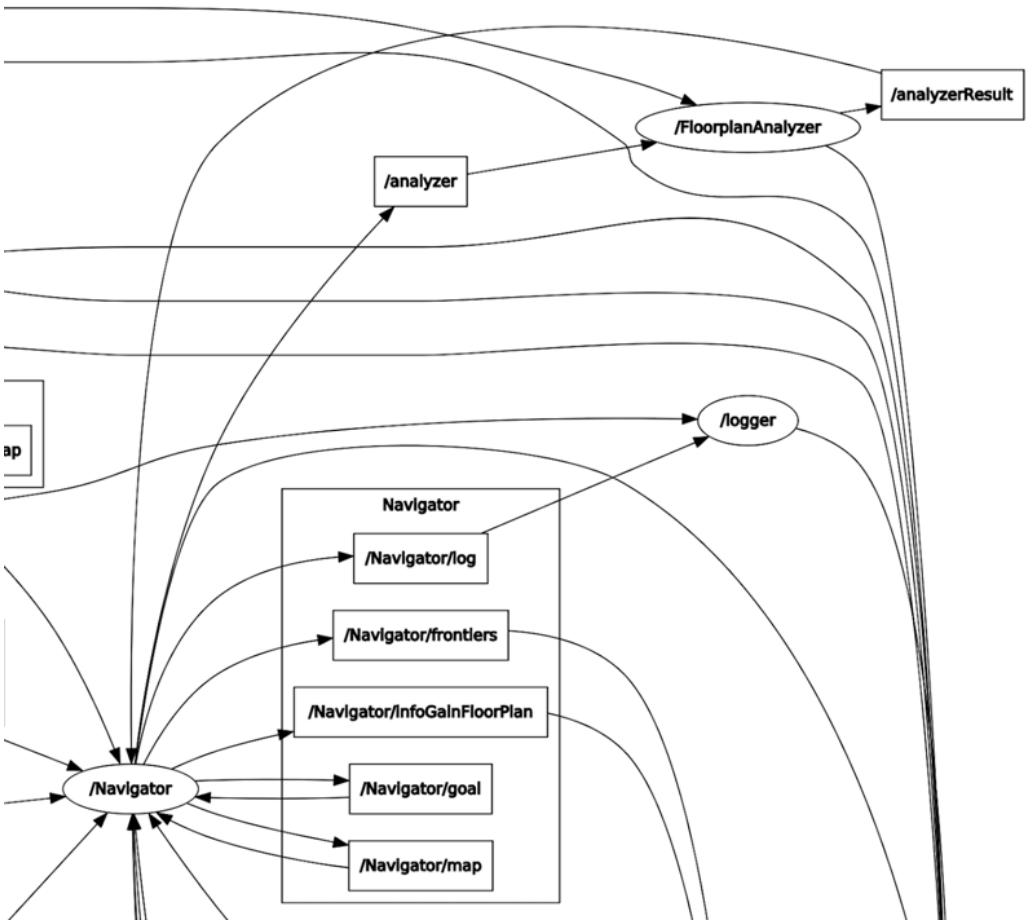


Figure 5.2: The communications between the navigator nodes and stopping criterion)

/analyzer	/analyzerResult
nav_msgs/OccupancyGrid	floorplan_analyzer/FrontierPrediction
std_msgs/Header header * uint32 seq * time stamp * string frame_id nav_msgs/MapMetaData info * time map_load_time * float32 resolution * uint32 width * uint32 height * geometry_msgs/Pose origin - geometry_msgs/Point position (float64 x/y/z) - geometry_msgs/Quaternion orientation (float64 x/y/z/w) * int8[] data	float32[] frontierPredictedArea int32[] frontierNumber bool[] touchMapEdges float32[] frontierPredictedAreaGreyPixel

Table 5.2: The topic information of /analyzer and /analyzerResult

current message. nav_msgs/MapMetaData is the *nav_msgs/MapMetaData*, in which includes the map data are saved in row-major order, starting with (0.0). Occupancy probabilities are in the range [0,100]. Unknown is -1.

The node */FloorplanAnalyzer* could handle the map information, do layout reconstruction and analysis on the map data. For the result of the analyzer, it would publish this result to */analyzerResult*, which would tell the */Navigator* whether it should stop or not.

5.1.2 Integration Of Deep Learning And ROS

Since we would like to integrate our deep learning model into ROS system, the way for implementing such function is to replace the original stopping decision module in the package *floorplan_analyzer* package.

In the end of training stage, the result of the convolutional neural network would be saved, or to be more specific, frozen into one static model. This will keep parameters, convolutional kernels, weights, and functions in the model.

Tensorflow is a popular tool to develop and train machine models [37]. To use this model, we have to load it again into the tensorflow sessions, and then feed it into specific models, with the same shape of defined tensors. After that, we could get the output of the model. So the implementation could be divided into these steps.

* develop the deep learning model.

* frozen graph.

* inplement into *floorplan_analyzer* package.

As for the details of the floorplan_analyzer integration, starting from the original ROS package, we are going to define a new ROS topic, known as */STOPCRITERION*. Our new */FloorplanAnalyzer* node would subscribe to the same ROS map information topic, and for each result, it would output a “True” or “False” message to the ROS topic of */STOPCRITERION*. Hence the */Navigator* node could decide its next move according to the result of the message it received.

According to our current model, since we would like to compare the difference between two kinds of stopping criteria, we did not deprecate the previous method in our project. We keep both methods and compare their performances in different environments.

Since we are not only looking for a method of doing the image binary classification, we would build a hybrid system, with both data driven models and the original feature extraction models.

5.2 Data Collection

The very first step in the training of data driven models is to get the proper amount of data. In our current dataset, we have about 100 environments, and for each environments, we have about 30 to 50 runs of exploration. In each run, we have ROS bag files and several maps sampled at different stages of exploration.

Those sampled maps have very clear features and boundaries, but the problem for this part of data is that only 3 or 5 maps are sampled during one run. Generally speaking, each run would take a time about 3600 seconds, and for some maps with particularly large rooms or complex structures, sometimes this time could reach over 7200 seconds. Only 3 to 5 maps are not enough to cover the whole exploration. We must have other methods to extract more effective data.

5.2.1 ROS Bag Play

Fortunately, we have the ROS bag files. ROS bag is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids de-serialization and re-serialization of the messages. In our runs in different environments, we have record the information related to the map.

Take the ROS bag files of environment 7A-2 for example. Table 5.3 is the result of "rosbag info 7A-2.bag".

The topics recorded in the bag file are the thing that is of significance for us. Here the three topics save information of autonomous agent's pose, locations, and sensor information in each timestamp. With this information, we could recover the observed map in any specific time stamp.

However, this map information replayed with high speed could not replay everything correctly. Some information may be lost, and during the map reconstruction some shift and distortion may appear, which may result map like Figure 5.3.

On the left of Figure 5.3 is the observed map recoved with ROS bag play with 2x speed, on the right of Figure 5.3 is the ground truth map. This problem becomes extremely frequent when we try to play the bag file with speed of 5x or 10x. So we have to monitor this process, and prevent it from saving a wrong map. Some manual filtering, which means that human has to check the data and drops those maps with distortion, must be executed to avoid use these kinds of confusing data to train our model, otherwise it would have bad performance.

5.2.2 Map_server Saver

ROS map is another package we use to extract data from ROS bag files. From the official document [43], *map_server* provides the *map_server* ROS node, which

path:	7A-2.bag
version:	2.0
duration:	30:04s (1804s)
start:	Jan 01 1970 01:00:00.50 (0.50)
end:	Jan 01 1970 01:30:05.10 (1805.10)
size:	31.1 MB
messages:	54141
compression:	lz4 [136/136 chunks; 29.80%]
uncompressed:	uncompressed: 102.1 MB @ 57.9 KB/s
compressed:	compressed: 30.4 MB @ 17.3 KB/s (29.80%)
types:	* nav_msgs/Odometry * sensor_msgs/LaserScan
topics:	* /base_pose_ground_truth 18047 msgs: nav_msgs/Odometry * /base_scan 18047 msgs: sensor_msgs/LaserScan * /odom 18047 msgs: nav_msgs/Odometry

Table 5.3: ROS bag info of one run in environment 7A-2

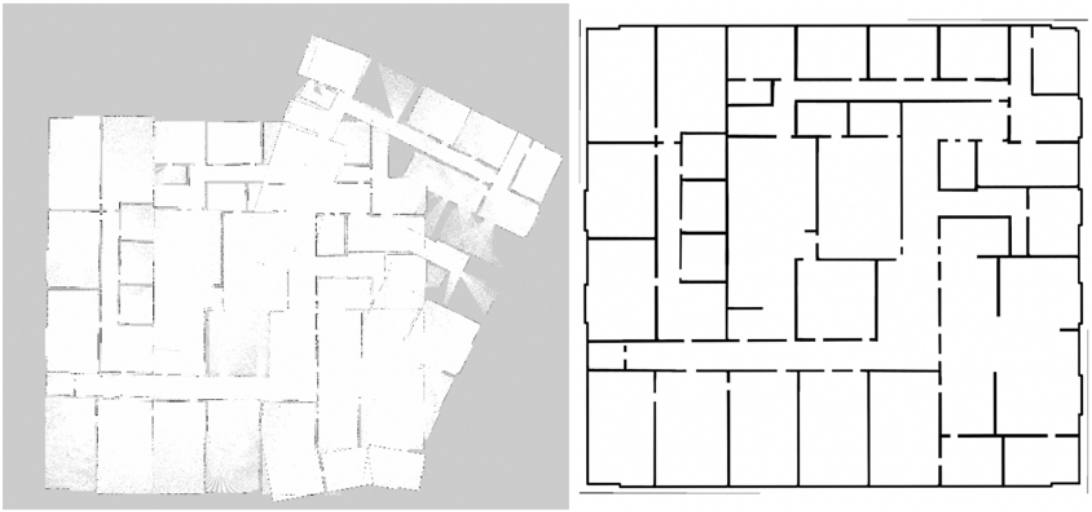


Figure 5.3: The map distortion between the replayed map and ground truth

offers map data as a ROS Service. It also provides the *map_saver* command-line utility, which allows dynamically generated maps to be saved to file. In the ROS bag replay, we are going to use this to save the map with format *.pgm* and *.yaml*.

5.2.3 Procedure

We write a python jupyter notebook to batch processing the input data. A while-loop is used to extract every environment. The whole process could be described in following structure:

* (open a new terminal) *roscore*

* (open a new terminal) *source Thesis/airlab-polimi-partialmapsexploration-01004f60f141/devel/setup.bash*

* *rosrun learning_tf nodes/turtle_tf_broadcaster.py*

* (get the time interval of this bag file)

  roslaunch replay.launch dur:=time srcbag:="/home/cesare/Thesis/Maps_-Data/7A-2.bag"

  [Here to be faster, in the launch file, we have set the rosbag play rate to 2.0.]

* (execute this command periodically)

  rosrun map_server map_saver -f /home/cesare/MAPS/

5.3 Data Pre-processing

Before the analysis of BC/FC methods in Section 2.3, we do some data preprocessing on the maps we obtained. These are mainly image processing algorithms, since we are dealing with a specific category of the images.

5.3.1 Crop And Resize

The partial map images saved by running the ROS bag are initially set to 4000 × 4000 pixels size. All the partial maps from the same environment have the same resolution, which means they all have the same scale. The center of the image is the starting position of the robot. The reason why we save the map at 4000 × 4000 pixels size is that we have no prior knowledge about how large the environment is. Therefore, we need to make sure the image is large enough to cover all the area of the environment.

In our convolutional neural network, the dimension of input data is bounded to (227, 227, 3), representing a map image with size 227 × 227 pixels and 3 channels (RGB). Therefore, we need to compress the initial images and resize them. However, this raises a problem. At the early stages of exploration, when the area of observed space is relatively small, it turns out that the observed space only contains a small area in the initial image (4000 × 4000 pixels). Then, after compression, the observed space turns into a very small area in the compressed image (227 × 227) with a high information loss due to compression.

In order to solve the problem, we need to crop the image and let the observed space cover approximately the same area in all initial images.

We do this by cropping the image step by step until an observed pixel or a wall pixel is detected. In our work, step is set to 100 pixels. We start from the outline border, crop the image into a new cropped image with size 3800 × 3800 (4000 - 2 × 100). Then if a pixel, which doesn't represent an unexplored area, is detected in the outline border, it means that we crop too much and need to backtrack a step. Otherwise, we continue cropping. By cropping and resizing the images generated from ROS bag, we are able to obtain the new dataset of images with size 227 × 227 pixels without too much information loss. As a cost, the resolution (real size of a pixel) is different in different images. However, this is not a critical issue, since our convolutional model aims at extracting the geometrical features and symmetries from the image, and usually doesn't care too much about the real size.

Figure 5.4 shows both the original image and the one after cropping and resizing.

As it can be seen from the above two images, after cropping, the explored area is still in the center of the image and covers much more space than the original one, which means we obtain a new image for training with less information loss.

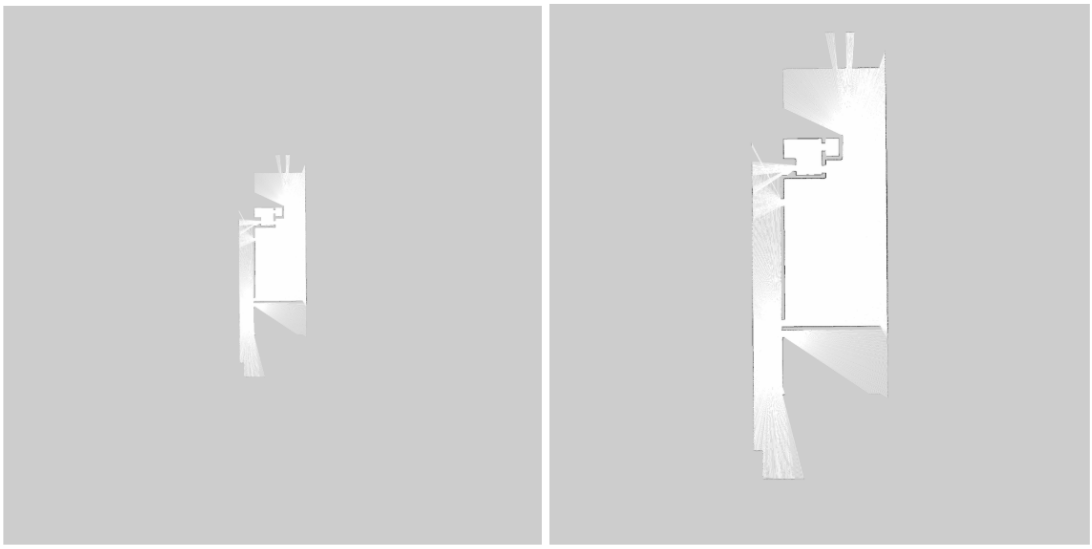


Figure 5.4: An example of crop and resize

5.3.2 Image Feature Augmentation

Due to the noise and error from the laser range scanner, the observed area is usually a sector composed of dozens of laser rays. In other words, intuitively this kind of area should be fully observed, but instead it's composed of half observed pixels and half unexplored pixels. As a consequence, it reduces the accuracy of layout reconstruction, and also increases the training cost of our convolutional model. Therefore, we should try to augment the image and turn this kind of area into a more recognizable one.

To do this, for each pixel, we compute a weighted average of its color and all of its adjacent pixels' colors by using a convolution kernel. Then we use a mapping function as a threshold and apply this function to all pixels of the image, by given the input of the weighted average calculated above. Closer to the position of the laser scanner, this value is higher since the laser range data are denser. By this method, we can find a suitable 'cut' of the observed sector and turn it into a more recognizable space, as shown in Figure 5.5.

5.4 Data Labelling

In our dataset, we have the samples of the same maps at different timestamps with total number of about 3500. To manually label these data would be very annoying and boring. So we would like to automatically labelling the maps. In this section we will discuss about several methods we used, their advantages and disadvantages.

But first thing we should start with, is the result of current BC/FC method

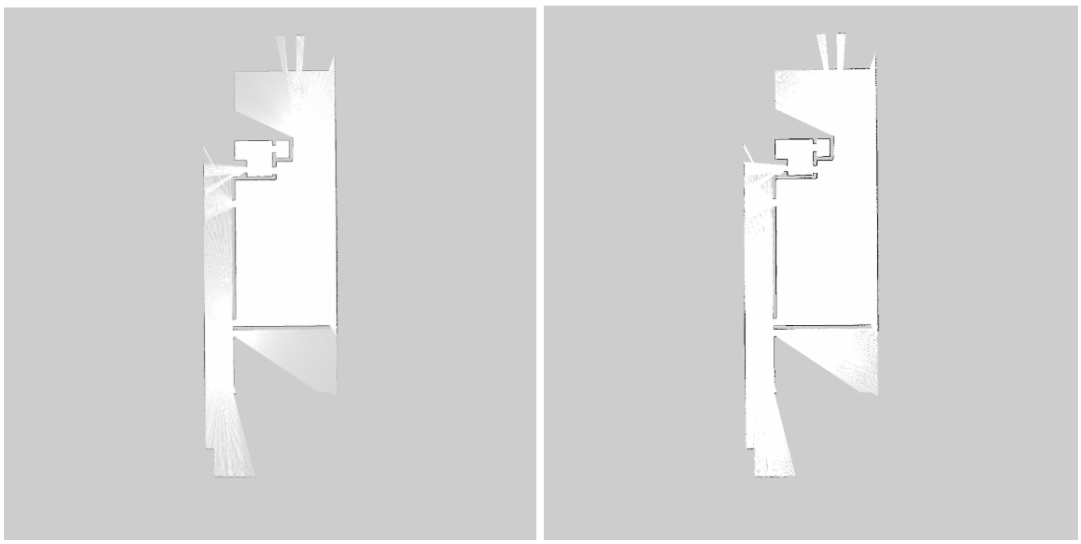


Figure 5.5: An example of image feature augmentation

introduced in Section 2.3, with manually attached labels as shown in Figure 5.6.

Below there are the BC/FC result for some of the environments. We could see that for each map (of one specific environment, in one specific run, under one specific timestamp), the BC/FC module in our system could give the output of *MapBC, MapFC, GroundTruthBC, GroundTruthFC (score value is calculated from these four BC/FC values), Map complete rooms, Map partial rooms, Map rooms, GroundTruth rooms Cells, GT area, Number of Frontiers.*

And our goal in this section is to find how to use above information to label our data.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Map Name	Run	Timestamp	Map BC	Map FC	GT BC	GT FC	Score	Map complete rooms	Map partial rooms	Map rooms	GT rooms	Cells	GT area	Frontier	Label
2	11-0	Run00	156250525	92.543357	84.823498	78.445962	71.902091	81.928727	5	0	5	15	46	1112813.5	1	0
3	11-0	run3	40Map	80.44883	96.129143	76.982114	91.986728	86.386704	19	0	19	15	152	1112813.5	0	1
4	11-4	run2	40Map	73.344905	85.730042	69.111267	80.781505	77.24193	23	0	23	23	210	869867.6785	0	1
5	130604-Plan_update	Run00	156250581	83.788777	74.913429	40.468459	36.181827	58.838123	13	0	13	38	107	1652228.5	0	0
6	130604-Plan_update	run3	50Map	87.440196	68.623964	89.156304	69.970784	78.797812	29	0	29	38	316	1652228.5	0	1
7	24-1	Run00	156250630	96.261183	95.92391	71.660353	71.409275	83.81368	17	0	17	23	50	1971005	1	0
8	24-1	run7	70Map	95.926966	96.64675	95.730974	96.449287	96.188494	23	0	23	23	135	1971005	0	1
9	324-Elm-Farm-Plan	run7	40Map	56.63276	80.05889	60.047892	84.886691	70.406558	40	0	40	26	157	585240.5	0	1
10	41-0	Run00	156250708	70.526216	61.994129	37.119012	32.628445	50.56695	9	0	9	23	78	1880669.5	2	0
11	41-0	run2	80Map	87.831021	88.655613	89.560156	90.400982	89.111943	20	0	20	23	195	1880669.5	1	1
12	41-1	run1	80Map	90.135497	83.365964	87.071835	82.464407	86.259423	16	0	16	16	126	1873398	0	1
13	41-2	run1	120Map	78.541095	73.865824	78.144284	75.48253	77.008433	29	0	29	30	359	1880786.5	0	1
14	43-2	Run00	156250883	65.889694	90.597095	28.278711	38.882698	55.91205	8	0	8	15	83	756610.5	10	0
15	43-2	run1	60Map	72.330625	61.292048	72.289279	61.257012	66.792241	14	0	14	15	165	756610.5	0	1
16	44-1	run1	100Map	90.101965	83.139524	90.842041	83.822412	86.976486	19	0	19	22	113	1599713.5	0	1
17	48-3	run3	60Map	81.300141	66.811003	85.383289	70.166461	75.913223	22	0	22	29	155	2231936.5	0	1
18	54-3	run3	40Map	86.197394	92.637907	86.433551	92.891709	89.54014	20	0	20	20	127	1279777	0	1
19	7A-2	run3	30Map	94.941295	97.508857	94.143079	96.689054	95.820571	11	0	11	9	68	798477.5	0	1
20	8-1	Run00	156251054	83.464623	78.730848	23.105656	21.795197	51.774081	8	0	8	28	34	2795800	2	0
21	8-1	run8	70Map	79.123833	78.99007	74.864575	74.738012	76.929122	24	0	24	28	174	2795800	0	1
22	9-00	Run00	156251157	97.572146	89.89263	54.073912	49.81797	72.839165	6	0	6	20	85	2766751.5	3	0
23	9-00	run1	50Map	94.0013	88.461982	92.744286	87.279941	90.621653	15	0	15	20	122	2766751.5	1	1
24	9-2	run3	80Map	87.877419	91.777196	79.704868	83.241968	85.650363	52	0	52	50	463	2476390.5	0	1
25	ES6-3	Run00	156251690	94.29118	90.62627	60.70248	58.343095	75.990756	18	0	18	41	72	1621015.5	0	0
26	FALAELEM1_updated	Run00	156251843	72.432567	82.84325	40.021443	45.773697	60.267739	29	0	29	37	225	1890113	1	0
27	Franchl_updated	Run00	156251983	85.636737	80.808747	77.877545	73.486999	79.452507	10	0	10	18	73	1466502	0	0
28	Liselottel_updated	Run00	156252287	95.982969	74.977498	88.966845	69.496824	82.356034	8	0	8	12	99	447681.5	0	1
29	MediaCache_updated	Run00	156252359	92.555922	91.986874	65.438892	65.036563	78.754563	22	0	22	26	120	1501470	1	0
30	N10-1	Run00	156252405	92.951715	92.90447	73.771979	73.734082	83.340662	18	0	18	21	174	1002934.5	2	0

Figure 5.6: The BC/FC output of 100 maps

5.4.1 Data Visualization

Before we start with the further details of this section, we do some data visualization on our current manually labelled 100 map samples. Besides those attributes mentioned above, we also consider attributes, like:

* avg(bc,fc)

  This is the average value of BC and FC values.

* diff(bc,fc)

  This is the difference between the BC and FC values.

* room_differences

  This is the differences between the number of the rooms in the reconstructed map and ground truth map.

* room_abs_differences

  This is the absolute value differences between the number of the rooms in the reconstructed map and ground truth map.

Their visualizations could be viewed in Figure 5.7 and Figure 5.8. We use different dots to represent different kinds of maps. The blue dots means "maps still need further exploration", and red cross means "well-observed maps"

It is quite obvious that the data is not linearly separable in low dimensions. Even if in some cases we can identify relatively separable patterns, we need to find some effective method to separate and label them.

5.4.2 Logistic Regression And Support Vector Machine

The first thing coming into our mind is the logistic regression and support vector machine (SVM). Both methods are mature classification models in the area of supervised learning. However, to apply these methods, one thing we must understand is that our data is too limited to executed these two kinds of learning methods.

The problem of limited data is that our model would overfit, which will make our model completely useless. Hence, we need to select a different model for this specific scenario.

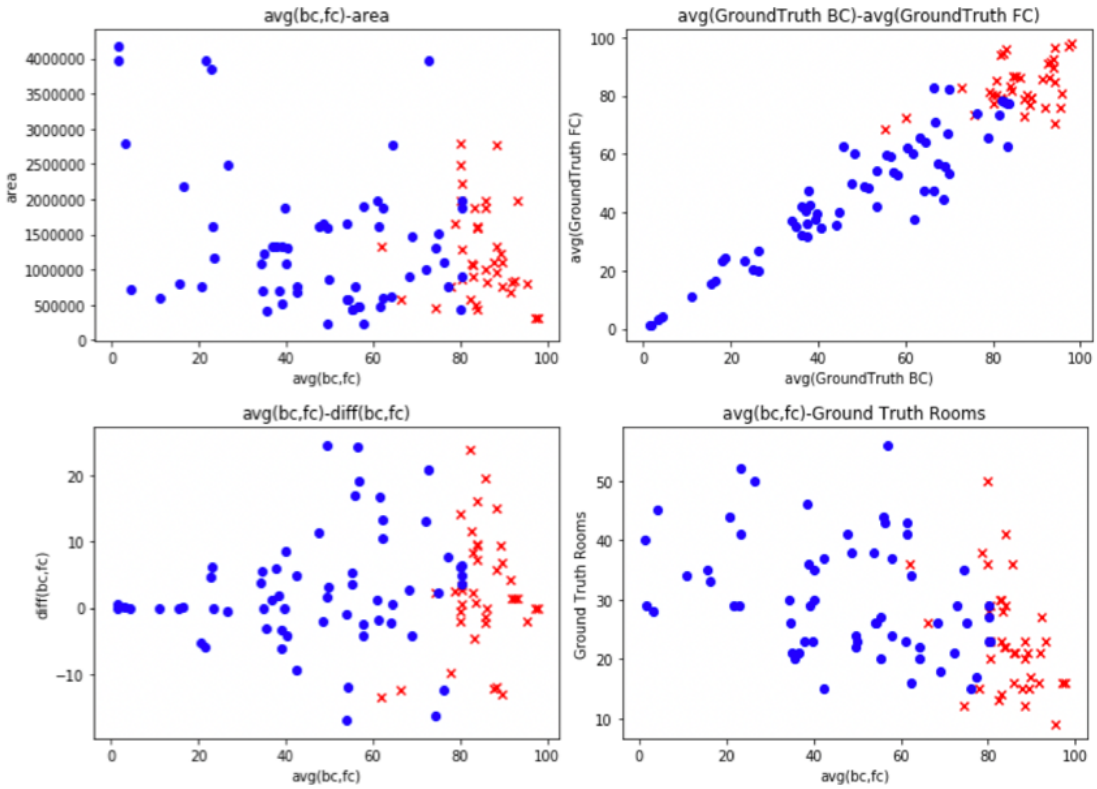


Figure 5.7: The visualization of data (blue are the maps that needs exploration, and red are the well observed maps)

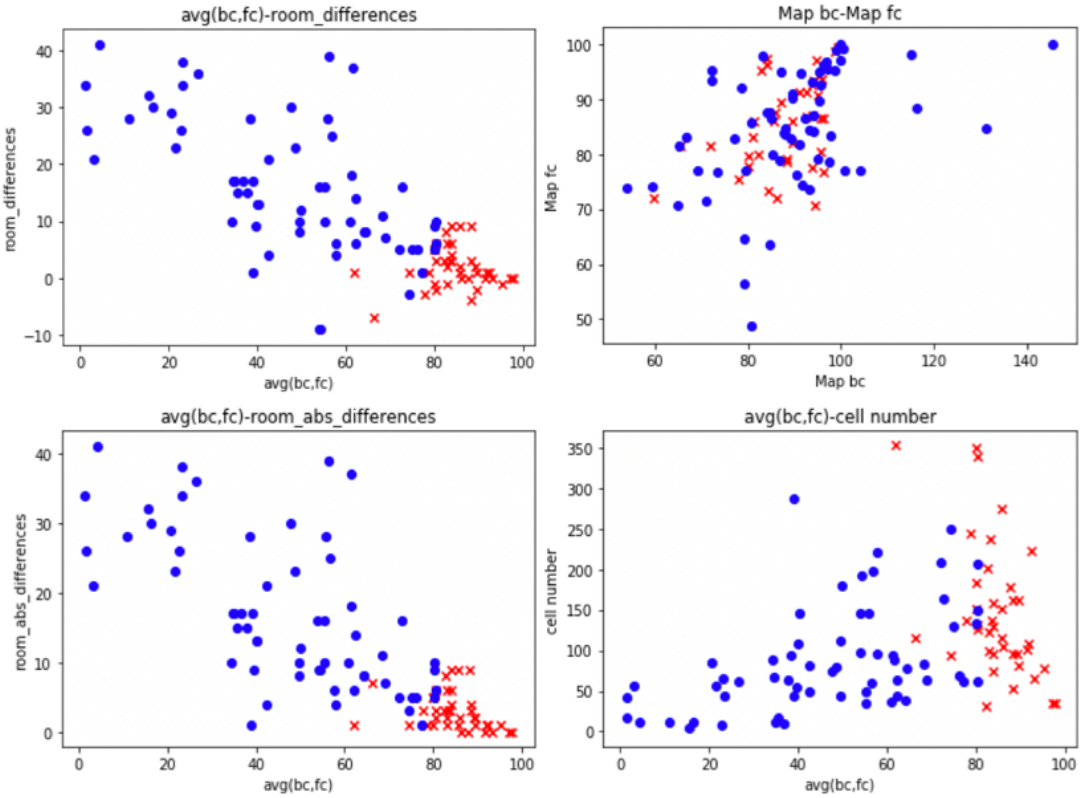


Figure 5.8: Another kinds of visualization of data (blue are the maps that needs exploration, and red are the well observed maps)

5.4.3 Fuzzy Rule

At the beginning, we supposed that BC/FC could be a great measure for the similarity between two maps. The problem we met is that, for maps with different scale (from about 500 m^2 to 2500 m^2) and different complexity (from about 10 rooms to over 50 rooms), a single BC/FC value threshold could be a bad idea.

We could see in Figure 5.9 are two observed maps, and we could try to check the features of BC/FC for both maps.

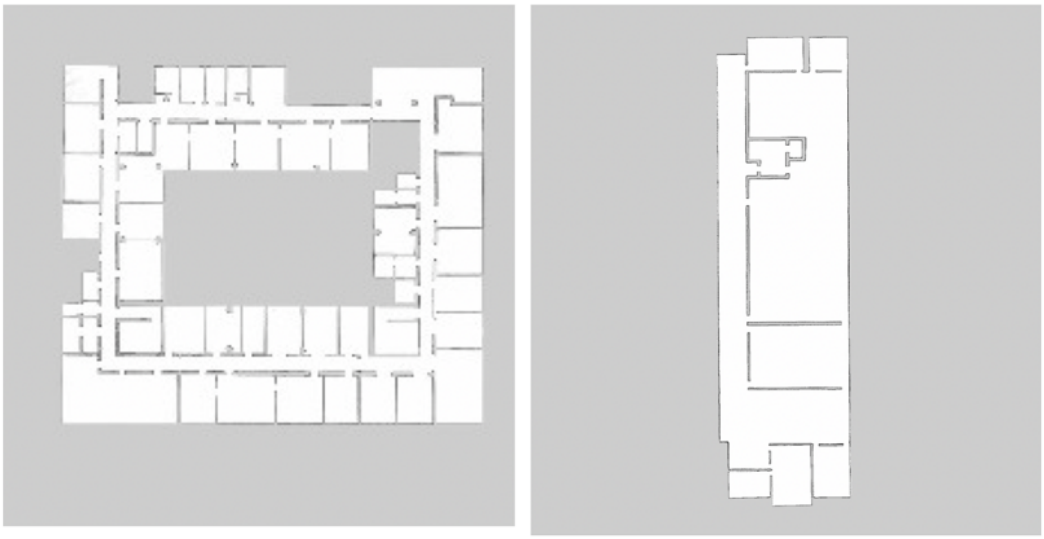


Figure 5.9: Two well observed maps

On the left of Figure 5.9 the observed map of environment NLB, and on the right of Figure 5.9 is the observed map of environment 7A-2.

	NLB	7A-2
Map BC	89.1709124814%	94.9412952601%
Map FC	35.7892492485%	97.5088574490%
GroundTruth BC	93.4221128548%	94.1430785241%
GroundTruth FC	37.4954925237%	96.6890539687%

Table 5.4: Two well observed maps and their BC/FC values

They are almost well explored and exploration in them could be stopped. However, if we concentrate on the BC/FC values shown in Table 5.4, we could find

that, the BC/FC value for the first environment is about 0.90, and for the second environment, the BC/FC value is about 0.60. Hence, it could be reckless that we just set one single value of BC/FC, without considering the complexity of the map.

So we need to integrate the BC/FC and other map information into a single model. The best way of doing this is to use the fuzzy logic. Because by fuzzy logic and fuzzy rules design [7], we have several benefits.

For the reconstructed maps, (from our observed map), it could be a "GOOD" map (map that is similar to the ground truth) or a "BAD" map (map that is not similar to the ground truth because it has been wrongly predicted). Instead of a crispy set, the concept of "GOOD" and "BAD" should be two fuzzy sets.

Hence, we would like to label the map "could be stopped" if the reconstruction of this map is "GOOD", and "still need further exploration" if the reconstruction of this map is "BAD".

To do the labelling, we would need to do the reconstruction for each instance in our training data, and label it by a fuzzy rule system.

How to design this fuzzy system? First we select several input variables that could be involved in the judgement of the similarity of the map.

Input Variables:

* MAP AREA Figure 5.10: the area of the ground truth map
* ROOM NUMBERS Figure 5.11: the number of the rooms in the map
* MIN(MapBC, MapFC, GroundTruthBC, GroundTruthFC) Figure 5.12: the min values of the BC FC values
* AVG(MapBC, MapFC, GroundTruthBC, GroundTruthFC) Figure 5.13: the differentce between the BC and FC values

Output Variables: MAP QUALITY: the similarity of the map, GOOD / BAD
We could write these rules to get the result of MAP QUALITY:

1. *IF MAP AREA IS BIG AND ROOM NUMBERS IS MANY AND MIN(BC,FC) IS MEDIUM AND AVG(BC,FC) IS MEDIUM, THEN MAP QUALITY IS GOOD*
2. *IF MAP AREA IS SMALL AND ROOM NUMBERS IS FEW AND MIN(BC,FC) IS SMALL AND AVG(BC,FC) IS SMALL, THEN MAP QUALITY IS BAD*
3. .....

The advantage of fuzzy rules is that the truth values of *MAP AREA*, *ROOM NUMBERS*, etc. may be any real number between 0 and 1. It could handle the

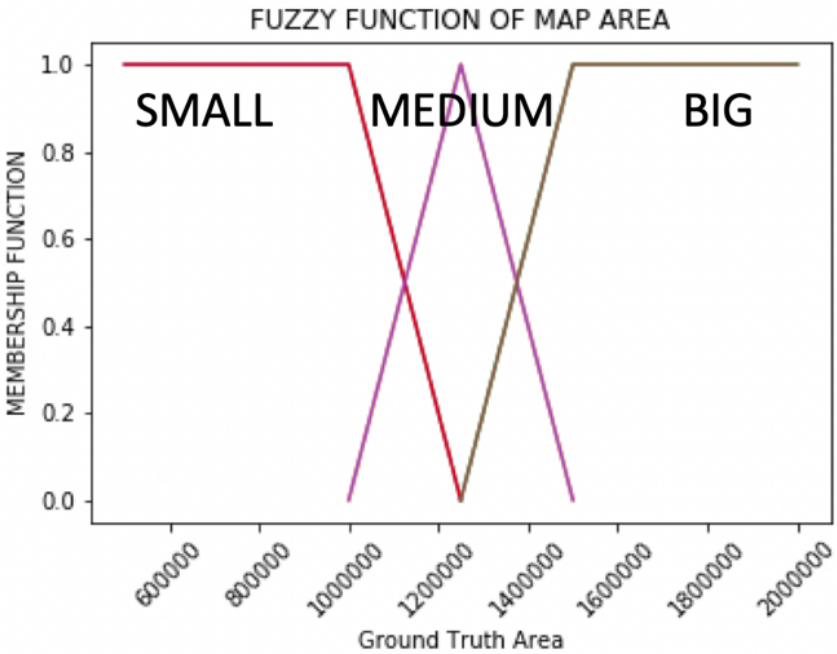


Figure 5.10: Fuzzy function of map area

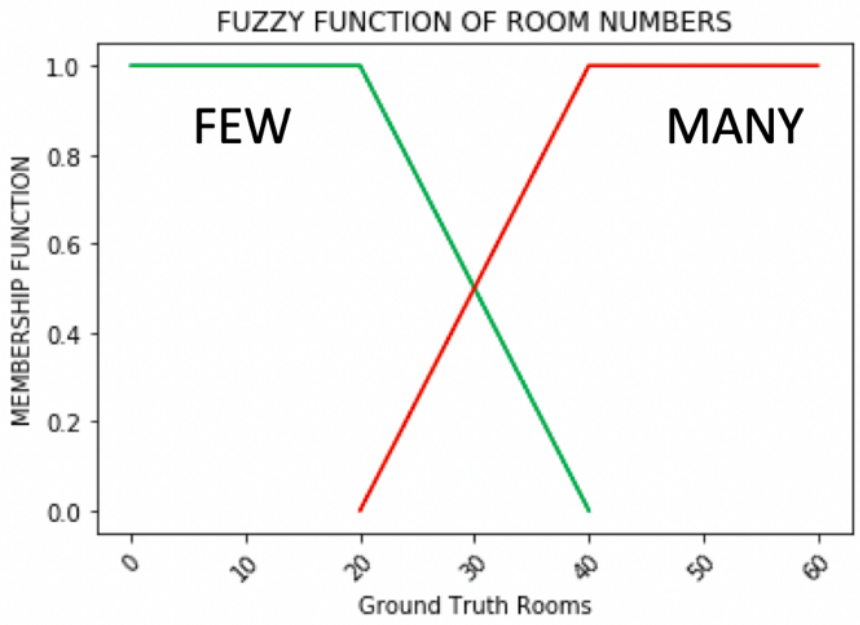


Figure 5.11: Fuzzy function of map area

concept of partial truth, where the truth value may range between completely true and completely false. These above rules are very readable and easy to understand.

However, it is not flawless. Fuzzy rules design, in most cases, are done by experts in certain area. However in our system, we have to continuously adjust our system and parameters to make it work. These may not be easy to handle.

5.4.4 Decision Tree

Decision trees are a more automatic method of rules design. They are very popular method in the area of today's data mining. A decision tree could apply a sets of tree-like rules and decision, to the data, and give relative accurate results according to our training data.

So we apply the decision tree models on the manually labelled 100 maps. Compared with logistic regression and support vector machine, it is easier to do generalization in decision trees by pruning and setting thresholds on the number of the rules and depth of tree.

To get several different models to evaluate, we tried to feed manually designed attributes, like the absolute values of difference between BC and FC, the average value of BC and FC, based on our current attributes. And we could get the following

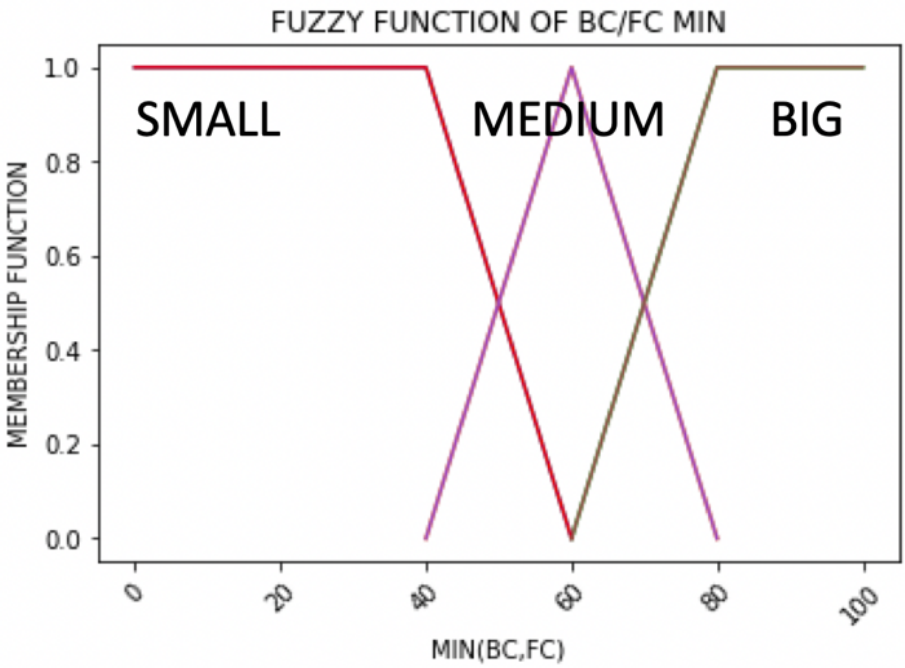


Figure 5.12: Fuzzy function of map area

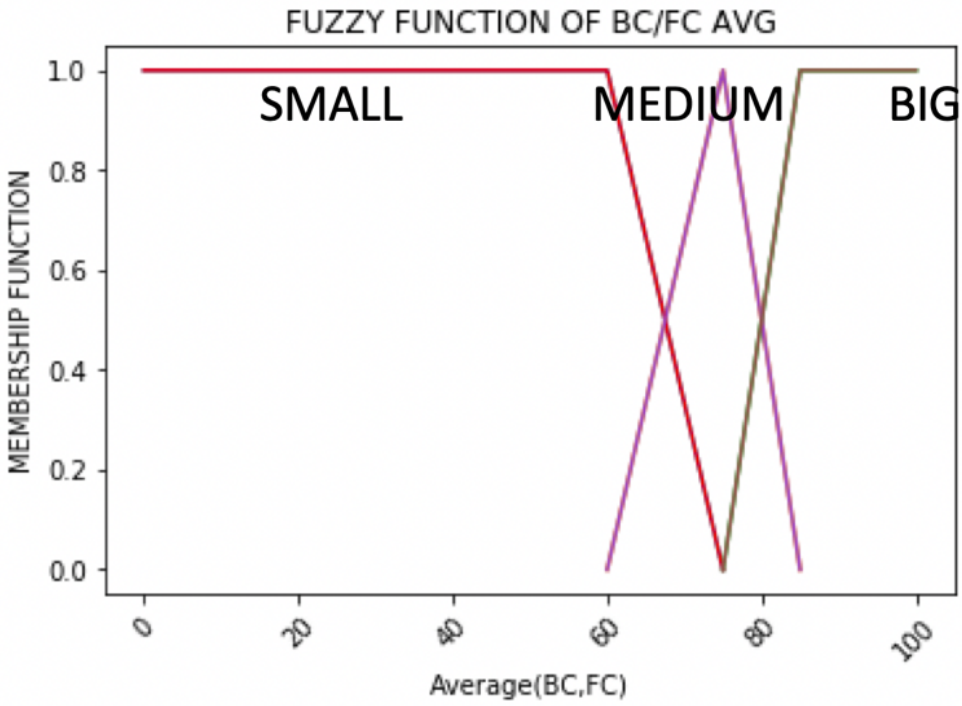


Figure 5.13: Fuzzy function of map area

result shown in Figure 5.14.

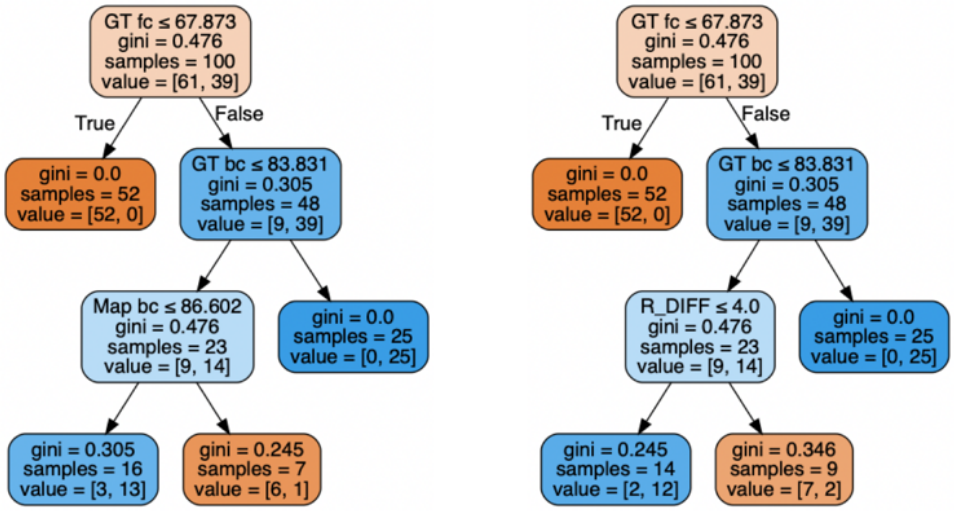


Figure 5.14: Decision trees obtained from different attributes

As we could seen from the above comparison, both decision trees have relatively close performances in the part of *Map_bc* branch and *R_DIFF* branch. And they both reached our expectations on the accuracy. Hence we could adapt the same trees on the testing data, which are the maps that has not been labelled. For the data to be used in the later training, it is labelled by the decision tree we obtained in this section.

5.5 Model Training

5.5.1 Parameter Definition

We first introduce multiple parameters which affect the performance of convolutional neural network:

- Learning rate: The learning rate refers to the magnitude of network weight updating in the optimization algorithm. The learning rate can be constant, decreasing, momentum-based, or adaptive, and the selection of learning rate depends on the type of optimization algorithm chosen, such as SGD, Adam, Adagrad, and so on.

- Epoch: The number of epochs refers to the number of times the entire training set is input to the neural network for training. When the difference of accuracy between test set and training set is small, the current number of epochs may be considered appropriate. Otherwise, it is necessary to increase the number of epochs, or adjust the network structure.

- Batch: The number of batches is a parameter in batch normalization. In the learning process of convolutional neural networks, small batches will perform better, and the number is generally within the interval [16, 128]. In addition, it should be also noted that the convolutional neural network is very sensitive to batch size adjustments.

- Activation function: The activation function is non-linear and theoretically allows the model to fit any function. Normally, the rectifier function (ReLU) works well in convolutional neural networks, while any other types of activation functions such as Sigmoid and Tanh are also available depending on the actual task.

- Number of hidden layers: Increasing the number of hidden layers to deepen the network depth will improve network performance to some extent, but when the test accuracy is no longer increasing, other improvements are needed. Increasing the number of hidden layers also raises the problem of increasing the computational cost of training the network.

- Number of neurons: When the number of neurons in the network is set too small, it may cause under-fitting. When the number of neurons is set too high, if an appropriate regularization method is used, there will be no adverse effects.

- Weight initialization: In the network, small random numbers are usually used to initialize the weights of each network layer to prevent inactive neurons. However, setting a too small random number may generate a zero-gradient network. In general, the uniform distribution method work wells.

- Dropout: As a common regularization method, adding the dropout layer can attenuate the overfitting effect of deep neural networks. The method randomly deactivates a certain proportion of neurons in each training epoch according to the preset probability parameters. The default value for this parameter is usually 0.5.

5.5.2 Evaluation Method

Confusion Matrix

Before we start the model evaluation, we need to select the evaluation methods. There are many evaluation methods which could be used in our convolutional model. And most of them are based on confusion matrix.

Confusion matrix is a standard format representing the evaluation of accuracy, which is also called error matrix. In artificial intelligence, confusion matrix is a tool, especially for supervised learning. In unsupervised learning, it's generally called a matching matrix. In the image accuracy evaluation, it's mainly used to compare the classification result with the actual measured value, and the accuracy of the classification result can be displayed in a confusion matrix.

Each column in a confusion matrix represents a prediction category. As for each row, it represents the actual attribution category of the data.

Confusion matrix is often used to describe the performance of a classification model on a set of test data for which the true values are known. In our work, the confusion matrix can be represented as a 2×2 matrix having 4 actual values: True Positive, True Negative, False Positive (Type 1 Error) and False Negative. All our following evaluation measurements are based on these 4 values. An example of confusion matrix in our work is shown in Table 5.6.

Accuracy

Accuracy is the most common metric, and it's easy to understand. Formally, it's defined as the number of samples correctly predicted divided by the number of all samples, calculated by the following equation:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Accuracy simplely tells us how many percent of sample we predict correctly. Usually the higher the accuracy, the better the classfier. However, sometimes accuracy is not representative for a model, especially when working with a class-imbalanced data set. A class-imbalanced data set means that there is a significant disparity between the numbers of labels. For example, digit recognition is not a class-imbalanced data set since commonly each digit has an approximately same probability of appearance, while a typical example of class-imbalanced data set is earthquake prediction. Usually earthquake has a significantly small probability, which means that the classifier can always predict False (no earthquake) to obtain a very high accuracy. To deal with this problem, we need some other measurements to help analysis.

AUC

For a class-imbalanced data set, accuracy is not the appropriate evaluation. AUC is a specific measure in the context of AUC ROC curve. For classification problems (especially binary classification), AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve is another performance measure of the classification at various threshold settings. ROC is a probability curve while AUC represents the degree of measure of the separability. It tells us how much the model is able to distinguish between different classes.

ROC curve, as shown in Figure 5.15 is plotted based on the real category of the sample and its prediction probability. Specifically, the x axis of ROC curve if the false positive rate, while the y axis is the true positive rate. Both of these two rates are calculated based on confusion matrix as following:

$$Truth\ Positive\ Rate(TPR/Recall/Sensitivity) = \frac{TP}{TP + FN} \tag{5.2}$$

$$False\ Positive\ Rate = \frac{FP}{TN} + FP \tag{5.3}$$

AUC as its name suggests, it's the area of the portion of space under the ROC curve. It's not so intuitive by this definition. From another point of view, AUC can be interpreted as the sorting ability of the classifier on the sample. For example, we randomly take a sample (sample 1) from all samples in category 1, and take another sample (sample 0) from all samples in category 0. Then we do the prediction on these two samples by our classifier. The probability of predicting sample 1 as category 1 is p_1, while the probability of predicting sample 0 as category 1 is p_0. Then the probability of $p_1 > p_0$ is equals to AUC. According to this explanation, if the classifier randomly classifies the samples, then we will obtain an AUC near 0.5. A excellent model has AUC near to 1 and it means it has good measure of separability, while a poor model will have a relatively small AUC.

As we can see, AUC is not sensitive to the consistency of sample categories. This is the reason why AUC is commonly selected as a evaluation measure for binary classification problems.

5.5.3 Underfitting And Overfitting

While tuning the parameters, we also need to pay attention to the problem of overfitting and underfitting, as shown in Figure 5.16. An important topic in machine learning is the generalization ability of the model. A model with strong generalization ability can be defined as a good model. For a model already trained, if the performance on the training set is poor, this may be caused by underfitting. On

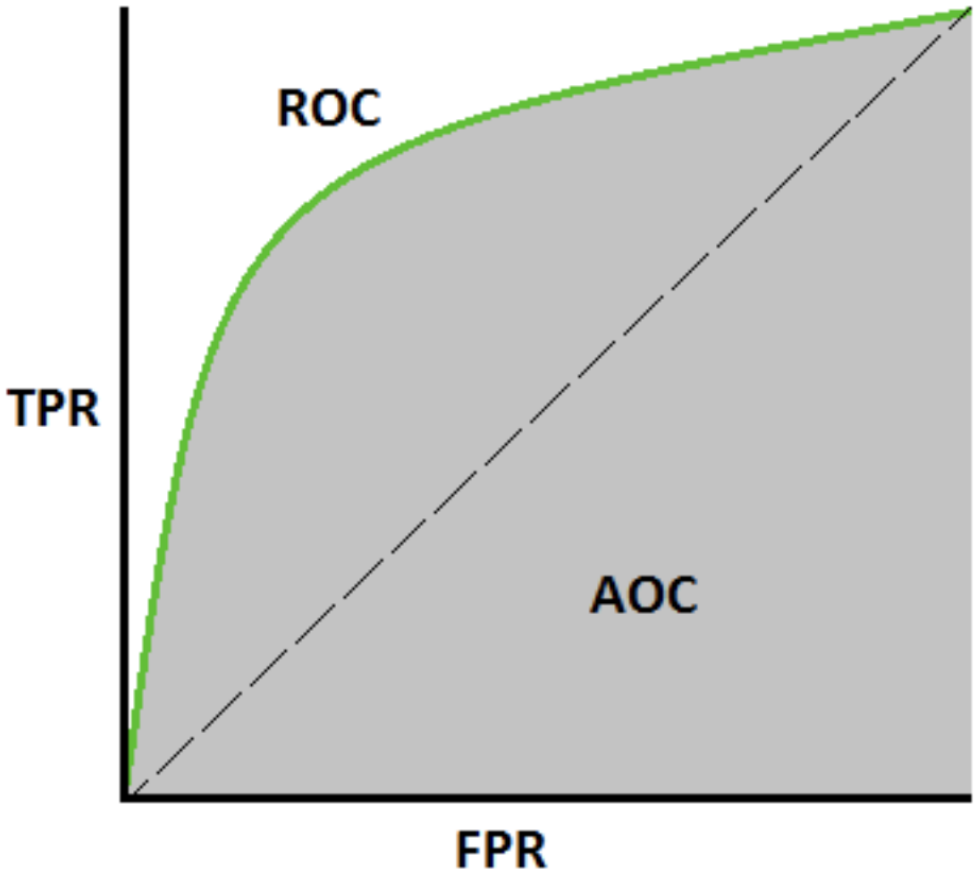


Figure 5.15: AUC ROC Curve [28]

the contrary, if the model performs very well in the training set but poorly in the test set, this is due to overfitting. Overfitting and underfitting can also be distinguished from the perspective of bias and variance. Bias is the difference between the expected output of the model and its real output, while variance describes the difference between the output of the model obtained from different training sets and their expected values. Underfitting leads to high bias, and overfitting leads to high variance. Therefore, a model needs to make a trade-off between both bias and variance.

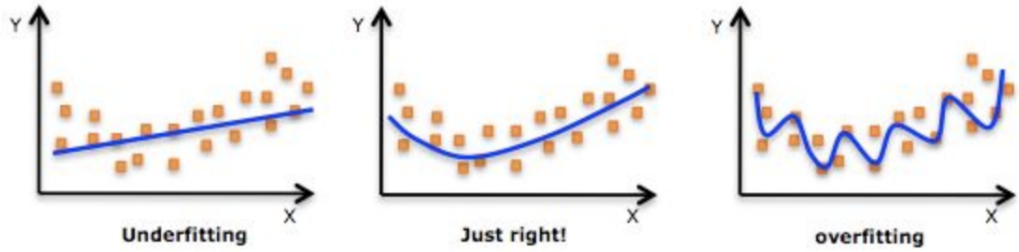


Figure 5.16: Visualization of underfitting and overfitting [36]

5.5.4 Experiments

In our training process, we start from the initial AlexNet structure, which contains five convolutional layers and two fully connected layers. We generate a set of configurations based on this structure and train on the data through each model of configuration in this set. Among all these configurations, we use an adaptive learning rate and a rectifier (ReLU) activation function, with a random initialization of weights. We train the model by 2800 partial map images and use a validation set of 400 images to calculate the performance of each configuration. Table 5.5 shows how the convolutional neural network based on AlexNet performs in each configuration.

Among all these configurations, we select the best one containing 5 convolutional layers and 2 fully connected layers, with 30 epochs and 0.5 dropout rate. We also plot the learning curve of this configuration as Figure 5.17. In these three plots, x-axis represents the number of epoches while y-axis represents the value of accuracy, loss, and AUC.

As it can be seen from Figure 5.17, the model converges after about 15 epochs. After that, the accuracy and loss on validation data tend to be stable. With an accuracy of 93.269%, we can conclude that the model fits well on both training set and validation set.

Model structure (CL = Convolutional layer FC = Fully connected layer, PL = Pooling Layer)	Parameters (Not shown below if it's same as origin)	Accuracy(%)
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2		93.269%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2 + FC3	Neurons of FC3 = 4096	92.696%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1		92.656%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + CL6 + PL3 + FC1 + FC2		92.5%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + PL3 + FC1 + FC2		88.125%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2	Neurons of FC2 = 2048	91.181%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2	Neurons of FC2 = 1024	91.875%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2	Dropout = 0.8	90.903%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2	Dropout = 0.2	89.236%
CL1 + PL1 + CL2 + PL2 + CL3 + CL4 + CL5 + PL3 + FC1 + FC2	Kernel size of CL1 = 9	89.514%

Table 5.5: Performance of cnn model in different configurations

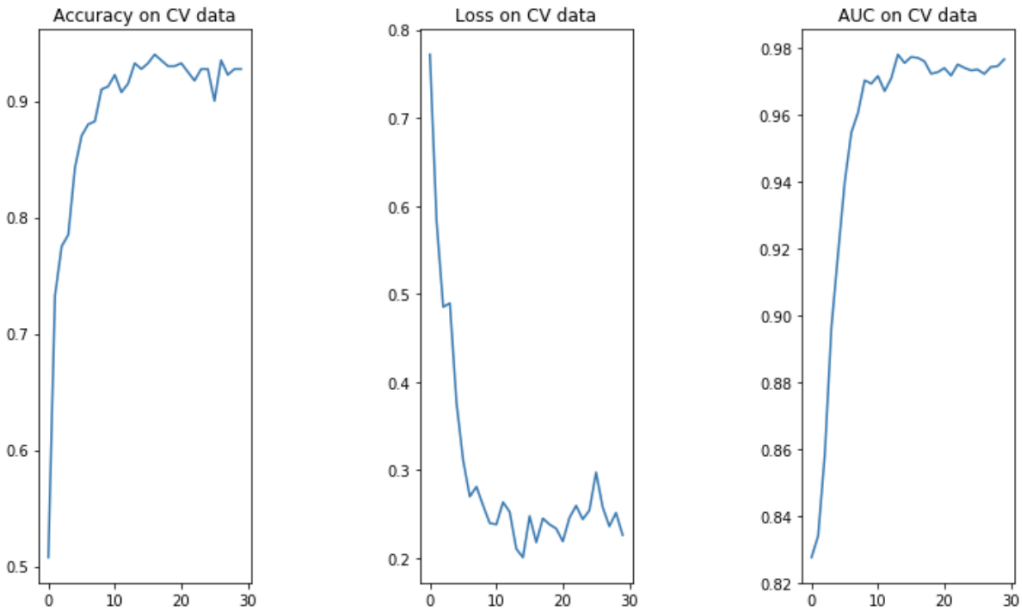


Figure 5.17: The learning curve of cnn model of the optimal configuration

Our test set contains 245 partial map images. The environments of these images are different from those of images in both training set and validation set, which means they are unknown to the robot. By feeding the test set to the model, we obtain an accuracy of 77.7%, with the confusion matrix shown in Table 5.6:

Origin/Prediction	True	False
True	0.33	0.18
False	0.03	0.46

Table 5.6: Confusion matrix of the optimal model

The accuracy on test set is not enough high, which means there could be a potential underfitting. However, as shown in the above confusion matrix, most of the errors occurs in True Negative (18%) instead of False Positive (3%). This means that our robot will never stop exploration before a map is completely explored, but will make some wrong prediction and continue to explore when a map is good enough. This result is actually reasonable in an practical point of view, since the cost of a non-completed observed map is much more than the time cost of over exploration.

5.6 Summary

In this chapter, we introduced some implementation details about our project. Methods like early stopping and dropout are implemented to avoid the risk of overfitting.

In the next chapter, we will discuss about the experimental results in offline test and online test.

Chapter 6

Experiment and Evaluation

In this chapter, the evaluation of the model we obtained would be executed. This evaluation includes two parts:

- Offline test: the test of the model without being integrated into the ROS system and performed on previously acquired metric maps. This also includes two sub-tests, the test on the validation data, which are the map information sampled on the environments used by the system for training, and the second test is on the test data, which are the data of unseen environments.
- Online test: the test of the model embedded in the whole exploration system, in an environment that has not been seen by the system.

6.1 Offline Test

Before we integrate our model into our robot system in ROS, we firstly test it offline. We select totally 8 environments for online test. For each environment, we randomly selects multiple runs in rosbag, redo the simulation in ROS, save map in every 90 seconds, and generate a set of partial observed maps. Then we sort them in alphanumerical order. Since images are named with format "environment_-run_timestamp.pgm', they are therefore sorted by timestamp for each run of each environment. Then we feed these images one by one to our convolutional model, and see how they behave in both validation set (seen environments) and test set (unseen environments).

The ideal behavior we expect is that at the early stages of exploration, the model tells that the current partial observed map is not completed, and the robot still needs do more exploration. At the middle stages of exploration at some timestamps, the model predicts that the current partial observed map is completed and the robot can early stop exploration, since the remaining unknown area can be reconstructed

correctly by layout reconstruction. At the late stages of exploration, the observed maps are of course almost completed, so the model should tell to early stop.

6.1.1 Validation Set (Seen Environments)

In validation set, all environments we select for evaluation are the ones our convolution model has seen, which means they has been used in either the training set or validation set. We feed these images to our convolutional model, and obtain the following results shown in Table 6.1.

Environment	Run	Total Time By Original Method	Total Time By Our Model	Improvements
1	1	4215	1434	65.98%
2	16	1804	284	84.26%
3	39	5415	4178	22.83%
4	38	1803	1803	None
5	30	3009	3009	None
6	17	1804	664	63.20%
7	43	1804	664	63.20%
8	25	1804	474	73.73%

Table 6.1: The performance of our model in validation set

Total time by original method is the original time the robot spends to explore and stop when the predicted amount of unexplored area is less than a specific threshold. Total time by our method is the exploration time when using our convolutional model as the early stopping criterion. And we calculate the improvement with the following equation:

$$SpeedUp = \frac{(TotalTimeByOriginalMethod - TotalTimeByOurModel)}{TotalTimeByOrignalMethod} \tag{6.1}$$

As we could see from the above table, in 75% of the environments, our model is able be save more than 50% of the time. And our model also works as expected. At first the outputs of our model are always False (which means do more exploration). From some timestamp, when the partial observed map is almost completed, the model outputs True (which means early stopping), and then keeps outputting True until the end.

6.1.2 Test Set (Unseen Environments)

In test set, all environments we select are the ones our convolution model has never seen, which means they are neither in the training set nor in the validation set. We feed these images to our convolutional model, and obtain the results shown in Table 6.2.

Environment	Run	Total Time By Original Method	Total Time By Our Model	Improvements
1	4	2406	2406	None
2	33	4212	1294	69.2%
3	37	3609	3609	None
4	9	6019	2864	52.41%
5	19	7825	3166	59.54%
6	21	3008	1392	53.72%
7	31	1804	1804	None
8	8	3610	3610	None

Table 6.2: The performance of our model in test set

As we could see from the above table, in 50% of the environments, our model is able be save more than 50% of the time. We show the example of the environment 2 in Figure 6.1.

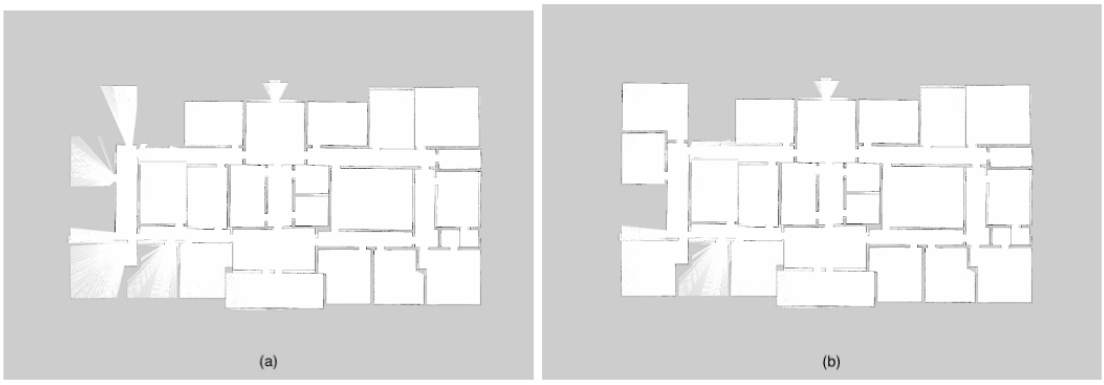


Figure 6.1: An example of early stopping in test set

Figure 6.1(b) is the partial observed map when the first time the model tells True for early stopping, while Figure 6.1(a) is the partial observed map at the timestamp before Figure 6.1(b). As we can infer from these two maps, our convolutional model extracts the geometrical features from the top left part and recognize these portions as the areas which still need more exploration. After those areas have been explored, as shown in Figure 6.1(b), most of the remaining unexplored areas

could be reconstructed correctly in layout reconstruction, shown in Figure 6.2 (still by learning the geometrical features by our convolution neural network model). Intuitively, this is just the correct timestamp for early stopping.

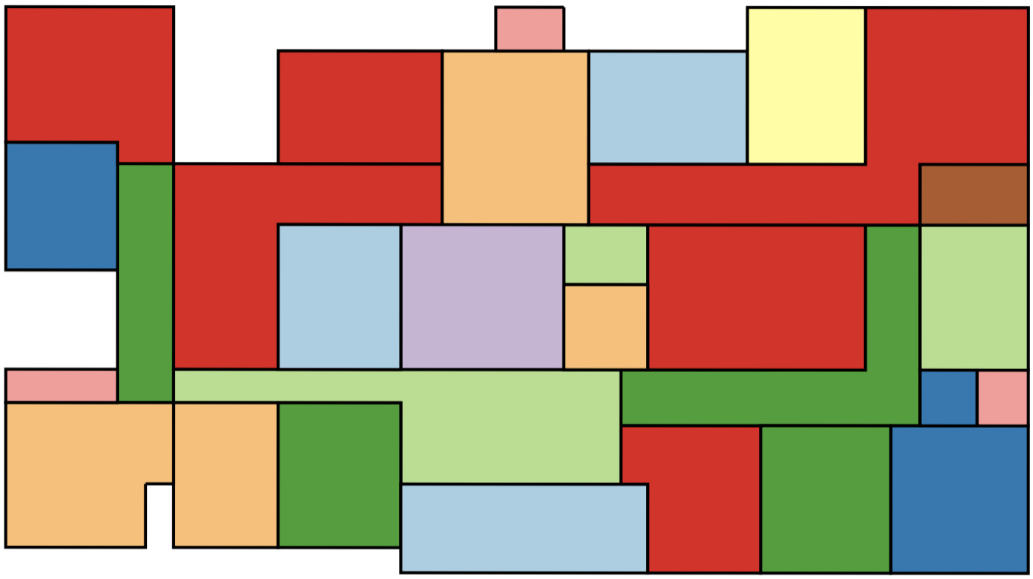


Figure 6.2: Reconstructed layout of partial map of environment 2.

Let's consider the environments which could not be predicted correctly in Figure 6.3.

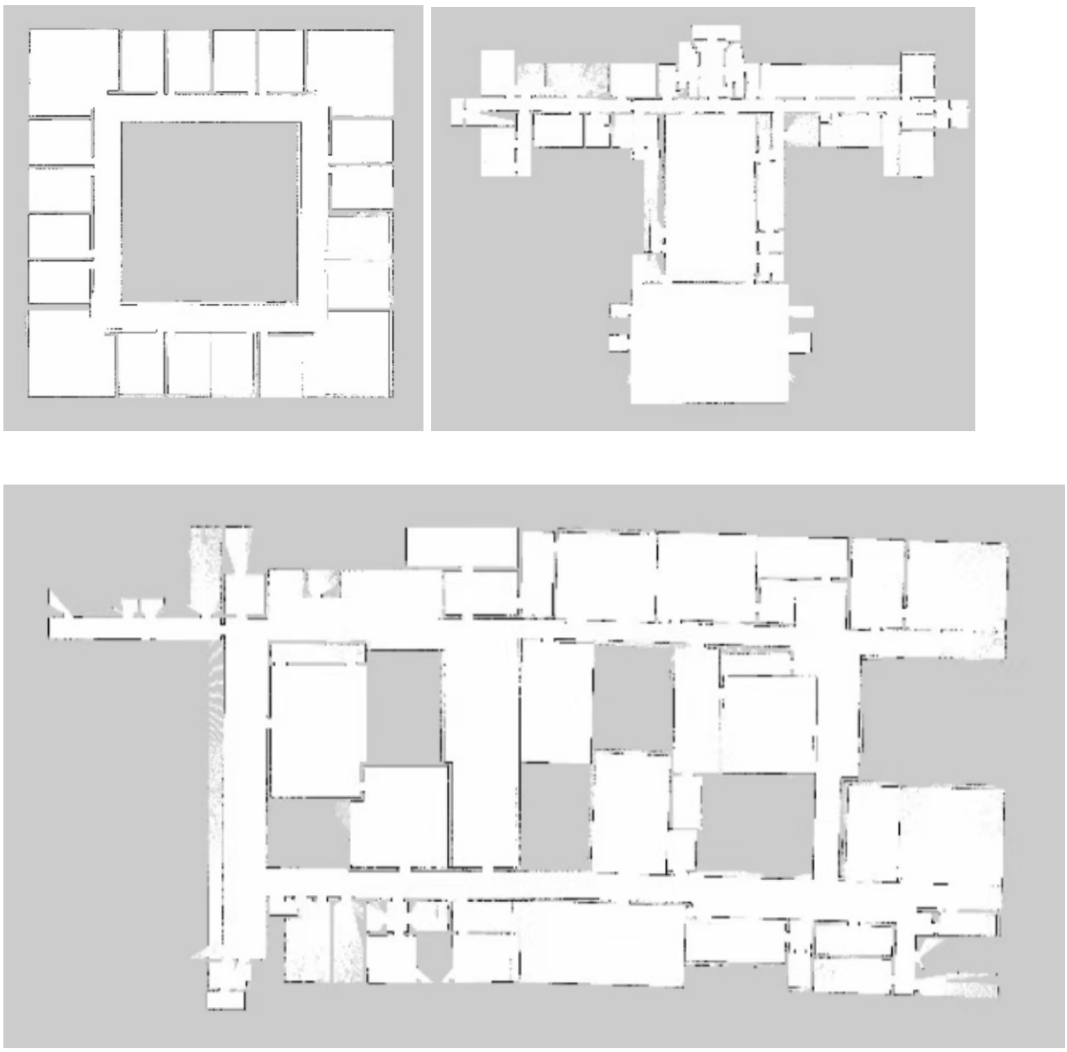


Figure 6.3: Environments in which early stopping perform badly.

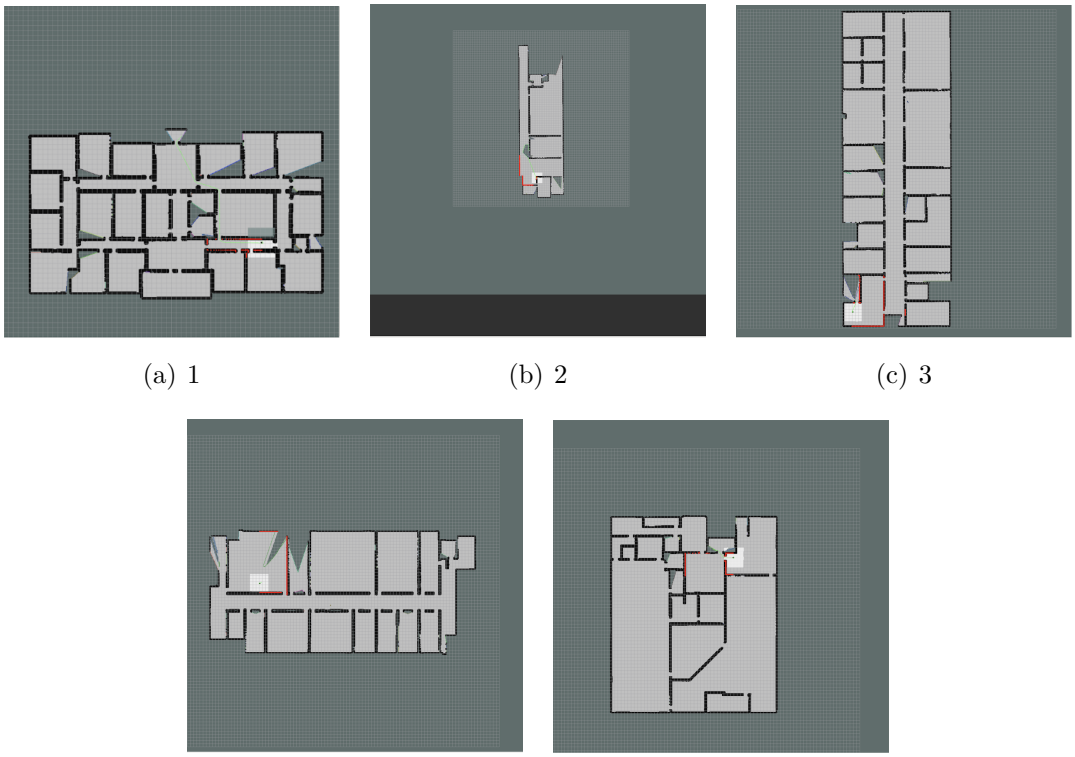
As shown in Figure 6.3, we see that these kinds of environments are usually big and complex. For the first environment (office_h), it has a hollow structure, however in our training set we don't have too many data of this specific structure. For the second and third environments (Neulengbach2_updated and Henderson_-high_school_updated), there are lots of rooms with different sizes and different shapes. In the third environment there are also multiple hollows in different portions. Therefore, we can conclude that our current model fits well on simple environments but is not guaranteed to work well in complex environments.

6.2 Online Test

In online test, we integrate our model into the robot system in ROS, and see how it behaves in exploration in ROS instead of feeding pre-generated testing images.

We do the integration by creating a module called 'predictEarlyStop' controlling the prediction of early stopping. Before each call to the layout reconstruction process, the partial observed map is post-processed into a 227 × 227 three channels images and passed to the module. The module feeds the post-processed image into the model and then publishes the result in a message to ROS node /navigation, which controls the navigation of the robot. If the result is True (early stopping), the robot stops and prints a stopping message including the current timestamp.

We run the simulation on 5 different environments. As shown in Figure 6.4, the robot stops when all remaining unexplored areas could be reconstructed correctly by layout reconstruction. It works in the same way in offline test. We obtain the performance table shown in Table 6.3. As a result, an average reduction of time cost about 34% is saved.



(a) 1 (b) 2 (c) 3

(d) 4 (e) 5

Figure 6.4: Online test on 5 different environments.

6.3 Summary

Environment	Total Time By Original Method	Total Time By Our Model	Improvements
1	2407	1470	38.9%
2	1805	1360	24.7%
3	4213	1823	56.7%
4	2406	1203	50.0%
5	3010	2997	0.4%

Table 6.3: The performance of our model in online test

In this chapter, we evaluated our model in both offline test and online test. First of all, we performed the offline evaluation both on validation set (seen environments) and test set (unseen environments). Both of them show a time cost reduction as we expected, with some limitations on specific environments which could be improved in future work. Then, we do the online test in ROS exploration. The robot stops at an early timestamp and saves 60% of the total exploration time, which further confirms the validity of our model.

Chapter 7

Conclusions

In this paper, we have presented a method that could shorten the exploration time of an autonomous mobile robot in an initially unknown environment by terminating the exploration process when the environment has been partially observed but the unobserved parts can be predicted reliably.

7.1 Conclusions

Experimental results show that our method performs well and could reduce the time spent on exploration of about 50% in many environments. This conclusion is consistent with the observation supported by data that the autonomous robot usually spends about half of the time on the exploration of small corners in the map.

However, in the previous chapter, we also observed that in some extremely large and complicate environments, the early stopping criterion may fail. To solve this problem, and increase the generalization over the maps, we should add more environments to the training set. So far we only have about 90 environments, and even though many data enhancement methods have been applied, this is still a small number. Better performance and accuracy could be expected, with a larger training set.

7.2 Future Works

Future work could concentrate on the improvements of the data set, providing more training environments. Our current training set mostly consists of environments with straight walls and symmetric rooms. However, in a relative large number of rooms encountered in the real world, the shape of the walls could be curved and with irregular shapes. If a more robust early stopping criterion is needed, we must

consider to add environments with different shapes into the training set.

Moreover, now we currently consider environments without any obstacle nor furniture. For environments with obstacles, we need to apply some algorithm to filter those obstacles. We may also have to re-train the model the environments with obstacles. These measures could help the early stopping criterion working better in the new environments.

Other interesting future directions include using a modified AlexNet, with a fully connected layer with input of other map information and context, like exploration time, areas covered, rooms covered, and so on. Moreover, the reconstruction method also directly influences the performance of our model, and different reconstruction algorithms could be considered to be combined with the model.

Appendix A

Tools

In this chapter, I would introduce the main tools we used.

A.1 ROS

ROS(Robot Operating System) is a robot software platform that provides operating system-like functions for heterogeneous computer clusters.

ROS provides some standard operating system services such as hardware abstraction, underlying device control, common function implementation, interprocess messages, and packet management. ROS is based on a graph-like architecture, in which processes at different nodes can accept, publish, and aggregate various information (such as sensing, control, state, planning, etc.). Currently ROS mainly supports Ubuntu. We used ros-kinetic in our project.

A.2 Tensorflow

TensorFlow is a symbolic mathematics system based on dataflow programming. It has a multi-tiered architecture that can be deployed on a variety of servers, PC terminals and web pages and supports GPU and TPU high-performance numerical computing. Therefore, it is widely used in the programming of various machine learning algorithms.

TensorFlow provides four different versions of the Python language: the tensorflow, the GPU-accelerated version (tensorflow-gpu), and their daily compiled versions (tf-nightly, tf-nightly-gpu). We used tensorflow-gpu in our training process.

Appendix B

Implementation

B.1 Multiprocess Partial Map Analysis

```
import time
import os
import glob
import cv2
import csv
import sys
import multiprocessing
from shutil import copy

import parameters as par
from floorplan import start_analysis
# ROS Object
from ros_object import FrontierObject as fo

# ------------------ Global variables
# PATH
INPUT_FOLDER = './data/INPUT/'
OUTPUT_FOLDER = './data/OUTPUT/'
INPUT_DATASET = 'NEW DATASET v2'
# CSV_FILEPATH = '/home/lc/Desktop/accuracy.csv'


# Define how score is computed based on bc and fc
def cal_score(map_bc, map_fc, gt_bc, gt_fc):
    score = (map_bc + map_fc + gt_bc + gt_fc) / 4.0

    return score

# Define job for each process
def job(work):
    map_full_path = work[0]
    parameter_obj = work[1]
    path_obj = work[2]
    frontierObject = work[3]

    map_full_name = map_full_path.split('/')[-1][:-4]

    # Extract map information
    map_name = map_full_name.split(' ')[0]
    run_name = map_full_name.split(' ')[1]
    timestamp = map_full_name.split(' ')[2]
    # print "Map: ", map_name
    # print "Run: ", run_name
    # print "Timestamp: ", timestamp

    # Create folder for storing layout image
    if par.DISEGNA or par.storeLayoutImage:
        SAVE_FOLDER = os.path.join(path_obj.OUTFOLDERS, path_obj.DATASET, map_full_name)
        path_obj.filepath = SAVE_FOLDER + '/'
        if not os.path.exists(SAVE_FOLDER):
            os.makedirs(SAVE_FOLDER)

```

```
53          im = cv2.imread(map_full_path)
54          path_obj.nome_gt = path_obj.INFOLDERS + 'XMLs/' + map_name + '.xml'
55          try:
56              analysis_data = start_analysis(parameter_obj, path_obj, im, frontierObject)
57          except:
58              return (False, map_full_name)
59
60          # print "Map BC: ", analysis_data['Map bc']
61          # print "Map FC: ", analysis_data['Map fc']
62          # print "GT BC: ", analysis_data['GT bc']
63          # print "GT FC: ", analysis_data['GT fc']
64          score = cal_score(analysis_data['Map bc'], analysis_data['Map fc'], analysis_data['GT bc'],
                 analysis_data['GT fc'])
65          # print "Score: ", score
66
67          analysis_data['Map Name'] = map_name
68          analysis_data['Run'] = run_name
69          analysis_data['Timestamp'] = timestamp
70          analysis_data['Score'] = score
71
72          return (True, analysis_data)
73
74      if __name__ == '__main__':
75          start_time = time.time()
76          print "Start!"
77          print "Dataset: ", INPUT_DATASET
78
79          # Create output folder
80          if not os.path.exists(os.path.join(OUTPUT_FOLDER, INPUT_DATASET)):
81              os.makedirs(os.path.join(OUTPUT_FOLDER, INPUT_DATASET))
82
83          # Maps already processed (where to start)
84          csv_filepath = os.path.join(OUTPUT_FOLDER, INPUT_DATASET, 'accuracy.csv')
85          maps_done = []
86          start_from_zero = False
87          if os.path.exists(csv_filepath):
88              with open(csv_filepath, 'r') as csv_file:
89                  csv_reader = csv.reader(csv_file)
90
91                  next(csv_reader)
92                  for row in csv_reader:
93                      maps_full_name = ' '.join(row[:3])
94
95                      maps_done.append(maps_full_name + '.png')
96          else:
97              start_from_zero = True
98          print "Maps done: ", len(maps_done)
99
100         # CSV writer
101         fieldnames = ['Map Name', 'Run', 'Timestamp', 'Map bc', 'Map fc', 'GT bc', 'GT fc', 'Score', 'Map
                 complete rooms',
102         'Map partial rooms', 'Map rooms', 'GT rooms', 'Cells', 'GT area', 'Frontier', 'Label']
103         csv_file = open(os.path.join(OUTPUT_FOLDER, INPUT_DATASET, 'accuracy.csv'), 'a+')
104         csv_writer = csv.DictWriter(csv_file, fieldnames)
105         if start_from_zero:
106             csv_writer.writeheader()
107
108         # Require parameters
109         parameter_obj, path_obj = par.Parameter_obj(), par.Path_obj()
110         path_obj.INFOLDERS = INPUT_FOLDER
111         path_obj.OUTFOLDERS = OUTPUT_FOLDER
112         path_obj.DATASET = INPUT_DATASET
113         frontierObject = fo.FrontierObject(False,[],[],[],[],[])
114
115         # Multi processing part
116         maps_all = os.listdir(os.path.join(INPUT_FOLDER, INPUT_DATASET))
117         maps_notdone = list(set(maps_all) - set(maps_done))
118         print "Maps not done: ", len(maps_notdone)
119
120         maps_notdone = [os.path.join(INPUT_FOLDER, INPUT_DATASET, m) for m in maps_notdone]
121         works = [[m, parameter_obj, path_obj, frontierObject] for m in maps_notdone]
122         works = works
123
124         cores = multiprocessing.cpu_count() - 1
125         pool = multiprocessing.Pool(processes=cores)
126
127         cnt = 1
128         for analysis_data in pool.imap_unordered(job, works):
129             sys.stdout.write('done %d/%d\r' % (cnt, len(works)))
130             sys.stdout.flush()
```

```
            cnt += 1

            if analysis_data[0]:
                csv_writer.writerow(analysis_data[1])
                csv_file.flush()
            else:
                print "Error in processing map: ", analysis_data[1]

        csv_file.close()

        end_time = time.time()
        print "Total time: ", end_time - start_time
        print "Done!"
```

B.2 Labeling

```
import os
import csv
from PIL import Image
from shutil import copy

# Path
INPUT_FOLDER = './data/INPUT/'
OUTPUT_FOLDER = './data/OUTPUT/'
INPUT_DATASET = 'NEW DATASET v2'
CSV_FILEPATH = './data/OUTPUT/NEW DATASET v2/accuracy.csv'

# Threshold for labelling
# Score greater than threshold is labelled TRUE
# Score less than threshold is labelled FALSE
THRESHOLD_GT_BC = 83.831
THRESHOLD_GT_FC = 67.873
THRESHOLD_DIFF_ROOMS = 4.0

# Label the image by analysis data
# Here we use a decision tree to implement that
def label(analysis_data):
    gt_bc = float(analysis_data['GT bc'])
    gt_fc = float(analysis_data['GT fc'])
    diff_rooms = int(analysis_data['GT rooms']) - int(analysis_data['Map rooms'])

    if gt_fc <= THRESHOLD_GT_FC:
        return False
    elif gt_bc > THRESHOLD_GT_BC:
        return True
    elif diff_rooms <= THRESHOLD_DIFF_ROOMS:
        return True

    return False



if __name__ == "__main__":
    dataset_path = os.path.join(INPUT_FOLDER, INPUT_DATASET)
    output_path = os.path.join(OUTPUT_FOLDER, INPUT_DATASET + ' LABEL')

    # Create label folders
    true_folder = os.path.join(output_path, 'True')
    if not os.path.exists(true_folder):
        os.makedirs(true_folder)
    false_folder = os.path.join(output_path, 'False')
    if not os.path.exists(false_folder):
        os.makedirs(false_folder)

    with open(CSV_FILEPATH, 'r') as csv_file:
        csv_reader = csv.DictReader(csv_file)

        for index, row in enumerate(csv_reader):
            if index == 0:
                continue

            map_name = row['Map Name'] + ' ' + row['Run'] + ' ' + row['Timestamp'] + '.png'
            map_label = label(row)
            print str(index) + ' ------------ ' + map_name
            print 'Label: ' + str(map_label)
```

```
60
61                  src = os.path.join(dataset_path, map_name)
62                  if map_label:
63                      dst = os.path.join(true_folder, map_name)
64                  else:
65                      dst = os.path.join(false_folder, map_name)
66                  copy(src, dst)
```

B.3 Model Training

```
1     # Import libraries
2     import numpy as np
3     import os
4     from PIL import Image, ImageOps
5     import PIL
6     import cv2
7
8     import sys
9     import datetime
10    from sklearn.utils import shuffle
11
12
13    import cv2
14    import matplotlib.pyplot as plt
15    import tensorflow as tf
16    import pandas as pd
17    from sklearn.metrics import roc_auc_score
18
19    original_train_data = np.load(os.path.join(os.getcwd(),'Dataset v2/training_set.npy'))
20    test_data = np.load(os.path.join(os.getcwd(),'Dataset v2/test_set.npy'))
21    print("the shape of train_data is: \t",(original_train_data.shape))
22    print("the shape of test_data is: \t",(test_data.shape))
23
24    # Split the train data in train set, validation set and test set
25    validation_data = original_train_data[2400:]
26    train_data = original_train_data[:2400]
27
28    X = np.array([i[0] for i in train_data])
29    Y = np.array([i[1] for i in train_data])
30
31    v_x = np.array([i[0] for i in validation_data])
32    v_y = np.array([i[1] for i in validation_data])
33
34    t_x = np.array([i[0] for i in test_data])
35    t_y = np.array([i[1] for i in test_data])
36
37    # Parameters
38    height = 227
39    width = 227
40    channel = 1
41
42    epochs = 30
43    batch = 8
44
45    steps = len(train_data)
46    remaining = steps % batch
47
48    validating_size = 40
49    nodes_fc1 = 4096
50    nodes_fc2 = 4096
51    output_classes = 2
52
53    IMG_SIZE_ALEXNET = 227
54
55    TRAIN_DIR = os.getcwd()
56
57    # Reset Calculate Graph
58    tf.reset_default_graph()
59
60    # Define Placeholder for a 3 Channel IMAGE
61    x = tf.placeholder(tf.float32,shape=[None,height,width,channel],name='x')
62    # x = tf.placeholder(tf.float32,shape=[None,IMG_SIZE_ALEXNET,IMG_SIZE_ALEXNET,1])  ///TBD///
63    y = tf.placeholder(tf.float32,shape=[None,output_classes],name='y')
64
65    #------------CNN Layer1------------
```

```
    # 3  channel input ///TBD///
    # 96 channel output 55*55 each
    #w1 = tf.Variable(tf.truncated_normal([11,11,channel,96], stddev=0.01))
    w1 = tf.Variable(tf.truncated_normal([11,11,channel,96], stddev=0.01))
    # w1 = tf.Variable(tf.truncated_normal([11,11,1,96], stddev=0.01))   ///TBD///
    b1 = tf.Variable(tf.constant(0.0, shape=[[11,11,channel,96][3]]))

    output1 = tf.nn.conv2d(x,w1,strides = [1,4,4,1], padding = 'VALID')
    output1 = output1+b1
    output1 = tf.nn.relu(output1)

    #------------Pooling Layer1------------
    # 96 channel input 55*55 each
    # 96 channel output 27*27 each
    output1 = tf.nn.max_pool(output1, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1], padding='VALID')

    #------------CNN Layer2------------
    # 96 channel input 27*27 each
    # 256 channel output 27*27 each
    w2 = tf.Variable(tf.truncated_normal([5,5,96,256], stddev=0.01))
    b2 = tf.Variable(tf.constant(1.0, shape=[[5,5,96,256][3]]))

    output2 = tf.nn.conv2d(output1, w2,strides=[1, 1, 1, 1], padding='SAME')
    output2 = output2 + b2
    output2 = tf.nn.relu(output2)

    #------------Pooling Layer2------------
    # 256 channel input 27*27 each
    # 256 channel output 13*13 each
    output2 = tf.nn.max_pool(output2, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1], padding='VALID')

    #------------CNN Layer3------------
    # 256 channel input 13*13 each
    # 384 channel output 13*13 each
    w3 = tf.Variable(tf.truncated_normal([3, 3, 256, 384], stddev=0.01))
    b3 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 256, 384][3]]))

    output3 = tf.nn.conv2d(output2, w3,strides=[1, 1, 1, 1], padding='SAME')
    output3 = output3 + b3
    output3 = tf.nn.relu(output3)

    #------------CNN Layer4------------
    # 384 channel input 13*13 each
    # 384 channel output 13*13 each
    w4 = tf.Variable(tf.truncated_normal([3, 3, 384, 384], stddev=0.01))
    b4 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 384, 384][3]]))

    output4 = tf.nn.conv2d(output3, w4,strides=[1, 1, 1, 1], padding='SAME')
    output4 = output4 + b4
    output4 = tf.nn.relu(output4)

    #------------CNN Layer5------------
    # 384 channel input 13*13 each
    # 256 channel output 13*13 each
    w5 = tf.Variable(tf.truncated_normal([3, 3, 384, 256], stddev=0.01))
    b5 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 384, 256][3]]))

    output5 = tf.nn.conv2d(output4, w5,strides=[1, 1, 1, 1], padding='SAME')
    output5 = output5 + b5
    output5 = tf.nn.relu(output5)

    #------------Pooling Layer3------------
    # 256 channel output 13*13 each
    # 256 channel output 6*6 each
    output5 = tf.nn.max_pool(output5, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1], padding='VALID')

    #------------Flatten------------
    # 256 channel input 6*6 each
    # output a vector or 6*6*256
    flattened = tf.reshape(output5,[-1,6*6*256])

    #------------Fully connected Layer1------------
    # input_size = int(flattened.get_shape()[1])=6*6*256
    # output neural nodes: nodes1_fc1
    input_size = int(flattened.shape[1])
    w1_fc = tf.Variable(tf.truncated_normal([input_size, nodes_fc1], stddev=0.01))
    b1_fc = tf.Variable(tf.constant(1.0, shape=[nodes_fc1]))

    output_fc1 = tf.matmul(flattened, w1_fc) + b1_fc
    output_fc1 = tf.nn.relu(output_fc1)
```

```
146
147     #------------Dropout Layer1------------
148     hold_prob1 = tf.placeholder(tf.float32,name = 'hold_prob1')
149     # output_fc1 = tf.nn.dropout(output_fc1,rate=1-hold_prob1)
150     output_fc1 = tf.nn.dropout(output_fc1,keep_prob=hold_prob1)
151     # Here we should also use rate  = 1-keep_prob = 1-hold_prob
152
153
154     #------------Fully connected Layer2------------
155     # input neurons: nodes1_fc1
156     # output neurons: nodes1_fc2
157     w2_fc = tf.Variable(tf.truncated_normal([nodes_fc1, nodes_fc2], stddev=0.01))
158     b2_fc = tf.Variable( tf.constant(1.0, shape=[nodes_fc2]))
159
160     output_fc2 = tf.matmul(output_fc1, w2_fc) + b2_fc
161     output_fc2 = tf.nn.relu(output_fc2)
162
163     #------------Dropout Layer2------------
164     hold_prob2 = tf.placeholder(tf.float32, name = 'hold_prob2')
165     # output_fc2 = tf.nn.dropout(output_fc2,rate=1-hold_prob2)
166     output_fc2 = tf.nn.dropout(output_fc2,keep_prob=hold_prob2)
167
168
169     #------------Fully Connected Layer 3------------
170     w3_fc = tf.Variable(tf.truncated_normal([nodes_fc2,output_classes], stddev=0.01))
171     b3_fc = tf.Variable(tf.constant(1.0, shape=[output_classes]))
172
173     prediction = tf.matmul(output_fc2, w3_fc)
174     prediction = tf.add(prediction, b3_fc, name = "op_to_predict")
175
176     # REMEMBER: In the output layer, we don't apply activate function, and we get a 2 dimension vector of y
177
178     #Defining loss function
179     cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y,logits=prediction))
180     #Define Loss function with specific confusion matrix ///TBD///
181
182     #Define objective
183     train = tf.train.AdamOptimizer(learning_rate=0.00001).minimize(cross_entropy)
184
185     #Define Accuracy
186     matches = tf.equal(tf.argmax(prediction,1),tf.argmax(y,1))
187     acc = tf.reduce_mean(tf.cast(matches,tf.float32))
188
189     #Global Initialization
190     init = tf.global_variables_initializer()
191
192     #Starting Empty lists to keep results
193     acc_list = []
194     auc_list = []
195     loss_list = []
196     saver = tf.train.Saver()
197
198     # GPU Training
199     config = tf.ConfigProto(allow_soft_placement=True)
200     config.gpu_options.allow_growth = True
201     config.gpu_options.allocator_type = 'BFC'
202
203     currentDT = datetime.datetime.now()
204     print("------------Strat to Train------------")
205     print(str(currentDT))
206
207
208     with tf.Session(config=config) as sess:
209         sess.run(init)
210
211         # This is the training part
212         for i in range(epochs):
213             for j in range(0,steps-remaining,batch):
214                 #Feeding step_size-amount data with 0.5 keeping probabilities on DROPOUT LAYERS
215                 _,c = sess.run([train,cross_entropy],
216                 feed_dict={x:X[j:j+batch] , y:Y[j:j+batch],hold_prob1:0.5,hold_prob2:0.5})
217
218
219             #Writing for loop to calculate test statistics. GTX 1050 isn't able to calculate all test data.
220             cv_auc_list = []
221             cv_acc_list = []
222             cv_loss_list = []
223             # Validation after one epoch
224             for v in range(0,len(v_x)-int(len(v_x) % validating_size),validating_size):
225                 acc_on_cv,loss_on_cv,preds = sess.run([acc,cross_entropy,tf.nn.softmax(prediction)],
```

```
                feed_dict={x:v_x[v:v+validating_size] ,y:v_y[v:v+validating_size] ,hold_prob1:1.0,hold_prob2
                     :1.0})

                auc_on_cv = roc_auc_score(v_y[v:v+validating_size],preds)
                cv_acc_list.append(acc_on_cv)
                cv_auc_list.append(auc_on_cv)
                cv_loss_list.append(loss_on_cv)
            acc_cv_ = round(np.mean(cv_acc_list),5)
            auc_cv_ = round(np.mean(cv_auc_list),5)
            loss_cv_ = round(np.mean(cv_loss_list),5)
            acc_list.append(acc_cv_)
            auc_list.append(auc_cv_)
            loss_list.append(loss_cv_)
            print("------------Epoch %i Finished------------" %i)
            currentDT = datetime.datetime.now()
            print(str(currentDT))
            print("Epoch:",i+1,"Accuracy:",acc_cv_,"Loss:",loss_cv_ ,"AUC:",auc_cv_)

        print("------------Training Stage Finished------------")

        # This is the validation part
        saver.save(sess, os.path.join(os.getcwd(),"CNN_MC_best.ckpt"))

    f,ax=plt.subplots(1,3,figsize=(12,3))
    pd.Series(acc_list).plot(kind='line',title='Accuracy on CV data',ax=ax[0])
    pd.Series(loss_list).plot(kind='line',figsize=(12,7),title='Loss on CV data',ax=ax[1])
    pd.Series(auc_list).plot(kind='line',figsize=(12,7),title='AUC on CV data',ax=ax[2])
    plt.subplots_adjust(wspace=0.8)
    ax[0].set_title('Accuracy on CV data')
    ax[1].set_title('Loss on CV data')
    ax[2].set_title('AUC on CV data')
    plt.show()
```

B.4 Model Integration

```
import os

import numpy as np
import cv2
import tensorflow as tf
from PIL import Image

class tfModel():

    def __init__(self):
        self.height = 227
        self.width = 227
        self.channel = 1
        self.nodes_fc1 = 4096
        self.nodes_fc2 = 4096
        self.output_classes = 2

        self.tf, self.x, self.prediction, self.hold_prob1, self.hold_prob2 = self.defineModel()
        self.saver = self.tf.train.Saver()
        self.session = tf.Session()
        self.saver.restore(self.session, os.path.join(os.environ['HOME'], "
             wholebuildingpredictiveexploration/src/floorplan_analyzer/src/data/MODEL/CNN_MC_best.ckpt"))

        self.step = 100
        self.size = 4000
        self.fix_width = 227

    def defineModel(self):
        # Reset Calculate Graph
        tf.reset_default_graph()

        # Define Placeholder for a 3 Channel IMAGE
        x = tf.placeholder(tf.float32,shape=[None,self.height,self.width,self.channel],name='x')
        # x = tf.placeholder(tf.float32,shape=[None,IMG_SIZE_ALEXNET,IMG_SIZE_ALEXNET,1])  ///TBD///
        y = tf.placeholder(tf.float32,shape=[None,self.output_classes],name='y')

        #------------CNN Layer1------------
        # 3  channel input ///TBD///
        # 96 channel output 55*55 each
        #w1 = tf.Variable(tf.truncated_normal([11,11,channel,96], stddev=0.01))
```

```
        w1 = tf.Variable(tf.truncated_normal([11,11,self.channel,96], stddev=0.01))
        # w1 = tf.Variable(tf.truncated_normal([11,11,1,96], stddev=0.01))   ///TBD///
        b1 = tf.Variable(tf.constant(0.0, shape=[[11,11,self.channel,96][3]]))

        output1 = tf.nn.conv2d(x,w1,strides = [1,4,4,1], padding = 'VALID')
        output1 = output1+b1
        output1 = tf.nn.relu(output1)

        #------------Pooling Layer1------------
        # 96 channel input 55*55 each
        # 96 channel output 27*27 each
        output1 = tf.nn.max_pool(output1, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1], padding='VALID')

        #------------CNN Layer2------------
        # 96 channel input 27*27 each
        # 256 channel output 27*27 each
        w2 = tf.Variable(tf.truncated_normal([5,5,96,256], stddev=0.01))
        b2 = tf.Variable(tf.constant(1.0, shape=[[5,5,96,256][3]]))

        output2 = tf.nn.conv2d(output1, w2,strides=[1, 1, 1, 1], padding='SAME')
        output2 = output2 + b2
        output2 = tf.nn.relu(output2)

        #------------Pooling Layer2------------
        # 256 channel input 27*27 each
        # 256 channel output 13*13 each
        output2 = tf.nn.max_pool(output2, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1], padding='VALID')

        #------------CNN Layer3------------
        # 256 channel input 13*13 each
        # 384 channel output 13*13 each
        w3 = tf.Variable(tf.truncated_normal([3, 3, 256, 384], stddev=0.01))
        b3 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 256, 384][3]]))

        output3 = tf.nn.conv2d(output2, w3,strides=[1, 1, 1, 1], padding='SAME')
        output3 = output3 + b3
        output3 = tf.nn.relu(output3)

        #------------CNN Layer4------------
        # 384 channel input 13*13 each
        # 384 channel output 13*13 each
        w4 = tf.Variable(tf.truncated_normal([3, 3, 384, 384], stddev=0.01))
        b4 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 384, 384][3]]))

        output4 = tf.nn.conv2d(output3, w4,strides=[1, 1, 1, 1], padding='SAME')
        output4 = output4 + b4
        output4 = tf.nn.relu(output4)

        #------------CNN Layer5------------
        # 384 channel input 13*13 each
        # 256 channel output 13*13 each
        w5 = tf.Variable(tf.truncated_normal([3, 3, 384, 256], stddev=0.01))
        b5 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 384, 256][3]]))

        output5 = tf.nn.conv2d(output4, w5,strides=[1, 1, 1, 1], padding='SAME')
        output5 = output5 + b5
        output5 = tf.nn.relu(output5)

        #------------Pooling Layer3------------
        # 256 channel output 13*13 each
        # 256 channel output 6*6 each
        output5 = tf.nn.max_pool(output5, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1], padding='VALID')

        #------------Flatten------------
        # 256 channel input 6*6 each
        # output a vector or 6*6*256
        flattened = tf.reshape(output5,[-1,6*6*256])

        #------------Fully connected Layer1------------
        # input_size = int(flattened.get_shape()[1])=6*6*256
        # output neural nodes: nodes1_fc1
        input_size = int(flattened.shape[1])
        w1_fc = tf.Variable(tf.truncated_normal([input_size, self.nodes_fc1], stddev=0.01))
        b1_fc = tf.Variable(tf.constant(1.0, shape=[self.nodes_fc1]))

        output_fc1 = tf.matmul(flattened, w1_fc) + b1_fc
        output_fc1 = tf.nn.relu(output_fc1)

        #------------Dropout Layer1------------
        hold_prob1 = tf.placeholder(tf.float32,name = 'hold_prob1')
```

```
            # output_fc1 = tf.nn.dropout(output_fc1,rate=1-hold_prob1)
            output_fc1 = tf.nn.dropout(output_fc1,keep_prob=hold_prob1)
            # Here we should also use rate  = 1-keep_prob = 1-hold_prob


            #------------Fully connected Layer2------------
            # input neurons: nodes1_fc1
            # output neurons: nodes1_fc2
            w2_fc = tf.Variable(tf.truncated_normal([self.nodes_fc1, self.nodes_fc2], stddev=0.01))
            b2_fc = tf.Variable( tf.constant(1.0, shape=[self.nodes_fc2]))

            output_fc2 = tf.matmul(output_fc1, w2_fc) + b2_fc
            output_fc2 = tf.nn.relu(output_fc2)

            #------------Dropout Layer2------------
            hold_prob2 = tf.placeholder(tf.float32, name = 'hold_prob2')
            # output_fc2 = tf.nn.dropout(output_fc2,rate=1-hold_prob2)
            output_fc2 = tf.nn.dropout(output_fc2,keep_prob=hold_prob2)


            #------------Fully Connected Layer 3------------
            w3_fc = tf.Variable(tf.truncated_normal([self.nodes_fc2,self.output_classes], stddev=0.01))
            b3_fc = tf.Variable(tf.constant(1.0, shape=[self.output_classes]))

            prediction = tf.matmul(output_fc2, w3_fc)
            prediction = tf.add(prediction, b3_fc, name = "op_to_predict")

            # REMEMBER: In the output layer, we don't apply activate function, and we get a 2 dimension
                vector of y

            #Defining loss function
            cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y,logits=
                prediction))
            #Define Loss function with specific confusion matrix ///TBD///

            #Define objective
            train = tf.train.AdamOptimizer(learning_rate=0.00001).minimize(cross_entropy)

            #Define Accuracy
            matches = tf.equal(tf.argmax(prediction,1),tf.argmax(y,1))
            acc = tf.reduce_mean(tf.cast(matches,tf.float32))

            #Global Initialization
            init = tf.global_variables_initializer()

            return tf, x, prediction, hold_prob1, hold_prob2


        def predict(self, im):
            im = self.processImage(im)
            im = im.reshape((self.width, self.height, 1))
            im = np.array([im])

            k = self.session.run([tf.nn.softmax(self.prediction)], feed_dict={self.x:im , self.hold_prob1:1,
                self.hold_prob2:1})

            r = np.round(k, 3).argmax()
            if r == 0:
                return True
            else:
                return False

        def processImage(self, im):
            self.size = im.shape[0]

            index1 = 0
            index2 = self.size -1

            flag = True
            while flag == True and index1<self.size/2:
                index1+=self.step
                for i in range(0,self.size):
                    if im[index1][i]!=205:
                        flag = False
                        index1-=self.step*2
                        if index1 < 0:
                            index1 = 0
                        break
                    elif im[i][index1]!=205:
                        flag = False
```

```
                        index1-=self.step*2
                        if index1 < 0:
                            index1 = 0
                        break
                    elif im[self.size-1-index1][i]!=205:
                        flag = False
                        index1-=self.step*2
                        if index1 < 0:
                            index1 = 0
                        break
                    elif im[i][self.size-1-index1]!=205:
                        flag = False
                        index1-=self.step*2
                        if index1 < 0:
                            index1 = 0
                        break

            im = im[index1:self.size-1-index1, index1:self.size-1-index1]

            # Calculate height
            im = Image.fromarray(im)
            wpercent = self.fix_width / float(im.size[0])
            height = int(im.size[1] * wpercent)
            im = im.resize((self.fix_width, height), Image.ANTIALIAS)

            im = np.array(im)

            return im
```

Bibliography

[1] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *arXiv e-prints* (Mar. 2018). arXiv: 1803.08375 (cit. on p. 41).

[2] Baike contributors. *Convolutional Neural Network — Baike*. [Online; accessed 10-Aug-2019]. 2019. URL: baike.baidu.com/item/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C (cit. on p. 10).

[3] Nicola Basilico and Francesco Amigoni. "Exploration strategies based on multi-criteria decision making for searching environments in rescue operations". In: *Autonomous Robots* 31.4 (2011), p. 401 (cit. on p. 5).

[4] Richard Bormann et al. "Room segmentation: Survey, implementation, and analysis". In: *Proceedings of the IEEE conference on robotics and automation (ICRA)*. Stockholm, Sweden, 2016, pp. 1019–1026 (cit. on p. 8).

[5] Steve Branson et al. "Bird species categorization using pose normalized deep convolutional nets". In: *arXiv preprint arXiv:1406.2952* (2014) (cit. on p. 13).

[6] Jeffrey A Caley, Nicholas RJ Lawrance, and Geoffrey A Hollinger. "Deep learning of structured environments for robot search". In: *Autonomous Robots* 43.7 (2019), pp. 1695–1714 (cit. on p. 14).

[7] Petr Cintula, Christian G. Fermüller, and Carles Noguera. "Fuzzy Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2017. Metaphysics Research Lab, Stanford University, 2017 (cit. on p. 64).

[8] Johannes Doellinger, Markus Spies, and Wolfram Burgard. "Predicting occupancy distributions of walking humans with convolutional neural networks". In: *Proceedings of the IEEE conference on Robotics and Automation Letters* 3.3 (2018), pp. 1522–1528 (cit. on p. 14).

[9] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016) (cit. on p. 12).

[10] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. "Image processing with neural networks—a review". In: *Pattern recognition* 35.10 (2002), pp. 2279–2301 (cit. on p. 13).

[11] Luca Fochetta. "Use of predicted layouts of indoor environments to improve exploration strategies for autonomous mobile robots". 2019 (cit. on pp. 1, 4, 6, 17, 23, 24).

[12] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Columbus, OH, USA, 2014, pp. 580–587 (cit. on p. 14).

[13] Héctor H González-Banos and Jean-Claude Latombe. "Navigation strategies for exploring indoor environments". In: *The International Journal of Robotics Research* 21.10-11 (2002), pp. 829–848 (cit. on pp. 1, 4, 5).

[14] Ian J Goodfellow et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks". In: *arXiv preprint arXiv:1312.6082* (2013) (cit. on p. 14).

[15] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377 (cit. on pp. 9, 11, 12, 14).

[16] Pan He et al. "Reading scene text in deep convolutional sequences". In: *Thirtieth AAAI conference on artificial intelligence*. Phoenix, Arizona, 2016, pp. 3501–3508 (cit. on p. 14).

[17] Jeff Heaton. "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning". In: *Genetic Programming Evolvable Machines* 19.1-2 (2018), s10710–017–9314–z (cit. on pp. 9, 11, 12).

[18] Jonathan Krause et al. "Learning features and parts for fine-grained recognition". In: *Proceedings of the IEEE conference on Pattern Recognition*. Stockholm, Sweden, 2014, pp. 26–33 (cit. on p. 13).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. Stateline, Nevada, USA, 2012, pp. 1097–1105 (cit. on pp. 39–41, 43).

[20] Christos Kyrkou et al. "DroNet: Efficient convolutional neural network detector for real-time UAV applications". In: *Proceedings of the IEEE conference on Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden, Germany, 2018, pp. 967–972 (cit. on p. 14).

[21] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995 (cit. on p. 9).

[22] Ziyuan Liu and Georg von Wichert. "A generalizable knowledge framework for semantic indoor mapping based on Markov logic networks and data driven MCMC". In: *Future Generation Computer Systems* 36 (2014), pp. 42–56 (cit. on p. 1).

[23] Matteo Luperto and Francesco Amigoni. "Extracting Structure of Buildings Using Layout Reconstruction". In: *International Conference on Intelligent Autonomous Systems*. Vol. 15. Springer. Milan, Italy, 2018, pp. 652–667 (cit. on pp. 5, 8, 47).

[24] Matteo Luperto, Valerio Arcerito, and Francesco Amigoni. "Predicting the Layout of Partially Observed Rooms from Grid Maps". In: *Proceedings of the IEEE conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada, 2019, pp. 6898–6904 (cit. on pp. 1, 5, 23, 47).

[25] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *Proceedings of the IEEE conference on Intelligent Robots and Systems (IROS)*. Hamburg, Germany, 2015, pp. 922–928 (cit. on p. 14).

[26] Claudio Mura et al. "Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts". In: *Computers & Graphics* 44 (2014), pp. 20–32 (cit. on p. 7).

[27] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Haifa, Israel, 2010, pp. 807–814 (cit. on p. 40).

[28] Sarang Narkhede. *Understanding AUC - ROC Curve*. [Online; accessed 11-Aug-2019]. 2018. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (cit. on p. 73).

[29] Andrew Ng. *Convolutional Neural Network — Coursera*. [Online; accessed 10-Aug-2019]. 2019. URL: en.coursera.org/learn/convolutional-neural-networks (cit. on pp. 10, 11, 13).

[30] Steven J Nowlan and John C Platt. "A convolutional neural network hand tracker". In: Denver, Colorado, 1995, pp. 901–908 (cit. on p. 14).

[31] *Powering the world's robots*. URL: https://www.ros.org/ (cit. on pp. 26, 47).

[32] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, 2016, pp. 779–788 (cit. on p. 14).

[33] Nitish Srivastava and Ruslan R Salakhutdinov. "Discriminative transfer learning with tree-based priors". In: *Advances in neural information processing systems*. Stateline, Nevada, USA, 2013, pp. 2094–2102 (cit. on p. 13).

[34] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. "Information gain-based exploration using rao-blackwellized particle filters". In: *Robotics: Science and Systems*. Vol. 2. Cambridge, MA, USA, 2005, pp. 65–72 (cit. on p. 3).

[35] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Boston, MA, USA, 2015, pp. 1–9 (cit. on p. 13).

[36] Srivastava Tavish. *How to avoid Over-fitting using Regularization?* [Online; accessed 11-Aug-2019]. 2015. URL: https://www.analyticsvidhya.com/blog/2015/02/avoid-over-fitting-regularization/ (cit. on p. 74).

[37] *TensorFlow*. URL: https://www.tensorflow.org/ (cit. on p. 53).

[38] Sebastian Thrun et al. "Robotic mapping: A survey". In: *Exploring artificial intelligence in the new millennium* 1.1-35 (2002), p. 1 (cit. on pp. 1, 3).

[39] Craig Tovey and Sven Koenig. "Improved analysis of greedy mapping". In: *Proceedings of the IEEE conference on Intelligent Robots and Systems (IROS 2003)*. Vol. 4. Las Vegas, NV, USA, 2003, pp. 3251–3257 (cit. on p. 4).

[40] Zhenhua Wang, Xingxing Wang, and Gang Wang. "Learning fine-grained features via a CNN tree for large-scale classification". In: *Neurocomputing* 275 (2018), pp. 1231–1240 (cit. on p. 13).

[41] *Wiki - ROS nav_msgs*. URL: http://wiki.ros.org/nav_msgs (cit. on p. 49).

[42] *Wiki : Dropout (neural networks)*. 2019. URL: https://en.wikipedia.org/wiki/Dropout_(neural_networks) (cit. on p. 40).

[43] *Wiki : ROS map server*. URL: http://wiki.ros.org/map_server (cit. on p. 54).

[44] *Wiki : ROS node*. URL: http://wiki.ros.org/Nodes (cit. on p. 49).

[45] Brian Yamauchi. "A frontier-based approach for autonomous exploration". In: *cira*. Vol. 97. Monterey, CA, USA, 1997, p. 146 (cit. on p. 3).

[46] Chengquan Zhang et al. "Automatic discrimination of text and non-text natural images". In: *Proceedings of the IEEE conference on Document Analysis and Recognition (ICDAR)*. Tunis, Tunisia, 2015, pp. 886–890 (cit. on p. 13).