Identifying Critical Regions for Robot Planning Using Convolutional Neural Networks

by

Daniel Molina

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Siddharth Srivastava, Chair
Baoxin Li
Yu Zhang

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

In this thesis, a new approach to learning-based planning is presented where *critical regions* of an environment with low probability measure are learned from a given set of motion plans. Critical regions are learned using convolutional neural networks (CNN) to improve sampling processes for motion planning (MP).

In addition to an identification network, a new sampling-based motion planner, *Learn and Link*, is introduced. This planner leverages critical regions to overcome the limitations of uniform sampling while still maintaining guarantees of correctness inherent to sampling-based algorithms. Learn and Link is evaluated against planners from the Open Motion Planning Library (OMPL) on an extensive suite of challenging navigation planning problems. This work shows that critical areas of an environment are learnable, and can be used by Learn and Link to solve MP problems with far less planning time than existing sampling-based planners.

To my mother, father, and brother for their unconditional love and support.

ACKNOWLEDGMENTS

I would first like to express my gratitude to Dr. Siddharth Srivastava for his patience and encouragement during my time working with him. I would also like to thank the many collaborators who have helped me along way, particularly Kislay Kumar for his help with neural networks and the members of the Autonomous Agents and Intelligent Robotics lab. Finally, I am extremely grateful to my parents, my brother, and my friends for keeping me motivated, and for their unwavering support. I could not have gotten this far without you all.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.1 Sampling-based Motion Planning

The motion planning problem deals with finding a feasible trajectory that takes a robot from an initial configuration to a goal configuration without colliding with obstacles. Reif (1979) showed that from a computational complexity point of view, even a simple form of the motion planning problem is NP-hard. In order to achieve computational efficiency, motion planning methods relax requirements of completeness. Sampling-based motion planners, such as Rapidly-exploring Random Trees (RRT) from LaValle and Kuffner Jr (2001) and Probabilistic Roadmaps (PRM) from Svestka *et al.* (1996), rely on *probabilistic completeness*, which assures a solution, if one exists, as the number of samples approaches infinity. Sampling-based motion planners sample a set of states from the configuration space (C-space) and check their connectivity without ever explicitly constructing any obstacles. This can reduce computation time considerably, especially as environments increase in complexity. Their performance, however, hinges on two main considerations: the way the C-space is sampled, and the particular order in which samples are chosen.

Lack of scalability of motion planning severely limits the applicability of symbolic planning algorithms in robotics problems, and precludes the development of versatile, autonomous robots. In order to improve the scalability of motion planning, a new approach for learning approximate, significant landmarks, or critical regions, for motion planning problems is presented. Lindemann and LaValle (2005) showed that these type of regions, such as narrow corridors, are less likely to be sampled, but are critical for solutions since most solutions must pass through them.

In this work, the sampling limitations of sampling-based motion planners are rectified using a CNN to identify critical regions prior to planning. Recent work on CNNs has demonstrated their utility in situations where the input data can be expressed as an image-based representation (Badrinarayanan *et al.* (2015); Ronneberger *et al.* (2015)). For a MP problem, an image representation can be created through recording the motion plans and environment. This approach begins using RRT-Connect (Kuffner and LaValle (2000)) to compute motion plans on a set of handmade training environments, though historical data or human demonstrations can be used as well. A raster scan is then used to construct images of the environments and trajectories. The environment images are based on collisions with the environment's obstacles, and the trajectory images are based on path intersections. Using the trajectory images, two labelling approaches are explored. The first method involves creating auto-generated saliency maps from the trajectory images using the Itti and Koch (2000) saliency model. The saliency maps are then thresholded to identify the most critical regions. The second method involves creating saliency labels using a cluster-and-fill approach on the critical regions of the trajectory image. This requires clustering the most salient areas, finding the concave hull of the region, and filling in the border described by the hull. This is done to produce smooth and clean regions so that the network is able to learn to distinguish between the free space and the critical regions more easily. Finally, a CNN is trained to identify critical regions using the saliency labels.

The model is then utilized by a new sampling-based motion planner, Learn and Link, to reduce the computation necessary to solve challenging navigation plans without compromising guarantees of correctness. Learn and Link has two modes: a single query mode, Learn and Link Planner (LLP), and a multi-query mode, Learn and Link

Roadmap (LL-RM). LL-RM is similar to PRM in that a roadmap is constructed, using both critical regions and random configurations, which can be used for multiple queries (see Figure 1.1). LLP is used for efficient single query plans when it is not necessary to construct a general roadmap, and one would like to bias a roadmap for a given start-goal pair. Currently, Learn and Link is usable for navigation planning and is being extended for general robot planning in the future. LLP and LL-RM are evaluated against sampling-based motion planners from OMPL.

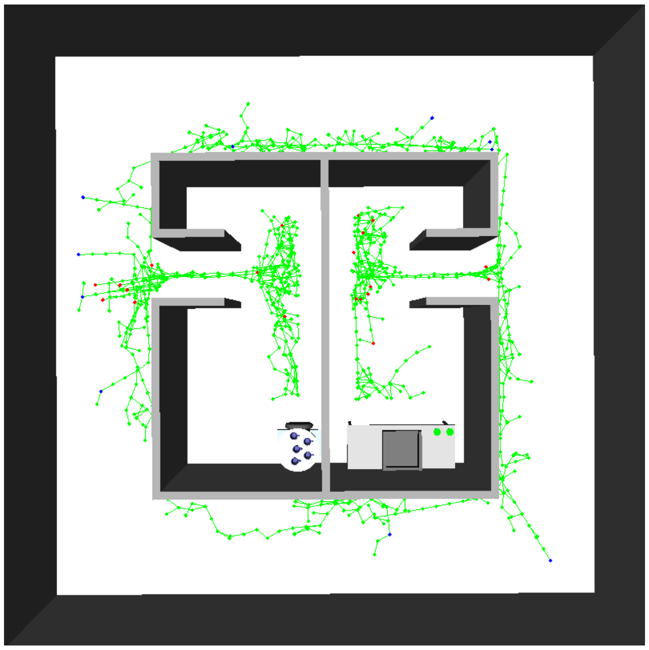


Figure 1.1: An example of the roadmap created using LL-RM for a Barrett WAM arm navigation task. The green points are states that were created when linking the vertices of the roadmap, the blue points are the vertices of the roadmap that were uniformly sampled, and the red points are the vertices of the roadmap that came from the critical regions.

Experiments reveal that areas of an environment that are critical for MP, but have a low probability of being sampled under a uniform distribution, are identifiable using CNNs. They demonstrate that these critical regions can be utilized by Learn and Link to robustly compute motion plans while requiring far less planning time than existing sampling-based motion planners. This approach is advantageous over pure sampling-based planners and pure learners: it leverages learning from experience to outperform sampling-based planners, but avoids the possibility of missing solutions that limits pure imitation learning, and remains probabilistically complete.

These results are believed to be general and hold across many domains. This approach is particularly useful in situations where one would have some prior knowledge on the class of environments being traversed, but not have the luxury of using time-expensive planners.

This approach presents a first step towards creating hierarchies for continuous planning problems by extracting critical regions for a given environment and defining actions as transitions between them.

1.2 Related Works

The notion of discrete landmarks has been used to improve the performance of classical planners in the past (Hoffmann *et al.* (2004)). The approach proposed in this thesis relates to this concept but differs in its consideration of the sets of states that are not only useful for reaching the goal, but are also less likely to be reached under a stochastic search paradigm.

Several methods have been proposed to guiding sampling-based motion planners to solutions. Heuristically-guided RRT (Urmson and Simmons (2003)) uses a probabilistic implementation of heuristic search concepts to create a reasonable bias towards exploration, as well as exploiting known good paths. Although this approach was able to produce less expensive paths, it required a high computational price. Anytime RRTs (Ferguson and Stentz (2006)) reuse information from previous RRTs to improve on the path by rejecting samples which have a higher heuristic cost. Batch Informed Trees (BIT*) (Gammell *et al.* (2015)) uses a heuristic to efficiently search a series of increasingly dense implicit RGGs while reusing previous information. In contrast, the method proposed in this thesis guides the planner to solutions without the need of a heuristic. Rather, the learned sampling distribution helps bias the sampling towards critical regions which have a lower probability of getting sampled, but in most scenarios are necessary for optimal solution.

The coupling of learning and motion planning has been extensively investigated in the past. Recent work by Ichter *et al.* (2018) uses a Conditional Variational Autoencoder to bias sample points for motion planning conditioned on encoded environment variables. This encoding is generalizable to higher dimensions, however it requires structuring the data to encompass the state of the robot, the environment, the obstacles (encoded as an occupancy grid), and the start and goal configurations. Moreover, during inference, the network model requires this costly data structuring again, which can take around 50 seconds. In contrast, this approach focuses on image-based learning where data can be easily generated using a top-view camera. Moreover, inferences can also be made using a top-view image of the environment to leverage the learned critical regions for motion planning in less than 5 seconds. This results in faster inference for situations demanding faster motion plans. Havoutis and Ramamoorthy (2009) use topology to learn sub-manifold approximations that are defined by a set of possible trajectories in the configuration space. This requires either motion plans that are generated through a motion capture device, or hand-crafted partial plans. Pan *et al.* (2013) use instance-based learning where prior collision results are stored as an approximate representation of the collision space and the free C-space. This is used to make cheaper probabilistic queries. Although their method shows significant improvement in some environments, their work is limited in finding solutions through narrow passages between obstacles where optimal solution may lie. In this work, the network learns the positions of regions that are critical for a given class of motion planning problems, but have low probability of getting sampled under uniform distribution. These critical regions can be leveraged by any motion planner for faster solutions.

1.3 Experimental Setup

The model's generalizability is evaluated on two domains: *SE(2)* and a 10-DOF C-space involving a 7-DOF Barrett WAM arm on a moveable platform (see Figure 1.2).

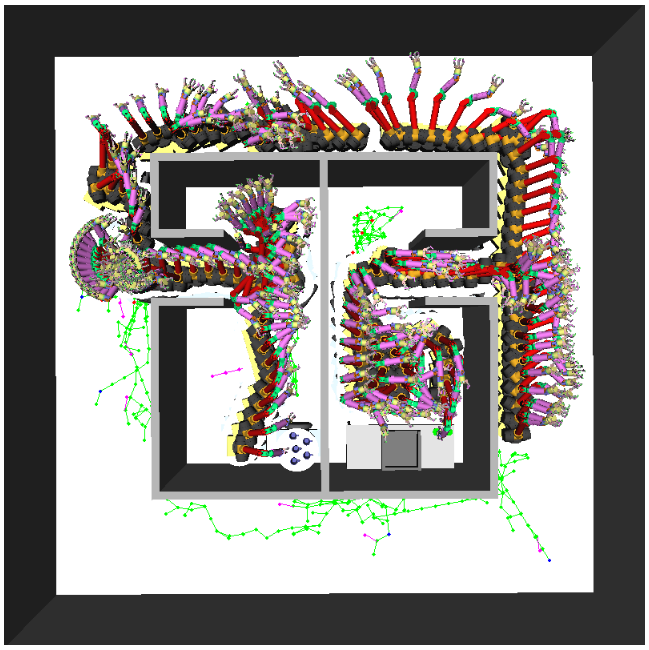


Figure 1.2: 7-DOF Barrett WAM arm on a movable base solves a transportation task using LL-RM. The pink points are states that were created when linking the start and goal configurations to the roadmap.

This thesis focuses on investigating two main questions:

1. Can CNNs be used to identify critical regions for motion planning?

2. Can Learn and Link solve challenging navigation problems using less time than prominent sampling-based planners?

The first consideration aims to see if the visual prowess exhibited by CNNs extends to critical region identification. The second consideration aims to see to what extent do critical regions help a planner in computation reduction.

To investigate these considerations, challenging motion planning problems were designed for *SE(2)* (see Figure 1.3) and the Barrett WAM arm (see Figure 1.4), and various network architectures were explored. The testing environments are comprised of extreme narrow channel situations and floor plan types. For both domains, 100 MP problems were constructed using the same start and goal pair, the same range, and a planning time limit of 60 seconds. LLP and LL-RM use 5% of the total non-collision causing critical region points identified as n, and $m = n/10$ and $m = 0$, respectively. OMPL PRM and LL-RM are both given 1 second to build a roadmap prior to planning. This approach is for robots with omnidirectional base movements, though an arbitrary local planner can be substituted in the *EXTEND* module. It is also important to point out that OMPL is written in highly optimized C++ code compared to this Python implementation.

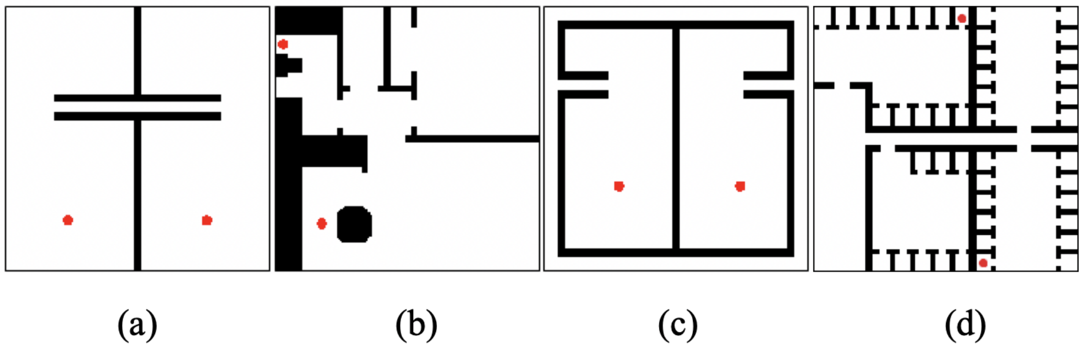


Figure 1.3: *SE(2)* test environments used to evaluate the model. Red dots represent the start and goal configurations.

The critical regions identified by a model are evaluated using the ground truth motion trace image for an environment. The model-identified critical regions are first clustered using k-Nearest Neighbors (Altman (1992)). Then, each critical region cluster is evaluated using the μ-criticality of the cluster, where $v(r)$ is estimated as the

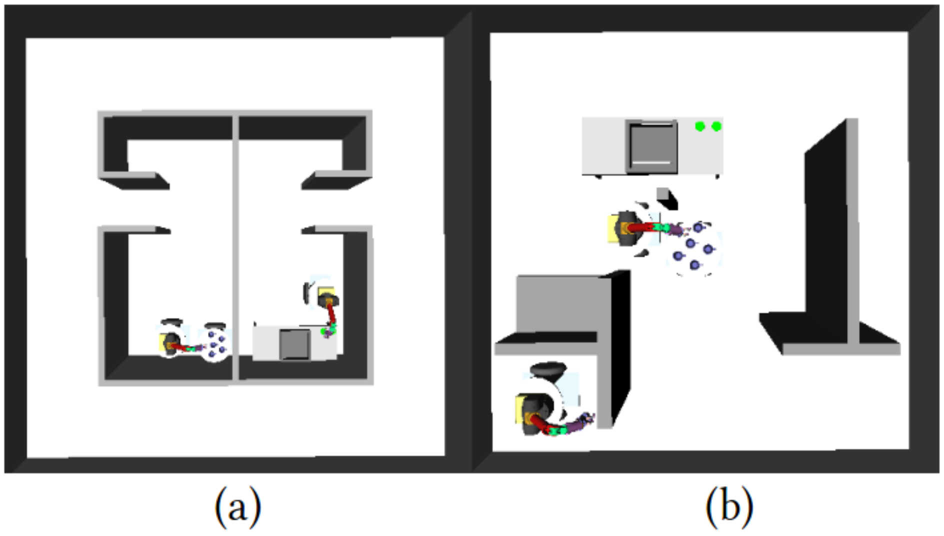


Figure 1.4: Barrett arm test environments used to evaluate the model. The robot is placed at the start and goal configurations.

area of the pixels in the cluster. The metric values for each cluster are then summed to obtain an evaluation of the environment as a whole. The higher the value, the better the critical regions.

This metric is used instead of comparing pixel accuracy with the ground truth label since the motion trace image is embedded with much more information regarding the quality of the critical regions than solely being able to locate them.

Chapter 2

LEARNING CRITICAL REGIONS

Given a robot R, an environment E, and a class of MP problems M, the *measure of criticality* of a Lebesgue-measurable open set $r \subseteq \mathbb{R}^n$, $\mu(r)$, is defined as $lt_{s_n \to^+ r} \frac{f(r)}{v(s_n)}$, where $f(s_n)$ is the fraction of observed motion plans solving tasks from M that pass through s_n, $v(s_n)$ is the measure of s_n under a reference (usually uniform) density, and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing r ($r \subseteq s_n$ for all n). Note that $\mu(r)$ is zero when $f(r) = 0$. While $\mu(r)$ can be infinite for a region, for all practical purposes, only regions r with $v(r) > 0$ under the uniform density are considered. Intuitively, regions with high criticality measures are those that are vital for solutions to problems in M, but have a low probability of exploration under a uniform density.

To learn critical regions, a set D_{train} of N_{train} MP problem instances $\{\Pi_1, ..., \Pi_{N_{train}}\}$, with corresponding solution trajectories $\{\tau_1, ..., \tau_{N_{train}}\}$, is used to construct the training images; and a set D_{test} of N_{test} MP problem instances is used to evaluate the learned model.

This approach consists of two phases: a data generation phase and a model training phase.

2.1 Data Generation

For each $\Pi_i \in D_{train}$, an off-the-shelf motion planner is ran to generate a corresponding motion plan τ_i consisting of 50 random MP problems from M for various handmade environments (see Figure 2.1). In this thesis, an OpenRAVE (Diankov and Kuffner (2008)) implementation of OMPL's RRT-Connect planner by `https://github.com/personalrobotics` is utilized, though any motion planner can be used instead.

Training images for each $\Pi_i \in D_{train}$ are constructed using a raster scan and a saliency model. The process for an *SE(2)* robot is described, though it can be easily extended to mobile manipulators, such as the Barrett arm on a mobile base. First, a pixel-sized obstacle is created based on the dimensions of the desired image and the bounds of a given environment, and it is scanned across the environment. For the input images, if a collision is detected with an environment's obstacles, a black pixel is selected, otherwise a white pixel is selected. For the motion trace images, a pixel value is assigned based on the fraction of paths that pass through the region the pixel encompasses. Using the trajectory images, two labelling approaches are explored. The first method (see Figure 2.2) involves using auto-generated saliency maps from the motion trace images using an implementation of Itti's saliency model by `https://github.com/mayoyamasaki`. The saliency maps are binned into two categories, high saliency (denoted by white pixels) and low saliency (denoted by black pixels) before being used as labels. The second method (see Figure 2.3) involves creating saliency labels using a cluster-and-fill approach on the critical regions of the trajectory image. This involves clustering the most salient areas, calculating the concave hull of each cluster, and filling the interiors described by each hull. This is done to produce smooth and clean regions so that the network is able to learn to distinguish between the free space and the critical regions more easily. The second labeling approach was devised to determine if the biases inherent to saliency models were affecting the performance of the network; though in the end, the auto-generated labels performed better with the final network.

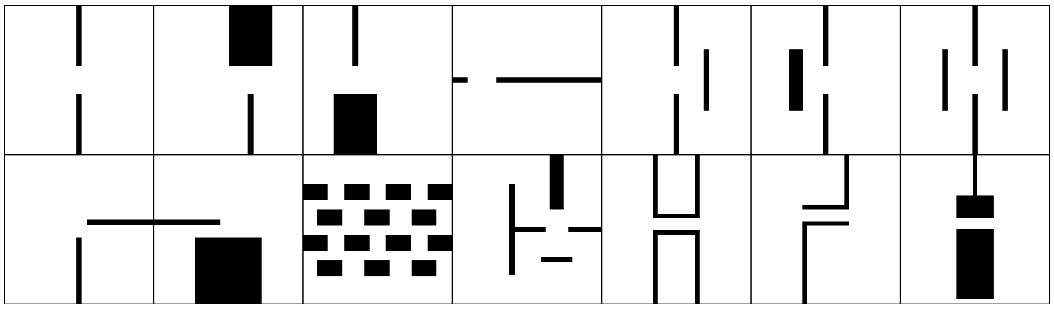


Figure 2.1: Handmade training environments used for the *SE(2)* domain (not including rotations). Training environments for the Barrett arm are similar, though scaled appropriately for the difference in robot size.

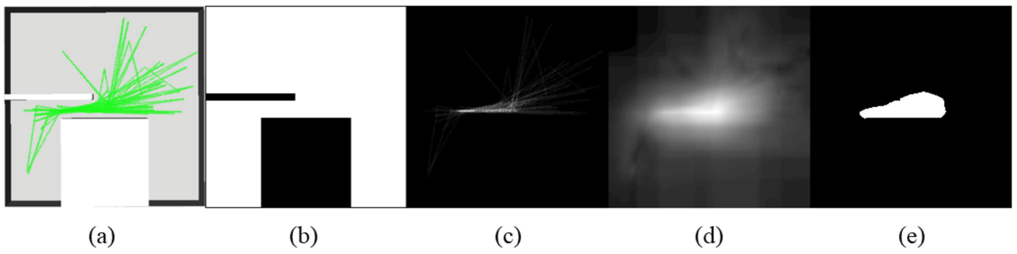


Figure 2.2: (a) Example training environment overlain with motion traces.(b) Model input obtained post raster scan.(c) Motion trace image based on the fraction of paths that pass through each pixel.(d) Saliency map obtained from the motion trace image.(e) Input label obtained after binning the saliency map based on pixel intensity.

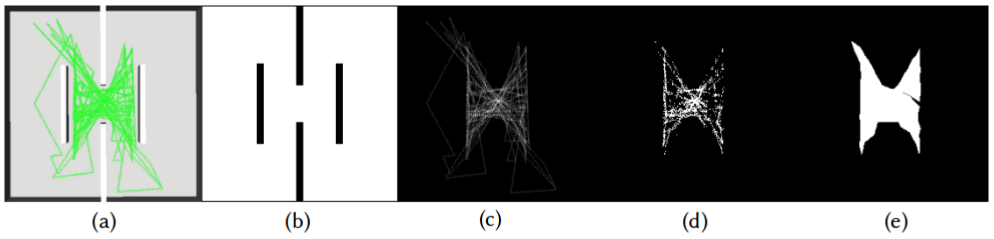


Figure 2.3: (a) Example training environment overlain with motion traces.(b) Model input obtained post raster scan.(c) Motion trace image based on the fraction of paths that pass through each pixel.(d) Thresholded motion trace image to visualize the most salient areas.(e) Input label obtained after clustering the salient areas and filling them in.

2.2 Network Architecture

A general structure for a convolutional encoder-decoder neural network which learns to detect critical regions is proposed.

The network, depicted in Figure 2.4, has 14 convolutional layers. 7 layers in the encoder network and 7 layers in the decoder network form the encoder-decoder architecture for pixel-wise classification. A pooling layer with stride 2 is introduced after each group of same number of filters to encode the learned representation. Similarly, an upsampling layer is added before each deconvolutional layer group of same number of filters. Inspiration is drawn from Badrinarayanan *et al.* (2015) for a learnable upsampling layer in the decoder network.

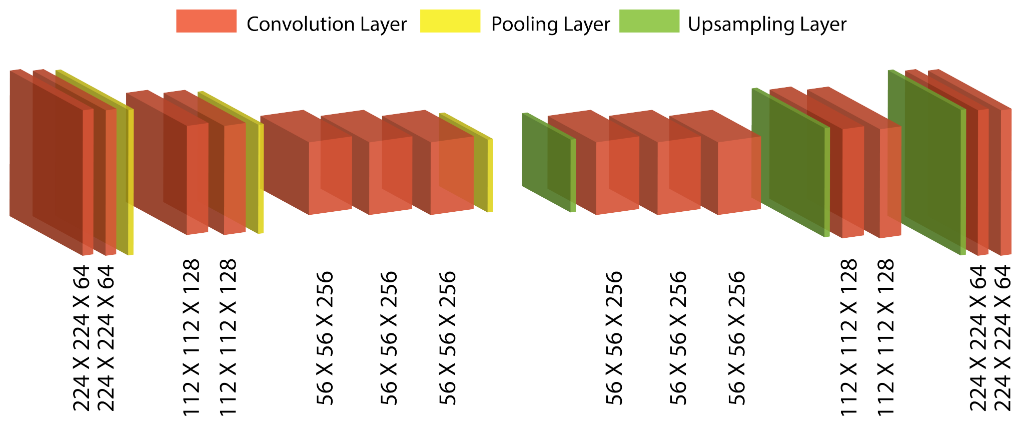


Figure 2.4: Network architecture selected for the CNN.

The first two convolutional layers have 64 filters with a 3×3 kernel. Motivated by the recent promising results of Simonyan and Zisserman (2014), 3 layers with 3×3 kernel size are stacked to obtain a similar receptive field as a 7×7 kernel, with 81% less parameters, and more effective training owing to the added non-linearity after every layer. For the initial layer group of filter size 64 and 128, only two layers of kernel size 3×3 are stacked. Though the receptive field is smaller than a 7×7 kernel, still only 2 layers are stacked as the problem statement does not require learning complex geometric features. The next 2 layers are of 128 filters with a 3×3 kernel. Finally, 3 layers of 256 filters each, with a 3×3 kernel, are added for a larger receptive field since Zeiler and Fergus (2014) showed that deeper layers learn invariant complex features.

In the decoder network, corresponding deconvolutional layers to the encoder network are used. The upsampled output is used for pixel-wise classification using a softmax cross entropy loss function. Each layer in the network is activated using ReLu non-linearity.

2.3 Network Training

The network was trained on a single Nvidia GTX 1080Ti using a mini-batch size of 16 and a dataset of 10,024 images. Following Ioffe and Szegedy (2015), the network was not trained with dropout (Srivastava *et al.* (2014)) since the output of every layer is batch-normalised, which also acts as a regularizer. The Adam Optimizer (Kingma and Ba (2014)) with a 0.1 learning rate is used to train the network. The network was trained for 50,000 epochs since the loss converges at this point. The training images are shuffled before each epoch and trained with mini-batch to ensure that every input to the network is different from the previous. This assists the optimizer to exit local minima. An implementation of SegNet (Badrinarayanan *et al.* (2015)) by `https://github.com/andreaazzini` is used for its data pipelines since they provide a fast and efficient input pipeline which reduces training time.

On average, training for the full dataset using mini-batch takes approximately 3 hours.

2.4 Processing Critical Regions

The following section discusses how to process the model output so that it can be used by Learn and Link.

Seeing as the model output is in image format, a mapping of pixel indices to the environment's coordinate system is required: $f : (i, j) \mapsto [p_{min_x}, p_{min_y}, p_{max_x}, p_{max_y}]$. Such a mapping is defined as follows:

$$f(i,j) = \begin{vmatrix} p_{min_x} & p_{min_y} \\ p_{max_x} & p_{max_y} \end{vmatrix}$$

$$f(i,j) = \begin{vmatrix} b_{min} & b_{max} \\ b_{min} & b_{max} \end{vmatrix} + \begin{vmatrix} i & -(j+1) \\ i+1 & -j \end{vmatrix} \times p_w$$

where $p_w = \frac{b_{max} - b_{min}}{224}$ and $0 \leq i, j \leq 224$.

In the equation: i is the horizontal pixel index, j is the vertical pixel index, b_{min} and b_{max} are the bounds of the square environment, p_w is the width of a pixel in terms of the environment's coordinate system, and $f(i, j)$ gives the bounds for a pixel located at (i, j) in terms of the environment's coordinate system. i and j are bounded since 224×224 is the desired dimension of the model input. It can be altered to accommodate the model.

Using *f*, the pixels of the model output are iterated through and the coordinate bounds of the pixels identified as critical regions, i.e. the white pixels, are stored. A list of critical region points are then taken and passed to Learn and Link.

Chapter 3

LEARN AND LINK

In this section, Learn and Link, and the various methods that compromise it, are discussed.

3.1 Algorithms

The two version of Link and Learn are discussed. A single query version: Learn and Link Planner, and a multi-query version: Learn and Link Roadmap.

3.1.1 Learn and Link Planner

Algorithm 1 is the base of LLP. In lines $11-17$, the graphs in the roadmap RM are initialized using a list of k configurations from the critical regions (CR) that do not result in collisions, the initial configuration (q_{init}), and the goal configuration (q_{goal}). In line 19, a random sample is taken to grow the current graph in its direction. In line 20, an attempt is made to extend the current graph to q_{new}, a new configuration in the direction of q_{rand}. If adding q_{new} to the graph results in a collision, *i.e.* *EXTEND* returns $Trapped$, q_{new} is not added to the graph; otherwise it is added. In line 21, a solution check occurs. If a solution is found, the path connecting the start and goal configurations is found using Dijkstra's algorithm (Dijkstra (1959)) in line 22. If the conditions in lines $20-21$ are not satisfied, shift to the next graph in the roadmap using a round-robin approach in line 24. If an explicit sample cap is reached, *i.e.* $S \neq \infty$, without a solution path being found, an empty solution path is returned.

Algorithm 2 is used in an attempt to link a subgraph to the remaining graphs in the roadmap (lines $9-11$), to remove dead graphs from consideration (line 12), and to check whether all the subgraphs in the roadmap have been linked (line 13). A subgraph is considered dead once it has been linked and added to another graph. Once only one graph remains in the roadmap list, *Linked* is returned to indicate that the roadmap is complete when used in LL-RM, or that a solution path is obtainable when used in LLP.

Algorithms 3 and 4 depict the methods reused and adapted from RRT-Connect. These methods are used to grow the current graph in the direction of the random samples taken. These are used in LL-RM as well.

3.1.2 Learn and Link Roadmap

In this section we discuss the additional methods that make up LL-RM.

Algorithm 5 is used to construct LL-RM's roadmap. In lines $10 - 13$, n random non-collision configurations are added as vertices to the roadmap from the critical regions identified by the model. In lines $14-17$, m random non-collision configurations are added as vertices to the roadmap using a uniform sampler. For the remainder of the algorithm, the subgraphs spawned from the vertices in the roadmap are attempted to be linked. In line 19, a random sample is taken to grow the current subgraph in its direction. In line 20, an attempt is made to extend the graph to q_{new}, a new configuration in the direction of q_{rand}. If adding q_{new} to the graph results in a collision, *i.e. EXTEND* returns $Trapped$, q_{new} is not added to the graph; otherwise it is added. In line 21, a connectivity attempt occurs to link the current subgraph to the remaining graphs in the roadmap; once all the subgraphs have been connected, *Linked* is returned and the building process is complete. Finally, the roadmap is returned on line 22. If the conditions in lines $20 - 21$ are not satisfied, shift to the next subgraph in the roadmap using a round-robin approach in line 23. If an explicit sample cap is reached, *i.e.* $S \neq \infty$, without a solution path being found, an empty roadmap, indicating a failure, is returned. This roadmap can be reused for multiple queries.

Algorithm 6 is the planning component of LL-RM. In lines $9 - 12$, two subgraphs are initialized from the initial (q_{init}) and goal (q_{goal}) configurations in an attempt to connect them to the existing roadmap *RM*. In lines $13 - 18$, the same approach used in the building process is employed to connect the start and goal subgraphs to the roadmap. In line 16, a solution check occurs. If a solution is found, the path connecting the start and goal configurations is found using Dijkstra's algorithm in line 17.

3.2 Extension to Mobile Manipulators

The extension to higher DOF robots follows simply. Since the model only gives base poses, append each configuration in *CR* with a random, non-collision causing, configuration for the additional DOF values prior to calling the planner. The algorithm then proceeds as usual.

3.3 Probablistic Completeness

Learn and Link maintains the probabilistic completeness property inherent to sampling-based motion planners. Since LLP and LL-RM only add a finite set of points to seed their roadmaps, it does not reduce the set of support (regions with non-zero probability) of its uniform sampler, and thus, this property is preserved. Even when the network paired with Learn and Link fails to identify any critical regions, no issue arises. In this scenario, LLP works analogously to RRT-Connect and LL-RM works analogously to PRM, both of which are probabilistcally complete.

Algorithm 1 `LLP`

```
1: Input
2:     $Q_{init}$:  start configuration
3:     $Q_{goal}$:  goal configuration
4:     $CR$:    list of critical region points
5: Output
6:     $P$:     non-collision path from start to goal, if it exists
7: procedure LLP($Q_{init}$, $Q_{goal}$, CR)
8:     $curr \leftarrow 0$
9:     $RM \leftarrow []$
10:    $K \leftarrow |CR|$
11:    $G_0.init(q_{init})$
12:    $G_1.init(q_{qoal})$
13:    $RM.append(G_0)$
14:    $RM.append(G_1)$
15:    for $k = 1$ to $K$ do
16:        $G_{k+1}.init(CR_k)$
17:        $RM.append(G_{k+1})$
18:    for $s = 1$ to $S$ do
19:        $q_{rand} \leftarrow UNIFORM()$
20:        if $EXTEND(G_{curr}, q_{rand}) \neq Trapped$ then
21:            if $LINK(RM, G_{curr}, q_{new}) == Linked$ then
22:                $P \leftarrow PATH(RM[0])$
23:                Return $P$
24:        $G_{curr} \leftarrow SWAP(RM, G_{curr})$
25:    Return []
```

Algorithm 2 `LINK`

```
1: Input
2:     $RM$:  roadmap created using LLP or BUILD LL-RM
3:     $G_{curr}$: current subgraph being grown
4:     $Q_{new}$: most recent configuration added to $G_{curr}$
5: Output
6:     $S$:    status of $G_{curr}$'s link attempt
7: procedure LINK(RM,$G_{curr}$,$Q_{new}$)
8:     $R \leftarrow []$
9:     for $G_i$ in $RM \setminus G_{curr}$ do
10:        if $CONNECT(G_i, q_{new}) == Reached$ then
11:            $R.append(G_i)$
12:    $RM.link\_and\_remove(R, G_{curr})$
13:    if $|RM| == 1$ then
14:        $S \leftarrow Linked$
15:    else if $|R| > 0$ then
16:        $S \leftarrow Connected$
17:    else
18:        $S \leftarrow Advanced$
19:    Return $S$
```

Algorithm 3 `CONNECT`

```
1: Input
2:     $G$:    graph being grown towards $q$
3:     $Q$:    configuration which $G$ is trying to connect to
4: Output
5:     $S$:    status of $G$'s connect attempt
6: procedure CONNECT(G,Q)
7:     repeat
8:         $S \leftarrow EXTEND(G, q)$
9:     until $S \neq Advanced$
10:    Return $S$
```

Algorithm 4 `EXTEND`

```
1:  Input
2:      $G$:    graph being grown towards $q$
3:      $Q$:    configuration which $G$ is stepping toward
4:  Output
5:      $S$:    status of $G$'s extend attempt
6:  procedure EXTEND(G,Q)
7:      $S \leftarrow Trapped$
8:      $q_{near} \leftarrow NN(q, G)$
9:      if $CONFIG(q, q_{near}, q_{new})$ then
10:         $G.add\_vertex(q_{new})$
11:         $G.add\_egde(q_{near}, q_{new})$
12:         if $q_{new} == q$ then
13:             $S \leftarrow Reached$
14:         else
15:             $S \leftarrow Advanced$
16:     Return $S$
```

Algorithm 5 `BUILD LL-RM`

```
1: Input
2:     $N$:   number of critical region states to include in the roadmap
3:     $M$:   number of uniform states to include in the roadmap
4:     $CR$:  list of critical region points
5: Output
6:     $RM$:  constructed roadmap
7: procedure BUILD(N,M,CR)
8:     $curr \leftarrow 0$
9:     $RM \leftarrow []$
10:    for $n = 0$ to $N-1$ do
11:        $s \leftarrow SAMPLE(CR)$
12:        $G_n.init(s)$
13:        $RM.append(G_n)$
14:    for $m = 0$ to $M-1$ do
15:        $s \leftarrow SAMPLE()$
16:        $G_{N+m}.init(s)$
17:        $RM.append(G_{N+m})$
18:    for $s = 1$ to $S$ do
19:        $q_{rand} \leftarrow UNIFORM()$
20:        if $EXTEND(G_{curr}, q_{rand}) \neq Trapped$ then
21:            if $LINK(RM, G_{curr}, q_{new}) == Linked$ then
22:                Return $RM$
23:        $G_{curr} \leftarrow SWAP(RM, G_{curr})$
24:    Return []
```

Algorithm 6 `PLAN LL-RM`

```
1: Input
2:     $Q_{init}$: start configuration
3:     $Q_{goal}$: goal configuration
4:     $RM$: roadmap created using BUILD
5: Output
6:     $P$: non-collision path from start to goal, if it exists
7: procedure PLAN($Q_{init}$,$Q_{goal}$,RM)
8:     $curr \leftarrow 0$
9:     $G_1.init(q_{init})$
10:    $G_2.init(q_{qoal})$
11:    $RM.append(G_1)$
12:    $RM.append(G_2)$
13:    for $s = 1$ to $S$ do
14:        $q_{rand} \leftarrow UNIFORM()$
15:        if $EXTEND(G_{curr}, q_{rand}) \neq Trapped$ then
16:            if $LINK(RM, G_{curr}, q_{new}) == Linked$ then
17:                $P \leftarrow PATH(RM[0])$
18:                Return $P$
19:        $G_{curr} \leftarrow SWAP(RM, G_{curr)}$
20:    Return []
```

Chapter 4

RESULTS

Results suggest that both LLP and LL-RM require far less time to obtain a solution than OMPL's RRT, RRT-Connect, and PRM planners, especially as the environments increase in difficulty. Figures 4.1 and 4.2 show a comparison of planning time used by the OMPL planners and LLP and LL-RM using the areas learned by the parsimonious network.

4.1 SE(2)

For *SE(2)*, LLP and LL-RM outperformed OMPL's planners in terms of average planning time and success rate, except on environment (d). On this floor plan environment, PRM edged out both LLP and LL-RM, requiring 62% and 78% less planning time on average, respectively; though RRT and RRT-Connect were still easily outperformed. The next closest was on the second floor plan environment (b) where LLP and LL-RM used 66% and 57% less time on average, respectively, than PRM, the best performing OMPL planner on this environment. The performance on these floor plan environments is attributed to there being a lot more open space and less narrow passages compared to the size of the robot, and the size of the C-space. Also recall that PRM and LL-RM both received an additional second for roadmap construction. On environments (a) and (c), whose passages allow for limited movement, the difference is more extreme. On environment (a), RRT, RRT-Connect, and PRM had success rates of 19%, 0%, and 53%, respectively. For successful plans, LLP and LL-RM required 97% and 99% less time on average, respectively, than PRM, the best performing OMPL planner on this environment. On environment (c), RRT,

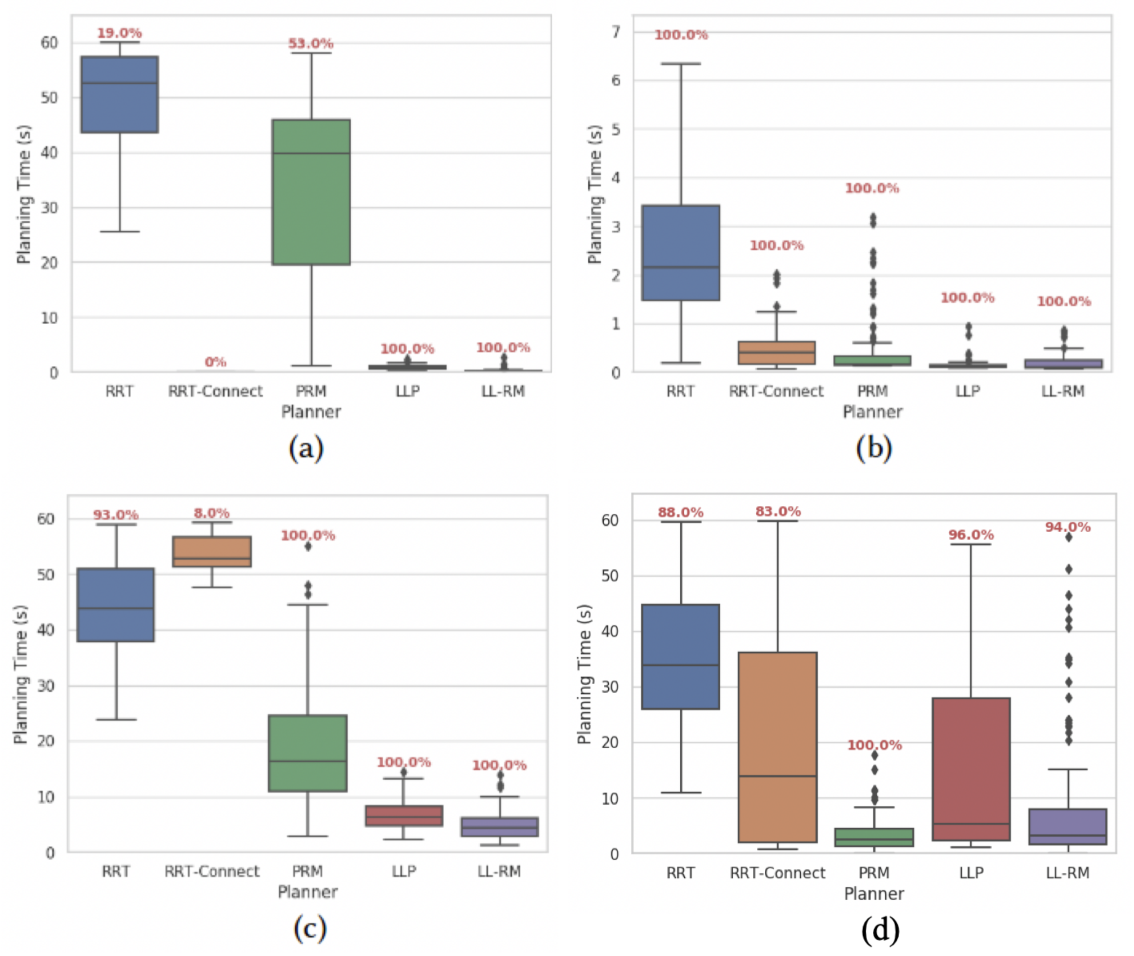
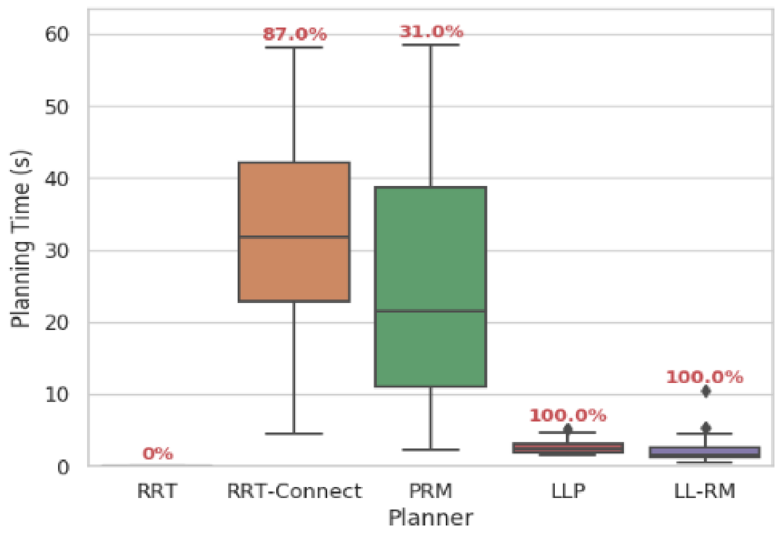


Figure 4.1: Boxplots of the 100 runs comparing the planning time used to arrive at a solution for the *SE(2)* domain. The success rate of the 100 plans for each planner is listed in red on the plot.

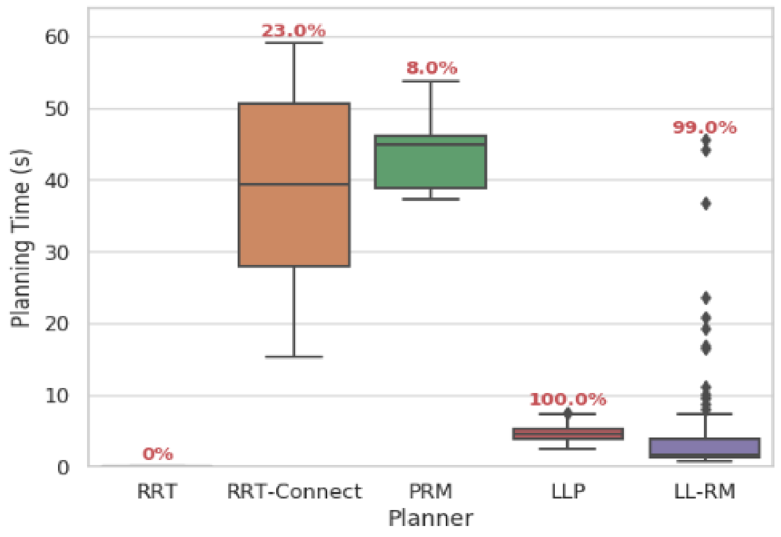
RRT-Connect, and PRM had success rates of 93%, 8%, and 100%, respectively. For successful plans, LLP and LL-RM required 64% and 74% less time on average, respectively, than PRM, the best performing OMPL planner on this environment.

4.2 10-DOF

For the transportation tasks using the movable Barrett arm, LLP and LL-RM require less planning time on average and had higher success rate than OMPL. On environment (a), RRT, RRT-Connect, and PRM had success rates of 0%, 87%, and



(a)



(b)

Figure 4.2: Boxplots of the 100 runs comparing the planning time used to arrive at a solution for the 10-DOF domain. The success rate of the 100 plans for each planner is listed in red on the plot.

31%, respectively. When comparing successful plans, LLP and LL-RM required 89% and 92% less time on average, respectively, than PRM, the best performing OMPL planner on this environment. On environment (b), RRT, RRT-Connect, and PRM had success rates of 0%, 23%, and 8%, respectively. When comparing successful plans, both LLP and LL-RM required 88% less time on average than RRT-Connect, the best performing OMPL planner on this environment.

4.3 Network Ablation Study

Since obstacles in an environment can be represented by bounding boxes, most of the objects in our dataset have regular geometric shapes. An ablation study was performed to find the simplest model that can learn the feature representation using as few layers as possible, without compromising the results. Two different types of neural networks are investigated and compared their performance with SegNet using the μ-criticality measure. The ablation study for both types of architecture is discussed below.

Convolutional Network

The main question in the network ablation study was to enquire whether a solely convolutional network would suffice in solving this problem.

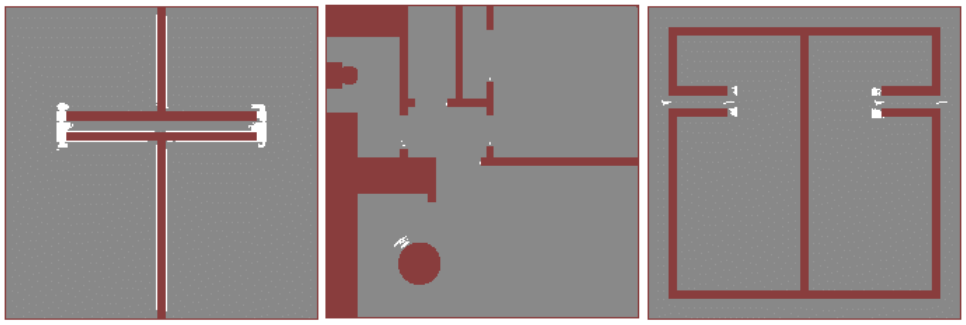
The CNN-based VGGNet learned only to trace obstacle borders. The μ-criticality for VGGNet as shown in the Figure 4.3(a) is 0 for all the test environment. Although the criticality values were not promising, it still shed light on network behaviour. The network was able to learn the geometry of the obstacles in the image, which CNNs are known to be good at, but was unable to identify the critical regions. Moreover, training VGGNet takes 16 hours on a single Nvidia GTX 1080Ti GPU for 50,000 epochs.

Encoder-Decoder

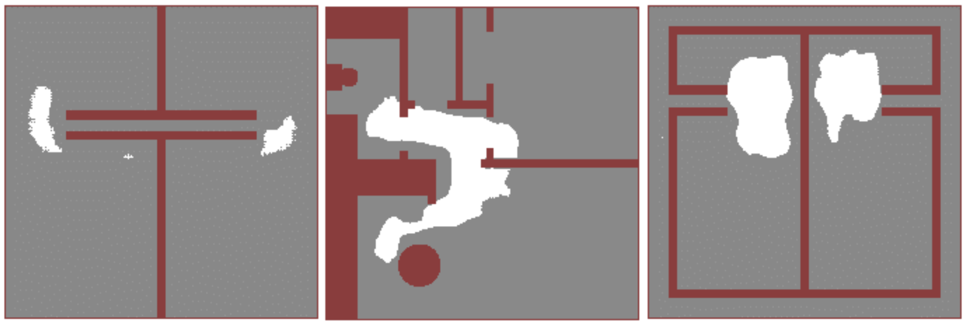
Following promising initial results using a SegNet architecture as shown in Figure 4.3(b), an encoder-decoder network which can learn the latent representation in a supervised manner for pixel-wise classification was investigated. In an encoder-decoder, the encoder can learn the feature representation and encode it into a latent space. While the decoder can learn the pixel-wise classification on the learned features.

A simple encoder-decoder network with 4 layers each in encoder and decoder sections of the network was able to somewhat learn the critical regions of the data well, obtaining μ-criticality scores of 0.0156, 0.384, and 1.043, respectively on the *SE(2)* environments, but tended to show a lot of checkerboard artifacts in the identified regions.

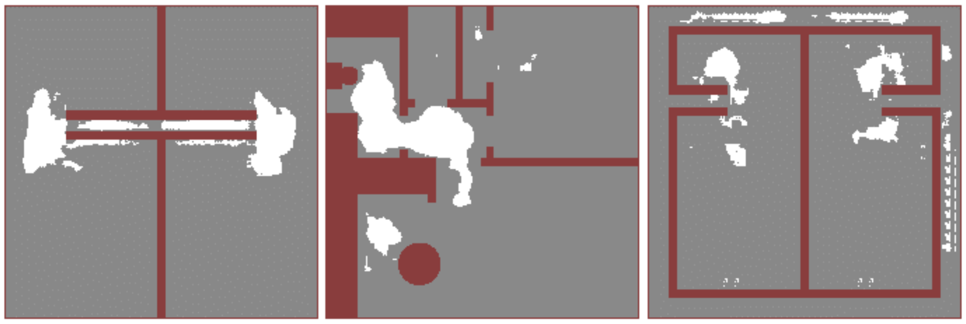
Building on top of the above architecture, 3 more batch normalized layers were added to increase the receptive field size in an attempt to smooth out the critical regions and generalize to the test set. μ-criticality scores of 0.604, 0.371 and 0.702 were achieved for the respective environments, as shown in Figure 4.3(c), indicating the network's ability to identify the critical regions for motion planning.



(a)



(b)



(c)

Figure 4.3: (a) Critical regions identified using VGGNet. From left to right, μ-criticality is 0, 0, 0.(b) Critical regions identified using SegNet. From left to right, μ-criticality is 0, 0.141, and 0.260.(c) Critical regions identified using the parsimonious network. From left to right, μ-criticality is 0.604, 0.371, and 0.702.

Chapter 5

CONCLUSION

A new approach to learning-based planning is presented using a new sample-based motion planner, Learn and Link. A fully convolutional encoder-decoder neural network is constructed to learn critical regions for navigation planning problems that generalizes across different domains. The model is used by Learn and Link to remedy the limitations of uniform sampling, without compromising guarantees of correctness.

Observed results on challenging navigation planning problems demonstrate that CNNs have the capability to extract important features relevant to robot planning problems.

Chapter 6

FUTURE WORK

In the future, this thesis could be extended to general planning tasks, in addition to navigation tasks. Currently, challenges are mainly in formulating the construction of data which could be used by a model to learn critical regions for general planning problems.

More work could also be done improving the second labeling approach. Either the parameters for calculating the concave hull of the clusters need to be tuned, or a different architecture is require to better utilize these labels.

REFERENCES

Altman, N. S., “An introduction to kernel and nearest-neighbor nonparametric regression”, The American Statistician **46**, 3, 175–185 (1992).

Badrinarayanan, V., A. Kendall and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”, arXiv preprint arXiv:1511.00561 (2015).

Diankov, R. and J. Kuffner, “Openrave: A planning architecture for autonomous robotics”, Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34 **79** (2008).

Dijkstra, E. W., “A note on two problems in connexion with graphs”, Numerische mathematik **1**, 1, 269–271 (1959).

Ferguson, D. and A. Stentz, “Anytime rrts”, in “Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on”, pp. 5369–5375 (IEEE, 2006).

Gammell, J. D., S. S. Srinivasa and T. D. Barfoot, “Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs”, in “Robotics and Automation (ICRA), 2015 IEEE International Conference on”, pp. 3067–3074 (IEEE, 2015).

Havoutis, I. and S. Ramamoorthy, “Motion synthesis through randomized exploration on submanifolds of configuration space”, in “Robot Soccer World Cup”, pp. 92–103 (Springer, 2009).

Hoffmann, J., J. Porteous and L. Sebastia, “Ordered landmarks in planning”, Journal of Artificial Intelligence Research **22**, 215–278 (2004).

Ichter, B., J. Harrison and M. Pavone, “Learning sampling distributions for robot motion planning”, in “2018 IEEE International Conference on Robotics and Automation (ICRA)”, pp. 7087–7094 (IEEE, 2018).

Ioffe, S. and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, arXiv preprint arXiv:1502.03167 (2015).

Itti, L. and C. Koch, “A saliency-based search mechanism for overt and covert shifts of visual attention”, Vision research **40**, 10-12, 1489–1506 (2000).

Kingma, D. P. and J. Ba, “Adam: A method for stochastic optimization”, arXiv preprint arXiv:1412.6980 (2014).

Kuffner, J. J. and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning”, in “Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on”, vol. 2, pp. 995–1001 (IEEE, 2000).

LaValle, S. M. and J. J. Kuffner Jr, “Randomized kinodynamic planning”, The international journal of robotics research **20**, 5, 378–400 (2001).

Lindemann, S. R. and S. M. LaValle, "Current issues in sampling-based motion planning", in "Robotics Research. The Eleventh International Symposium", pp. 36–54 (Springer, 2005).

Pan, J., S. Chitta and D. Manocha, "Faster sample-based motion planning using instance-based learning", in "Algorithmic Foundations of Robotics X", pp. 381–396 (Springer, 2013).

Reif, J. H., "Complexity of the mover's problem and generalizations", in "20th Annual Symposium on Foundations of Computer Science (sfcs 1979)", pp. 421–427 (IEEE, 1979).

Ronneberger, O., P. Fischer and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in "International Conference on Medical image computing and computer-assisted intervention", pp. 234–241 (Springer, 2015).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556 (2014).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", The Journal of Machine Learning Research **15**, 1, 1929–1958 (2014).

Svestka, P., J. Latombe and L. Overmars Kavraki, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation **12**, 4, 566–580 (1996).

Urmson, C. and R. Simmons, "Approaches for heuristically biasing rrt growth", in "Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on", vol. 2, pp. 1178–1183 (IEEE, 2003).

Zeiler, M. D. and R. Fergus, "Visualizing and understanding convolutional networks", in "European conference on computer vision", pp. 818–833 (Springer, 2014).

APPENDIX A

CODE

Code written for this thesis is located at:
`https://github.com/AAIR-lab/WaypointLearning`

APPENDIX B

DATA

Data generated and used to train the model is located at:
`https://drive.google.com/open?id=1NmT7f9XSA1mnNguTzaqSV4KLnrFfRKy`