ROAD EXTRACTION FROM HIGH-RESOLUTION SATELLITE IMAGERY USING DEEP REINFORCEMENT LEARNING

Nima Sarang

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science in Computer Science
Concordia University
Montréal, Québec, Canada

April 2022

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Nima Sarang**

Entitled: **Road Extraction from High-Resolution Satellite Imagery using Deep Reinforcement Learning**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Ching Yee Suen	Chair
René Witte	Examiner
Ching Yee Suen	Examiner
	Examiner
Charalambos Poullis	Supervisor

Approved _____
Lata Narayanan, Chair of Department

_____ 2022 _____ _____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Road Extraction from High-Resolution Satellite Imagery using Deep Reinforcement Learning

Nima Sarang

Reinforcement learning (RL) has emerged as one of the most promising and powerful techniques in deep learning. Despite its success, there have been very few practical applications of RL in computer vision tasks, where supervised learning is most dominant. In this thesis, we reformulate Road Extraction from Satellite Imagery as an RL problem. We aim to address some of the challenges of supervised learning methods and open the door for future works. To the best of our knowledge, this is the first time RL has been successfully applied to the complex real-world problem of road extraction, where the challenges are partially-observable, large-scale environments, and long time horizons. First, we design an environment, an action space, and a reward function that fully captures the problem as a partially observable Markov decision process. We propose a novel neural network architecture that captures the multi-modality in the input data. Then, we propose methods to address the challenges that come with the long time horizon aspect of the environment, and optimize and improve the policy efficiently. Due to the large-scale nature of the problem, we employ self-supervised representation learning to reduce the computational cost and increase the performance of the policy. We present experiments on satellite images of fifteen cities that demonstrate comparable performance to state-of-the-art methods.

Acknowledgments

My sincere thanks go to my supervisor, Professor Poullis, for all his support and guidance. I am extremely grateful for all the valuable lessons he taught me and made me become a better researcher. I would also want to thank my dear friend Shima Shahfar who helped me with some of my ideas and listened patiently to my endless talking. Lastly, I want to thank my family for their unconditional love and support throughout these years. I am certain that I couldn't have come this far without their help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Updating road maps is a very challenging and non-trivial task. Nowadays, maps of road networks are being widely used in location-finding and navigation services in mobile applications, which prompts the incentive to invest in the production of accurate maps. Many companies spend millions of dollars to create their own maps and sell them as a software as a service (SaaS) to other companies. Even though there are publicly available datasets such as OpenStreetMap (OSM), there is still a significant gap in terms of accuracy [40] to that of commercial products. In recent years, deep learning has been successfully used to produce quality maps from high-resolution satellite imagery [5, 6, 40, 53]. Historically, road extraction from satellite imagery has been formulated as a semantic segmentation problem, where the objective is to classify whether each pixel on the image is part of a road or not. Since the images can have ultra-high resolution, each image would be split into smaller patches and fed to the model to predict the semantic mask. After that, post-processing methods would be used to generate road network graphs from the semantic masks [19, 40, 60].

Recently, [5, 53] showed that the road extraction could be solved by training a decision function operating on smaller patches and iteratively constructing the graph, which produced state-of-the-art results. We believe training a decision function in a simulated environment over many trajectories can be closely modeled using a Markov Decision Process. Our motivation for using RL modeling is that it would provide better exploration during the training, which would result in a more robust agent. To our knowledge, this is the first time that RL has been used for road extraction. Prior work did not investigate RL for this problem not because it is not appropriate -road

extraction can be naturally formulated as an RL problem- but rather because of the complexity and intractability of the policy optimization. It is noteworthy that the two SOTA [5, 53], are both inspired by RL and. In particular, their processing pipeline, e.g. use of a decision function for choosing which direction to follow, resembles an RL framework and it is acknowledged in their work.

To reformulate the problem in the reinforcement learning framework, the environment, the state representation, the reward function, and the action space must be carefully designed and tested. During the experiments, we realized that solving the problem and optimizing the policy is a major challenge and requires significant computational resources. Firstly, road extraction from high-resolution satellite imagery is, by definition, a large-scale problem. The resolution of the images in the RoadTracer dataset [5] is 16384x8192 on average, which limits the application of any deep-learning-based model only on smaller patches of the input. Consequently, this makes it a partially-observable problem with a Long Horizon. As we discuss in Section 3.5, it will take at least 10,000 steps for the agent to extract the road network from a single aerial image. Thus, we can conclude that Road Extraction exhibits all of the characteristics of a large-scale problem. As such, we allocated a significant portion of Chapter 3 to propose an RL framework that reduces the computation cost and increases the convergence speed when training agents on large-scale and partially-observable environments.

In Chapter 2, we review some of the topics that are necessary to understand the methods. Chapter 3 contains our contributions to solving the road extraction problem and proposing a new framework for tractable RL. Finally, in Chapter 4 we present our conclusion and discuss the future works.

Contributions In this work, we address the following research questions:

- How can we define the road extraction problem in the reinforcement learning framework? Previous methods have tried to leverage some techniques from RL in a supervised learning setting. However, designing an RL environment from scratch has its challenges. What will the reward function be? The reward function is the only guiding signal to the agent as to how to approach the problem. Poorly designed reward functions cannot even guarantee convergence to a sub-optimal solution.

- How can we deal with partial observability of the environment? The state representation plays a vital role as all information necessary to make a decision must be supplied to the agent. The architecture of the agent's network will depend on the state representation as well. We propose a novel neural network architecture that captures the multi-modality in the input data. Can we use continuous action spaces? The implication of the type action space is the choice of the optimization algorithm.

- How to make the learning process more efficient? It is known that convergence in RL is much slower than supervised learning due to the gradual improvement in policy and requiring a considerable amount of sampled trajectories at each iteration. Furthermore, we are dealing with a large-scale problem which makes learning even more difficult. What techniques can be used to reduce the training time? It is imperative that we reduce the training time to the point that we can experiment on commodity GPUs in a reasonable amount of time. We address these issues by proposing several techniques, some of which are task-specific and the others are task-agnostic. Specifically, we propose a self-supervised loss function to reduce the training time and increase the performance of the agent.

Chapter 2

Background and Related Work

In this chapter, we'll review the basic concepts of reinforcement learning, self-supervised learning, convolutional neural networks, and road extraction, which are necessary for understanding our proposed method presented in Chapter 3.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that attempts to solve interaction-driven sequential decision-making problems. In RL, the agent is not supervised as to what action to take in each situation and there are no training examples of the correct behavior. Instead, the only feedback is a scalar reward signal that may be provided after taking an action. The objective of the agent is to learn a decision-making *policy* that maximizes these rewards during its interaction with the environment. Depending on the complexity of the problem, the reward signal can be either frequent or very sparse. Delayed rewards are challenging because the value of a decision is not known until further into the future. Another aspect to consider is that the agent's actions influence the situations it will encounter. While a particular action may have been chosen correctly in hindsight, subsequent wrong actions can render that choice ineffective. As such, assessing how much value an action can bring is difficult and depends on the agent's overall performance.

For example, consider reformulating chess in an RL framework. At each time step, we have full access to the state of the game, and all of the legal actions that can be taken at each step are known a priori. The only feedback that we expect to receive is

whether we have lost or captured a piece, or that the game is over. How would one approach this problem? Without prior knowledge or strategies, we can solve this game using RL, which shows how powerful RL can be in decision-making problems where minimal feedback is available. While producing a sub-optimal solution is possible, the challenges are non-trivial. In chess, there are at least 10^{50} positions [1] which makes it computationally impossible for the agent to see all of them during the training. Another question is how the value of an action can be estimated? The true impact of an action may not be known after a handful of steps. When should we stop the exploration and start utilizing the knowledge that we have learned? These questions form the backbone of RL and will be addressed in the next sections.

2.1.1 Bandits Problem

A classic example in Reinforcement learning is the **k-armed bandit problem** [51]. Let us assume there are k slot machines where each one has a lever, and at each time step t we can select a machine a_t and pull its lever. The probability of hitting the jackpot and receiving the reward R_t differs from one machine to another, and this probability distribution is not known to us. The objective of the learning agent is to maximize the total reward over the course of playing for T number of time steps. One way to solve this problem is that if we had an estimate of the expected reward given selecting a certain machine, we could always greedily select the one with the highest expected reward. We can define this function as:

$$Q(a) = \mathbb{E}[R_t \mid A_t = a] \tag{2.1}$$

But how can this expectation be estimated? Through trial and error, we can try each machine for some steps to have an estimate of the average reward. This is known as *exploration*. After finding the estimations, we can always select the machine that we expect to give us the higher reward. This step is referred to as *exploitation* and utilizes the learned knowledge. The balance between exploration and exploitation is one of the main challenges in RL, and all the optimization algorithms must address it in some way or another. If in a learning example we did not leverage enough exploration, we would end up with exploiting sub-optimal actions. On the other hand, if we never utilize what has been learned and get stuck with exploring different states then we would also end up with sub-optimal rewards. For most problems,

exploration and exploitation are not separate stages but go hand in hand.

2.1.2 Markov Decision Processes

Reinforcement Learning environments can be formally expressed using Markov decision processes (MDPs) [51]. An MDP M is the tuple

$$(S, A, P, R, \gamma)$$

where S is the finite state-space, A is the set of actions, $P : S \times A \longrightarrow S$ is the transition probability function which for every $(s, a) \in S \times A$ and $s' \in S$, defines the probability of going from state s to s' using action a. R is the reward function $R : S \times A \longrightarrow \mathbb{R}$, which describes the reward associated with taking action a from state s, and a discount factor $\gamma \in [0, 1]$ representing the importance of future rewards. The sequence of interactions between the agent and MDP is called a *trajectory* or a *rollout*:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots)$$

After taking action a_t the agent receives reward r_{t+1} and transitions into state s_{t+1}. The starting state is s_0. If there are terminal states in MDP, states that do not transition into any other state, then the trajectory will be finite. MDPs that always termination at some point are called *episodic*. This is common in most RL problems where at some point, we expect to reach a final state.

One important characteristic of MDPs is that they are memory-less random processes. The probability of the next state and reward only depends on the preceding state and action, and anything before that is unnecessary. Therefore, the Markov property holds for the probability transition function and the reward function:

$$P(s_t \mid s_{t-1}, a_{t-1}) = P(s_t \mid s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, s_{t-3}, a_{t-3}, \ldots) \tag{2.2}$$

$$R(s, a) = \mathbb{E}[R_t \mid s_{t-1}, a_{t-1}] \tag{2.3}$$

The solution to an MDP is a decision-making *policy* π_θ (parameterized by *theta*) that maximizes the expected return. The return for time step t is the sum of discounted future rewards after step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.4}$$

As such, the objective function maximization can be formally defined as [51]:

$$J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}[G_0] = \mathbb{E}_{s,a\sim\pi_\theta}\Big[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\Big] \tag{2.5}$$

2.2 Self-Supervised Learning

Self-supervised learning methods have gained significant traction over the past few years. The most critical challenge is always the data when it comes to supervised learning. Labeling and annotating datasets is a very costly task, and one can only go so far with the publicly available datasets. On the other hand, we have access to unlimited unlabeled data that is not being utilized: images, videos, books, etc. This is where self-supervised learning becomes important. It provides a solution to employing unlabeled data in supervised learning settings. In some methods, we can manipulate the labeled data to create new learning tasks [11, 21, 22, 42] to improve the performance on the main objective. In other cases, unlabeled data can be directly used for better representation learning, pre-training and weight initialization [10, 24]. Recently, language models such as [17, 28, 32, 38] have seen major success by pre-training on the unlabeled data. Some of the tricks used are masking random words from the text and training the model to predict those words. Another method is to predict the next word given a number of the previous words. In [21], some sequences of frames were randomly shuffled and the model was trained to classify whether the order of frames was correct. All of these methods require no extra labeling and have shown that the model can learn better representation from the data.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) or ConvNets are a special kind of neural network that use convolutional layers instead of linear layers [35]. Each convolution layer consists of kernels that are convolved with the input. A 2D convolution with kernel K with input image I is defined as:

$$O(m, n) = \sum_j \sum_i I_{(m-i)(n-j)} * K_{ij} \tag{2.6}$$

Convolutional layers are mostly used for images since they can be convolved over the image with shared parameters and as a result, they are far more efficient than the linear layers. Each kernel can be thought of as learning aggregating specific features from the image such as edges, patterns, shapes, etc. Due to the nature of convolution layers, CNNs are translation-invariant. CNN models are mostly used in computer vision tasks such as image classification, semantic segmentation, objection detection, motion estimation, etc [31, 34, 57].

2.3.1 Semantic Segmentation

Semantic Segmentation is a computer vision task of partitioning an image into different categories. Formally, it can be defined as a pixel-wise classification task where each pixel is assigned a category or label. In semantic segmentation, instances of the same category are not distinguished. For example, if we have a dog category and there are three dogs in an image, all of their pixels are labeled equally. Semantic segmentation has many use-cases such as in background and foreground separation, image filters used in mobile apps, self-driving cars, etc [41].

Figure 1: **Cityscapes.** A semantic segmentation dataset with 30 classes [14].

Loss Function

Since semantic segmentation is represented as a pixel-wise classification task, the output of the model will be a categorical distribution for each pixel. As such, we can use the categorical cross-entropy to calculate the loss term:

$$\mathcal{L}(p, y) = -\frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{c=0}^{|C|} y_{ijc} \log p_{ijc} \tag{2.7}$$

where $p, y \in \mathbb{R}^{H \times W \times |C|}$ are the predicted semantic mask and ground-truth respectively, and C is the set of classes.

Evaluation Metrics

All of the classification metrics can also be used for semantic segmentation. However, two notable metrics that are widely used are:

- **Intersection over Union (IoU)**: Also known as the *Jaccard Index* is defined as the area of the intersection between the predicted segmentation map and the ground-truth, over the area of the union between the predicted segmentation map and the ground-truth. The range of values is between $[0, 1]$ and higher valeus are better.
$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{2.8}$$
- **F1-score**: Is the harmonic mean between precision and recall. It is most useful when there's an imbalance between the distribution of the classes in the dataset.

$$F1 = 2\,\frac{Precision \times Recall}{Precision + Recall} \tag{2.9}$$

2.4 Transformers

The Transformer architecture was introduced as a successor to recurrent neural networks (RNNs) to improve the performance on the Natural Language Processing (NLP) tasks such as language understanding, machine translation, etc. While this architecture is often used in NLP tasks, it can also be used in other tasks such as computer vision [18], where we want to capture the relationships between the entities in a sequence.

The backbone of the Transformer architecture is the self-attention module called Multi-Head Attention (MHA). The idea is that in order to capture the context of the sentence, for every word in the sentence we can compare it to every other word in sentence and extract some features. Then, we can aggregate these features to have a global understanding of the sentence.

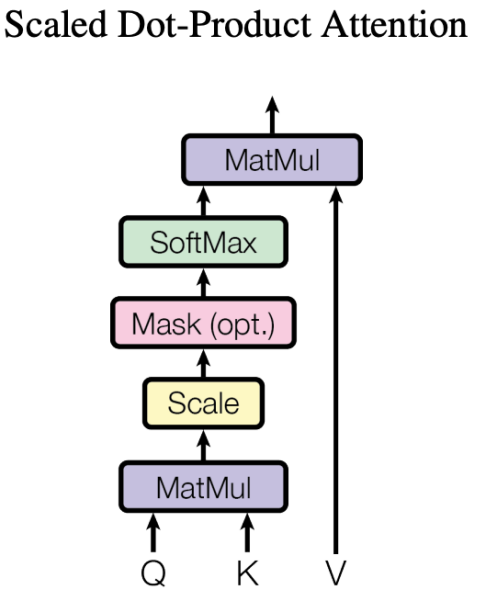


Figure 2: Multi-Head Attention Module [56].

2.5 Road Extraction

Over the recent years, there has been significant progress in the automatic extraction of road networks from satellite imagery. There are two important criteria to consider when assessing the quality of the generated road networks; Firstly, the alignment and the topology of the generated graph compared to the ground truth are important. Secondly, the connectivity of the graph also plays an important role since one of the main applications of the road networks is for navigation, and having disjoint connected components makes the task impossible. Traditionally, semantic-segmentation-based methods [6, 37, 40, 61] have been proposed to generate binary segmentation mask to segment road pixels from the background. Having the segmentation mask,

post-processing techniques are then applied to generate the road graph. [12] used morphological transformations to thin the road segments enough to map them to edges. [40] used heuristics search to connect disjoint line fragments and [37] leveraged sparse affinity maps. In [6], the authors proposed learning a model for road orientation and a model for road connectivity refinement. These methods mainly focus on addressing the connectivity issues of the inferred graphs. More recently, another category of methods that are iterative-based was able to produce state-of-the-art results [5,53]. In these methods, the idea is to train a decision model that starting from a random location on the road network, iteratively traces the near unexplored roads until it fully explores all of the roads in the city. Algorithm 1 is the pseudocode for the overall extraction process.

Algorithm 1: Iterative Road Extraction

Input: Location v
Output: Road graph
$S \leftarrow$ Initialize an empty stack
$G \leftarrow$ Initialize an empty graph
add v to S
while $S \neq \emptyset$ **do**
    $u \leftarrow$ pop from S
    if unexplored road p near u exists **then**
        add u to S
        add p to S
        Add edge (u, p) to G
return G

One major advantage of these methods is that they guarantee the connectivity of the generated graph since the algorithm iteratively expands on the current graph at each step. [5] trains a CNN model using an Oracle, which decides the next move. [53] expanded upon the previous method by adding auxiliary segmentation tasks to the model. They also introduced an exploration mechanism where the next set of moves is determined using pixel-wise classification. One of the limitations of these methods is the restrictive exploration. It is possible that during the training phase, not all

the unique trajectories are explored, and as such in the inference phase, if the model slightly deviates from the road, it will be in a scenario where it has not observed before during the training, and this may result in unexpected behavior.

2.5.1 Dataset

There are several large-scale datasets [5, 16, 54] that are used in the literature. Our focus will be on the RoadTracer dataset [5] that is well studied by the previous methods. The dataset consists of satellite images of 40 major cities such as Toronto, New York, Paris and Chicago. The images cover inner cities, urban and sub-urban areas and there is a balance between various roads such as highways, metropolitan roadways and rural roads. The imagery is available through Google at 60cm/pixel resolution, and the size of the images ranges from 8096×8096 to 24576×24576 pixel resolution, depending on the size of the city. As for the target graph, OpenStreetMap (OSM) which is a publicly available geographic database is used. Minor roads such as parking lots, short service roads and tunnels are excluded. For training and testing, the dataset is split into two sets of sizes 25 and 15 respectively.

2.5.2 Evaluation Metrics

There are several criteria to measure the quality of the inferred road graphs. Firstly, the alignment of the road segments between the inferred graph and the ground truth is important since it shows how accurately the topology of the roads are preserved. Another evaluation is the connectivity of the graph. Below, we will discuss two metrics.

Junction F1

Introduced in [5] and then refined in [53], this metric measures how well the junction vertices in the generated graph match with the ground truth for a single city-wide image. A junction is a vertex or an intersection in the road network that has more than two connections. For every junction vertex v in the inferred graph G_I, we check whether it can be matched a vertex in ground truth graph G_T. If a match exists, then we classify v as a True Positive (TP). Otherwise it will be flagged as a False Positive (FP). Similarly, for every junction vertex u in G_T, if u is unpaired

then it is considered as a False Negative (FN). Based on this definition, it will be straightforward to calculate the precision, recall and F1 terms. Precision $P = \frac{TP}{TP+FP}$ is the fraction of generated junctions that are correctly identified, and Recall $R = \frac{TP}{TP+FN}$ is the fraction of the true junctions that are correctly recovered. Given the precision-recall values, the F1 score can be calculated using $F_1 = 2\,\frac{P \times R}{P+R}$ which is the harmonic mean between the precision and recall terms.

Average Path Length Similarity (APLS)

The Average Path Length Similarity metric (APLS) [54] measures the difference of the shortest paths between every pair of vertices $(u,\ v)$ in the inferred graph G_I and ground truth G_T. If the generated graph is accurate, we expect the length of shortest path between $(u,\ v)$ in G_I and G_T to be similar. Formally, the similarity metric for two arbitrary graphs G and H is defined as:

$$S_{G \longrightarrow H} = 1 - \frac{1}{N} \sum_{u,v} min\left(1,\ \frac{|d_G(u,v) - d_H(u,v)|}{d_G(u,v)}\right) \tag{2.10}$$

Where N is number of the all unique paths in G and $d_G(u,v)$ is distance between u and v in graph G. If a path doesn't exist H, then for the value of inside the summation, the value of 1 will be considered which is the maximum error for a path. Given this definition, APLS for a single city-wide image is defined as the harmonic mean of the similarity values $S_{G_I \longrightarrow G_T}$ and $S_{G_T \longrightarrow G_I}$:

$$APLS = \left(\frac{S^{-1}_{G_I \longrightarrow G_T} + S^{-1}_{G_T \longrightarrow G_I}}{2}\right)^{-1} = \frac{2}{\frac{1}{S_{G_I \longrightarrow G_T}} + \frac{1}{S_{G_T \longrightarrow G_I}}} \tag{2.11}$$

Chapter 3

Tractable Deep Reinforcement Learning

This chapter is based on the conference paper "Tractable Large-scale Deep Reinforcement Learning", Nima S., Poullis C., Under review.

3.1 Abstract

Reinforcement learning (RL) has emerged as one of the most promising and powerful techniques in deep learning. The training of intelligent agents requires a myriad of training examples which imposes a substantial computational cost. Consequently, RL is seldom applied to real-world problems and historically has been limited to computer vision tasks, similar to supervised learning. This work proposes an RL framework for complex, partially observable, large-scale environments. We introduce novel techniques for tractable training on commodity GPUs, and significantly reduce computational costs. Furthermore, we present a self-supervised loss that improves the learning stability in applications with a long-time horizon, shortening the training time. We demonstrate the effectiveness of the proposed solution on the application of road extraction from high-resolution satellite images. We present experiments on satellite images of fifteen cities that demonstrate comparable performance to state-of-the-art methods. To the best of our knowledge, this is the first time RL has been applied for extracting road networks.

3.2 Introduction

In recent years, reinforcement learning (RL) has seen significant success in research and applications primarily due to the introduction of deep neural networks as function approximators [43], from robotic tasks [62] to complex games such as Dota 2 [8] and AlphaGo [49], where world champions were defeated. Despite this progress, there are significant challenges that hinder the application of RL in real-world applications, especially when compared to supervised learning (SL). The main challenge is that RL agents require more data, considerable training and are more complex to design than SL methods. For example, training the OpenAI agent required 5000 GPUs and AlphaGo agent 5000 TPUs. The hide-and-seek agents were trained on 31 billion frames of data [2].

Offline RL methods [36] have attempted to improve efficiency, but there is still a significant gap between SL and RL methods. Hence, SL is the preferred framework in cases where the problem can be expressed as a mapping from one domain to another. In contrast, RL is primarily used for research to train agents on simple, well-defined tasks.

In this paper, we demonstrate that with careful design considerations, RL can successfully solve real-world problems involving complex, partially observable, large-scale environments using commodity GPUs. We demonstrate the application of our novel solution on the real-world problem of road extraction from high spatial resolution (HSR) remote sensing data, where all the solutions to date employ procedural techniques or SL methods, such as semantic segmentation with deep neural networks. Major challenges in road extraction are the considerable imbalance between the background and foreground, occlusions, high resolution of the images, and preservation of the topology of the road segments.

Specifically, our technical contributions are,

1. An RL framework for training agents in complex, partially observable, large-scale environments using commodity GPUs.
2. The design of an RL environment based on the road extraction, with reasoned design decisions involving the state representation, action space and reward structure.

3. A self-supervised regularization loss function that improves the learning stability of the model and the efficiency during training.

Paper organization. The paper is organized as follows: Section 3.3 gives a brief overview of the recent progress on improving the efficiency of deep reinforcement learning techniques and a brief review of the most relevant work in road extraction. In Section 3.4 we present how we reformulate road extraction as an RL problem and provide details on all the main components, including the environment, reward, policy optimization, and training scheme. Finally, we present our experimental results in Section 3.5 followed by the conclusion and future work.

3.3 Related Work

3.3.1 Tractable Reinforcement Learning

There have been several attempts to improve the efficiency of policy optimization in RL. Rainbow [25] combined several enhancements to DQN [43] such as Multi-step learning [51] and Distributional RL [7] to increase both the performance and the learning speed of the policy. [27] introduced distributed prioritized experience replay to gather experiences from multiple actors, and [58] proposed applying augmentation on the input to get a more accurate estimate of the Q-values. [45] proposed decoupling the learning representation from the RL objective to reduce the learning complexity.

3.3.2 Road Extraction

In recent years, with the success of CNNs, several semantic segmentation-based methods formulated the problem as a supervised pixel classification where road pixels are separated from the background. Consequently, post-processing techniques need to be applied to generate topological networks. [61] used Dilated Convolutions to capture the global information without sacrificing the resolution. [37] improved the feature aggregation flow by generating sparse affinity maps, and [6] proposed learning the road orientation and a model for the road connectivity refinement. One limitation of this formulation is that incorporating information about the spatial relation of pixels is non-trivial. As such, the primary focus in these methods has been on the

model's architecture to incorporate the interdependencies of the pixels and preserve the topology of the road segments.

Another category of methods focus on the iterative extraction of the road network [5] where the objective is to output the road network as a graph, which is its natural representation. Starting from a small patch of the aerial image, a CNN-based decision function predicts the direction of the move to the neighbour patches and extends the graph. [5] trains the CNN model using an Oracle, which decides the next move. [53] introduced an exploration detector where the set of its next moves are predicted by the model using pixel-wise classification. One of the limitations of these methods is the restrictive exploration. At each step, the next move for the decision function is generated on the fly based on the ground-truth graph. During the inference phase, if the model slightly deviates from the road, it will face an unfamiliar scenario that has not been observed before during the training. This error can accumulate and have detrimental effects on the tracking the road. [5] attempted to address this using graph matching techniques. In contrast, [53] uses segmentation and junction cues. We argue that training the model in an RL framework addresses this issue where the model is guided only by rewards based on how close it is to the road, and exploration and exploitation can be tuned or incorporated into the policy. The agent will have explored many different trajectories during every policy iteration and becomes better at each step. This formulation also gives us the ability to specialize in corner cases where roads are close to each other or occlusion occurs.

Figure 1: **Challenging cases.** Dealing with occlusions caused by shadows, trees, tall buildings and overpasses poses a significant challenge to RL and SL methods alike.

3.4 Method

In this section, we describe the iterative graph construction algorithm [5] and explain how we formulate this in an RL framework. Furthermore, we propose a set of policy improvement techniques, including a self-supervised regularization loss function.

3.4.1 Formulation

The topological structure of road networks can be naturally represented as a graph where the nodes correspond to intersections and junctions, and the edges correspond to straight-line road segments. We reformulate the road extraction problem as an iterative graph construction, which intuitively can be considered analogous to a Depth-First Search (DFS). Starting from a node corresponding to a road location, the agent moves to a neighbouring location after choosing an action from a finite set of actions, and thus, iteratively constructs the graph representing the road network. The agent maintains a record of the visited nodes to avoid revisitation. In case of a dead-end road, the agent can backtrack to its previous location and explore other unvisited neighbours, if any. The objective is for the agent to return to its starting location after having visited all roads.

We design the RL environment in way that each environment is tied to a single aerial image associated with a city. In what follows, we describe the components of the environment.

3.4.2 State Representation

The state is represented as aerial images of urban areas. Due to the large-scale nature of the data, the agent has access to only a tiny patch of the aerial image at each step. Therefore, in this context, the state refers to the approximation of the state of the system. A state is associated with an actual location $c = (x, y)$, representing the agent's location on the global scale. The state consists of an RGB image patch $I_c^{W \times H}$ of size $W \times H$ centred around c. To approximate the information of the observation history, we annotate the agent's previous movements onto the patch, which provides information about the overall trajectory of the movement. Additionally, the last L actions A^L of the agent are included as part of the current state. Hence, the state is

defined as,

$$S =< c, I_c^{W \times H}, A^L >$$

Our emphasis on including multiple sources for providing temporal information is that occlusion by tall building and trees (Figure 1) is challenging and can lead the agent to misidentify a dead-end and leave a whole area unexplored. Finally, to decouple the representation learning from RL, we trained a separate segmentation-based model to generate road and junction masks, which are added as features to the state representation.

3.4.3 Action Space

Two distinct actions must be considered to replicate the DFS behaviour: a) transitioning to the next location, i.e. to neighbouring vertices on the graph (Figure 3), and b) backtracking to the previous location when reaching a dead-end (Figure 2). Consequently, we define the action space $\mathcal{A}$ as having two discrete components $\mathcal{A} =< a_b, a_d >$, where a_b is a binary flag indicating whether a backtrack action should occur, and a_d is the direction of the next movement expressed in degrees i.e. $a_d \in [0, 2\pi]$. If the agent chooses to backtrack, then the value of a_d is ignored. Otherwise, the agent moves forward in the direction of a_d with step-size s.

3.4.4 Reward Function

Our approach requires a substantial number of steps (>10000) to capture all roads. Due to this (very) long time horizon, we adopt potential-based reward shaping functions [44] to reduce the reward horizon. Different reward functions can produce similar results; however, the computational cost for training can differ significantly. The reward function must also be designed to maximize the RL objective and construct an accurate road graph. The rewards act as a guide for the agent when constructing the road network graph. Based on the action's component a_b, two different scenarios are considered: the agent performs a) a regular movement or b) backtracks.

Movement Reward.

When transitioning to a next location, the reward function $r_{movement}$ is defined as,

$$r_{movement} = r_{align} + r_{divergence} + r_{revisit} \tag{3.1}$$

Figure 2: **Backtrack action.** Four ordered frames taken from the same sequence. The orange lines show the ground-truth road network. Visited locations are drawn in white, and visited locations from which the agent backtracked are drawn in green. The blue cross marks the agent's location. In frame 1, the agent reaches the image border and decides to backtrack for several subsequent steps, shown in frame 2. Backtracking continues until another viable action becomes available. Frame 4 shows the agent turning right when reaching the junction to explore unvisited roads.

where r_{align} penalizes for deviations from the road while encouraging it to move in the right direction and is given by,

$$r_{align} = \alpha \times (-d_{agent}) + \beta \times l_{traversed} \tag{3.2}$$

d_{agent} is the distance between the agent and the closest location on the road, thus the goal is to minimize this distance or equivalently maximize $-d_{agent}$. $l_{traversed}$ is the length of the traversed road in the previous movement and is in the range of $[0, s]$

depending on the angle between the vector of movement and the road segment. α and β are weighting hyperparameters. If $\alpha = 0$, the agent cannot recover from a bad position to return closer to the road. If $\beta = 0$, then the agent will tend to move in a zig-zag fashion since the priority would be to only minimize the distance. $r_{divergence}$ is a penalty when moving farther away from one road segment and closer to another. Although performing this move increases the $-d_{agent}$ penalty -which is preferred- the penalty is negligible in cases like highways where the lanes are very close to each other. Figure 3 shows a visualization of the reward landscape.

Backtrack Reward.

The backtrack action must only be used when reaching a dead-end where no roads are left unexplored, and therefore no available options. The reward for this action is defined as,

$$r_{backtrack} = \begin{cases} -d_{agent} & \text{if backtracking is justified} \\ r_{bt\_penalty}, & \text{otherwise} \end{cases} \tag{3.3}$$

If the backtracking action is correct, the agent is given a reward of $-d_{agent}$. Based on our experiments, giving a fixed positive reward leads to abuse the backtracking action. As such, the reward is proportional to the distance of the agent from the road.

Figure 2 shows an example of the backtracking action in four ordered frames taken from the same sequence. The orange lines show the ground-truth road network, visited locations are drawn in white, and visited locations from which the agent backtracked are drawn in green. The blue cross marks the agent's location. Frame 1 shows the agent reaching an image border and after that in frame 2 choosing the backtracking action for several steps. At frame 3, it reaches the junction and chooses to turn right to explore unvisited roads.

3.4.5 Learning

To learn the policy, we adopt Double Q-learning (Double DQN), which is a model-free off-policy algorithm [55]. Off-policy algorithms are sample-efficient compared to on-policy since generated experiences can be re-used several times at different stages

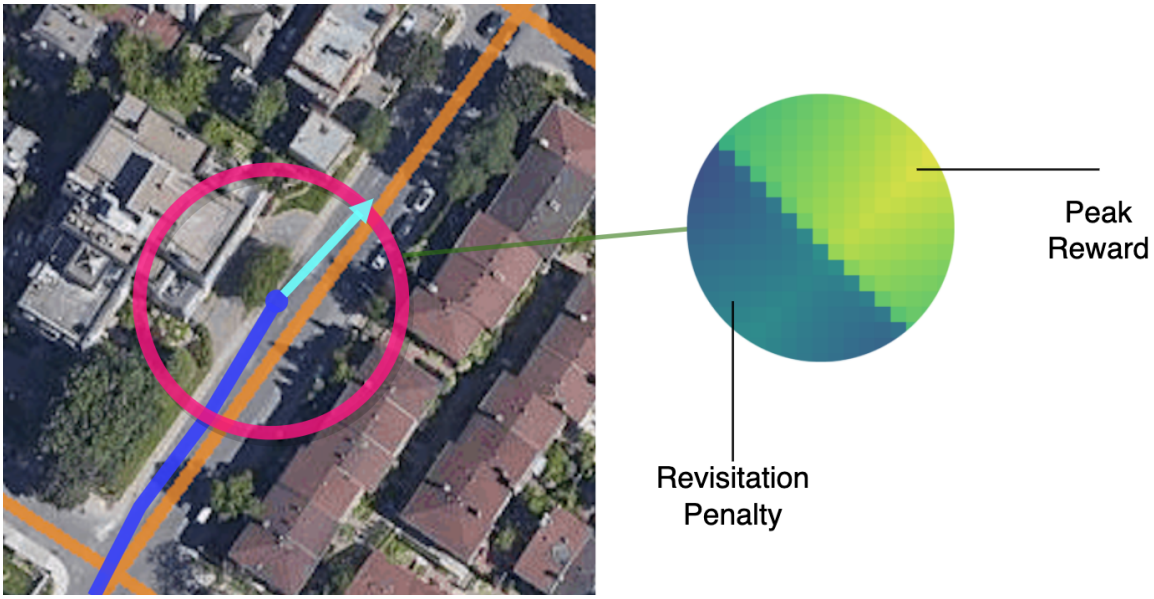


Figure 3: **Reward Landscape.** The orange lines show the ground-truth road network. The blue line shows the previously visited locations. The agent's location is in the center of the purple circle. The right image shows the reward for the area within the purple circle, rendered as a colour-mapped surface. Moving backwards would result in a negative reward. In contrast, moving forward in a straight line will result in the maximum reward.

of policy evaluation. Moreover, Double DQN is a simple method with few hyper-parameters, reducing the computational cost for fine-tuning. Moreover, in Double DQN bootstrapping is naturally used for the Temporal Difference (TD) update rule, on the other hand, partial bootstrapping should be used for the on-policy algorithms due to the long time horizon of the problem.

Network Design

One factor in the design of the architecture is dealing with the multi-modality in the input space. To learn the image representation, we use EfficientNet-V2 [52] since it provides a good trade-off between speed and accuracy. We use the Transformer encoder with Multi-head Attention to embed action indices. Learning temporal information regarding the previous action is useful since, most of the time, we have repeated actions when the agent is moving in a fixed direction or backtracking from a dead-end location. To extract the representation, the class-token technique [18] is

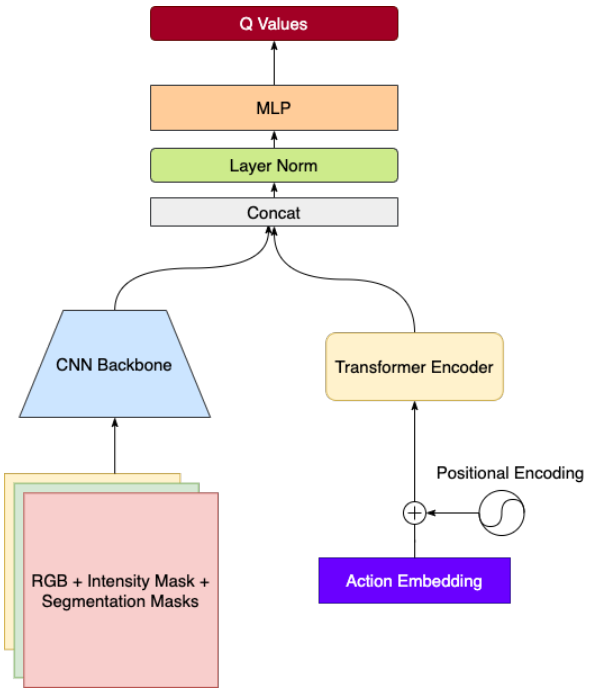


Figure 4: **Network Architecture.** The input consists of the RGB aerial image, the intensity mask, and the semantic segmentation mask. The CNN backbone is based on EfficientNet-v2 [52] and we use the Transformer encoder with Multi-head Attention [56] to embed action indices.

used to fuse information from all of the inputs. We use LayerNorm to normalize the activation statistics when fusing the image and action representations.

To learn the segmentation and junction masks, we used DDRNet [26]; a lightweight architecture specialized for road scenes. We modified this network to add BlurPool [59] for anti-aliasing purposes since road masks are tiny, and essential information (especially at the center where the agent is located) could be lost during the down-sampling layers. To train the model, we use cross-entropy with the addition of clDice [48] to emphasize the skeleton and the road segments' topology.

3.4.6 Policy Optimization

Distributed Data Collection. We leverage a distributed data collection pipeline similar to [27]. Several environments are initialized with the aerial image of a unique city and each environment has an experience replay buffer where the gathered data is stored. This distributed pipeline significantly increases the diversity of experience data and has also been shown to increase the performance in the learning phase [27, 33]. Additionally, we use a priority replay buffer where experiences with higher TD error are more likely to be sampled during the optimization phase.

Long Time Horizon.

One of the main challenges of this type of vision task is the long time horizon. Due to the scale of the problem, it can take at least 10,000 steps for the agent to extract the complete graph. In the worst-case scenario, the agent will aimlessly wander without exploring any new area. Due to the size of the observation space and memory constraints, we can store at most 9,000 observations. As a result, it is imperative to introduce new termination conditions that would end a *bad* episode prematurely. For example, if the agent gets an average reward below a certain threshold for the last n actions, we terminate the episode. This termination threshold must be tuned. If the threshold is low, it produces low-quality experiences where the agent is far off the track. In contrast, if it is set too high, it prevents exploration. Adding the early termination mechanism introduces a side-effect: the expected length of an episode depends entirely on the agent's performance. Therefore, having long-length episodes becomes improbable since if the agent deviates from the road at any point, the episode will terminate. Consequently, having experiences from later stages of the extraction process becomes rare. For instance, backtracking can only be invoked when the agent reaches a dead-end which can take a while for the agent to reach those areas from its starting location. We address this by adding an *Undo* and *Resume* functionality to the environment where at the end of a terminated episode, with a given probability, we entirely reverse the changes of the last m steps and resume from there as the starting point of the next episode, thus, the agent does not have to start from scratch again. One issue with adding the resume functionality is that the samples stored in the buffer will become highly correlated if the agent fails at a certain spot several times. However, since we are using a prioritized replay buffer with a high degree of prioritization, samples with high errors are more likely to be used in the optimization step, making it unlikely that the agent would get stuck in an area for an extended period.

Self-supervised Loss. Inspired by self-supervised methods [10,11,22], we introduce an auxiliary self-supervised loss that would make the agent more robust to perturbations in the input by augmenting the data and minimizing the difference between the output of the model on the augmented data and the output on the unperturbed data. Formally, the loss is given by,

$$L_{sl} := \|q_{\oslash}(s) - r(q(t(s)))\|_2^2 \tag{3.4}$$

Figure 5: **Bad episodes.** This figure shows that the agent can easily wander around and move-in circles in the early episodes.

Where q is the agent, s is the state, and t and r are transformation functions. Specifically, t can be a spatial transformation such as rotation, random shift, or a pixel augmentation such as adding noise or blurring. In the case this changes the orientation of the actions, the reverse transformation r in combination with t must be applied to compensate for the changes. The $\oslash$ symbol means that the gradient flow from $q(s)$ is stopped. In contrast to [11], we do not use a predictor on top of the model. Instead, to avoid collapsed solutions, we apply the self-supervised loss once for every k optimization iterations on the main RL objective, where k is a hyperparameter.

3.5 Experiments

3.5.1 Dataset

We evaluate our method using the dataset introduced by [5]. The dataset consists of aerial images from 40 cities worldwide, collected using the Google Maps API at a 60cm/pixel resolution. The road network is collected and processed from OpenStreetMap using OSMnx [9], enabling us to access information such as the type and

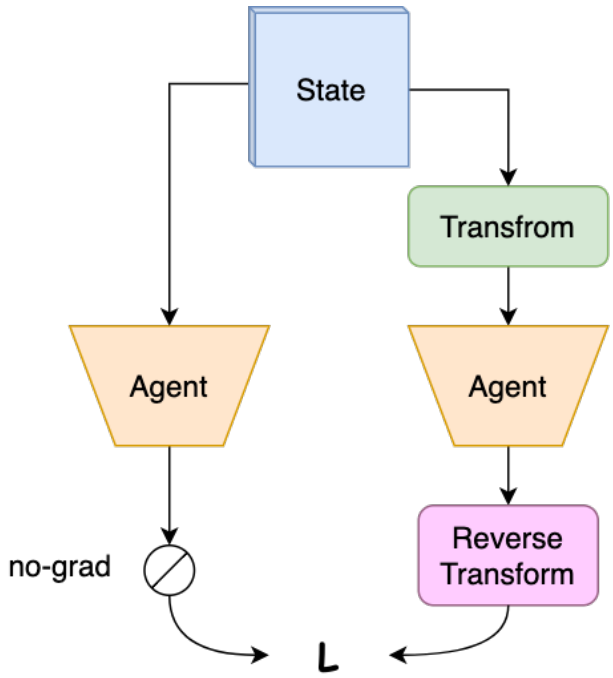


Figure 6: **Self-supervised loss function**

width of the road. The dataset is split into 25 and 15 images for train and test, respectively.

3.5.2 Evaluation metrics

We evaluate our method based on two sets of metrics. The first set is based on the obtained reward and characteristics of the episodes during training and testing. Specifically, we are interested in the average reward and the episode's length; the average reward is a good measurement of the alignment of the extracted graph with the ground truth, and episode length shows how far the agent went without triggering the stopping conditions.

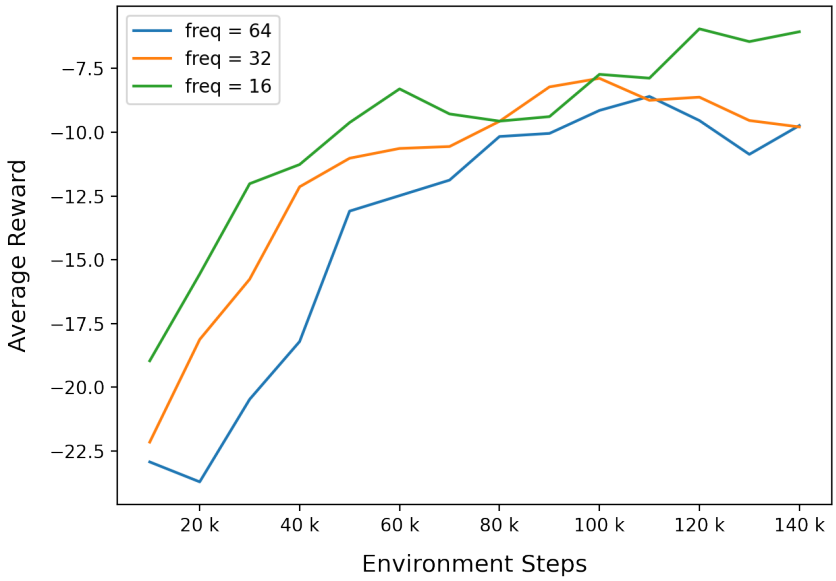
The second set is to measure the overall quality of the final graph. We use the junction-metric [53] to measure the accuracy of the junction, and APLS [20] for the connectivity of the graph.

3.5.3 Design decisions

Designing the environment in tandem with the policy and the optimization techniques is a significant challenge. Design choices relating to the environment include the action space, reward function and space representation. The type of action space directly influences the choice of the policy algorithm. We experimented with both on-policy and off-policy methods and experimentally confirmed that off-policy methods are indeed more efficient. Early experiments primarily used DQN [43] as the baseline, and the smaller number of hyperparameters made it easier to focus on the design of the environment. A full list of hyperparameters is provided in Appendix B.

3.5.4 Ablations on Self-supervised Loss

We measure the impact of the self-supervised loss with different frequency updates, while other hyperparameters remain fixed. Figure 10 shows that the agents gain on average a higher reward per episode which implies that the alignment quality of the road network has increased. Furthermore, there is a noticeable relationship to the TD loss.



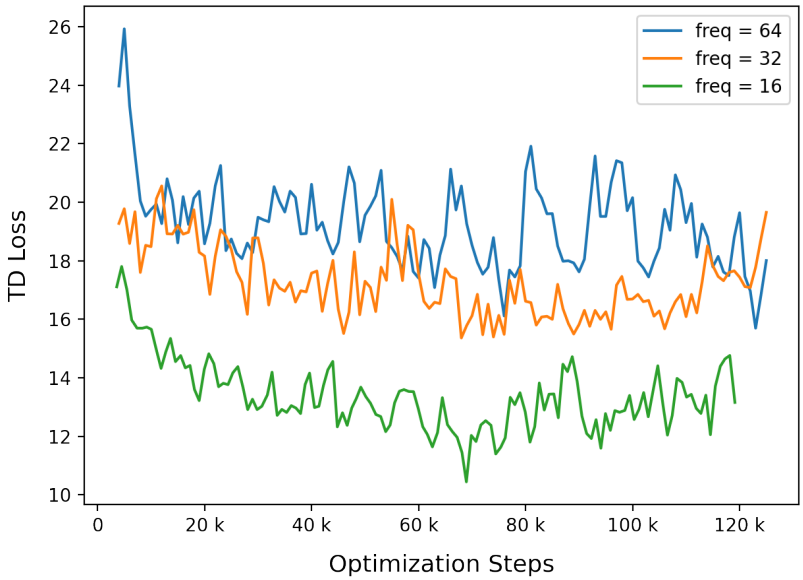


Figure 7: **Self-supervised Experiment.** The orange, gray and green curves correspond to the experiments where the self-supervised loss was applied every 16, 32, 64 steps respectively. Looking TD error and average reward, we can conclude that increasing the self-supervision effect directly increases the performance of the agent.

3.5.5 Early Termination

One limitation of the iterative construction design (Section 3.4) is that backtracking causes the agent to move to its previous location and not necessarily where it wants to. If the previously taken actions were not optimal, for example, moving around in circles and going back and forth, it would result in the agent repeating all those movements while backtracking. This makes it challenging to extract essential temporal information (intensity mask and action embedding) and degrades the effectiveness of the optimization updates. Thus, it is necessary to end an episode early if the average reward received is below a certain threshold. Figure 5 shows examples where the agent got trapped in certain areas and could not escape.

3.5.6 Reward Function

We used several shaping functions in our reward function which needed several experiments to be fine-tuned. To facilitate the design process, we developed a rendering tool to visually observe if the reward cues are working as expected. Surprisingly, compared to the evaluation metrics we already had, this gave us more information as to how change the function, and in some cases help us find out bugs in our code. Figure 8 is a frame of an episode that gives information regarding the quality of the graph, changes in the distribution of the Q-values and the history of movements. A summary of our experiments with the reward functions is provided in Appendix C.

3.5.7 Comparisons with state-of-the-art results

We compare our method to state-of-the-art road extraction methods [5, 6, 40, 53]. Table 1 shows the results of the comparisons. Our proposed RL method can achieve competitive results compared to the state-of-the-art supervised-learning techniques. Due to memory constraints, we could only train our agent on 40% of the training data without post-processing. We believe gearing towards more complex policies such as SAC [23] for the continuous control part of the problem could yield better results.

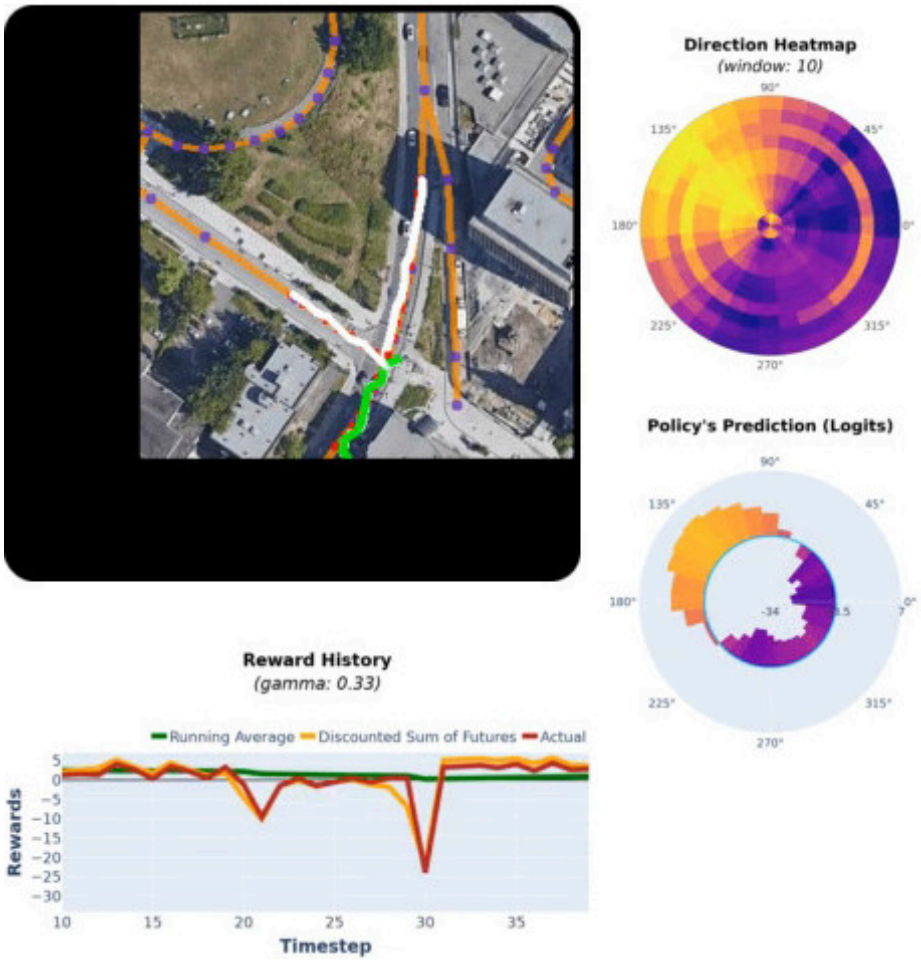


Figure 8: **Episode visualization.** The agent is moving towards the left boundary. The top-right heatmap shows the agent's preference in moving towards a direction based on the choices made in the previous ten steps, i.e. the preferred direction is towards 135 deg. In the bottom-right heatmap, we visualize the Q-values of the last timestep corresponding to each direction. As shown, the agent's prediction of the reward is significantly smaller for moving backwards -since this would incur a revisitation penalty- compared to moving forward. The bottom chart shows the value of the rewards, i.e. running average (green), discounted sum of futures(yellow), and actual(red), as a function of time. The orange lines and purple points show the ground-truth road network. Visited locations are drawn in white, red points indicate the visited locations' projections on the closest ground-truth road segment, and visited locations from which the agent backtracked are drawn in green. Sample videos of the training of the RL agent are included in the supplemental material.

Method	JF-1	APLS
DeepRoadMapper [40]	29.05	21.27
RoadTracer [5]	52.19	-
ImprovedConnectivity [6]	55.21	56.89
VecRoad [53]	63.13	64.59
Ours	53.25	55.97

Table 1: **Quantitative comparison of JF-1 and APLS with state of the art methods.**

3.6 Conclusion

We present an RL framework for complex, partially observable, large-scale environments and introduce novel techniques for tractable training of RL agents on commodity GPUs. We demonstrate the effectiveness of our approach and its significant reduction of computational costs on road extraction. Road extraction from aerial images is reformulated as an RL problem where the agent iteratively constructs a road network graph by traversing road locations on the image. Our self-regularization loss addresses instabilities typically caused by the long-time horizon. Furthermore, we discussed several design considerations, including introducing the backtracking action for training the RL agent, and presented experiments of their effect on the agent's learning. Our extensive experiments showed that although we addressed a significantly more complex learning task using RL, we achieved comparable performance with state-of-the-art SL methods on the extracted road networks' accuracy using commodity GPUs. To the best of our knowledge, this is the first time RL has been successfully applied to the complex real-world problem of road extraction, where the challenges are partially-observable, large-scale environments and long-time horizons. In the future, we intend to explore the use of an additional multi-task semantic segmentation loss. Thus, the CNN backbone will be conditioned on both taking actions for the RL agent and classifying pixels into road/non-road pixels, which we believe will improve the accuracy of the results and further reduce the computational cost of the training.

3.7 Appendix

3.7.1 Discrete and Continuous Action Representation

To further improve the performance and training cost of the agent, we experimented on representing the a_d action component as a continuous space. This would enable us to employ more advanced policy-optimization algorithms such as SAC [23] that are more stable and efficient compared to the Double DQN. Since the SAC is designed only for continuous spaces and we have both discrete and continuous components, some modifications are needed. We follow [13, 15] to design a Hybrid-SAC suitable for hybrid action spaces and the conditional entropy of $a_d|a_b$ is also added to the original entropy term. The architecture of the actor and critic models is similar to Figure 4, and we followed the same process as [58] for weight sharing and frequency of the critic's updates.

In our experiments, SAC failed to produce any meaningful results. At some point in the training, the policy would become deterministic and only choose the backtrack action, even though that resulted in a very high penalty. Our analysis showed that the magnitude of the local gradient coming to the actor through the critic would eventually converge to zero. As a result, the actor gets tiny gradient updates and cannot recover from that state. We tried many methods proposed in the literature to alleviate the issue, such as using orthogonal initialization [29], increasing the actor's learning rate, and changing the effect of the entropy term. However, this only delayed the collapse and did not solve the issue entirely. We believe there are still possible ways to address this, such as using orthogonal regularization [4] to preserve the gradient magnitude and using biases for the discrete component.

3.7.2 Implementation Details

Table 2 summarizes the hyperparameters employed by our proposed solution.

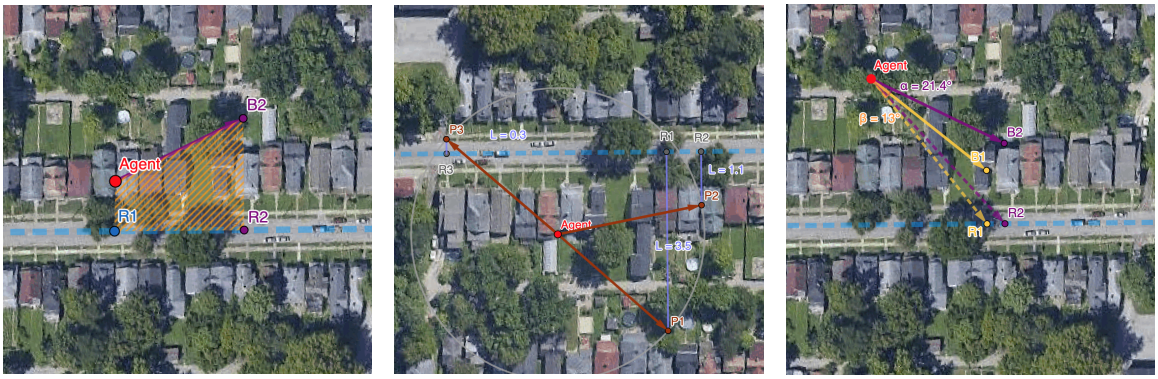
Component	Hyperparameter	Setting(s)
Action Scheme	Step-size s	10
	Number of directions a_d	16 / 32
Reward Function	α	2
	β	1
	$r_{divergence}$	-35
	$r_{revisit}$	-25
	$r_{bt\_penalty}$	-55
	Clipping	$[-55, \infty]$
Early Termination	Reward Window	10
	Threshold	-5
State Representation	$W \times H$	256×256
	L	10
Undo & Resume	Undo steps m	15
	Full-reset probability	0.1
Network	Action embedding size	16
	MLP layers	128 x 3
	Feature vector size	96
Experience Buffer	Capacity	9000
	Prioritization α	0.8
	Importance-sampling β	$0.4 \rightarrow 1$
Policy	Discount γ	0.75 / 0.8
	Target update freq	25
	n_step	3
	ϵ-greedy	$0.3 \rightarrow 0.05$
	Regularization freq	0.125
	Update per step	0.2

Optimizer	Adam	
	Learning rate	0.0005
	Weight decay	0.0001
	Epsilon	1e-8
	Batch size	128
	add	

Table 2: **Hyperparameters.**

3.7.3 Reward Functions

We experimented with many reward functions for the r_{align} reward (Section 3.5.6, where the objective is to keep the agent as close to the road as possible. The reward functions were based on different characteristics such as area (Figure 9a), angle (Figure 9b), and distance (Figure 9c). The *Area Reward* is a penalty that is given based on the area between the agent's vector of movement and the corresponding road segment. The *Angle Reward* is a penalty defined as the negative of the angle between the agent's vector of movement and the optimal direction of movement. Finally, the *Distance Reward* is the negative of the distance between the agent and the road. One side effect of the *Area reward* is that the penalty decreases as the agent moves orthogonal to the road since the area in-between shrinks. This is an undesirable property since we want to keep the agent close to the road. As such, this type of reward must be coupled with traversal rewards such as l_{length} discussed in Section 3.5.6. Our experiments found this type of reward to be very unstable, as it can vary significantly between different actions. One disadvantage of the *Angle Reward* that we found is that irrespective of what action is taken, it does not differentiate between the cases where the agent is very close to the road or is very far from it. We desire that the agent should receive a much higher penalty when it is farther from the road. As such, we chose the *Distance Reward* as one of the shaping functions used in the reward function.



(a) (b) (c)

Figure 9: **Reward Functions.** (a) Area Reward, (b) Distance Reward, (c) Angle Reward. In (a), the agent chose B2 as its next step of movement, and the negative of the area between the agent's vector of movement and the vector of the corresponding points on the road is given as the penalty. In (b), P1 to P3 are the three possible choices for the agent's next move, alongside the distances of the destination points to the road. In (c), B1 and B2 are two possible choices for the next step, and the angle between the vector of these movements and the optimal directions are calculated. In this type of reward function, the objective is to minimize the angle.

3.7.4 Interpretability Analysis

We performed several interpretability analysis using [50] to visualize how much each pixel in the image contribute to the final prediction. We took advantage of this to find the optimal input resolution and reduce the computational and storage costs.

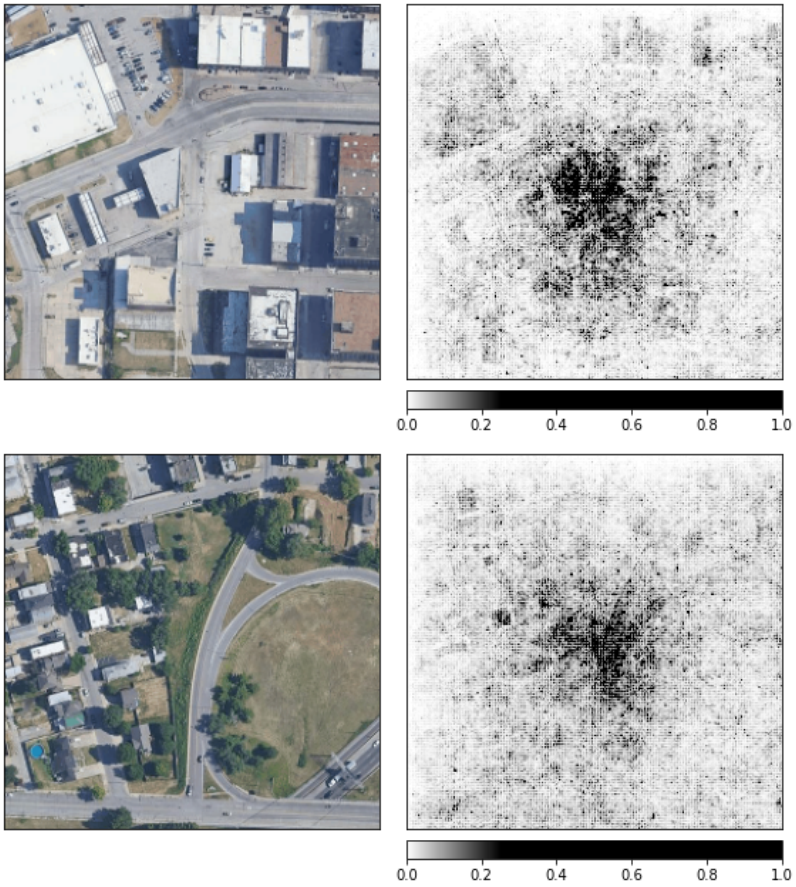


Figure 10: **Integrated Gradients.** Attributing the prediction of the CNN backbone to its input features. The integrated gradients are normalized between $[0, 1]$. Higher value means large contribution.

3.7.5 Additional Quantitative Results

Table 3 summarizes the evaluation of our method on each city in the test set.

City	JF-1	APLS
Amsterdam	46.72	47.16
Boston	53.29	56.96
Chicago	56.52	60.17
Denver	52.36	58.76
Kansas City	54.72	58.84
Los Angeles	59.16	55.46
Montreal	54.02	58.40
New York	53.48	57.48
Paris	54.61	57.63
Pittsburgh	48.83	50.71
Salt Lake City	52.49	50.72
San Diego	49.15	52.69
Tokyo	54.09	55.64
Toronto	52.81	57.76
Vancouver	56.42	61.09

Table 3: **Quantitative comparison of JF-1 and APLS for each city.**

Chapter 4

Conclusion

In this thesis, we introduced a reinforcement learning solution to road extraction from high-resolution satellite imagery. We designed an RL environment from scratch and addressed the challenges of the task such as partial observability, large-scale input, long-time horizon, and high computational cost through design choices and novel techniques that makes it possible to perform tractable training of RL agents on commodity GPUs. We designed and tested several reward functions based on reward shaping to reformulate the iterative graph construction algorithm. The agent's novel neural architecture is capable of learning from both image and categorical features using CNN and NLP methods. Our proposed self-regularization loss results in better and more stable representation learning without requiring additional data or extra supervision. Furthermore, we discussed several design considerations, including a novel backtracking action for training the RL agent and presented experiments of their effect on the agent's learning. We demonstrate the effectiveness of our approach and its significant reduction of computational costs on road extraction. Our experiments showed that although we addressed a significantly more complex learning task using RL, we achieved comparable performance with state-of-the-art supervised learning methods on the extracted road networks' accuracy using commodity GPUs. To the best of our knowledge, this is the first time RL has been successfully applied to the complex real-world problem of road extraction.

Possible future directions and extensions of this work are as follows:

- To improve the accuracy of the generated maps, we intend to explore the use

of an additional multi-task semantic segmentation loss. Thus, the agent will be conditioned on both estimating the action values for the RL objective and classifying pixels into road/non-road pixels, which we believe will improve the accuracy of the results and further reduce the computational cost of the training.

- In Section 3.7.1 we leveraged SAC, a more complex off-policy algorithm, that supports both hybrid action spaces and better exploration. However, the designed pipeline failed to converge to a solution. In our experiments, the critic that is responsible for estimating the q-values, always diverged at some point during the training. We hypothesize that it may be possible to address this issue by using a similar, but on-policy algorithm, like PPO [47]. The main difference between the critics in SAC and PPO is that in PPO, the critic is responsible for estimating the advantage of the action, as opposed to its actual value. Using advantage functions was shown to greatly reduce the variance of estimation [46]. We believe this can lead to a more stable training process.

- Our framework and design choices can be extended to other applications of deep learning in remote sensing such as weather forecasting, land cover mapping, and object tracking [3, 30, 39]. We believe using an RL solution can be more appropriate when designing a supervised learning solution is non-trivial.

Bibliography

[1] Victor Allis. *Searching for Solutions in Games and Artificial Intelligence*. University of Limburg, Maastricht, 1984.

[2] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2020.

[3] John E Ball, Derek T Anderson, and Chee Seng Chan Sr. Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community. *Journal of applied remote sensing*, 11(4):042609, 2017.

[4] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? *Advances in Neural Information Processing Systems*, 31:4261–4271, 2018.

[5] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4720–4728, 2018.

[6] Anil Batra, Suriya Singh, Guan Pang, Saikat Basu, CV Jawahar, and Manohar Paluri. Improved road connectivity by joint learning of orientation and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10385–10393, 2019.

[7] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[8] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[9] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[10] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[11] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[12] Guangliang Cheng, Ying Wang, Shibiao Xu, Hongzhen Wang, Shiming Xiang, and Chunhong Pan. Automatic road detection and centerline extraction via cascaded end-to-end convolutional neural network. *IEEE Transactions on Geoscience and Remote Sensing*, 55(6):3322–3337, 2017.

[13] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[15] Olivier Delalleau, Maxim Peter, Eloi Alonso, and Adrien Logut. Discrete and continuous action representation for practical rl in video games. *arXiv preprint arXiv:1912.11077*, 2019.

[16] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raskar. Deepglobe 2018: A challenge to parse the earth through satellite images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 172–181, 2018.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[19] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[20] Adam Van Etten, Dave Lindenbaum, and Todd M. Bacastow. Spacenet: A remote sensing dataset and challenge series. *CoRR*, abs/1807.01232, 2018.

[21] Basura Fernando, Hakan Bilen, Efstratios Gavves, and Stephen Gould. Self-supervised video representation learning with odd-one-out networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3636–3645, 2017.

[22] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic

actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[24] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[25] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[26] Yuanduo Hong, Huihui Pan, Weichao Sun, Yisong Jia, et al. Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*, 2021.

[27] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

[28] Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. In *International Conference on Machine Learning*, pages 4411–4421. PMLR, 2020.

[29] Wei Hu, Lechao Xiao, and Jeffrey Pennington. Provable benefit of orthogonal initialization in optimizing deep linear networks. In *International Conference on Learning Representations*, 2019.

[30] Ali Jamali. Land use land cover mapping using advanced machine learning classifiers: A case study of shiraz city, iran. *Earth Science Informatics*, 13(4):1015–1030, 2020.

[31] Huaizu Jiang and Erik Learned-Miller. Face detection with the faster r-cnn. In *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*, pages 650–657. IEEE, 2017.

[32] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[33] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[35] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[36] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[37] Xiangtai Li, Hao He, Xia Li, Duo Li, Guangliang Cheng, Jianping Shi, Lubin Weng, Yunhai Tong, and Zhouchen Lin. Pointflow: Flowing semantics through points for aerial image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4217–4226, 2021.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[39] Lei Ma, Yu Liu, Xueliang Zhang, Yuanxin Ye, Gaofei Yin, and Brian Alan Johnson. Deep learning in remote sensing applications: A meta-analysis and review. *ISPRS journal of photogrammetry and remote sensing*, 152:166–177, 2019.

[40] Gellért Máttyus, Wenjie Luo, and Raquel Urtasun. Deeproadmapper: Extracting road topology from aerial images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3438–3446, 2017.

[41] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[42] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer, 2016.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[44] Andrew Y Ng, Daishi Harada, and Stuart J Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[45] Kei Ota, Devesh K Jha, and Asako Kanezaki. Training larger networks for deep reinforcement learning. *arXiv preprint arXiv:2102.07920*, 2021.

[46] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[47] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[48] Suprosanna Shit, Johannes C Paetzold, Anjany Sekuboyina, Ivan Ezhov, Alexander Unger, Andrey Zhylka, Josien PW Pluim, Ulrich Bauer, and Bjoern H Menze. cldice-a novel topology-preserving loss function for tubular structure segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16560–16569, 2021.

[49] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[50] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[51] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[52] Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.

[53] Yong-Qiang Tan, Shang-Hua Gao, Xuan-Yi Li, Ming-Ming Cheng, and Bo Ren. Vecroad: Point-based iterative graph exploration for road graphs extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8910–8918, 2020.

[54] Adam Van Etten, Dave Lindenbaum, and Todd M. Bacastow. Spacenet: A remote sensing dataset and challenge series, 2018.

[55] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[57] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[58] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2020.

[59] Richard Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*, pages 7324–7334. PMLR, 2019.

[60] Tongjie Y Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

[61] Lichen Zhou, Chuang Zhang, and Ming Wu. D-linknet: Linknet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 182–186, 2018.

[62] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.