Autonomous Fire Front Detection in Satellite Images with Deep Reinforcement Learning

Marianna Velessioti

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Panagiotis Tsakalides*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

This work was supported by the TITAN ERA Chair project (contract no. 101086741) within the Horizon Europe Framework Program of the European Commission.

University of Crete
Computer Science Department

Autonomous Fire Front Detection in Satellite Images with Deep Reinforcement Learning

Thesis submitted by
Marianna Velessioti
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Marianna Velessioti

Committee approvals: _____
Panagiotis Tsakalides
Professor, Thesis Supervisor, University of Crete, Greece

_____
Grigorios Tsagkatakis
Assistant Professor, Thesis Co-supervisor, University of Crete, Greece

_____
Panos Trahanias
Professor, Committee Member, University of Crete, Greece

Departmental approval: _____
Polyvios Pratikakis (director of master studies)
Associate Professor , Director of Graduate Studies

Heraklion, November 2023

Autonomous Fire Front Detection in Satellite Images with Deep Reinforcement Learning

Abstract

An enduring concern that has troubled humanity for generations is the outbreak of wildfires in both forests and areas where communities thrive. This phenomenon has catastrophic implications for the ecosystems and wildlife in the impacted regions, as well as for human lives and properties that are lost in the wake of these fire events. Earth observation satellites play a vital role in the surveillance of land areas, primarily for fire detection. The spatial resolution of these observations is a matter of great importance for the agencies responsible for gathering and processing space-based data. Obtaining high-resolution geographical data systematically from private use satellites is a costly endeavor. In contrast, publicly accessible satellites offer global coverage at considerably lower spatial resolution. Furthermore, there is a trade-off in temporal resolution analysis; public satellites revisit the same geographical position after intervals of days or even weeks to capture updates, while private satellites schedule targeted observations based on the specific requirements of different users, thus not ensuring a systematic monitoring. Despite the potential benefits of satellite-based remote sensing, current solutions are unable to provide both high spatial and temporal resolution simultaneously. This presents an opportunity for research aimed at developing a system that combines lower-resolution observation with the advanced capabilities of private satellites equipped with high-resolution cameras. Such a system could enable more effective observation of specific areas of interest in the images. Within a dual-satellite system featuring high and low-resolution cameras respectively, the challenge at hand is the prompt and accurate detection of fire fronts. The proposed solution involves the utilization of a Deep Reinforcement Learning framework to train an AI agent. The objective of this agent is to pinpoint the location of the fire front within the low-resolution image by delineating a discrete area of interest. This is achieved through the optimization of the agent's predictive model using Policy Gradient methods. Guided by the model's predictions given the low-resolution observations, the high-resolution camera captures observations of the selected areas, i.e. those with a higher likelihood of fire. This thesis involves conducting experiments to assess the algorithm's performance and the influence of its hyperparameters. We also investigate how the algorithm behaves across various dataset sizes. Furthermore, we delve into the dataset itself by performing exploratory analysis to unearth patterns that could improve its handling during the training process. Based on the insights gained from the data, we refine the algorithm. Our analysis demonstrates that the proposed method exhibits better generalization compared to alternative approaches and displays a greater capability to identify the fire front, even in its initial stages.

Αυτόνομη Ανίχνευση Πύρινων Μετώπων σε Δορυφορικές Εικόνες με χρήση Βαθιάς Ενισχυτικής Μάθησης

Περίληψη

Ένα διαρκές πρόβλημα που απασχολεί την ανθρωπότητα εδώ και γενιές είναι το ξέσπασμα πυρκαγιών τόσο σε δασικές εκτάσεις όσο και σε περιοχές όπου αναπτύσσονται πολιτισμοί. Το φαινόμενο αυτό έχει καταστροφικές συνέπειες για τα οικοσυστήματα και την άγρια ζωή στις πληγείσες περιοχές, καθώς και για τις ανθρώπινες ζωές και τις περιουσίες που χάνονται λόγω των πυρκαγιών. Οι δορυφόροι παρακολούθησης της Γης, διαδραματίζουν ένα ζωτικό ρόλο στην επιτήρηση των χερσαίων περιοχών με στόχο τον εντοπισμό πυρκαγιών. Η χωρική ανάλυση αυτών των παρατηρήσεων αφορά άμεσα τις υπηρεσίες που συλλέγουν και επεξεργάζονται δεδομένα διαστήματος, καθώς η συστηματική αποτύπωσή τους σε υψηλή ανάλυση από ιδιωτικούς δορυφόρους είναι μια δαπανηρή επιχείρηση. Αντιθέτως, οι δημοσίως προσβάσιμοι δορυφόροι προσφέρουν παρατηρήσεις με πολύ χαμηλότερη χωρική ανάλυση. Επιπλέον, υπάρχει και ο συμβιβασμός στη διάσταση της χρονικής ανάλυσης, καθώς οι δημόσιοι δορυφόροι επισκέπτονται ξανά την ίδια τοποθεσία μετά από διαστήματα ημερών ή ακόμα και εβδομάδων για να καταγράψουν ενημερώσεις, με αποτέλεσμα να μην υπάρχει συστηματική παρακολούθηση, ενώ οι ιδιωτικοί δορυφόροι προγραμματίζουν τις παρατηρήσεις τους στοχευμένα, βάσει συγκεκριμένων αναγκών των χρηστών. Παρά τα οφέλη της δορυφορικής τηλεπισκόπησης, οι υφιστάμενες λύσεις δεν παρέχουν ταυτόχρονα υψηλή χωρική και χρονική ανάλυση. Αυτό δημιουργεί έδαφος για έρευνα με στόχο την ανάπτυξη ενός συστήματος που συνδυάζει την αξιοποίηση των εικόνων χαμηλότερης ανάλυσης με τις δυνατότητες ιδιωτικών δορυφόρων, εξοπλισμένων με κάμερες υψηλής ανάλυσης. Ένα τέτοιο σύστημα θα μπορούσε να επιτρέψει την αποτελεσματικότερη παρακολούθηση συγκεκριμένων περιοχών ενδιαφέροντος στις εικόνες. Προτείνουμε ένα σύστημα δύο δορυφόρων, που διαθέτει κάμερες υψηλής και χαμηλής ανάλυσης αντίστοιχα, για την άμεση και ακριβή ανίχνευση των μετώπων πυρκαγιάς. Η προτεινόμενη λύση περιλαμβάνει τη χρησιμοποίηση ενός πλαισίου Βαθιάς Ενισχυτικής Μάθησης για την εκπαίδευση ενός πράκτορα τεχνητής νοημοσύνης. Στόχος αυτού του πράκτορα είναι να εντοπίσει τις θέσεις του μετώπου πυρκαγιάς στην εικόνα χαμηλής ανάλυσης, σηματοδοτώντας διακριτές περιοχές ενδιαφέροντος. Αυτό επιτυγχάνεται μέσω της βελτιστοποίησης του προβλεπτικού μοντέλου του πράκτορα με χρήση μεθόδων Policy Gradient, μιας προσέγγισης ενισχυτικής μάθησης που δεν εξαρτάται από κάποια συνάρτηση απόδοσης, αλλά από την πολιτική του πλαισίου εκπαίδευσης. Στη συνέχεια, η κάμερα υψηλής ανάλυσης καταγράφει παρατηρήσεις των επιλεγμένων περιοχών, δηλαδή αυτών με την υψηλότερη πιθανότητα πυρκαγιάς, βάσει των εκτιμήσεων του μοντέλου. Στην παρούσα εργασία, εκτελούμε πειράματα που εξετάζουν την απόδοση του αλγορίθμου και την επίδραση των υπερπαραμέτρων του. Ερευνούμε επίσης πώς συμπεριφέρεται ο αλγόριθμος σε διάφορα μεγέθη συνόλων δεδομένων. Έπειτα, προβαίνουμε σε εξερεύνηση των δεδομένων ώστε να ανακαλύψουμε

μοτίβα που μπορούν να βελτιώσουν τη χειρισμό τους κατά τη διάρκεια της διαδικασίας εκπαίδευσης και επανεξετάζουμε τον αλγόριθμο με την αποκτημένη γνώση πάνω στα δεδομένα. Από την ανάλυση προκύπτει ότι η προτεινόμενη μέθοδος έχει καλύτερη γενίκευση σε σύγκριση με εναλλακτικές προσεγγίσεις και μεγαλύτερη ικανότητα ανίχνευσης του πύρινου μετώπου ακόμα και σε πολύ αρχικά στάδια.

Ευχαριστίες

Θα ήθελα πρωτίστως να ευχαριστήσω τον επιβλέποντα καθηγητή της εργασίας μου Γρηγόρη Τσαγκατάκη, για την καθοδήγηση, την υπομονή και την άμεση ανταπόκριση του καθ΄όλη τη διάρκεια της συνεργασίας μας. Ο Γρηγόρης συνέβαλε σημαντικά εξαρχής απαντώντας σε όλα μου τα ερωτήματα σχετικά με το σκοπό της εργασίας και τα τεχνικά της μέρη, μου παρείχε πολύτιμες ερευνητικές κατευθύνσεις και με ενθάρρυνε υποστηρίζοντας τις ιδέες μου. Ευχαριστώ πολύ τον επόπτη μου, κ. Παναγιώτη Τσακαλίδη για τη συνεπίβλεψη και την προθυμία να με βοηθήσει σε κρίσιμες πτυχές της εργασίας όπως η μοντελοποίηση μέσω κατανομών πιθανότητας. Είμαι ευγνώμων σε όλο το Εργαστήριο Επεξεργασίας Σήματος (SPL), τους συμφοιτητές μου και το ερευνητικό προσωπικό, με τους οποίους εργαστήκαμε στον ίδιο χώρο, αλλά και αποκομίσαμε από κοινού αξέχαστες εμπειρίες μέσω εθελοντισμού και συμμετοχής σε ερευνητικά δρώμενα. Ευχαριστώ, λοιπόν, τα παιδιά και ιδιαιτέρως την Αρετή, η οποία μου παραχώρησε τον υπολογιστή της ώστε να τρέχω τα πειράματά μου για εβδομάδες. Θα ήθελα επιπλέον να ευχαριστήσω τον καθηγητή Πάνο Τραχανιά, που συμμετείχε ως μέλος στη τριμελή μου επιτροπή. Θα προεκτείνω την εγκάρδια ευγνωμοσύνη μου στον ακαδημαϊκό μου σύμβουλο Γιώργο Καφεντζή, για τη διαρκή του στήριξη και παρουσία, τόσο κατά τη διάρκεια της συνεργασίας μας στο πτυχίο, όσο και μετά, στην πρακτική μου και στο master. Δεν θα ήθελα να παραλείψω τη γραμματεία του τμήματος Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης, η οποία ήταν πάντοτε συνεργάσιμη και βοηθητική, τόσο στο προπτυχιακό όσο και στο μεταπτυχιακό. Τέλος, θα ήθελα να ευχαριστήσω τις φίλες μου, Κατερίνα και Μαίρη που ήταν στο πλευρό μου από την πρώτη μέρα που ήρθα στο πανεπιστήμιο και στάθηκαν μαζί μου τόσο στα ευχάριστα όσο και στα δύσκολα αυτής της σταδιοδρομίας, τον αγαπημένο μου Νίκο για τη στήριξη και την κατανόηση του στη διάρκεια του μεταπτυχιακού μου, τους γονείς μου, τη γιαγιά μου και την αδερφή μου.

στην οικογένεια μου

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem and Motivation

The outbreak and spread of fire in open geographical territories is a crucial issue, that occupies many countries internationally for centuries. Several studies have been carried out on fire spread conditions and causes that bring forward the problems that are still lurking [5, 59, 1]. Since this phenomenon has devastating consequences for the environment and the human safety, it is imperatively urgent to take some countermeasures to address the problem by taking advantage of the modern technology.

In an era where the fire fronts are a large scale problem, space agencies seek to provide solutions with satellite surveillance. Earth observation satellites which are available for public use (institutional satellites) capture broad geographical areas periodically in moderate spatial resolution, whereas private satellites (tasked) capture high resolution images on demand, based on the needs of different users. The spatial resolution of satellite images is a real-world challenge, as the institutional satellite observations are characterized by lack of it, due to ground sampling distance, whereas private satellite observations are costly and do not ensure systematic monitoring. Also, the issue of temporal resolution in satellite surveillance cannot be overlooked; public satellites sample an observation and return to re-sample this same geographical area after a significant time interval, whereas private satellite observations are more adaptable.

Various fire detection and localization methods utilizing satellite images have been proposed [17, 21, 54, 27, 46, 9, 11] which provide useful insights for the use case of the current thesis. However, none of them investigates the aforementioned issues and therefore, ground arises for relevant research in the field. In this study, we address the problem of fire localization in low-resolution image sub-areas and propose a system that can be used for real-time supervision on the Earth's surface in order to prevent fire spread. Such a detector plays a crucial role in wildfire monitoring, environmental research, and disaster management. It provides valuable information for decision-makers to respond effectively to ongoing fire incidents

and mitigate their impact on communities and ecosystems.

1.2 Proposed solution: AI-based decision making system for fire front detection

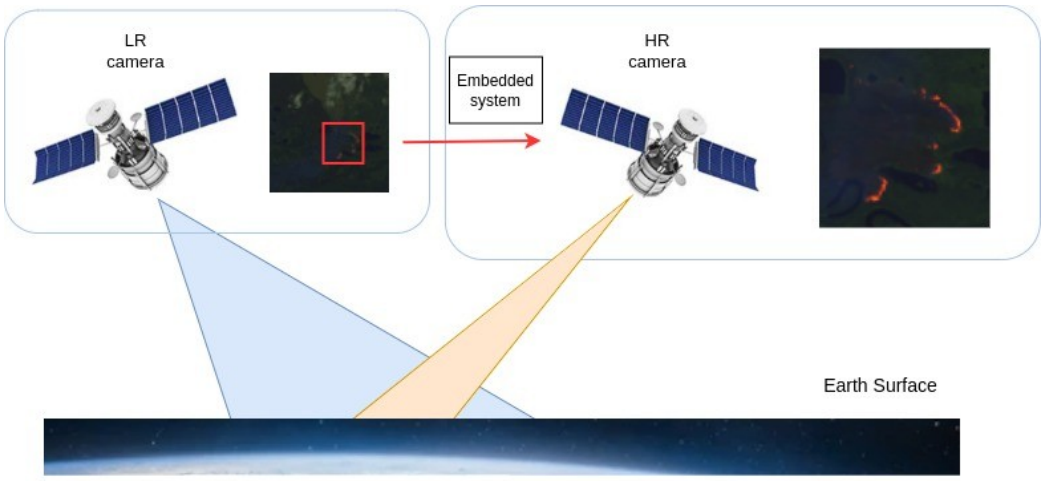


Figure 1.1: The autonomous surveillance system consists of two artificial satellites: The first satellite captures low resolution images periodically (left). The embedded system informs the second satellite to capture targeted areas in high resolution (right).

We address the challenge described above with an embedded system that combines the revisit frequency of low-resolution (LR) imaging satellite, with the capabilities of tasked satellites equipped with high-resolution (HR) cameras. Figure 1.1 illustrates the system of the two-satellite supervision in a high-level diagram. The institutional satellite captures images periodically, that are processed by an AI agent, trained to localize Areas of Interest (AoI) which, in this case, are the fire fronts. After marking those areas, the private satellite is notified with a signal in order to sample targeted observations in high resolution.

The control system of the agent mainly comprises of a neural network, called policy network, that inputs LR images and outputs a patch-level decision indicating presence of fire or not. To train this network to produce accurate predictions, we are following a Reinforcement Learning scheme involving a second neural network, that performs segmentation on the input image. Throughout the process, the LR image sub-areas are mapped to their high resolution correspondences; those constitute the input of the segmentation network. The latter one is providing feedback to the policy network so that it can "learn" the patterns of the fire-present input images in a self-supervised manner.

1.3 Contribution

In this master thesis we propose an Autonomous **Fire Front Detection** framework. The contribution lies in the following aspects:

- Propose a feasible and effective solution to counterpoise spatial and temporal resolution limitations in remote sensing data. This is achieved by embedding to tasked satellites an on-board AI system, trained to detect fire fronts with a data-driven approach.
- Exhibit the capabilities of a Reinforcement Learning algorithm to localize Areas of Interest (AoI) in satellite images based on their semantic segmentation. The current thesis analyses the RL design in technical depth and delves into the explanation of our AI agent's behavior performance-wise.
- Take profit of the relevant literature to incorporate a U-Net architecture for Semantic Image Segmentation to an RL framework, in order to achieve pixel-level localization of the target.
- Delve into model-free Reinforcement Learning concepts and, specifically, Policy Gradient Optimization that is used to solve the AoI detection task.
- Give insights into a well-known remote sensing dataset regarding fire-presence and form, based on the acquired knowledge, individual sets of different sample sizes to train and evaluate our algorithm.
- Develop baseline methods to propose alternative solutions to the fire front detection problem, that are less time and memory consuming; we compare these methods with the deep RL method and show by inference that the latter one achieves better generalization.
- Finally, design a novel method to model random paths in the context of Markov Decision Processes; select relevant image patches in multiple steps via Monte Carlo sampling.

1.4 Organization

The rest of this thesis is organized as follows: In Chapter 2 we present the important concepts that constitute the theoretical cornerstone of the current work. Chapter 3 includes an overview of relevant studies in the field of active fire detection and localization of AoI in images. Chapter 4 comprises our approach to the problem with the description of the overall framework and the technical details of the design and training. In Chapter 5 we present the experiments and results that support this work and finally, in Chapter 6 we state some conclusions and visions for future investigation.

Chapter 2

Theoretical Background

2.1 Deep Neural Networks

Deep neural networks (DNNs) [61, 43, 44] are a class of machine learning models designed to emulate human brain functions in processing and learning from data. They consist of interconnected layers of artificial neurons, where information flows through these layers to perform complicated tasks, such as image classification, regression and more [23, 30, 49]. Fully connected layers [31] connect all neurons from adjacent layers, allowing intricate relationships to be learned. Convolutional neural networks (CNNs) [25, 23] specialize in processing grid-like data, like images, by utilizing convolutional layers that apply filters to capture spatial hierarchies. Residual neural networks (ResNets) [15], a subset of CNNs, combat vanishing gradient issues by introducing skip connections. These connections allow the network to learn residual mappings, making it easier to train significantly deep architectures. The innovation of these network types has propelled advancements across various domains, achieving state-of-the-art results in tasks requiring intricate pattern recognition and data transformation [15, 16]. The contribution of CNNs and ResNets is decisive in the current thesis, as they constitute the backbone of our machine learning training pipeline.

2.2 Towards Unsupervised Learning

The quest for developing machines with human-like cognitive abilities has driven significant research towards the paradigm of unsupervised learning. Unlike **supervised** learning, which relies on labeled data, **unsupervised** learning seeks to enable machines to extract meaningful patterns, representations, and structures from unlabeled data. This paradigm shift holds the promise of unlocking the potential to discover hierarchical dependencies and intrinsic characteristics within complex datasets. Therefore, as data acquisition continues to outpace labeling efforts, the exploration of unsupervised learning becomes increasingly pertinent.

Significant works that have contributed to the development and understanding

of deep unsupervised learning are Boltzmann Machines [2], autoencoders and deep belief networks [4], as well as energy-based models [26]. This concept is decisive in the field of Reinforcement Learning, where agents learn to make sequential decisions through trial and error. Their goal instead is not to find hidden structure in unlabeled data, but to maximize a reward signal while the agent interacts with its environment [52].

Traditional learning algorithms often require extensive labeled data or expert demonstrations, making them unsuitable for many real-world applications. However, recent advancements in reinforcement learning techniques [34, 48, 28, 47], have shown promising results in enabling agents to learn complex tasks without the need for explicit supervision. By leveraging unsupervised learning paradigms, these methods highlight the agents' ability to explore diverse environments and generalize knowledge efficiently. This special kind of learning is examined thoroughly in the next section.

2.3 Key Concepts in Reinforcement Learning

Reinforcement Learning (RL) is a subfield of Machine Learning that focuses on training agents to make sequences of decisions in order to maximize a cumulative reward signal and thus, find an optimal way to take actions in an environment. This section encapsulates all the important information on Markov Decision Processes and Reinforcement Learning methods that are useful to understand the concepts used in the current thesis. Specifically, some key concepts are listed below.

- An **agent** is an entity that learns to make decisions and take actions within an environment. In RL, the agents are learning by trial and error.
- The **environment** is an external system or context in which the agent operates. The environment provides feedback to the agent in the form of rewards and states, as shown in figure 2.1.
- The **state** (S) is a representation of the environment at a given time. It encapsulates all relevant information about the current position/situation of the agent.
- As **actions** (A) we define the set of all possible transitions or decisions the agent can take in the environment in order to move from one state to another. Actions are chosen by the agent based on its policy.
- The **policy** (π) is a function that maps states to actions. In practice, it is a probability distribution assigned to the set of actions that determines the agent's behavior in the RL framework.
- The **reward** (R) is a scalar signal from the environment that provides the agent immediate feedback about the current policy after it takes an action

from a particular state. The agent's goal is to maximize the cumulative reward over time, called return.

- As **trajectory** or **episode** (τ) we define a sequence of states, actions and rewards that the agent experiences while interacting with the environment. These are forming the path the agent follows until it reaches the goal or comes across some stopping condition.

- Exploration vs. Exploitation: this encapsulates the trade-off in RL between exploring new actions and exploiting known pathways with high expected rewards. Striking the right balance is crucial for effective learning.

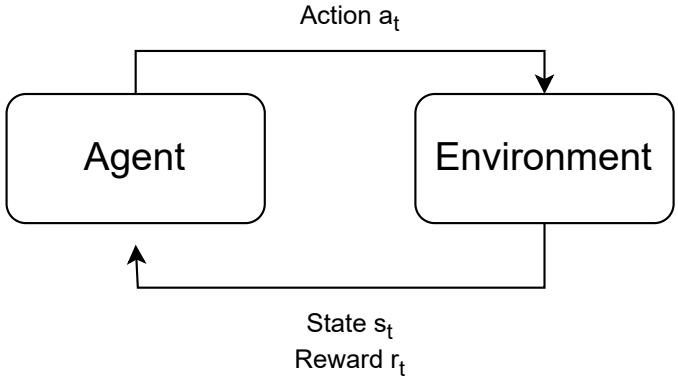


Figure 2.1: Agent - environment interaction loop; the agent takes an action at time-step t, then the environment sends back a reward signal and the new state.

2.3.1 Markov Decision Processes

Markov Decision Processes (MDPs) are a fundamental framework in the field of Reinforcement Learning (RL). An MDP (S, A, π, R, γ) is a process that obeys the Markov Property [1]. Its aforementioned components are: the state space **S**, the action space **A**, the state transition probabilities π i.e. the policy, the reward function **R**, and the discount factor $\gamma \in [0, 1]$.

In figure 2.2 one can see a complete trajectory τ of a Markov Decision Process in a flow diagram. The trajectory example, in a world with these four states, is:

$$\tau = \{(s_0, a_0), (s_1, a_1), (s_2, a_2), (s_1, a_1), (s_2, a_3)\} \tag{2.1}$$

where each action a_i, with $i \in \{0, 1, 2, 3\}$, is accompanied by the respective Reward, R_i. The very first state of the world, s_0, is randomly sampled from a start-state

[1] transitions only depend on the most recent state-action and no prior history

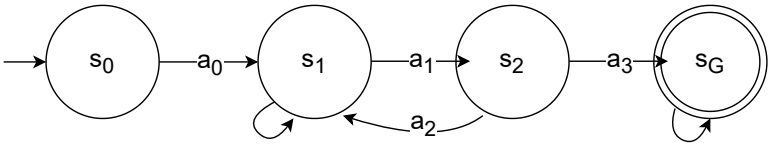


Figure 2.2: Example of a MDP with four states

distribution, denoted as ρ_0. State transitions, are governed by the natural laws of the environment (edges), and depend only on the most recent action:

$$P_{ss'} = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a) \tag{2.2}$$

where P is the state transition probability, s the current state, and s' the next state. Finally, the agent is located at the goal state s_G where the process ends.

Action Spaces and Stochastic Policies

Action spaces can vary depending on the environment; in **discrete** action spaces, only a finite number of moves are available to the agent, whereas in **continuous** action spaces, actions are real-valued vectors. Policies determine what actions the agent takes, and those can be **deterministic** or **stochastic**. Also, deep RL problems are formulated with policies whose outputs are computable functions that depend on a set of parameters. Actions are estimated by the policy given the current state as follows:

$$a_t \sim \pi_\theta(.|s_t) \tag{2.3}$$

where π_θ is the parameterized stochastic policy. Parameters are usually adjusted via some optimization algorithm in order to change the agent's behavior in future episodes.

For training stochastic policies, two are the main computations; the first one, is sampling actions from policy distributions, while the second one, is computing the log-likelihoods of particular actions:

$$\log \pi_\theta(a|s) = \log P_\theta(s)|_a \tag{2.4}$$

where $P_\theta(s)$ denotes the vector of probabilities with as many entries as there are actions. In this scenario, the parameterized policy is proportional to the probability distribution that determines action-state transitions over trajectories.

Rewards and Value Functions

The reward function R has a critical role in the Reinforcement Learning loop. Its output depends on the current state, the current action and the next state of the environment:

$$r_t = R(s_t, a_t, s_{t+1}) \tag{2.5}$$

The agent's goal is to maximize some notion of the cumulative reward over a trajectory τ. Averaging the rewards of a trajectory, we get the **Expected Utility** i.e. the **Value** or simply the **return** of a path. When the number of time-steps is known and finite, we define the finite-horizon undiscounted return as the sum of rewards in T steps:

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{2.6}$$

Otherwise, the discount factor γ is multiplied with the individual rewards, which sum up to give the infinite-horizon return.

Regarding the evaluation of complete paths, an MDP may compute different Value Functions depending on the agent's behavior upon the policy. The on-policy Action-Value Function, or Q-function, which is primarily used in many RL methods, is defined as:

$$Q_\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_t = s, a_t = a] \tag{2.7}$$

The above formula gives the expected return starting from state s, taking an arbitrary action a, and then acting according to policy π until the end of the episode. The optimal Action-Value Function, gives the maximum possible expected return of an MDP starting in state s, taking action a, then until the end acting according to the optimal policy:

$$Q^*(s, a) = \max_\pi Q_\pi(s, a) \tag{2.8}$$

This gives the maximum action-value function over all policies, which solves the primary MDP problem.

Finally, it is worth mentioning the concept of the **advantage function**. This function represents the relative advantage of taking a specific action a in a given state s, over the expected value of actions under a certain policy. It measures how much better (or worse) the randomly selected action (according to policy π) is compared to the average or expected action in the given state. The formula for the advantage function is typically defined as follows:

$$A(s, a) = Q_\pi(s, a) - V_\pi(s), \tag{2.9}$$

where $V_\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_t = s]$, the on-policy Value Function.

2.3.2 The RL Objective

Reinforcement Learning, regardless of the choice of Reward and Value functions, seeks to optimize an agent's behavior by finding an optimal policy that maps states to actions, maximizing the cumulative reward of a Markov Decision Process. This behavior can be approached by modeling probability distributions over trajectories:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.10}$$

The above distribution is considering a stochastic policy and a T-step trajectory. The first state of the world is determined by the start-state distribution. Thereafter, states occur based on the actions taken according to policy. Each stochastic policy can produce multiple trajectories. Each trajectory comes with an average reward $R(\tau)$. Integrating over all possible trajectories of a policy, altogether with their rewards, we conclude in the general formulation of the objective:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)], \tag{2.11}$$

which is simply the expectation of Values of all possible paths. Maximizing this average solves for the central optimization problem of RL:

$$\pi^* = \arg\max_{\pi} J(\pi), \tag{2.12}$$

with π^* being the optimal policy. Intuitively, this means that we aim to find the policy that leads the agent to make on average the best possible decisions.

2.3.3 Reinforcement Learning Methods

In a reinforcement learning problem, the transition probability distribution and the reward function of a Markov Decision Process are considered as the "model" of the environment. Model-free and model-based RL are two broad categories of RL algorithms (see figure 2.3) which differ in whether the agent has access to (or learns) the model. Their major differences are presented below along with examples of relevant works.

In **model-based RL**, the model parameters are known from the beginning - so the agent can use them to plan and make decisions - however, they might keep getting updated throughout the interaction of the agent with the environment. This can be advantageous for sample-efficient learning but requires accurate modeling. A family of model-based approaches are those that are doing pure planning, like Model Predictive Control (MPC) [19, 36], where the agent computes a plan that is optimal in relation to the model each time it observes the environment. The plan outlines all the actions to be taken across a predefined temporal window. Another family of model-based approaches is leveraging data augmentation for model-free methods; MBVE [10] is using a model-free algorithm to augment real experiences with fictitious ones, whereas Recurrent World Models [14] are using purely fictitious experience to update the agent. Lastly, a straightforward follow-on to pure planning is Expert Iteration (Exit) [3], which involves using and learning an explicit representation of the policy $\pi_\theta(a|s)$ by leveraging Monte Carlo Tree Search. AlphaZero [50] is another popular example of the last approach. It involves training a DNN to evaluate board positions and learn a policy for selecting moves, enabling it to play complex games such as chess, shogi, and Go at an incredibly high level.

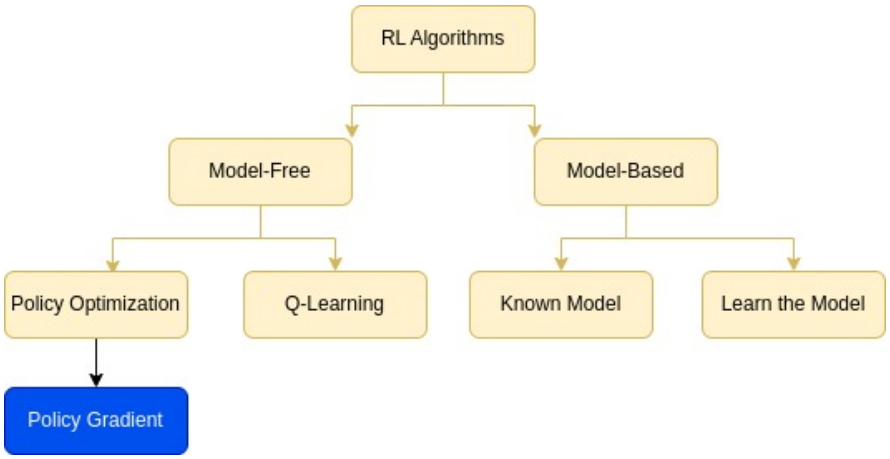


Figure 2.3: A taxonomy of RL algorithms

In **model-free RL**, the agent directly learns from interactions with the environment without building an explicit model of the environment's dynamics. It focuses on estimating the Value or Action-Value functions to make decisions based on past experiences, often employing techniques like Q-learning or policy gradients. Model-free RL is often preferred when the environment is complex and modeling is challenging. A brief survey of the most predominant relevant research in model-free RL, from classic methods to more contemporary, is following in the next paragraphs.

Q-Learning methods try to learn an approximator $Q_\theta(s, a)$ for the optimal Action-Value function $Q^*(s, a)$, and use an objective function based on the Bellman equation[2]. Unlike policy gradients, optimization here is performed off-policy, meaning that policy updates are decoupled from data collection; the agents can learn from data generated by any policy, not just the current one (the one being optimized). The actions taken by the Q-learning agent are given by:

$$a_t \leftarrow \arg\max_a Q_\theta(s, a) \tag{2.13}$$

Learning from Delayed Rewards [58] is the original Q-learning algorithm that uses a tabular approach to estimate Q-values and iteratively updates them based on the Bellman equation. Deep Q-Network (DQN) [33] extends Q-learning to handle high-dimensional state spaces by using deep neural networks to approximate the Q-function. This classic example substantially launched the field of deep RL.

Policy gradient methods optimize the parameters θ of the model either by

[2]The Bellman equation expresses the Value (or Action-Value) Function as the maximum expected reward achievable by making a current decision and then optimally solving the remaining subproblem.

gradient ascent on the performance objective $J(\pi_\theta)$ or by maximizing local approximations of $J(\pi_\theta)$. Pertinent examples are REINFORCE (Monte Carlo Policy Gradient) [60] and Actor-Critic [22]. The REINFORCE algorithm estimates gradients after performing Monte Carlo sampling on the policy distributions and updates the policy parameters to maximize expected rewards. Actor-Critic methods combine the advantages of both policy-based (actor) and value-based (critic) methods; the actor learns the policy, while the critic estimates the value function and provides feedback to the actor. Some more contemporary research has been done on Asynchronous Advantage Actor-Critic (A2C/A3C) [32], which is a distributed RL method that combines the actor-critic architecture with parallelization, allowing multiple agents to interact with their own environments and share information. Proximal Policy Optimization (PPO) [48], on the other side, is maximizing the agent's performance by using a clipped surrogate objective function which gives a conservative estimate of how much $J(\pi_\theta)$ will change as a result of the update. The aforementioned works are some of the most prevalent examples of policy gradient methods, however the relevant research on them is even more broad.

2.3.4 Policy Optimization

Policy optimization is a pivotal aspect in the field of Reinforcement Learning, and particularly in the context of this thesis. It represents the core methodology for training intelligent agents to make sequential decisions in complex, uncertain environments. By delving into the theoretical foundations of policy optimization, we aim to establish a robust understanding of the techniques and algorithms that form the backbone of modern intelligent systems, like the one developed in the present work.

Theoretical foundation of policy gradient

As **policy** in the context of policy optimization, we define a probability distribution the agent uses to make actions. In deep reinforcement learning, policies are often approximated using neural networks with parameters θ, whose output represents the probabilities of taking different actions given a particular state. The policy function, in this case, is a mapping from states to action probabilities, and it is defined by the output of the neural network, hereinafter referred to as f_θ.

$$\pi_\theta(a_t|s_t) = \frac{e^{f_\theta(a_t,s_t)}}{\sum_i e^{f_\theta(a_i,s_t)}} \tag{2.14}$$

In the above equation, the output of the parameterized policy network f_θ, gives the raw score (logit) for action a in state s. A softmax activation function is generally applied in the output layer to ensure that the output is a valid (categorical) probability distribution over all possible actions.

In the previous sections, we defined the policy performance objective $J(\pi_\theta)$ (formula 2.11), the undiscounted return $R(\tau)$ (formula 2.6) and the probability of a trajectory $P(\tau|\pi_\theta)$ (formula 2.10). The derivation of the analytical gradient of policy performance (**policy gradient**) relies on these terms and makes use of the log-derivative trick to form the final expression, which takes into account the gradient log-prob:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \int_\tau P(\tau|\pi_\theta) R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\pi_\theta) R(\tau) \\ &= \int_\tau P(\tau|\pi_\theta) R(\tau) \nabla_\theta \log P(\tau|\pi_\theta) \\ &= \mathbb{E}_{\tau\sim\pi}[R(\tau)\nabla_\theta \log P(\tau|\pi_\theta)]\end{aligned} \tag{2.15}$$

We shall start by substituting formula 2.11, we then bring the gradient under the integral and apply the log-derivative trick. We then return to the expectation form and derive to the final expression of the parameterized policy gradient by substituting with the log-probability of a trajectory (2.10). To note that, the environment has no dependence on θ, so gradients of $\rho_0(s_0), P(s_{t+1}|s_t, a_t)$ and $R(\tau)$ are zero.

Since we also know that the probability of a trajectory given the policy is equal to the probability of each action given the current state and the policy parameters:

$$P(\tau|\pi_\theta) = \prod_{t=0}^{T} P(a_t|s_t;\theta) \tag{2.16}$$

where T is the last trajectory index, we can conclude in the final expression for the gradient of the objective by taking into account the formula 2.4 and expanding the product under the logarithm:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{2.17}$$

In order to maximize the performance objective, the parameter updates follow the **gradient ascent** rule:

$$\theta_{t+1} = \theta_t + \alpha\nabla_\theta J(\pi_{\theta_t}) \tag{2.18}$$

where the term α above denotes the learning rate.

In conclusion, assuming that we represent our policy in a way that allows to calculate $\nabla_\theta \log \pi_\theta(a|s)$, and that we are able to run the policy in the environment to collect the trajectory data, we can compute the policy gradient and take an update step towards the direction of policy optimization.

About the Loss Function To make a disclaimer regarding the objective function, even though we describe this as a negative loss function, it is not really a loss function, in the sense that it is used in supervised learning. This "loss" is only useful to us because, when evaluated at the current parameters, it has the negative gradient of performance. Therefore, minimizing this "loss" function, for a given batch of data, does not signify any improvement on the expected return.

The REINFORCE algorithm

REINFORCE [60] is a Monte-Carlo variant of policy gradients, used for optimizing the parameters of a stochastic policy π_θ. It works by adjusting the probabilities of actions in proportion to the observed rewards, encouraging highly rewarded actions to be taken more frequently. The pseudocode for REINFORCE is given in Algorithm 1.

Algorithm 1 REINFORCE, A Monte-Carlo Policy Gradient Method

Input: a differentiable policy parameterization π_θ.
Initialize θ arbitrarily, start with state s_0.
for each episode $\tau = \{(s_0, a_0, r_1), ..., (s_{T-1}, a_{T-1}, r_T)\}$ **do**
  $R(\tau) = 0$
  for each time step t $= \{0, ..., T-1\}$ **do**
    $a_t \sim \pi_\theta(.|s_t)$
    $r_{t+1} \leftarrow$ evaluate reward for a_t
    $R(\tau)$ += r_{t+1}
  end for
  $\theta_{k+1} \leftarrow \theta_k + \alpha \mathbb{E}[R(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)]$
end for

Assuming a neural network to be trained within an episodic process, where each episode consists of T time steps, we initialize randomly its parameters θ and perform a trajectory roll-out using the current policy π_θ. The algorithm stores log probabilities of policy and reward values at each step; then it calculates the cumulative reward. At the end of an episode, it computes the policy gradient and updates the policy parameters θ by gradient ascent. This process repeats until we reach the maximum number of episodes. Despite its apparent simplicity, REINFORCE serves as a foundational concept in policy gradient methods and has paved the way for more advanced algorithms in the field of RL.

2.4 Semantic Image Segmentation

An important part of our RL framework is the network that performs semantic segmentation on input images. Thus, we dedicate a special section that explains the concept of Semantic Image Segmentation and its advancements with the use of deep neural networks.

The goal of this task is to produce a **pixel-wise segmentation map** of an image, where each pixel is assigned to a specific class or object. Unlike object detection, where bounding boxes are used to outline objects, semantic segmentation provides a fine-grained understanding of the visual scene by assigning each pixel a specific class label, thereby enabling machines to comprehend images with human-like precision.

Traditionally, semantic segmentation methods heavily relied on handcrafted features and shallow machine learning algorithms [24, 45, 35]. However, the advent of deep learning, particularly Convolutional Neural Networks (CNNs) [25, 23], revolutionized this field. Deep learning models, especially Fully Convolutional Networks (FCNs) [29], U-Net [42], and DeepLab [6], have exhibited unprecedented performance in semantic segmentation tasks. These models leverage hierarchical feature representations to capture contextual information, enabling them to delineate intricate object boundaries with great precision.

2.4.1 U-Net

Among various architectures developed to solve the task of semantic image segmentation, U-Net stands out as a seminal model renowned for its effectiveness and versatility. Proposed by Ronneberger et al. in 2015 [42], U-Net introduced a novel and intuitive architecture characterized by a U-shaped network structure, with a contracting path to capture context and a symmetric expanding path to achieve precise localization. A U-Net architecture, built to perform segmentation on the Landsat-8 dataset images, is depicted in figure 2.4.

U-Net is an extension of FCN that can result in accurate segmentations - even with a relatively small dataset. A Fully Connected Network (FCN) [29] is basically a CNN whose fully-connected layers are replaced with a convolutional layer with large receptive field (known as the kernel size) in order to output high resolution images. A U-Net, however, consists of two FCNs; an encoder and a decoder networks with a bottleneck in between. The horizontal arrows in the figure above correspond to the **skip connections** that allow concatenation of each encoder block output to the corresponding stage of the decoder. Subsequently follows an explanation of the U-Net building blocks.

Encoder

The encoder of U-Net performs the downsampling of the image - it is the contracting path in the example of fig. 2.4. It consists of repeating blocks containing

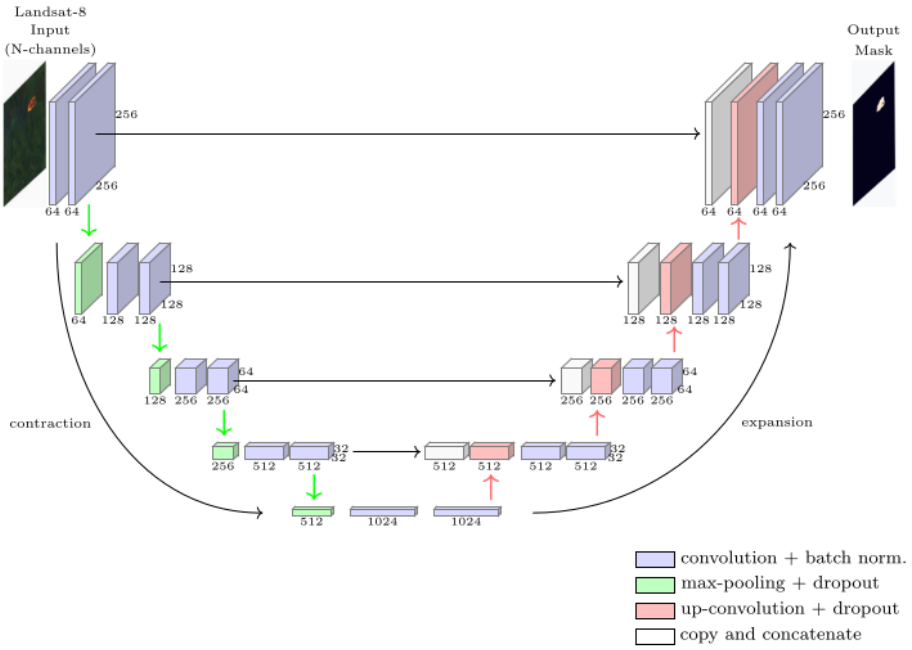


Figure 2.4: The U-Net architecture proposed by de Almeida Pereira et al. [9] to perform image segmentation on Landsat-8 dataset.

convolutional layers (blue) activated by a ReLU function, followed by a max pooling and a dropout layer (green). The number of filters increases at each contraction stage and, because of the pooling layer, the feature dimensionality gradually decreases.

Bottleneck

The bottleneck follows the encoder block and is used to extract more features. It is in the middle of the two (contracting and expanding) paths, where the input tensor has been flattened. This stage does not have a pooling layer, so the dimensionality remains the same.

Decoder

This is the last part of U-Net before acquiring the final segmentation mask. The decoder upsamples the features back to the original image size i.e. in the opposite direction of the encoder. At each upsampling level (red), it takes the output of the corresponding encoder block and concatenates it with the intermediate tensor (white) to feed it to the next decoder block.

U-Net has been proved efficient for both binary and multiple class problems. The features extracted at different levels of the contracting path are combined with the feature maps in the expansion path, bearing this way more features that contribute to a more accurate segmentation. The significant impact and the desired properties of U-Net and its variants in the field of image processing, led to the decision to use this model as a part of the Reward Evaluation of our RL framework.

2.4.2 Evaluation Metrics

Evaluating the performance of an image segmentation algorithm is a critical aspect in computer vision and differs from the way we evaluate classification or regression algorithms. Choosing the appropriate evaluation metrics assumes grasping the nature of the problem and it is essential to understanding the strengths and limitations of the segmentation model. Prevalent performance metrics for image segmentation are listed below.

Pixel Accuracy calculates simply the percentage of pixels in the image that are classified correctly. Although intuitive and easy to compute, it can be misleading in the presence of class imbalance.

Intersection over Union (IoU) is the area of overlap between the predicted segmentation and the ground truth over their union:

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}} \tag{2.19}$$

This metric ranges from 0 to 1, meaning zero overlap and perfectly overlapping segmentation respectively. It takes into account both the false positives and the false negatives all along with the true positive classifications. This metric is also known as Jaccard index.

Dice Coefficient (F1-dice) measures the spatial overlap between the predicted and true segmentations, providing a more fine-grained evaluation of segmentation accuracy. It is twice the overlap of the two images over the total number of pixels in both images:

$$F1 = \frac{2 * \text{Overlap}}{\text{Total number of pixels}} \tag{2.20}$$

IoU and F1-dice are positively correlated, which means that if the one says that model A is better than model B for semantic image segmentation, the other will say the same.

The above metrics evaluate **pixel-level** accuracy; there are also other metrics that focus on region-level accuracy like precision and recall, as well as class specific metrics, which are practically projections of the existing metrics tailored to individual classes. The choice of evaluation metric actually depends on the specific segmentation task and the characteristics of the dataset.

Chapter 3

Related Work

3.1 Zooming into Image Sub-spaces with Deep Learning

Attention Networks. In order to localize semantically important parts of images, the relevant literature has proposed various Attention Mechanisms [55]. Saliency-based Sampling [39] involves training an extra layer in convolutional networks dedicated for selecting specific features in an image based on their visual saliency. Utilizing computational models, saliency maps that are generated for input images, highlight regions that are visually conspicuous, capturing features such as edges, colors, or textures that draw human attention. Residual Attention Networks [56], is another type of neural networks that integrate attention mechanisms in the residual connections [15]. By doing so, they enable the network to focus on important features for classification, while maintaining the benefits of learning identity mapping. This has been demonstrated by improving the classification accuracy on various benchmark datasets while focusing only on the target-specific pixels.

Region Proposal Networks. RPNs [40] are a critical component of modern object detection systems. They generate a set of candidate bounding box proposals that are likely to contain objects in the input image. In subsequent stages of the pipeline, they utilize these predictions to attain the final object classification and localization. Thus, by focusing computational resources on regions of interest, they significantly improve the efficiency and accuracy of object detection. Faster R-CNN (Region-based Convolutional Neural Networks) [40] combines RPN and Fast R-CNN [13] into a single unified model. RPN generates region proposals directly from the convolutional feature maps, which are then used by the Fast R-CNN detector for final object classification and bounding box regression. By sharing convolutional features between RPN and Fast R-CNN, Faster R-CNN avoids redundant computations, making the entire object detection process faster.

Nevertheless, the last approaches still operate a costly per-region subnetwork hundreds of times to achieve object detection. R-FCN [8] is a fully convolutional alternative proposed to defeat the latter drawback of Region-based CNNs for even more efficient object detection.

Deep Reinforcement Learning. This section cites relevant works that combine the benefits of both the aforementioned categories with Reinforcement Learning methods. drl-RPN [38] is a deep RL-based visual recognition model, consisting of a sequential Region Proposal Network (RPN) and an object detector. Replacing the greedy RoI selection process of RPN with a sequential attention mechanism, drl-RPN is trained via deep reinforcement learning to optimize an objective closer to the final detection task. The aforementioned method conditions the attention modules on high-resolution images. Dynamic Zoom-in Network [12], on the other hand, is pursuing fast object detection from the down-sampled versions of large images. It consists of a coarse detection network (R-net) that outputs the accuracy gain for the classification of high-resolution image regions, and a Q-function network (Q-net) that sequentially selects regions that are worth zooming in. This approach exhibits qualitative improvements compared to other object detection algorithms and speeds-up the process of object detection without manipulating the underlying detector's structure. A final work by which the current thesis is radically inspired is [53]; **PatchDrop** is a patch sampling system that conditions on low-resolution images and learns based on classification signals, similarly to [12]. The authors present a general framework for zooming in the image space that consists of a Residual Network and a two-stream classifier, both built upon convolutional layers. They model the RL episodes with a two-step Markov Decision Process and hence, define two different policies and the corresponding action spaces. The networks interact incorporating Policy Gradients to optimize the objective. The Residual Network is forming the RL policy based on which are inferred the semantically important parts of the image. The classifier outputs a prediction that determines the RL reward; this signal determines the training upon the policy parameters. Both the Residual Network and the two-stream classifiers are trained independently and fine-tuned jointly to achieve an optimal policy. The results are noticeable in all the datasets on which they evaluated the model, including CIFAR-10, CIFAR-100, ImageNet and fMoW.

3.2 Active Fire Detection in Landsat-8

Various algorithms are employed to detect active fires in Landsat-8/OLI data. One common approach is to use the Shortwave Infrared (SWIR) and Thermal Infrared (TIR) bands to identify hotspots [7, 20]. Active fires emit energy in these bands, allowing algorithms to distinguish them from the surrounding environment. Thresholding techniques or machine learning algorithms can be applied to identify pixels or areas with temperatures indicative of active fires. [45], [24] and [35] are

prevalent examples of algorithms that rely on comparisons to fixed threshold in certain bands and statistics from their surrounding region. Active fire detection using Landsat-8/OLI (Operational Land Imager) data [46] involves utilizing the relevant satellite imagery to identify and monitor active fires on the Earth's surface with a handcrafted algorithm expanded with the use of multi-temporal analysis to improve pixel classification.

A recent study that is closely related to this thesis is [9]. The authors process the Landsat-8 imagery and form the dataset we describe in section 5.1. This work involves the utilization of a U-Net model [42] trained with targets produced by the three sets of thresholding conditions mentioned above, their intersection, the voting scheme (intersection of at least two of them) and a manually annotated dataset. They propose three different convolutional neural network architectures to approximate these handcrafted algorithms: a U-Net-10c, a U-Net-3c and a U-Net-Light-3c. The first and second architectures are tailored to 10-channel and 3-channel inputs respectively. The third one is a variant of U-Net-3c, but with significantly less training parameters, hence a "lightweight" version. The authors investigate the potential improvement of the networks' performance with numerous benchmarks. Finally, they achieve high-quality segmentation masks that delineate fire fronts in Landsat-8 imagery and present the comparative evaluation among the different labeling schemes and the architectures they use.

Chapter 4

Methodology

In this chapter, we present the proposed solution for the problem of fire front detection in low spatial resolution (LR) images. The computational sensing of fires with Deep Reinforcement Learning (deep RL) is a domain-specific contribution relying in prior works on active fire detection with semantic segmentation and image patch sampling with deep RL [9, 53]. It is designed to serve the use case of fire patch detection in multi-spectral satellite images. We define a Reinforcement Learning setting for each of the two methods we propose, and explain subsequently the training pipeline over the fire-present data of Landsat-8. Finally, we discuss some implementation details of the model architectures we use to compose the overall framework.

4.1 A Reinforcement Learning setting

In a setting where we theoretically possess only LR images, we want to find a mechanism that distinguishes areas of interest while dropping the non-relevant areas in the image. We design an RL setting in two different scenarios: as a **single-step** and as a **multi-step** episodic Markov Decision Process (MDP). In both cases, we start with a dataset of images D. The RL **environment** is formed by the images space, where each image is divided into a grid of K square sub-areas of equal size $(x_h^1, x_h^2, ..., x_h^K)$. Those areas are observed by the RL agent, to take actions, switch states, compute rewards, and finally form an optimal policy deriving by its interaction with the environment. These scenarios can be perceived as two different pathways for a RL agent to achieve its goal: **learn the optimal policy** that solves for a specific task. Our approach involves a high resolution input network (U-Net) pretrained, and a low resolution input network (LR-ResNet) that interact within the RL loop. In the next sections, we develop the problem statements and solutions for both scenarios by following the formalism below:

x_l: the low resolution (LR) image
x_h: the high resolution (HR) image

x_h^m: the masked high resolution image
$\bar{y}$: target segmentation mask
y: predicted segmentation mask
f_p: the policy network (input x_l)
f_q: the segmentation network (input x_h^m)

To note that, the agent does not observe HR images at all, but their projections in the spatial dimensions of the LR image instead, which means that x_h, in this problem, is latent. Also, the agent does not have immediate access to any ground truth. A U-Net network is responsible to provide reward signals to the agent so that it forms its policy throughout the RL episodes.

4.1.1 Modeling a single-step episodic MDP

This scenario involves single state-action pairs in each *episode*, and thus the definition of states is simply the initial image x_h and the produced masked image x_h^m. Since we have a discrete action space, we formalize patch sampling *actions* with a square **action matrix** A_p, whose cells correspond to image patches. Action cells can take binary values indicating presence of fire (1) or not (0). Although this is the theoretical ground truth, the agent is not provided any, as already mentioned. Using this action matrix, the agent passes from *start state* s_0 to *final state* s_1. A state-transition example is illustrated in figure 4.1. The positions of the action array are K patch-specific independent random variables following a Bernoulli distribution:

$$P(A_p^i) = \begin{cases} p_i & \text{if } A_p^i = 1 \\ 1 - p_i & \text{if } A_p^i = 0 \end{cases} \tag{4.1}$$

for $i \in [0, K-1]$. Our **policy**, which defines the state transitions, is parameterized by θ as:

$$\pi_\theta(A_p|x_l) = P(A_p|x_l;\theta) \tag{4.2}$$

where $\pi_\theta(A_p|x_l)$ is a function mapping the observed LR image to a probability distribution over the patch sampling action A_p. This function outcome should be equal to the probability of an action given the LR image and the policy parameters. Thus, the policy can be described by the following joint distribution as a function of A_p:

$$\pi_\theta(A_p|x_l) = \prod_{i=1}^{K} p_i^{A_p^i}(1-p_i)^{(1-A_p^i)} \tag{4.3}$$

where $\boldsymbol{p}$ is a vector of K positions indicating the probability of each patch to be sampled. These distributions are estimated by the **policy network** f_p given the LR image and the policy parameters:

$$p = sigmoid(f_p(x_l;\theta)) \tag{4.4}$$

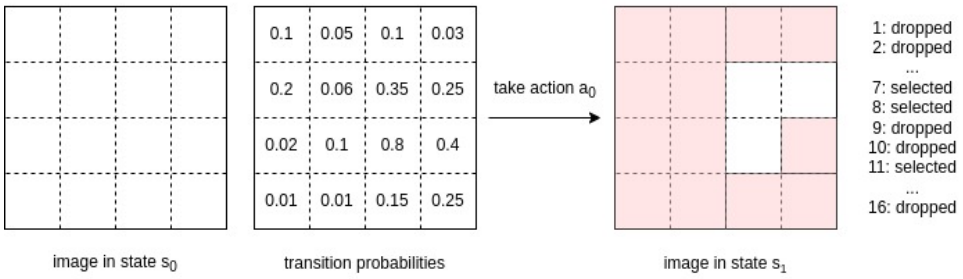


Figure 4.1: State transition for action a_0 after Monte Carlo sampling given the policy π. In the new state s_1 all patches are dropped except patches with indexes 7, 8 and 11.

In order to convert output logits to probabilistic values we use the *sigmoid* function. The agent's **reward** R depends on the action matrix and the prediction of f_q; actions determine the quantification of the performance and the sparsity of selected patches. Hence, the reward of a path can be evaluated as a function of the L1-norm of matrix A_p and the dice coefficient metric (2.20) between the f_q output prediction and the ground truth $\bar{y}$:

$$R(\tau) = dice(f_q(x_h^m), \bar{y}) - \left(\frac{|A_p|_1}{K}\right)^2 \tag{4.5}$$

Finally, the overall objective J can be defined as maximizing the **expected return** of all the episodes our agent is going through:

$$\max_{\theta} J(\pi_\theta) = \max_{\theta} \mathbb{E}[R(\tau)] \tag{4.6}$$

4.1.2 Modeling a multi-step episodic MDP

In contrast with the previous method, this one involves multiple state transitions over the process. At each step, the agent observes the current state, samples an action from a categorical distribution over patch indexes, and computes a cost-aware reward for the specific action. The policy, in this case, is a function of the trajectory state-action pairs, based on which the agent shall gather a minimal subset of image patches so that it maximizes the long-term expected return.

Action. In correspondence with the single-step case, the action representation is encapsulated in a square binary matrix A_p whose cells correspond to the K image patches, with consecutive indexing, called the **action matrix**. A_p is initially all zeros, indicating that none of the HR image patches is sampled. In every step of the process the agent samples exactly one patch until some condition is satisfied; thus it ends up with ones in the positions of selected HR patches. Figure 4.2 illustrates the evolution of an action matrix, projected in the original HR image

dimensions and applied in the image to change state, over a three-step episode. The state matrix is reset in the beginning of a new episode.

State. The definition of state is totally different in each episode, since we examine a different image. States can be perceived as x_h^m with all the possible combinations of patch samplings. However, the possible states where an agent can transition at a given time are exactly $K-i$, with $i \in \{0, ..., K\}$, if we consider that it explores a full path of transitions.

Policy. Given the low resolution image x_l, the agent must be able to produce the state-transition probabilities according to which it will perform the single patch selection. In the first step of the example 4.2, the agent has K possible moves, though in every next state those moves decrease by 1. Hence, we must reformulate the policy distribution by assigning zero probability to the action that is excluded from the next steps. Patch selection action is a single categorical variable of $K-i$ possible outcomes, with $i \in \{0, ..., K\}$, in contrast with the single-step case where it is a vector of K Bernoulli variables. The parameterized policy π_θ can be then expressed by the conditional probability distribution of state transitions as a function of the action matrix A_p:

$$\pi_\theta(A_p|S, x_l) = P(A_p|S, x_l; \theta) = P(a_0|S_0) \cdot P(a_1|S_1) \cdot ... \cdot P(a_{T-1}|S_{T-1}) = \prod_{t=0}^{T-1} P(a_t|S_t) \tag{4.7}$$

where S_t is the state occurring by the union of all the previous actions, a_t the action in time-step t and T denotes the total number of time-steps. We conclude in a final expression for the policy as a joint distribution of categorical variables:

$$\pi_\theta(A_p|S, x_l) = \prod_{t=0}^{T-1} \prod_{i=1}^{K} p_i^{A_p^i} \tag{4.8}$$

where $\mathbf{p}$ is a vector of K positions indicating the probability of each patch to be sampled. In this case, each category in $\{1, ..., K\}$ is mutually exclusive and sampling shall be performed without replacement in each trial. We approximate this distribution with a policy network f_p whose output logits are converted to probabilities with the *softmax* function:

$$p = softmax(f_p(x_l; \theta)) \tag{4.9}$$

Reward. The formula of the individual rewards R_t is 4.5. Nevertheless, at the end of an episode we must compute the expected utility of a trajectory $\mathbb{E}[R(\tau)]$ for $t \in \{0, ..., T-1\}$, in order to update the policy. To note that, each time the agent visits a state, can choose among a different set of actions. Each action leads to radically different future returns. However, the agent discards prior knowledge that comes from the returns of previous episodes, whereas it learns from them and starts exploring the environment from scratch. The objective J, similarly to before, will follow the equation 4.6.

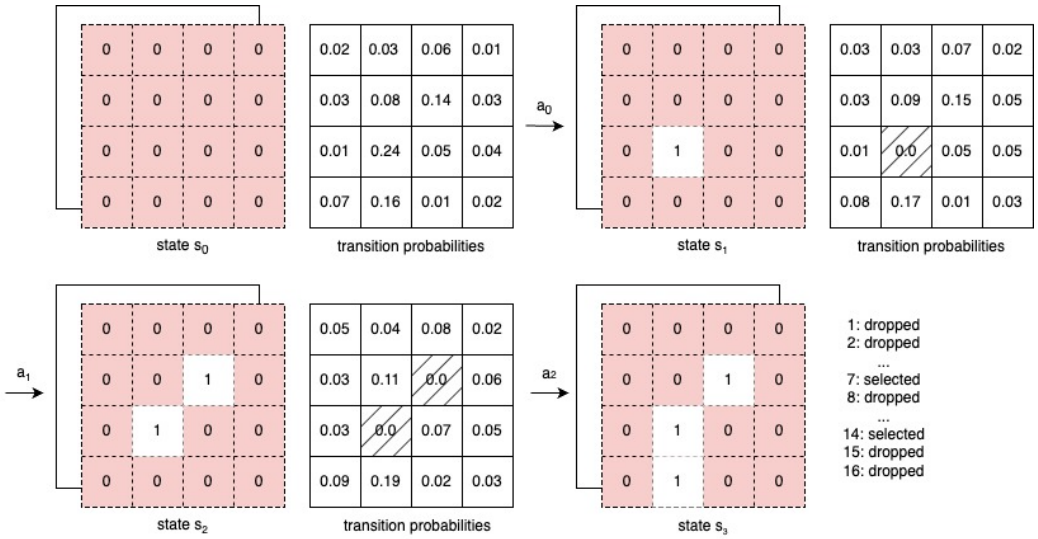


Figure 4.2: State transitions for a complete MDP with three steps. The agent is sampling one patch per step and the probability distribution is reformulated until some stopping condition is satisfied. On the bottom right one can see the selected (white) and the dropped (red) patches of x_h^m.

We conclude the formulation of the proposed framework by considering the Markov Property. In each step, we compute the probability of an action given the current state. Thus, the agent is not conditioning on previous actions, but only on their impact on the current state.

4.2 Training pipeline of the deep RL system

After defining the Markov Decision Processes, we detail the training pipeline of the RL system. The agent's goal is to learn the optimal parameters θ of the policy network f_p. To reach this goal we employ model-free reinforcement learning and specifically policy gradient [52]. In order to train the agent, we implement f_p with an architecture based on convolutional layers, and set the number of patches to K = 16. For the policy evaluation step, we make use of a pre-trained U-Net model [9] to produce a segmentation mask y of the original image. These two networks interact in an actor-critic way [22, 51]; the first one is learning based on the Rewards calculated upon the output of the second one. The training pipeline described below is illustrated in summary in figure 4.3.

4.2.1 Single-step episodes

In the beginning of an episode, the LR image x_l is observed by the agent to estimates state-transition probabilities for all actions at once, as shown in figure

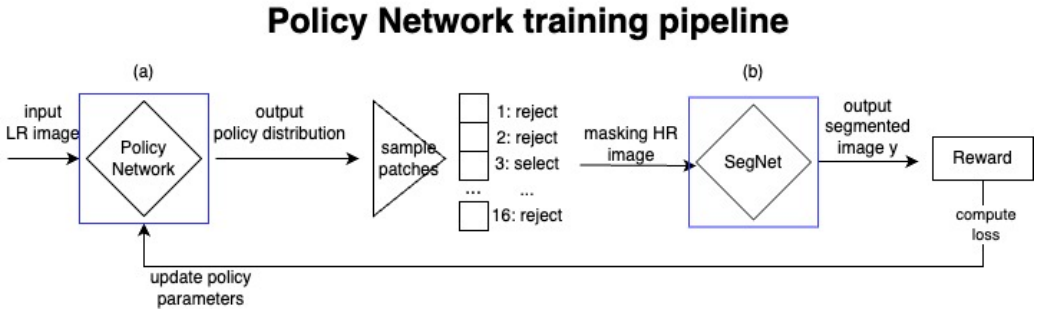


Figure 4.3: A flow diagram of training our computational sensing system with an agent interacting with an image segmentation mechanism. Network (a) is responsible for outputting probability distributions over patches in order to perform Monte Carlo sampling. Network (b) outputs the predicted segmentation mask that will be used to evaluate the reward and update the policy parameters in every iteration.

4.1. We introduce here the exploration-exploitation parameter $\epsilon \in [0, 1]$ with temperature scaling:

$$p = \epsilon \cdot p + (1 - \epsilon) \cdot (1 - p) \tag{4.10}$$

to encourage the agent to explore rather than clinging to the policy. Next, a sampling action is performed to get an outcome from each probability distribution p_i[1]:

$$A_p \sim p(s_G | x_l; \theta) \tag{4.11}$$

where s_G is the goal state. The masked HR image is then obtained by an element-wise multiplication with the action array as $x_h^m = x_h \odot A_p$. This is also depicted in fig. 4.4. The resulting image is then introduced to the U-Net model, which outputs the predicted segmentation mask. We binarize this mask by applying a threshold[2] to its pixel values and compare it with the ground truth $\bar{y}$ by calculating the *dice coefficient* metric (2.20) between the two masks. The reward of the path can be then evaluated using the equation 2.6. The utility of this reward is two fold; it considers the segmentation error between y and $\bar{y}$ while also penalizing the agent for selecting a large number of HR patches at the same time.

The possible transitioning states are in the order to 2^K which is a relatively large number to perform optimization with Q-learning [57]. Therefore we optimize by policy gradients with REINFORCE [52], and hence, the gradient of the objective is the following:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}[R(\tau) \nabla_\theta \log \pi_\theta(A_p | x_l)] \tag{4.12}$$

[1]When specific actions have lower probability, it doesn't mean they won't be picked at all, but that they are less likely to be picked.

[2]pixel values of the segmentation mask above 0.25 are marked as fire-present

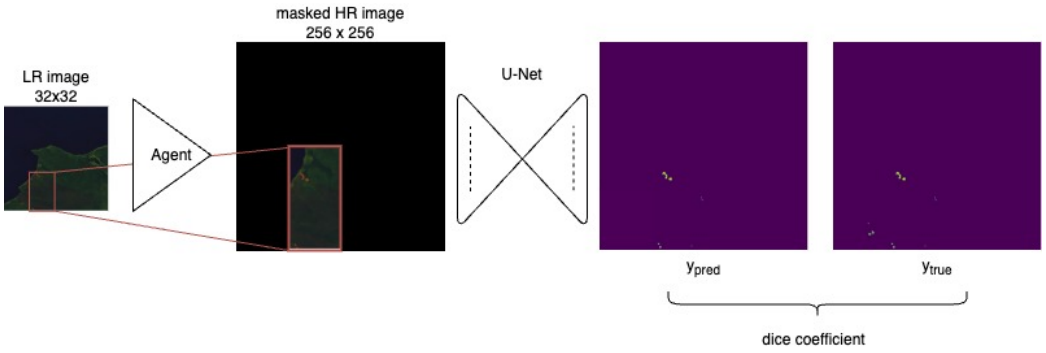


Figure 4.4: Visualization of the AoI mapping in the HR image and the corresponding segmentation mask of x_h^m altogether with the ground truth

where $R(\tau) = R(A_p, y)$ the reward for taking actions A_p and predicting y in a single-step trajectory. Averaging across a mini-batch via Monte-Carlo sampling produces an unbiased estimate of the expected value, but with potentially large variance. We have seen in practice that this can lead to an unstable training process, so we introduce a **baseline** action $\bar{A}_p$ and replace $R(\tau)$ with an advantage function [52] to reduce the variance:

$$A = R(A_p, y) - R(\bar{A}_p, y) \tag{4.13}$$

where $\bar{A}_p$ represents the baseline action matrix. To form it, we select the most likely patches i.e. the ones with predicted probability $p_i > 0.5$. This way we are following a self-critical baseline [41]. So the gradient now becomes:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}[A \sum_{i=1}^{K} \nabla_\theta \left(A_p^i \log p_i + (1 - A_p^i) \log(1 - p_i) \right)] \tag{4.14}$$

after substituting with the log-policy formula 4.3:

$$\begin{aligned} \log \pi_\theta(A_p|x_l) &= \log(p_1^{A_p^1}(1-p_1)^{(1-A_p^1)} \cdot \ldots \cdot p_K^{A_p^K}(1-p_K)^{(1-A_p^K)}) \\ &= \log(p_1^{A_p^1}(1-p_1)^{(1-A_p^1)} + \ldots + \log(p_K^{A_p^K}(1-p_K)^{(1-A_p^K)}) \\ &= \sum_{i=1}^{K} \log p_i^{A_p^i} + \log(1-p_i)^{1-A_p^i} \\ &= \sum_{i=1}^{K} A_p^i \log p_i + (1 - A_p^i) \log(1 - p_i) \end{aligned} \tag{4.15}$$

Algorithm 2 condenses the whole training pipeline. The procedure is repeated until the agent explores all the available dataset for a number of training **epochs**.

Algorithm 2 Training the policy network with **single** step episodes

Input: $x_l = \{x_l^1, x_l^2, ..., x_l^N\}$, the policy parameters θ.
for each episode τ **do**
$p \leftarrow f_p(x_l; \theta)$
$p \leftarrow \epsilon p + (1 - \epsilon)(1 - p)$
$A_p \sim \pi_\theta(.|p)$
$x_h^m = x_h \odot A_p$
$y = f_q(x_h^m)$
$R(\tau) \leftarrow -error(y, \bar{y}) - \left(\frac{|A_p|_1}{K}\right)^2$
$\theta_{k+1} \leftarrow \theta_k + \alpha \mathbb{E}[R(\tau) \nabla_\theta \log \pi_\theta(A_p|x_l)]$
end for

4.2.2 Multi-step episodes

To train the agent with multi-step episodes, we follow the model of 4.1.2 and propose the Algorithm 3. The difference from the previous training procedure lies in the formulation of the Markov Decision Process. Each episode composes of T time-steps so the policy network f_p does not output the probability of each patch at once but a probability distribution over all the K patches summing up to 1. The agent is then selecting one patch by Monte Carlo sampling and stores the action on the action matrix A_p. For every next move it samples from a new distribution p_t, appends the action to A_p and computes the individual reward R_t for being in the current state x_h^m. Likewise, in the end of an episode, the expected reward $\mathbb{E}[R_t]$ is evaluated and an update step is performed, considering the log-probability of the initial output distribution of f_p. In case we are assuming batches of data for training, the update rule will consider the expectation of $R(\tau_i)$ which is the utility of M trajectories, for $i \in 1, ..., M$.

4.3 Choice of Neural Network Architectures

Depending on our dataset size, we are using a Convolutional Neural Network (CNN) [23] or a Residual Neural Network (ResNet) [15] to model the policy network. The CNN architecture consists of three convolutional layers followed by a ReLU function [37]. The first one outputs 16 channels, the second one 32 and the third one 64. A max pooling layer in applied to the input tensor after each convolutional layer. Lastly, a fully connected layer flattens the final feature map and outputs K scores. If needed, we apply a sigmoid or a softmax function to obtain probabilistic values on the last layer. The total number of trainable parameters is 28,096.

For the ResNet we are using two variants; a ResNet18 (18 layers overall) and a smaller ResNet with two convolutional layers in each block (12 layers overall). Both models consist of four residual blocks whose convolutional layers are followed

Algorithm 3 Training the policy network with **multiple** step episodes

Input: $x_l = \{x_l^1, x_l^2, ..., x_l^N\}$, the policy parameters θ.
for each episode τ **do**
  Start with state s_0.
  $p \leftarrow f_p(x_l; \theta)$
  $p \leftarrow \epsilon p + (1 - \epsilon)(1 - p)$
  for each time step t $= \{0, ..., T - 1\}$ **do**
    $a_t \sim \pi_\theta(.|p_t)$
    $A_p^t = A_p^t + a_t$
    $x_h^m = x_h \odot A_p^t$
    $y_t = f_q(x_h^m)$
    $R_t \leftarrow -error(y_t, \bar{y_t}) - \left(\frac{|A_p^t|_1}{K}\right)^2$
  end for
  $\theta_{k+1} \leftarrow \theta_k + \alpha \mathbb{E}[R_t \nabla_\theta \log \pi_\theta(A_p|x_l)]$
end for

by a Batch Normalization layer [18] and a ReLU. After the global average pooling, the feature map is finally flattened by a fully connected layer to produce the final scores. The first model (ResNet18) has 11,184,720 trainable parameters while the second one has 4,914,000 trainable parameters.

Regarding the U-Net architecture, we are using the lightweight version of U-Net proposed in [9] with 3-channel input (U-Net-Light-3c) and 16 filters in the first layer. In the process of selecting the model, we noted that, with less parameters and reduced number of input channels the model performs similarly or even better in our dataset. This can be confirmed by the benchmarks of the relevant work. We implement the model structure in PyTorch and adapt the pre-trained weights to this architecture. Those weights, were estimated after training with target masks obtained by the combination of three handcrafted algorithms discussed in the relevant paper.

Chapter 5

Experimental Evaluation

5.1 Dataset

Landsat-8 is an American Earth observation satellite launched in 2013 as a product of cooperation between NASA and the United States Geological Survey (USGS). The objective of the Landsat Data Continuity Mission (LDCM) launched by the aforementioned organizations is to collect and archive medium resolution multispectral image data affording seasonal coverage of the global landmasses. The captured images are segmented into scenes with 185 $\times$ 180 km, defined according to the second World-wide Reference System (WRS) in a 16-day revisit period. The Landsat-8 dataset that we use to train and evaluate our system, consists of real time images available for August 2020 around the globe.

Specifically, each data point contains eleven spectral bands $c_1, c_2, ..., c_{11}$ among which we stand out three c_7, c_6, c_2 (RGB) for the purpose of image segmentation. The spatial resolution of the original images is 7,600 $\times$ 7,600, albeit they were split to 256 x 256 sized HR patches by de Almeida Pereira et al. [9] for the purposes of the relevant work, forming a manually annotated dataset of images and segmentation masks. The images are captured from different geographical areas, categorized into 6 continents: Africa, Asia, Europe, North America, South America and Oceania. For the current thesis, we made use of the **Europe** dataset and performed data preparation and preprocessing to result in the final image and target sets. Finally, we performed exploratory analysis to handle and manipulate the data optimally in all phases of the experimental evaluation.

5.1.1 Data Preparation

The Europe dataset consists of 46,659 items overall (images and masks). 14,110 of them are 11-band images, among which we separated 6,179 that correspond to masks obtained by the **voting** scheme. The latter one is a setting, according to which the target masks are the intersection of at least 2 of the handcrafted sets of conditions proposed by Kumar and Roy (2018) [24], Murphy et al. (2016) [35], and Schroeder et al. (2014) [45].

Altogether with the big datasets of HR patches, a smaller one of 100 samples is given by the authors of [9]. From the dataset of 6,179 we randomly sample 256, 512 and 1024 images with their corresponding mask targets for the realization of separate experiments. The choice of sizes of the data subsets was made with a view to optimizing GPU capacity in every training iteration.

5.1.2 Preprocessing

As a preprocessing step, we first extract the channels 7, 6 and 2 of the multi-spectral image, we perform normalization based on the max pixel value and lastly, we sub-sample the image to 32x32 (or 64x64) which corresponds to 8 (or 4 respectively) times smaller spatial dimensions. This last step is made in order to feed the images to the deep neural network which forms our agent's policy. For the second neural network (U-Net) the HR images are used as is by projecting the action array to the actual HR dimensions resulting in the final masked image.

5.2 Agent Performance on Landsat-8

We train our RL agent with the framework proposed in 4.2.1 and delve into the explanation of key metrics obtained during the training process. The system was designed with a focus in the Reward, which is a means of maximization in RL environments. Also, we assess the evolution of dice coefficient throughout the process, as this is the most suitable metric to quantify the error between image segmentation masks. Figures 5.1, 5.2, 5.3 and 5.4 illustrate the evolution of reward, dice coefficient and sparsity during training on 100, 256, 512 and 1024 samples respectively.

Dice coefficient (middle plot) measures the similarity of U-Net prediction with the target mask and takes values in the range $[0, 1]$. Thus, we should see the value of this metric increase in order to verify that the algorithm works correctly. **Sparsity** (right plot) measures the average number of sampled patches in the images; in our problem it takes values from 0 to 16. Thus, we should see its value

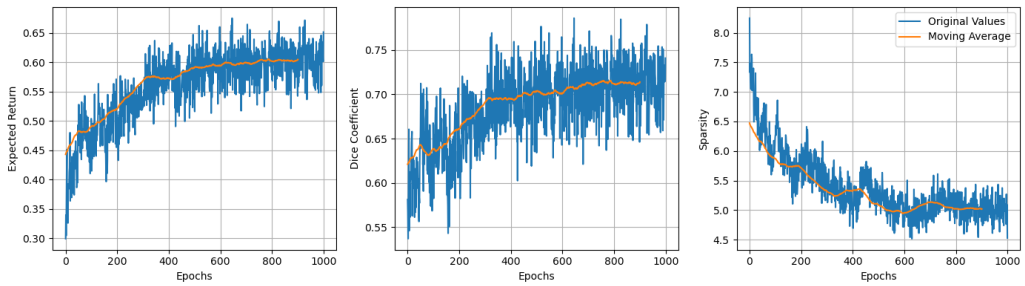


Figure 5.1: Training the Agent with 100 samples: the average return is a function of dice coefficient and sparsity

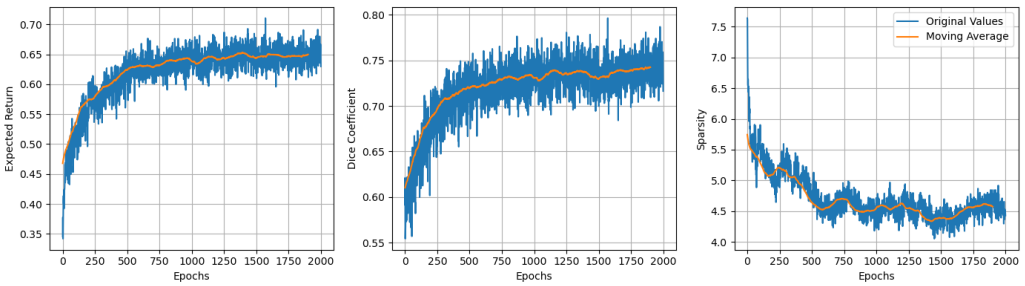


Figure 5.2: Training the Agent with 256 samples: the average return is a function of dice coefficient and sparsity

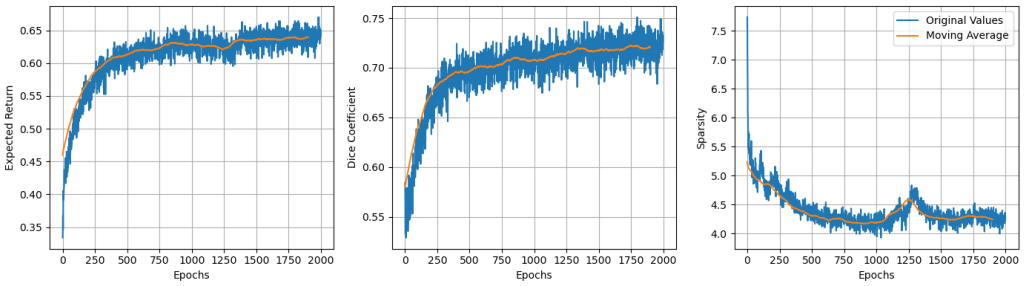


Figure 5.3: Training the Agent with 512 samples: the average return is a function of dice coefficient and sparsity

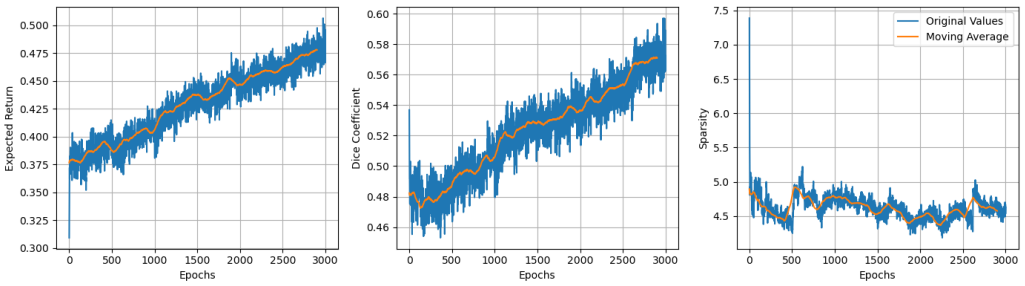


Figure 5.4: Training the Agent with 1024 samples: the average return is a function of dice coefficient and sparsity

decrease over epochs. A rational behavior for the agent is to start with selecting a lot of patches and gradually learn to select less while maintaining its dice coefficient score.

The **Reward** is a function of dice coefficient and sparsity (2.6). With an increasing F1-dice and decreasing sparsity, the reward tends to increase, and this is a desired behaviour; by maximizing the expected return, we maximize the Agent's objective 2.11. However, each time the agent visits a state, can choose different actions which leads to radically different future returns. The reason for this lies in two folds; (1) the agent performs Bernoulli random sampling after observing the policy and (2) we introduce the exploration-exploitation parameter ϵ. Both the aforementioned factors introduce **stochasticity** in the process and hence, force the agent to try different actions. This way, it explores its environment while it avoids getting stuck in a local optimum.

By plotting the return for every epoch gives us the average reward over all the episodes[1]. We observe that this line is not monotonically increasing, instead it shows a lot of **noise**. In fact, training via Policy Gradients exhibits this behavior due to the reasons we mention above; so we should get the sense of an increasing trend by fitting a **moving average** line. Apart from optimizing with the Expectation of Rewards over a batch of trajectories, introducing a **baseline** (4.13) also helps decrease the variance between iterations.

Figures 5.1-5.4 demonstrate that increasing the number of sample size, we need more epochs to converge, which confirms our intuition. Specifically, training with 100 samples reaches an average of 60% for the return. F1-dice score converges over 72% and the average number of sampled patches starts from 8 (half of the image patches) and decreases to 5. Similar behavior we observe for the 256 and 512 sample sizes, but with a slight increase in performance. Despite the fact that the smaller datasets follow the same trends at training, for the dataset of 1024 samples we observe a continuous improvement until epoch 3000, which testifies that -although the improvement is slower- with more epochs we can achieve even better scores.

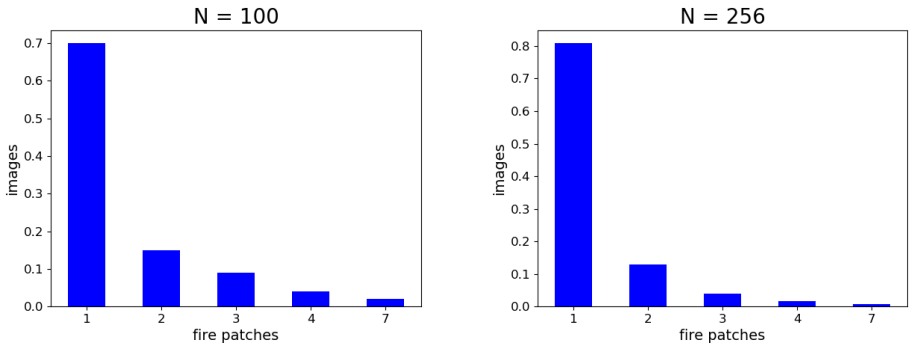
5.2.1 Data Exploration

In this section, we discuss the exploratory analysis we performed on the data, in order to detect specific features and proceed to a correct data split for training and testing. This way we shall unearth patterns that will aid towards improving the performance of our algorithm.

As a first step, we need to create **custom labels** for the images, indicating which patches include fire pixels and which do not. This is represented with a binary vector of K positions. To infer a label, we apply **thresholding** on the percentage of fire pixels in the patch. By applying a very small threshold (1%) we discriminate the fire-present images from those who do not contain any fire patch

[1] until the goal state, which is the end of the dataset

and proceed with processing the fire-present ones.



(a) 100-sample dataset (b) 256-sample dataset

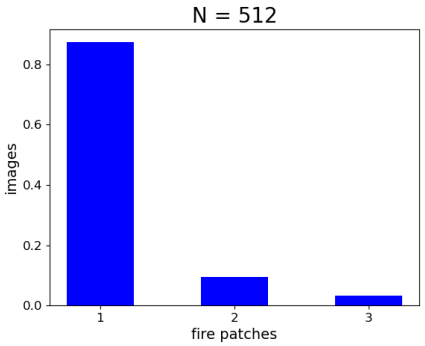
Figure 5.5: The distribution of fire patches in the dataset of 100 samples (a) and the dataset of 256 samples (b).

An interesting finding here, is that both the datasets of 100 and 6,179 samples contain only **fire-present images**. Figures 5.5 and 5.6 illustrate the fire-patch distribution over the different datasets. The dataset of 256 samples is a superset of the 100-sample dataset, occurring after randomly adding observations from the (independent) dataset of 6179 samples. The datasets of 512 and 1024 samples are random subsets of the 6179-sample dataset.

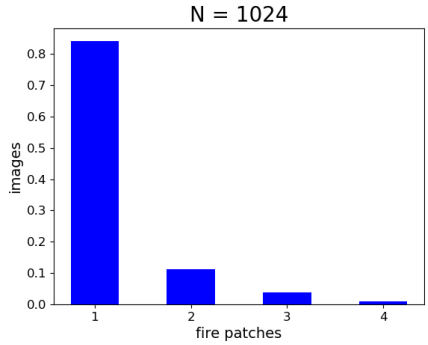
We observe that all datasets seem to follow a **geometric distribution** with the majority of samples clustering on the value 1. The frequency of fire-patch representation decreases exponentially. Moreover, the dataset of 6179 contains extremely few images with 4, 5 and 6 fire-patches. Similarly, the 100-sample dataset contains just 4 images of 4 fire-patches and 2 images of 7 fire-patches. Certainly, we would not wish those samples to be attributed to the test set; with a random split, however, we cannot control this outcome.

After this exploratory step, it is a natural sequent to perform **stratification** on the dataset. We customize this procedure by crafting the splits as shown in figure 5.7, which demonstrates the splitting of the 100-sample dataset. We conclude in M sets of image-target splits, where M is the number of different fire patch counts in the images of the dataset. Then we concatenate them and perform **randomization** to form the final train-test splits. We end up with a robust dataset, ensuring that the distribution of fire patches in the images is the same in both training and testing phases.

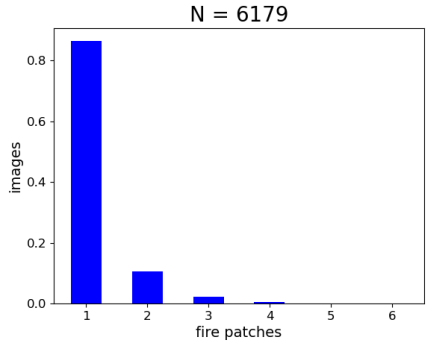
The importance of this kind of splitting for training a RL algorithm lies in the fact that the agent should explore as many states of its environment as there are possible during training, and then being able to recognize the patterns in unseen data based on the obtained knowledge. This cannot be achieved if the training set is missing a representative sample, i.e. images with at least the plenity of



(a) 512-sample dataset



(b) 1024-sample dataset



(c) 6179-sample dataset

Figure 5.6: The distribution of fire patches in the full dataset of 6179 samples (c) and its subsets of 512 (a) and 1024 samples (b).

fire-presence in the test set.

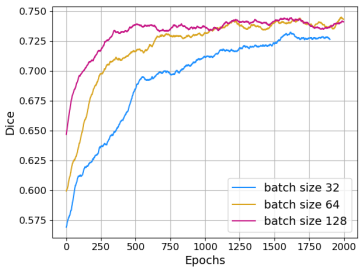
Figure 5.7: Data split of the 100-sample dataset based on the number of fire-present patches

5.2.2 Choice of hyperparameters

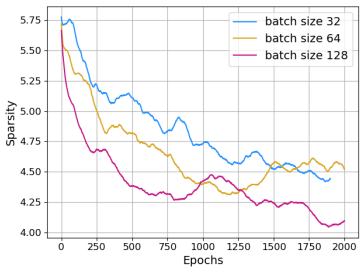
The configurations of figures 5.1, 5.2, 5.3 and 5.4 were trained with a **batch size** of 32, 64, 128 and 256 respectively. We increase the batch size as the dataset size increases because with more data we have a more versatile environment and thus, the agent should be able to explore more states in every iteration. Figure 5.8 highlights the impact of the batch size in the reward, dice coefficient and sparsity level. We tune this hyperparameter for the dataset of 256 samples, as it contains few enough samples to save training time but, at the same time, the desired versatility to exhibit the pattern. However this behavior has been observed regardless of the dataset size; the higher the batch size, the quicker the convergence rate and the smaller the variance of the performance metrics. Lastly, one can reduce the noise further, by running much more episodes. Hence, with a higher batch size not only we have a faster convergence, but also a more stable training process.

The rest of the hyperparameters are the **learning rate** (α), the **epsilon greedy parameter** ϵ, the **sub-sampling ratio** and the number of **epochs**. At training time, we are using Adam optimizer and set $\alpha = 0.001$. ϵ is used to balance exploration and exploitation trade-off; the relevant literature [53] is tuning ϵ to 0.8, so we employ this value too. We sub-sample the original images so they have 8 times smaller spatial dimensions, hence the low-resolution image size is 32 x 32. In this setting, and depending on the dataset, the performance of the algorithm converges to an optimum within 2000 epochs or more.

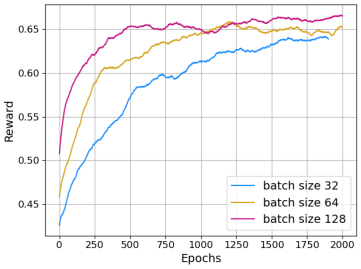
Although we do not perform model selection, we conclude that the datasets of 256 and 512 samples exhibit their best performance with a ResNet of 12 layers. The dataset of 100 samples is learnt by a CNN and the one of 1024 samples by a



(a) Dice Coefficient



(b) Sparsity level



(c) Average Return

Figure 5.8: Training the Agent with different batch sizes; we observe faster convergence and better performance overall as the batch size increases.

ResNet18. With an increasing dataset size, we need more complex models, albeit complex models have difficulty to learn the patterns of smaller datasets. Finally, it is worth mentioning that the agent's **penalty** for bad segmentation is not a static parameter; instead, it stems from dice coefficient and patch use.

5.2.3 Improving performance with stratification

Training with a data split described in 5.2.1 and computing the moving average lines of rewards and metrics, exhibits the results of figures 5.9, 5.10 and 5.11. Regardless of the dataset size, the improvement in performance is noticeable after deducting fairly the data split, in context of fire presence. The red line represents the training with stratified datasets, whereas the black, training with non-stratified, ignoring the fire-patch distributions. The improvement is especially highlighted in the dataset of 1024 samples (5.11), where the model achieves a reward 0.48 after 3000 epochs when training with randomly attributed training set, whereas 0.55 with the stratified.

In this case, it is worth considering the test performance of the algorithm which is depicted in table 5.1.

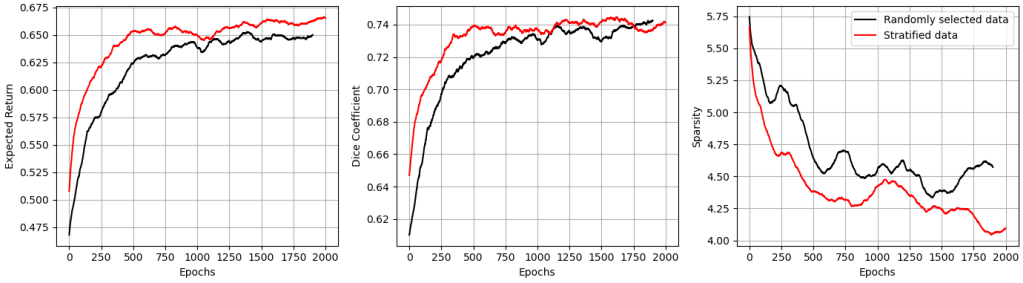


Figure 5.9: Training with stratified vs. non-stratified 256 samples

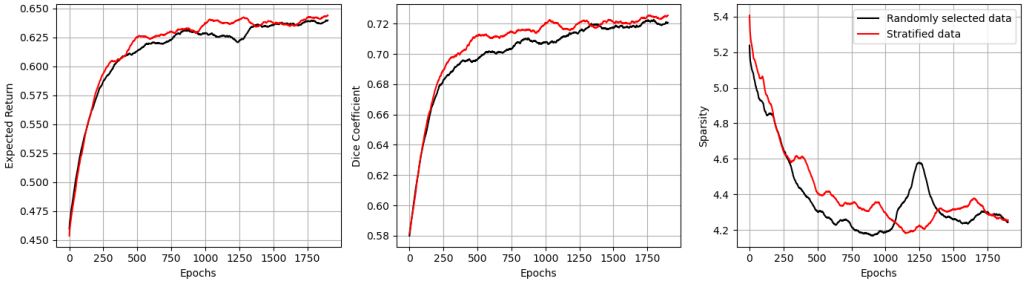


Figure 5.10: Training with stratified vs. non-stratified 512 samples

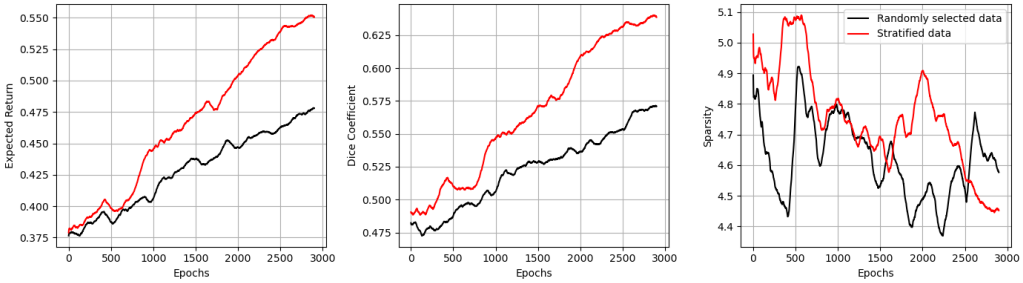


Figure 5.11: Training with stratified vs. non-stratified 1024 samples

Samples	256	512	1024
stratified	0.533	0.515	0.488
non-stratified	0.485	0.453	0.424

Table 5.1: Rewards obtained after testing with stratified vs. non-stratified datasets under the same hyperparameter sets. The first category hits higher scores in unseed data.

5.3 Baseline Methods

In order to evaluate our system, we design simpler, alternative methods to detect fire indexes from low resolution image inputs. These methods are evaluated on the stratified test sets and compared successively in section 5.4 with our deep RL model.

5.3.1 No patch sampling

In this approach, we simply observe the LR images and up-sample them to HR dimensions in order to feed-forward them to U-Net. This approach involves an up-sampling error stemming from the interpolation and assumes that we must capture the full image area if needed, via remote sensing.

5.3.2 Random sampling from a statistical distribution of fire-present areas

This approach performs stochastic patch sampling; we model the distribution of fire patches and proceed in randomly sampling a number of patches k to draw. We then uniformly sample k indexes from the image. In contrast with the previous method, we do not up-sample the LR image, instead we apply the mask grid immediately to the HR image excluding the non-selected patch indexes. Although in this case we escape the spatial resolution drawback, there is no guarantee of improvement in segmentation.

5.3.3 Exploring fire threshold with multi-label classification

This approach requires the custom labeling described in the *Data Exploration* section. To evaluate on this approach:

- We create five different datasets for labels produced by the thresholds $\{0.01, 0.02, 0.03, 0.04, 0.05\}$. In order to do so we split the image in individual patches and apply the thresholding on the fire-pixel ratio to label each patch.

- We train a multi-label ResNet classifier with binary cross-entropy with logits error function and Adam optimizer; we then test it on LR images.

- We are using F-score to quantify the training performance as other metrics like accuracy do not consider the class imbalance; in this case the black pixels of the mask outweigh significantly the white.
- Based on the ResNet predictions we obtain masked images conditioning on the maximum likelihood criterion. These models will be compared with the RL approach in regard to the predicted segmentation mask of U-Net, given the masked image. Thus, we feed masked images to U-Net and compute the F1-dice and IoU scores between prediction and target of the corresponding threshold.

The results are shown in figure 5.12. We see that the loss converges quickly to zero and the F-score to one after 20 epochs. However, the question that arises is whether this model has good performance on data that has not been trained. This is depicted in table 5.2 where the dice coefficient and IoU scores are indicated for the different dataset sizes on test set.

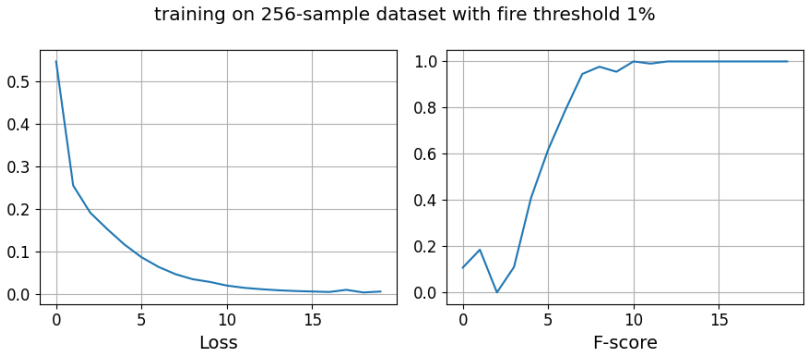


Figure 5.12: The classifier trained with custom labels maximizes quickly its training performance. This is an indicative example of training with 1% fire threshold on custom labels.

To note that, for testing we hold-out the 15% of each dataset, so the scores are obtained from these data subsets. Holding out such a small amount out of 100 samples, it is expected for the model to struggle at recognizing the patterns of the test set, even with stratified splits. Moreover, as the sample size increases, we observe a general increase in performance which is also an expected behavior. However, the overall test performance shows that the system **overfits** the data.

The cells in bold represent the best configuration based on the F1-dice (they all occur for the smallest threshold value) and we rely on these to compare with the other approaches. Nevertheless, setting a fixed threshold to the pixel values is not an optimal solution as, apart from the difficulty in generalization, we cannot rely on custom labeling due to (1) inability to examine the full range of real numbers to obtain an optimal threshold and (2) the selection of a very small threshold might

	threshold	0.01	0.02	0.03	0.04	0.05
100 samples	F1-dice	**0.348**	0.301	0.268	0.268	0.268
	IoU	0.067	0.033	0.0	0.0	0.0
256 samples	F1-dice	**0.303**	0.296	0.294	0.294	0.294
	IoU	0.030	0.013	0.011	0.003	0.0
512 samples	F1-dice	**0.312**	0.279	0.275	0.241	0.242
	IoU	0.020	0.0	0.0	0.0	0.0
1024 samples	F1-dice	**0.372**	0.354	0.346	0.338	0.338
	IoU	0.021	0.014	0.009	0.0	0.0

Table 5.2: Inference scores of the multi-label classifier for the different sample sizes.

be sensitive to potential U-Net prediction error. In conclusion, setting a threshold in fire pixels percentage to obtain the labels might work in practice, but the choice is arbitrary and impossible to be explored exhaustively.

5.4 Comparative Evaluation

In this section we discuss the performance of our algorithm compared to the proposed baseline methods. Tables 5.3 to 5.6 present the inference scores stemming from the best model of each approach.

100 samples	Up-sampling	Stochastic sampling	Multilabel	**RL Agent**
F1-dice	0.252	0.352	0.348	**0.483**
IoU	0.009	0.086	0.067	**0.147**

Table 5.3: Testing IoU and F1-dice for 100 samples

256 samples	Up-sampling	Stochastic sampling	Multilabel	**RL Agent**
F1-dice	0.268	0.336	0.303	**0.528**
IoU	0.005	0.054	0.030	**0.168**

Table 5.4: Testing IoU and F1-dice for 256 samples

We observe that stochastic sampling exhibits better scores than the up-sampling method; at some cases it performs even better than the multi-label framework (testing for 100, 256 and 1024 samples). However, for 512 samples it shows lower scores, and this is due to lack of diversity in the patch outcomes, as testified in the

512 samples	Up-sampling	Stochastic sampling	Multilabel	**RL Agent**
F1-dice	0.259	0.292	0.312	**0.544**
IoU	0.005	0.029	0.020	**0.173**

Table 5.5: Testing IoU and F1-dice for 512 samples

1024 samples	Up-sampling	Stochastic sampling	Multilabel	**RL Agent**
F1-dice	0.302	0.381	0.372	**0.505**
IoU	0.019	0.103	0.021	**0.190**

Table 5.6: Testing IoU and F1-dice for 1024 samples

distribution of figure 5.6 (b) during the exploratory analysis. The best segmentation scores are achieved by the RL Agent; the latter one is using the maximum likelihood criterion on the given policy conditioning on the LR images of the test set, and then inputs the masked HR images to U-Net to obtain the prediction. The results indicate that our self-supervised model outperforms the alternative probabilistic and supervised models, when trained and tested for a small amount of data, as we examine in our use case. The multi-label classifier overfits the training data, while the other two methods do not have enough expressivity to highlight the patterns.

5.5 Qualitative results

In this section we visualize how the learnt policies of the agent produce actions upon the image grid and discuss the agent's behavior at inference time. The figures 5.13 to 5.16 depict several cases of fire front detection from low-resolution (LR) image input. We visualize the extracted bands of the original image in high-resolution (HR), the produced masked HR image and the target segmentation mask to evaluate and explain visually both the successful cases and the possible limitations or errors.

Figure 5.13 demonstrates indicative cases where the model selects all the correct patches. Some of them show that the fire was detected although it occupied a small part of the image, meaning that the fire was either in an initial stage, or partially covered by clouds, or simply indiscernible due to the relative sampling distance. Figure 5.15 demonstrates cases where the correct patches were detected altogether with redundant patches. We observe that this is the most common behavior, as the agent rarely selects exactly the areas where fire front occurs; it usually samples 1 or 2 additional patches. Even in this case, the fire issue is addressed, but with some penalty of sampling extra areas. Figure 5.16, on the other side, shows the cases where the model did not detect the fire fronts at all, instead it signaled false positives. This occasional behavior comes with the cost of capturing only redundant areas in high resolution.

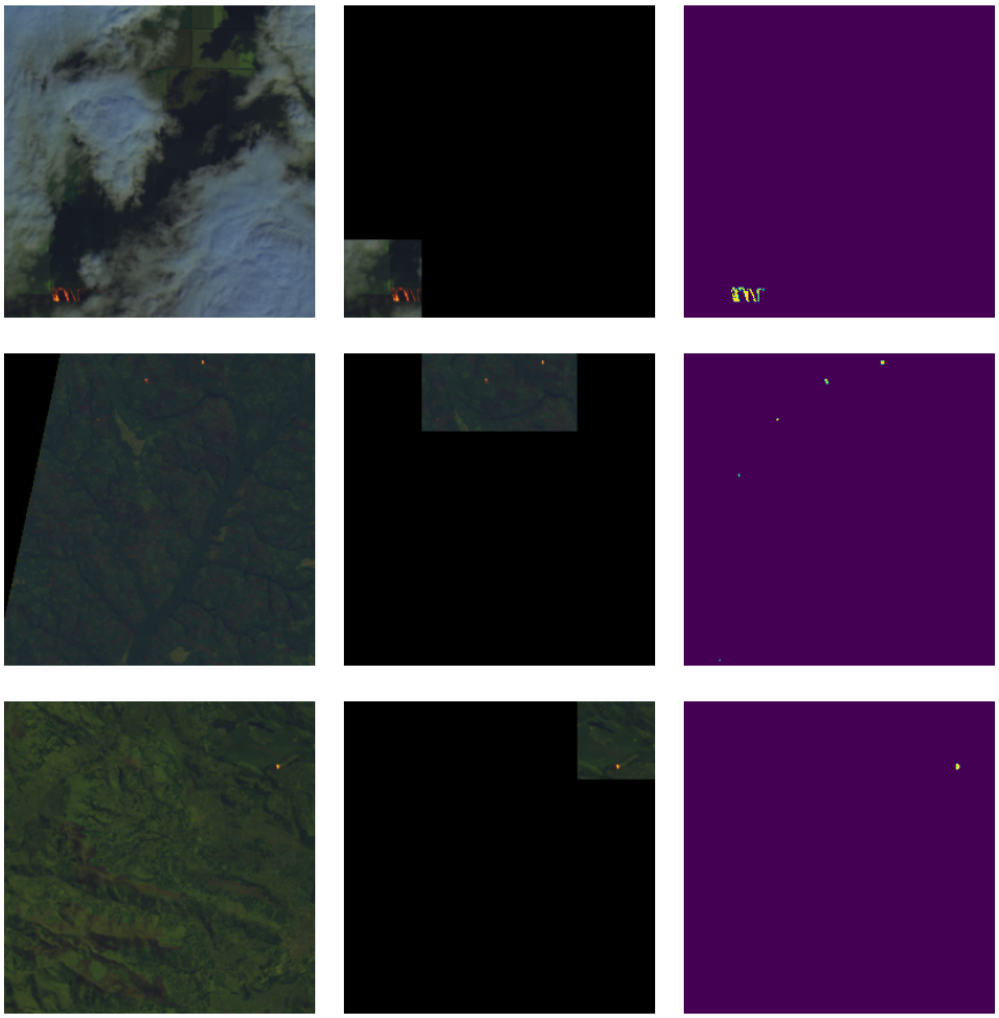
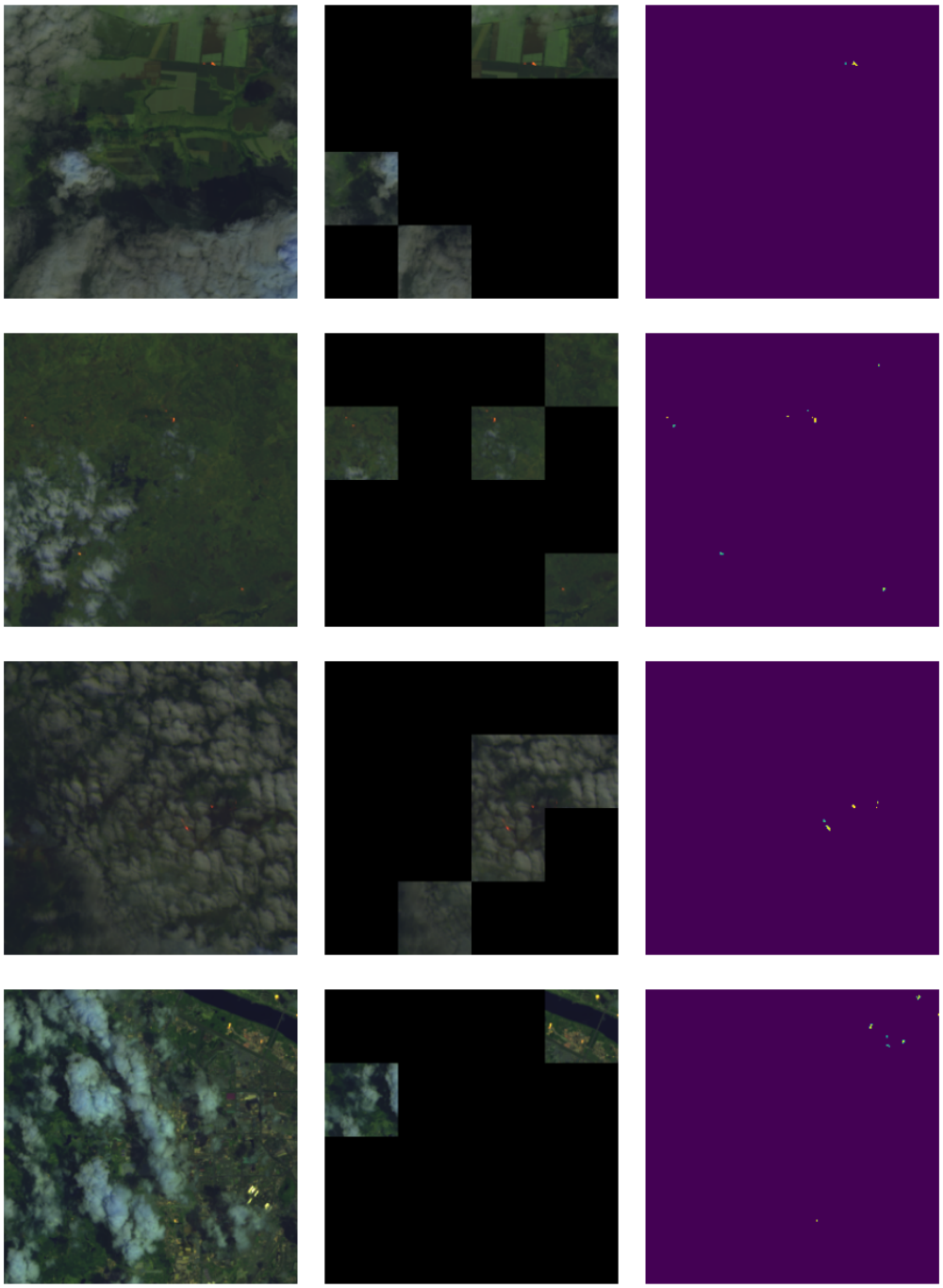


Figure 5.13: Policies learnt from LR images of Landsat-8 - cases of **correct** prediction: The original HR image (left), the masked HR image (middle) and the target segmentation mask (right)



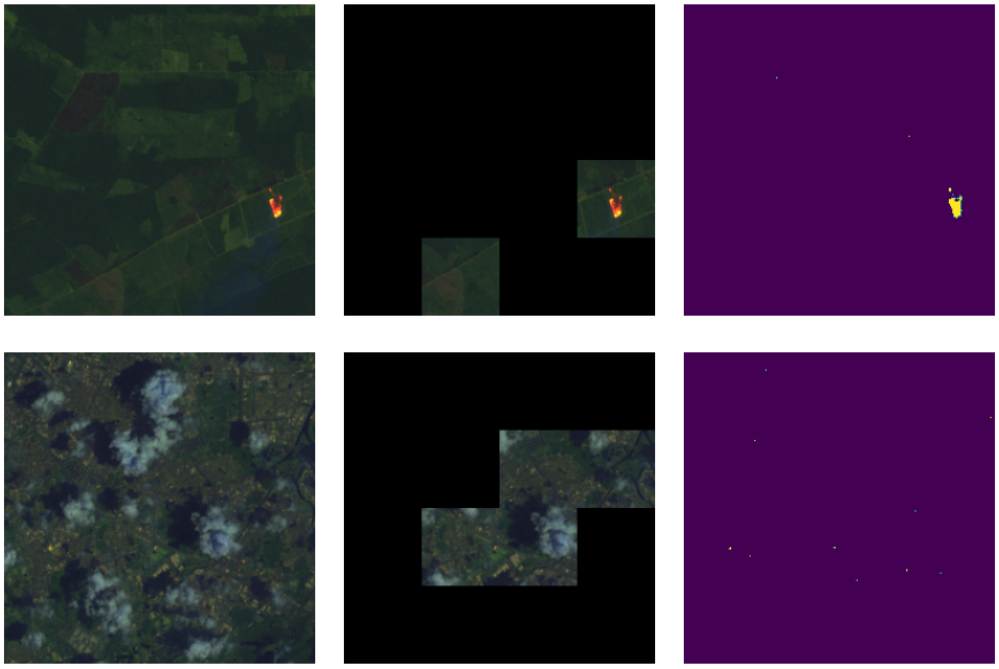


Figure 5.15: Policies learnt from LR images of Landsat-8 (cases of **partially correct** prediction): The original HR image (left), the masked HR image (middle) and the target segmentation mask (right)

In contrast with **classification** problems, the AoI does not occupy necessarily the center of the image, or the biggest part of it. Instead, the agent shall be able to detect the front even if it appears at an extreme corner or it is not distinct by the human eye. Furthermore, the front might appear in discontinuous areas, thus the detection should not focus on specific area of pixels.

In contrast with **object detection**, we do not aim at drawing bounding boxes around distinct objects and assign them a label. We simply aim at delineate discrete areas where we would wish to zoom-in; that will help us address the desired mapping of the AoI to spatial dimensions via pixel-level classification.

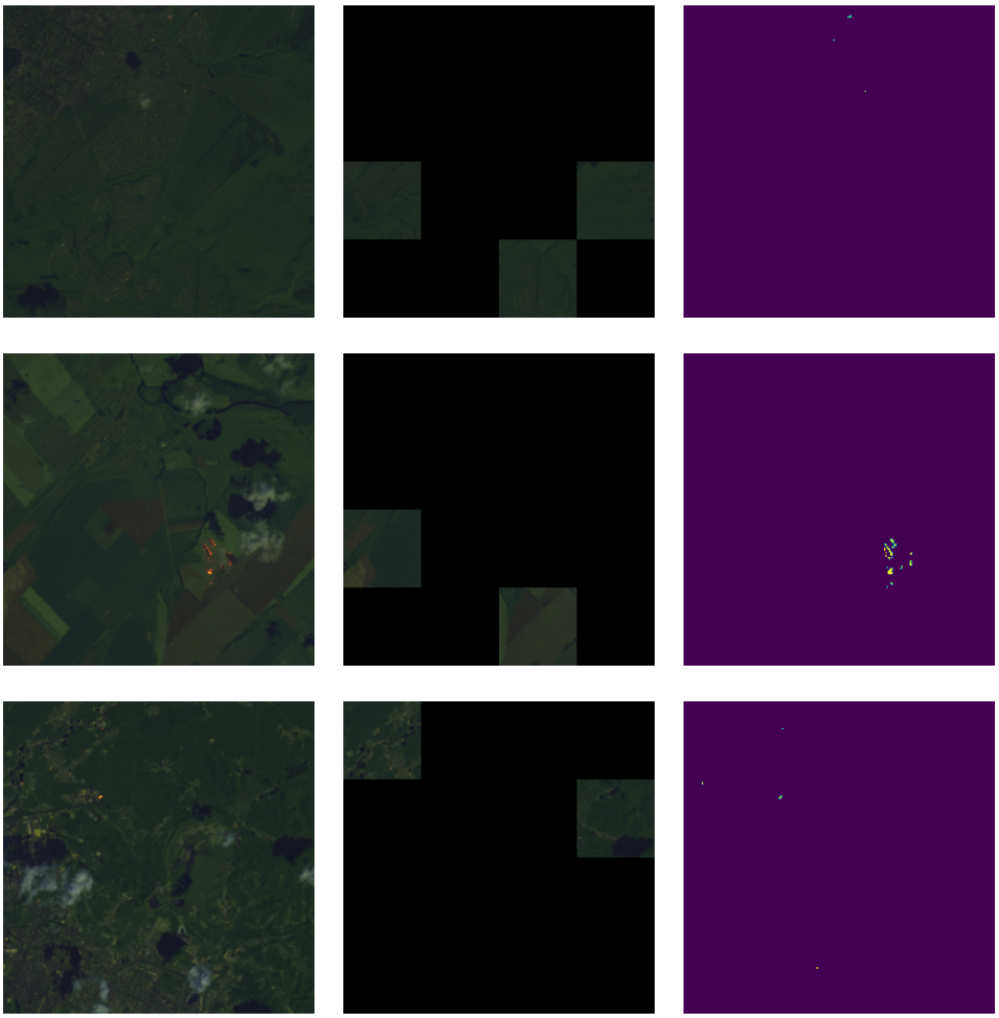


Figure 5.16: Policies learnt from LR images of Landsat-8 (cases of **incorrect** prediction): The original HR image (left), the masked HR image (middle) and the target segmentation mask (right)

Chapter 6

Conclusions

6.1 Discussion

Active fire detection from images is a demanding challenge that has been addressed by numerous studies in the field of Signal Processing. In this thesis, we examine the problem of fire front localization in discrete image sub-areas, opting to balance the trade-off between spatial and temporal resolution of satellite observations. We propose a Machine Learning pipeline whose predictive model operates leveraging Reinforcement Learning (RL) principles. Throughout the training process, our RL agent learns to recognize patterns that lead to better future decisions regarding the selection of AoI in fire-present images.

We conduct experiments that highlight the properties of our learning method; Policy Gradient methods exhibit smoothly evolving action choices towards the direction of the objective, and hence result in a long-term increasing Reward. With the right choice of hyperparameters, considering a specific environment, the algorithm can have good convergence properties. On the other hand, due to various limitations, it can be very slow or even inefficient in learning the desired patterns.

Our agent is designed to operate in a model-free RL scheme; it does not focus on the overall dynamics of the environment, on the contrary it only solves for the policy parameters. Additionally, RL methods often involve exploration, where the agent pursues different actions to gather information about the environment and learn an optimal policy. This exploration can lead to fluctuations in the rewards as the agent explores different actions and learns from the outcomes. Our agent is also considering a baseline policy, and thus avoids static selection of actions with maximum probability; this leads to a more stable training process.

We found out that balancing the observations in the train and test sets considering the fire presence, improves significantly the detection performance. Also, the hyperparameters to which the model is more sensitive are the batch size and number of epochs. Therefore, we opt for increasing their values to the extent that our resources allow it. We train and test our system for various data sizes and

develop baseline methods to comparatively evaluate its performance. The promising supervised approach we proposed with custom labeling, although it exhibited fast convergence at training, it underperformed on the test sets. We infer that the RL system outperforms the developed baseline methods in terms of dice coefficient and IoU metrics for small sample sizes at test time.

Finally, we seek to explain the agent actions via their visualization upon the latent image. Guided by the policy, the agent drops the larger amount of the image, which contains non-relevant information with respect to fire presence. It has been observed that, in the majority of cases, the agent samples the correct patches altogether with 1 or 2 redundant patches on average. The latter ones, however, undermine the evaluation performance of the agent as they dynamically penalize it with increased patch use.

6.2 Future work

Considering the limitations but also the potential of this work, we propose the following future steps:

- Increasing the size of the dataset; this can provide the agent more diverse experiences and potentially improve its learning and convergence properties and speed. This endeavor requires extremely powerful resources.
- Finetuning the multi-label classification framework with the proposed RL scheme; this approach would consider the prior multi-label classifier as the policy network and optimize its parameters w.r.t the RL objective. We expect to combine the speed of training of the first approach with the good generalization of the second.
- Further investigating the impact of the hyperparameters *epsilon greedy* and *subsampling ratio* to the system performance.
- Showcase the proposed multi-step MDP to model the agent trajectories. The reason why we believe in the prosperity of this approach, is that the model shall produce a series of state-action pairs altogether with the respective rewards. Thus, at inference time we will be able to pick out the series of actions that give the highest reward. In practice, this might be preferred than statically apply the policy to all patches at once.

In conclusion, this work has been fulfilled with a view to limit the outbreak of fires in land areas with the involvement of satellite surveillance. We studied the problem from various aspects and proposed valid points for future investigation, that hold the potential to face the initial real-world problem with greater efficiency and timely action.

Bibliography

[1] John T Abatzoglou and A Park Williams. Impact of anthropogenic climate change on wildfire across western us forests. *Proceedings of the National Academy of Sciences*, 113(42):11770–11775, 2016.

[2] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[3] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. arxiv 2012. *arXiv preprint arXiv:1206.5538*, 2012.

[5] David MJS Bowman, Jennifer K Balch, Paulo Artaxo, William J Bond, Jean M Carlson, Mark A Cochrane, Carla M D'Antonio, Ruth S DeFries, John C Doyle, Sandy P Harrison, et al. Fire in the earth system. *science*, 324(5926):481–484, 2009.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[7] Kangjoon Cho, Yonghyun Kim, and Yongil Kim. Disaggregation of landsat-8 thermal data using guided swir imagery on the scene of a wildfire. *Remote Sensing*, 10(1):105, 2018.

[8] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, 29, 2016.

[9] Gabriel Henrique de Almeida Pereira, Andre Minoro Fusioka, Bogdan Tomoyuki Nassu, and Rodrigo Minetto. Active fire detection in landsat-8 imagery: A large-scale dataset and a deep-learning study. *ISPRS Journal of Photogrammetry and Remote Sensing*, 178:171–186, 2021.

[10] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[11] Leonardo N Ferreira, Didier A Vega-Oliveros, Liang Zhao, Manoel F Cardoso, and Elbert EN Macau. Global fire season severity analysis and forecasting. *Computers & Geosciences*, 134:104339, 2020.

[12] Mingfei Gao, Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Dynamic zoom-in network for fast object detection in large images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6926–6935, 2018.

[13] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[14] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[17] Zhonghua Hong, Zhizhou Tang, Haiyan Pan, Yuewei Zhang, Zhongsheng Zheng, Ruyan Zhou, Zhenling Ma, Yun Zhang, Yanling Han, Jing Wang, et al. Active fire detection using a novel convolutional neural network based on himawari-8 satellite images. *Frontiers in Environmental Science*, 10:794028, 2022.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[19] Sanket Kamthe and Marc Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In *International conference on artificial intelligence and statistics*, pages 1701–1710. PMLR, 2018.

[20] Soushi Kato, Hiroki Miyamoto, Stefania Amici, Atsushi Oda, Hiroyuki Matsushita, and Ryosuke Nakamura. Automated classification of heat sources detected using swir remote sensing. *International Journal of Applied Earth Observation and Geoinformation*, 103:102491, 2021.

[21] Alexander Koltunov, Susan L Ustin, and Elaine M Prins. On timeliness and accuracy of wildfire detection by the goes wf-abba algorithm over california during the 2006 fire season. *Remote sensing of environment*, 127:194–209, 2012.

[22] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

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

[24] Sanath Sathyachandran Kumar and David P Roy. Global operational land imager landsat-8 reflectance-based active fire detection algorithm. *International Journal of Digital Earth*, 11(2):154–178, 2018.

[25] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[26] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[27] Pu Li and Wangda Zhao. Image fire detection algorithms based on convolutional neural networks. *Case Studies in Thermal Engineering*, 19:100625, 2020.

[28] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[30] D Matthew Zeiler and Fergus Rob. Visualizing and understanding convolutional neural networks. ECCV, 2014.

[31] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479(480):104, 1969.

[32] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[34] 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.

[35] Sam W Murphy, Carlos Roberto de Souza Filho, Rob Wright, Giovanni Sabatino, and Rosa Correa Pabon. Hotmap: Global hot target detection at moderate spatial resolution. *Remote Sensing of Environment*, 177:78–88, 2016.

[36] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[37] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[38] Aleksis Pirinen and Cristian Sminchisescu. Deep reinforcement learning of region proposal networks for object detection. In *proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6945–6954, 2018.

[39] Adria Recasens, Petr Kellnhofer, Simon Stent, Wojciech Matusik, and Antonio Torralba. Learning to zoom: a saliency-based sampling layer for neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 51–66, 2018.

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[41] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7008–7024, 2017.

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[43] Frank Rosenblatt et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, volume 55. Spartan books Washington, DC, 1962.

[44] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[45] Wilfrid Schroeder, Patricia Oliva, Louis Giglio, and Ivan A Csiszar. The new viirs 375 m active fire detection data product: Algorithm description and initial assessment. *Remote Sensing of Environment*, 143:85–96, 2014.

[46] Wilfrid Schroeder, Patricia Oliva, Louis Giglio, Brad Quayle, Eckehard Lorenz, and Fabiano Morelli. Active fire detection using landsat-8/oli data. *Remote sensing of environment*, 185:210–220, 2016.

[47] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[49] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[50] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

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

[52] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[53] Burak Uzkent and Stefano Ermon. Learning when and where to zoom with deep reinforcement learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12345–12354, 2020.

[54] K Vani et al. Deep learning based forest fire classification and detection in satellite images. In *2019 11th International Conference on Advanced Computing (ICoAC)*, pages 61–65. IEEE, 2019.

[55] 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.

[56] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for

image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2017.

[57] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[58] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[59] Anthony L Westerling, Hugo G Hidalgo, Daniel R Cayan, and Thomas W Swetnam. Warming and earlier spring increase western us forest wildfire activity. *science*, 313(5789):940–943, 2006.

[60] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[61] Andreas Zell. *Simulation neuronaler netze*, volume 1. Addison-Wesley Bonn, 1994.