Sequential Analysis of High-Resolution, Aerial Video with Deep Reinforcement Learning

Maximilian Ulmer

Master's thesis

Sequential Analysis of High-Resolution, Aerial Video with Deep Reinforcement Learning

Maximilian Ulmer

20. December 2018

Institute for Data Processing
Technische Universität München

Maximilian Ulmer. *Sequential Analysis of High-Resolution, Aerial Video with Deep Reinforcement Learning.* Master's thesis, Technische Universität München, Munich, Germany, 2019.

Supervised by Prof. Dr.-Ing. K. Diepold and Supervisor; submitted on 20. December 2018 to the Department of Electrical Engineering and Information Technology of the Technische Universität München.

Institute for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de`.

I confirm that this master's thesis is my own work and
I have documented all sources and material used.

21.12.2018 _____

Maximilian Ulmer

Acknowledgments

First and foremost, I would like to thank Prof. Diepold for his genuine interest in my ideas, honest feedback, guidance and constant engagement in this thesis. Moreover, I would like to thank the Institute for Data Processing and Technische Universität München for providing the framework that made this thesis possible.

At last, I need to express my profound gratitude to my parents, Martin and Elke, my siblings, Lisa and Felix, and my girlfriend, Maria, for their relentless support and unfailing encouragement during my years of study.

Abstract

Recent technological advancements have made Unmanned Aerial Vehicles — commonly knows as "drones" — an indispensable and powerful tool for numerous urgent use-cases, such as disaster relief and environmental monitoring. Contrary to popular belief, piloted drones require a high degree of human involvement, which is a major source of limitations, inefficiencies and error. To overcome these obstacles, researchers propose the notion of "autonomous" drones which can operate without human intervention or guidance. Autonomous flight entails delicate synergy of a plethora of different software and hardware technologies to operate while ensuring safety for the vessel and its environment. Its most powerful sensor, which yields the most intricate data, is the camera. Computer vision, the technology poised to translate high-level semantics contained in visual data to machine code, has recently been revolutionized by a flurry of deep learning based methods. The problem of applying novel computer vision techniques to aerial data is twofold: First, aerial imagery is inherently high-resolution with a wide aspect ratio; Second, a drone is an exceedingly resource-constrained platform. Modern object detection algorithms are designed for low-resolution image data and require substantial computational capabilities. Hence, with today's methods real-time detection in high-resolution aerial video is beyond the bounds of possibility. To move towards autonomous drones we propose a novel approach to aerial video object detection. Inspired by the mammalian visual system, we propose to sequentially analyze video in a coarse-to-fine topology. Our approach consists of three stages. A deep reinforcement learning agent, called ScopeNet, observes a downsampled representation of the camera sensor and predicts which subregion of the current frame is rich in new information. If no new information is contained in the observation the agent can decide to skip the frame entirely. Thus, we only sparsely sample so called key-frames. If the agent predicts an information gain, a object detector is applied to the subregion to extract knowledge. Then, the knowledge is sent to the tracker which exploits temporal dependencies and propagates detected object to adjacent frames. In contrast to existing approaches, we learn the key-frame scheduling policy directly from raw data. The fundamental goal of this approach is to empower state-of-the-art object detectors by zooming-in on subregions and reducing computations with sparse key-frame sampling and visual tracking. Our experiments show, that the pipeline developed within the scope of this thesis, enhances the accuracy of the object detector substantially. ScopeNet increases the accuracy of object detectors by an average of 237% while only processing $\sim 20\%$ of the frames.

The code of ScopeNet will be available at `https://github.com/mwulmer`.

Contents

1 Introduction

The market of unmanned aerial vehicles (UAV) has seen significant growth in the last decades. Whilst in the past they were primarily developed by the military industry, the technology has recently been adopted by many companies, start-ups, scientists and hobbyists. Piloted UAVs — popularly known as "drones" — are already indispensable for a variety of critical use-cases, such as environmental monitoring [1, 2] and disaster support [3]. However, it is apparent that a major source of error and inefficiency are human operators [4]. This led to the vision of drones that can operate in a human-shared space and fulfill tasks without any intervention or guidance. Such robots are referred to as "autonomous" UAVs. First experiments have already shown promising results in isolated environments, namely indoors or on trails [5, 6].

Nonetheless, there is still a plethora of unsolved issues. The problem space of autonomous drones can roughly be divide into four categories: Perception, Cognition, Platform and Security. **Perception** covers all problems related to the immediate interface between the UAV and its environment, such as sensor technology, computer vision and signal processing. **Cognition** is the central piece of the autonomous system. It receives data from other modules and determines the next best action to fulfill the task. Problems in this class include decision engine, localization, path-planning and navigation. **The Platform** is the basis for hardware components, integration of hardware/software and interfaces between modules. Additional research topics of the category are locomotion and control theory. **Security** can be broken down into intrinsic security, considerations concerning the surroundings of the drone and data security. Drones are an exceedingly sensitive topic in terms of public perception. Manifesting a save relation between the drone and its environment is pivotal for the progress of the technology.

Henceforth, we impose two requirements on the system. First, the system can only be considered autonomous if all processing is done on-board. Secondly, to evade potentially harmful situations, it is imperative that the drone can react in real-time. Thus, all the critical data processing has to execute in real-time as well.

This thesis is concerned with problems regarding the perception of UAVs. More specifically, the visual system.

1.1 Problem Statement

The fundamental purpose of the visual system of any robot is to extract relevant information from the camera sensor. In case of a UAV, the goal is to analyze and understand the

high-level semantics of aerial video data. Thus, a solution has to consider both domains, aerial imagery and video.

Existing state-of-the-art deep convolutional object detectors have shown that they can operate in resource constrained environments [7, 8, 9]. However, video content entails many intrinsic properties which are not addressed by naively applying an object detector to every frame. First, effects like motion blur, defocus and occlusion are deteriorating computed features and detection accuracy [10]. Second, video contains a very high degree of temporal redundancy, because image content varies slowly over video frames, especially high level semantics [11, 12]. This property can be exploited to reduce computational cost [13]. Moreover, existing approaches rarely benchmark in embedded environments. Even the best embedded systems operate at a fraction of the performance of modern ultra-high-end Graphical Processing Units (GPU), which are commonly used in studies.

Aerial imagery is a very challenging medium for computer vision algorithms. Most recent competitive neural network based detectors and classifiers are built for low-resolution [14, 15, 16, 17, 8, 18, 19]. The primary reason is that they are engineered to perform best on the most popular computer vision datasets, such as ImageNet (∼500x400) [20], MS COCO (∼600x400) [21] and PASCAL VOC (∼500x400) [22]. However, aerial imagery is usually captured in high-resolution with a wide aspect ration and comprises extreme scale variance and unusual angles. Scaling the networks up, to be applicable to high-resolution data, is unfeasible due to the exponential growth of computational cost. Likewise, even the best current GPUs do not have enough memory to effectively train such large networks. The opposite approach, downsampling the data, does not retain enough information which results in very poor performance. A qualitative experiment to validate that claim can be seen in Figure 1.1.

In this thesis, we propose a novel approach to merge resource-constraint video object detection and aerial imagery for autonomous UAVs. A lightweight neural agent, called ScopeNet, learns the best policy from raw video data to sequentially analyze high-resolution aerial data, in resemblance to the human visual system. This agent is then placed in a pipeline with a state-of-the-art object detector and tracker to exploit inter-frame temporal dependencies. A key advantage of this approach is that it can leverage the constant improvements made by the object detection and tracking communities. Thus, it is possible to separate the behavior of the agent from the performance of the object detector and tracker, which allows us to keep up with recent developments in the field.

1.2 Contribution

This thesis contributes to the field of computer vision. More specifically, it applies recent techniques from deep reinforcement learning to it. The objective of the thesis is to test the hypothesis that

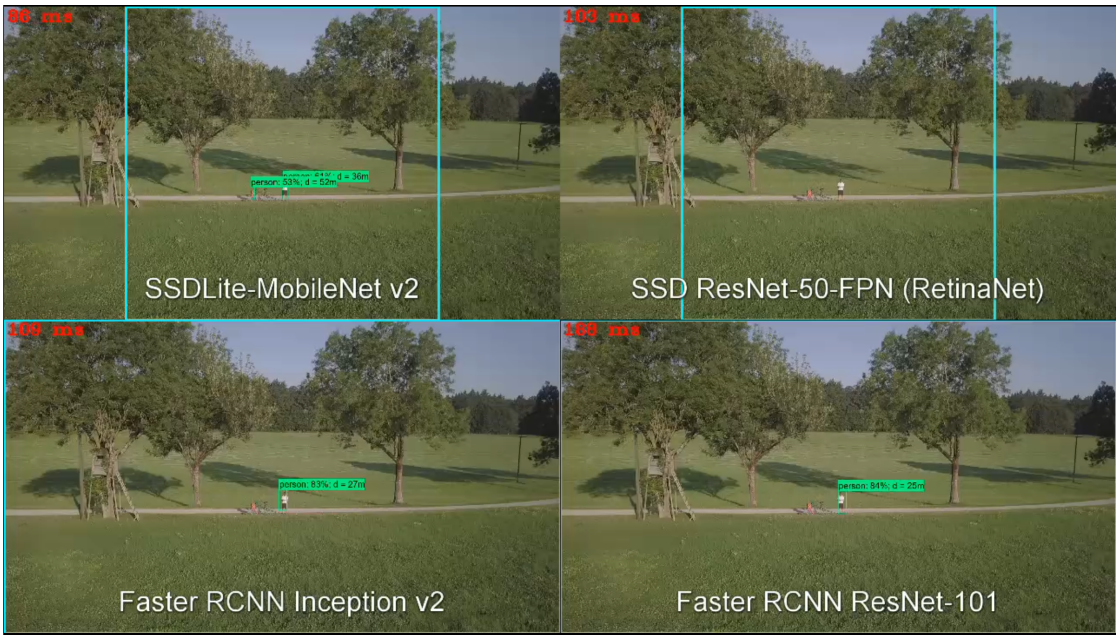


Figure 1.1: The video compares four state-of-the-art object detectors with different backbone networks for long distance object detection. The experiment consists of a drone which steadily increases the distance to the subject, to get an estimation of the performance of different detectors at varying view distance. In the top left corner of each video is the computation time for each single frame and the teal rectangle represents the receptive field. Above the detection boxes is an approximation of the distance between the drone and the person, calculated with the triangle similarity. More on the architecture of the four detectors follows in Chapter 3. The full video is available here: `https://www.youtube.com/watch?v=LXqMVgUPeEw`

- a reinforcement learning agent can learn a policy to (a) sequentially analyze high-resolution aerial imagery and (b) be aware of computational cost, and
- the agent, embedded in a framework, outperforms the baseline algorithms in terms of speed, computations and accuracy.

In the scope of this thesis it is not possible to provide formal proof of the correctness or falsehood of the hypothesis. However, it hopefully provides strong experimental evidence for or against its validity.

The contributions of this thesis are as follows:

- ScopeNet: A novel approach to sequentially analyze high-resolution aerial video data in real-time using policy gradient deep reinforcement learning (RL);
- A neural agent with a Temporal Difference (TD) policy gradient model, a critic which evaluates the policy and a Long Short-Term Memory [23] recurrent neural

network to learn temporal dependencies, which is trained using multi-task loss function to improve convergence of the convolutional layers;

- Detailed training progress of three different models to derive best practices for future research;
- An easily extendable environment for the agent to practice in, on the basis of a challenging real-world UAV dataset [24];
- Integration of the agent into an easily adaptable computer vision pipeline, consisting of the agent, a detector and a tracker;
- Experimental validation of the architecture using the validation set of the training data.

At its core, the proposition is as follows. A UAV with a high-resolution camera sensor captures a stream of images. The task is to understand the semantics of the data. Depending on the mission, this can imply very different things. In a similar fashion to human perception of large images, the RL agent first has a coarse look at the entire frame. In terms of signal processing that means we downsample the image to a smaller size and extract deep convolutional features. Then, the agent picks an action depending on the contents of the coarse image. For instance, if there is information which is not understood yet he can draw his attention to that region of the image. In contrast, if all the information in the image is already analyzed, he can choose to save computational resources and wait for the next observation from the camera sensor. Figure 1.2 shows the different stages of the process.

This work builds on a few key concepts which will be covered throughout the thesis. First, applying an object detector to a cropped subregion of a large image with small objects can increase the accuracy of said detector over applying the detector to the entire image. Second, computing detections in *key-frames* of the video and propagating them to non-key-frames can drastically increase the processing speed. Additionally, *key-frames* feature propagation is more resilient to deteriorating video effects. Finally, choosing which frame is a *key-frame* and which subregion to process can be solved with a deep reinforcement learning agent which is trained on raw data.

The proposed approach unifies object detection, tracking and sequential analysis in one framework. After the agent is trained successfully, it is embedded into the framework. We show, that the developed pipeline empowers state-of-the-art object detectors and increases the accuracy on average by 237% on the validation set. Moreover, we apply the computationally expensive detector on only $\sim$ 20% of the frames. In the case of single object tracking, our pipeline is twice as fast as the next fastest implementation.

Before introducing the building blocks, Chapter 2 offers background information on deep learning and autonomous UAVs. Additionally, it details related work to the different domains as well as primary influences of this thesis. Chapter 3 and 4 introduce methods

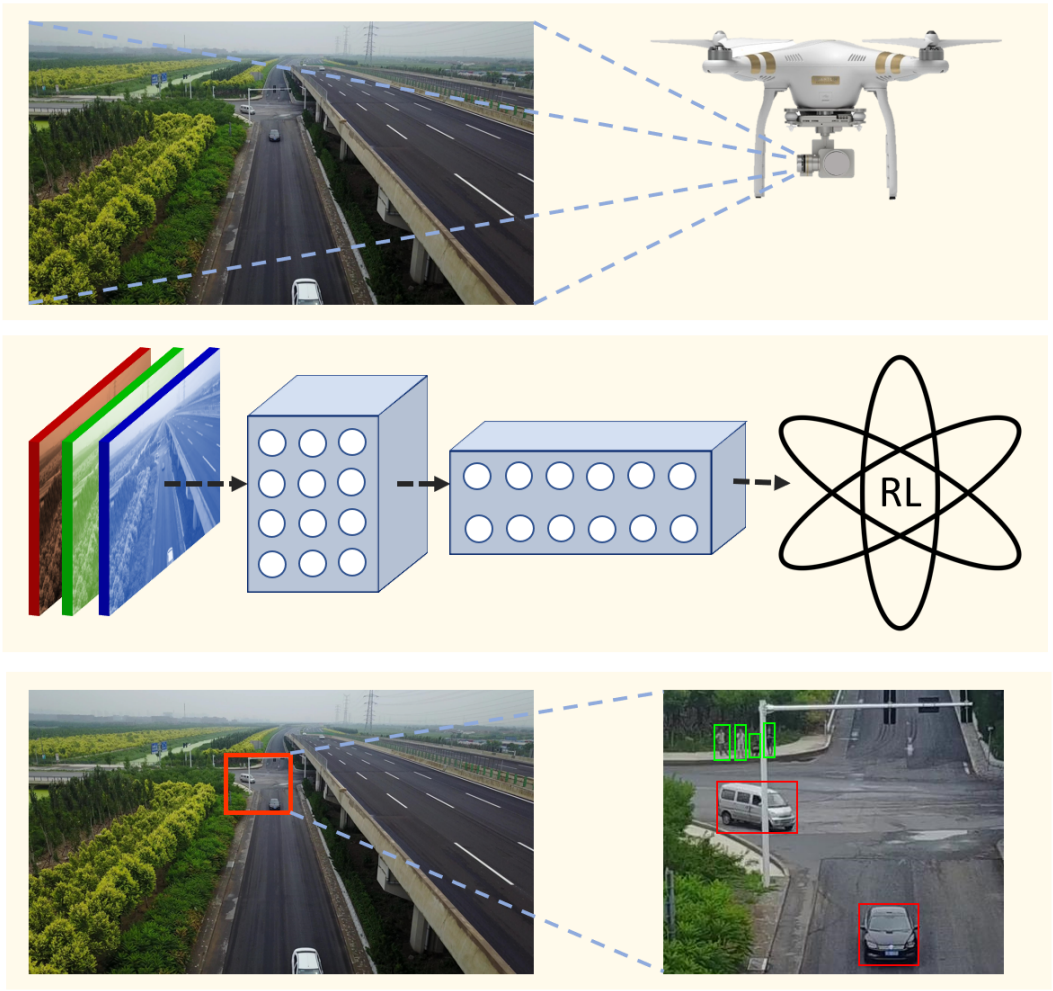


Figure 1.2: The proposition of this thesis can be divided into three stages. First, the drone captures images at high resolution (HD, FHD, QHD). Then, the agent analyzes a coarse representation of the frame to decide on the next, most rewarding action. At last, the visual system performs the action and zooms-in on a region of the high-resolution image.

which are relevant to learn the policy from raw data. At first, an overview of the entire pipeline will be introduced in Section 3.1. From there on, the next sections outline the theory of each building block until Chapter 4 explores the central piece of the thesis.

After the architecture is introduced, Chapter 5 covers details of the implementation. First, Section 5.2 explores the UAV dataset. Consequently, the dataset is embedded into the environment in which the agent learns. Then, the training setup of ScopeNet is described. In Chapter 6, finally, the hypothesis will be examined. At last, Chapter 7

will discuss the results of the study thoroughly, before Chapter 8 concludes this thesis and outlines future research directions.

2 Background

The goal of this chapter is to build the foundation for the rest of the thesis. First, some background information about autonomous Unmanned Aerial Vehicles (UAV) is on order. After a brief history and definition, the section will present a few important use-cases in which autonomous drones can have an important positive impact. Then, section 2.2 will focus on computer vision which is the main topic of this work. Finally, the last section of Chapter 2 lays out related work to video analysis in high-resolution and major influences to this thesis.

2.1 Autonomous Unmanned Aerial Vehicles

The very first UAV was developed by the British Royal Navy in 1933 for gunnery practice and since then, the technology has been primarily driven by the military industry [25]. However, lately market shares shift towards civil and commercial developers. Since 2005, the production of non-military drones has increased fivefold and market research projects the total, global UAV revenue to grow from $6.8 million in 2016 to $36,9 million by 2022 [26].

Contrary to popular belief UAVs require a high degree of human involvement. In fact, behind most UAV operations an entire team is stationed with jobs ranging from remote control, camera management, maintenance, sensor operations to spatial disorientation [4].

Drones are a powerful tool for near-surface operations. In contrast to satellites or manned aircraft, they are quickly deployed, relatively cheap, have high spatio-temporal resolution, and are not bound to atmospheric factors [1]. There are several different variations of UAVs, most notably fixed-wing and rotary-wing. Fixed-wing drones excel in long distance operations but have limited maneuverability. Rotary-wing UAVs has short reach but a high level of agility and maneuverability [26]. They both complement each other and cover a wide variety of use-cases.

Developing UAVs to act autonomously in challenging environments has been a proposition of researchers and practitioners for years. Especially with the current exposure of self-driving cars, the prospect of "self-flying" drones appears to be more in reach than ever before. Companies like DJI, in cooperation with NVIDIA, are already paving the way by building drones which can be equipped with powerful processing units like the NVIDIA Jetson TX Series [27]. These units are designed with a similar architecture as

modern Graphics Processing Units (GPU) to process large amounts of data in parallel and tackle high dimensional problems.

2.1.1 Use-Cases

Before we delve into different high-level building blocks of an autonomous drone, some use-cases for autonomous drones are in order. From a sheer hardware perspective, human operated drones are a fully functional product. They are an integral part of operations such as remote sensing [28], environmental monitoring [2, 29], disaster management [3], the multimedia industry and many more. Although there are some limitations of UAV applications induced by technology most are attributed to human factors [4]. If a drone is human operated, the video data of the camera sensor has to be transmitted to the pilot, this creates latency which limits the drone in its capabilities. Additionally, humans are generally error-prone and subject to fatigue, command-control inefficiency and loss of situational awareness. On-board information processing and decision making has the potential to solve many of those problems. It follows a short list which highlights the potential impact that autonomous UAVs could have on some use-cases:

Environmental Monitoring is an integral part of our understanding of nature, climate change and its impact on the ecosystem. In the past, satellites and and manned aircraft were used to collect data. Both technologies have severe limitations and challenges, such as low spatial resolution, low temporal frequency and high cost [1]. Thus, researchers recently started using drones to monitor nature. The process, albeit more efficient than satellites and regular aircraft, still takes a howling amount of time and effort. To monitor a region of interest, the operator has to fly a certain pattern over an area and analyze the pile of data manually. An autonomous system can drastically increase the productivity of the process. It can fly for an extended period of time, filter the area for cues and preprocess data before an expert evaluates the results. In total, the process would be more efficient, allow researchers to cover wider areas and react faster.

Humanitarian Crisis are happening at a shocking frequency. Natural disasters like hurricanes, tsunamis or earthquakes, and their aftermath, threaten thousands of human lives each year. Such disaster often rupture the landscape, making it difficult for aid to reach the affected zones. In such situations, the life of victims often depends on the speed of reinforcement. Drones can work beyond human capabilities, without putting more people in harm's way [30] and autonomy would help supply more victims faster. Additionally, they can be used to rebuild wireless communication infrastructure [31] and search for missing or buried people.

Public Safety and law enforcement is important for policy makers, albeit a controversial topic. A key element to it is observation and surveillance. Getting a hawks-eye view of a crowd to evaluate the situation, searching for people in a burning house, chasing suspects or gathering information. There is a plethora of use-cases that autonomous drones can be beneficial to ensure public safety.

Logistics, retail and delivery is an enormous market with millions of products being delivered every single day. Delivering thousands of items per city by car strains the environment, increases traffic and worsens air conditions. Amazon is already actively developing autonomous drones for retail deliveries[1]. However, when they first announced their project the public reception was mixed and in the end the customers will decide whether or not the technology will prevail.

2.1.2 Platform

The platform represents the entire stack of software and hardware that grants the robot the ability to complete its task autonomously. It is paramount to ensure safety for itself and the environment, which is often shared with people. The process of building such a platform involves a wide variety of disciplines [32]. To give the robot the ability to perceive its environment, the engineer has to have knowledge in sensor technology, computer vision and signal processing. In order to solve problems concerning its locomotion, knowledge in control theory and kinematics is needed. To plan paths, navigate and localize itself in the environment, the engineer requires knowledge of probability theory, information theory and mapping. Finally, the implementation of the algorithms and the integration of hardware, require proficiency in software and hardware development. Figure 2.1 merges the different parts into one model.

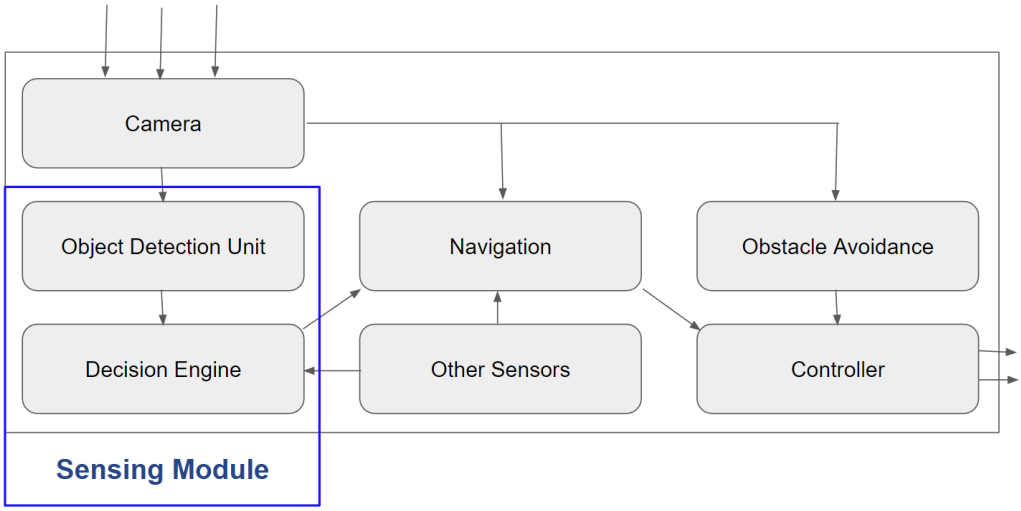


Figure 2.1: An exemplary realization of the platform. The camera sensor takes a central role in this configuration because it provides data for three integral modules. The sensing module is the only task specific unit in the model.

[1] https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011

The central idea of the this architecture is to be as flexible as possible. The sensing module consists of a task-specific object detection unit, which extracts the high-level semantics from the video stream, and a task-specific decision engine. Hence, the Sensing Module can be easily adapted to different tasks in a plug-and-play fashion and the rest of the system is does not need to change. The camera, as the most powerful and versatile sensor, takes a central role. Its information is required for navigation, decision making and obstacle avoidance. The latter has to run in a separate thread, such that the safety of the vessel can be guaranteed at all times. Once the vision unit has analyzed the situation captured by the camera sensor, the decision engine decides what action to take next. In order to complete the objective, it takes into account the analysis of the vision unit, other sensors and the decision rules. The action is then formulated as a navigation command and passed to the navigation unit. From there on, the navigation unit communicates with the controller of the UAV to reach the goal.

It is apparent that the vision unit plays a pivotal role for autonomous drones. Computer vision is one of the computationally most taxing tasks of the drone and many parts rely on the information it provides. The next section will introduce the area and provide some historical background.

2.2 Computer vision

> *I propose to consider the question, 'Can machines think?'*
> - Alan Turing, *Computing Machinery and Intelligence*

This opening statement of Alan Turing's 1950 paper [33] marks the inception of the evolution of artificial intelligence (AI) as we know it. Turing himself estimated that by the year 2000 humanity is able to build intelligent machines which pass his famous "Turing Test". The principle of the test is to evaluate whether a machine exhibits intelligent behaviour, indistinguishable from a human. Turing's estimation, however, turned out to be wrong.

Since then, AI has grown with staggering speed to a vast sphere of countless research topics and practical applications [34], exercised by scientists and the commercial industry alike. As a matter of fact, AI has already conquered most industries and many parts of our everyday life. Intelligent systems are built to power web search and social media, automate labor, diagnose medical patients, analyze language and assist law enforcement.

In the early days of AI research an interesting insight, a duality, became apparent. Formal and abstract problems, which are among the intellectually most difficult for humans, are often trivial for machines. Simultaneously, tasks that are intuitive for us, are among the most challenging for machines. Just very recently, an old technology empowered by faster processors and greater databases reappeared in the AI universe and allowed us to tackle some of these more intuitive problems for the first time. The approach is called **deep learning** [35].

Conventional approaches build on hand-crafted features and decision rules, combined with shallow architectures. In contrast, deep learning allows the machine to learn from experience. Experiences are gathered through data which is fed to the algorithm. It then automatically learns a representation of the data that is needed to solve the task. By learning representations from raw data, deep learning avoids error-prone human operators which is especially beneficial for high-dimensional problems. Deep learning architectures are built using a hierarchy of multiple representation layers which helps the algorithm to build complex concepts from simpler ones. Due to its very structure of multiple, deep layers it is called deep learning.

Computer vision is precisely such an intuitive problem. For humans, vision is essential. We use our vision every day, it allows us to interact, live, work and survive. What our visual cortex and brain is especially good at, is understanding high-level semantics of visual information. The information we perceive with our eyes is inherently noisy. At any given time, of all the information that hits the retina of our eyes, only a very small percentage is needed to understand what we see. Most of the contribution to our visual understanding is prior knowledge. This means, that there is more to understanding the semantics of a scene than just processing the visual data. *Smeulders et al.* [36] defined this phenomenon as the *Semantic Gap* and it represents one of the major challenges in computer vision.

Before Deep Convolutional Neural Networks (DCNN) started dominating the computer vision landscape, tasks like image classification, object detection, segmentation and motion estimation [37] used simple features, such as templates [38] and key-points [39]. Then, the first statistical approaches revolutionized the field [40, 41]. These Machine Learning (ML) algorithms used shallow regressors and classifiers on hand-crafted features, such as different histograms [41, 42, 43], Haar Wavelets [44] and covariance descriptors [45].

As briefly said before, neural networks have been around since the 1940s [46] and were originally developed to imitate the structure of the human brain, in order to solve learning problems. However, it was not before 1986 that *Hinton et al.* [47] developed an effective method to train such structures, namely the back-propagation algorithm. Hinton's research kicked off the triumph of neural networks in many ML disciplines, such as dimensionality reduction [48] and speech recognition [49]. One class of neural networks are Convolutional Neural Networks (CNN). They are specialized to process data that has a grid-like topology [34]. Although, they were first developed by *LeCun et al.* [50] in 1989 for a digit recognition task, it took until 2012 before the technology was widely adopted for computer vision. In 2012 *Krizhevsky et al.* [14] won the ImageNet classification competition [20] with their seminal work: AlexNet. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is widely considered the most prestigious classification competition which consists of millions of images containing 1000 classes. From there on, other computer vision disciplines like localization, segmentation and object detection started adopting CNNs. The latter will be discussed thoroughly in the next chapter.

2.3 Related Work

The ultimate goal of the thesis is to unite two challenging fields in one solution: video object detection and object detection in large images.

Video Object Detection. Recently, there has been rising interest in video object detection. All recent solutions use a method to exploit temporal redundancy and limit video intrinsic properties. At its core, they can be divided into feature-level methods and box-level methods [10].

Feature-level methods have gained a lot of traction lately. A feature-level implementation [13, 51] won the video division of the ILSCRC 2017 [2]. Their solution is based on the concept of a deep neural network that predicts optical flow [52]. Similar to the approach of this thesis, they only compute convolutional features on sparse key-frames and then propagate these feature maps to adjacent frames with an iteration of FlowNet [52]. To diminish deteriorating video effects on certain frames, they additionally aggregate features from nearby frames. For their work, they used a fixed key-frame scheduling which means they sampled key frames simply at a fixed interval. Later, *Zhu et al.* [10] introduced a more versatile implementation to key frame scheduling which is temporally adaptive based on a feature consistency indicator. Feature-level methods are a very elegant solution to video object detection. However, currently it suffers from inaccurate optical flow which will become less of a problem in the future. Although they are developed for video content, flow predictors are computational expensive and currently barely run in real-time on ultra-high-end GPUs. This dismisses them for our use-case.

Box-level methods try to improve detection accuracy along the temporal domain by tracking boxes. Tubelets - which are essentially a sequence of boxes - based methods [53] use a special proposal network, bounding box tracking and a recurrent neural network to incorporate temporal information. T-CNN [54] is a heaviliy engineered video object detection solution, which won the ILSCRC 2015 competition. However, it requires feature computation per frame and is thus unfeasible for the embedded environment. Detect & Track from *Feichtenhofer et al.* [55] incorporates detecting and tracking into one network using multi-task learning. Their method links frame detection to tracklets and uses correlation feature which aid the network tracking. Seq-NMS of *Han et al.* [56] proposes post processing high-confidence sequences of bounding boxes which are rescored to their average confidence and suppresses boxes close to that sequence. Finally, *Chen et al.* use a Scale-Time Lattice in their work [57] which achieves a competitive 79% mean average precision (mAP) on the ImageNet VID dataset at 62 fps on a Titan X. As this thesis does, their approach uses a coarse-to-fine methodology. They use an adaptive scheduling to sample sparse key-frames and an acyclic graph which refines detection results.

The proposal of this thesis uses a box-level method with a simple approach to bounding box propagation: sparse-adaptive key-frame scheduling with interpolation tracking. *Chen*

[2] `http://image-net.org/challenges/LSVRC/2017/results`

et al. showed in [57] that such an architecture can run at a respectable speed and accuracy trade-off with one absolute key-advantage. Because all trackers and detectors basically do the same on an abstract level, they are interchangeable and thus, we can keep up with industry developments. Most approaches use some kind of key-frame scheduling, either constant or adaptive. They calculate metrics to determine whether or not a frame is a key-frame. Our approach is solely based on learning a key-frame scheduling policy from raw data. We do not use additional hand-crafted metrics to determine the scheduling. **To the best of our knowledge, this has not been proposed before.**

Object Detection in Large Images. As noted briefly in the introduction, even the best GPUs available do not have enough memory to effectively train modern object detectors on large images. If the object class is spatially large, many current architectures can be accurate. Especially networks based on region proposals have reasonable performance, such as Faster R-CNN [17], Cascade R-CNN [58] or RetinaNet [59]. Nonetheless, when it comes to detecting smaller objects, such methods are unfeasible.

Traditional methods rely on splitting up the image and performing sequential search on the candidates. To increase the accuracy beyond an ordinary deep convolutional object detector, *Meng et al.* [60] propose an approach in which they break down the large image into smaller patches which they feed into a specialized CNN. Additionally, they use resolution pyramids to establish scale-invariance. Processing multiple images at different resolution of every subregion can be powerful when maximum accuracy is needed, however in our case run-time is critical. *Alexe et al.* [61] use a multi-scale deformable part model [62] with histograms of oriented gradients (HoG) [41] as features. They explore locations by making sequential observations that were successful in the past. However, the method requires a long time to process a single frame. *Zhang et al.* [63] build on their proposal by using a CNN to extract features and a Bayesian optimization search algorithm that sequentially proposes candidates. The work of *Lu et al.* [64] studies the benefits of super-resolution. They train a model to determine if a image should be further divided into subregions to increase accuracy. Then, they detect objects in each subregion separately and achieve performance comparable to R-CNN.

Reinforcement Learning. RL has become a popular method to learn search policies. *Google's Deepmind* seminal work on Deep Q-Networks (DQN) [65], which they developed to beat Atari games, is powering the work of *Caicedo et al.* [66] to achieve active object localization. Their model is class-specific and performs actions to sequentially deform bounding boxes according to a trained implicit policy. They designed their rewards according to the Intersection-over-Union (IoU) between the target object and the predicted box. A similar approach to object detection have *Bueno et al.* [67]. Their work entails two different networks. First, an agent that crops a certain object class from an image and second, an agent that learns to zoom on an object in an image. Hence, they divide the images into a hierarchical structure and use a deep Q-Network [65]. A significant advantage of this scenario is that they can define the task as a Markov Decision Process

(MDP) because the agent can directly observe the entire state. In contrast, the problem that this thesis characterizes can only be formulated as a Partially Observable Markov Decision Process (POMDP). Another approach that uses RL to sequentially locate objects is proposed by *Jie et al.* [68]. To incorporate global dependencies between objects, they propose an tree-structureed traversing scheme to search for objects. *Gao et al.* [69] builds on reinforcement learning to sequentially scan large images. Their iterative approach is a highly engineered model, consisting of a regression network, which calculates an accuracy gain map, a policy network and region selection.

All methods covered apply RL to image data which allows them to formulate the problem as a MDP. Most don't penalize the agent when he takes an action, thus their agent has no cost-sensitivity which is one of the main goals of this thesis. Besides the problems inherited by POMDPs, there is no reason to not apply reinforcement learning to multiple frames instead of one image at a time. Our approach is an extension of [67] to the video domain with a more powerful base model. Instead of a DQN, we use a *Synchronous Advantage Actor-Critic* which is its successor.

3 Visual Attention and Detection Framework

This chapter presents the framework to sequentially analyze aerial video data using only a downsampled observation of the camera sensor stream. The Visual Attention and Detection pipeline is composed of a discriminative correlation filter tracker, a deep convolutional object detector and ScopeNet. The chapter grants the opportunity to offer a chronological walk through computer vision. The correlation filter makes use of many traditional machine learning techniques as well as classic signal processing. The object detector is based on recent deep learning developments which are mostly well researched. Finally, ScopeNet uses technology which is largely in its infancy, especially in combination with the medium. However, before we delve into the components, section 3.1 will briefly give an overview of the functionality of the pipeline, its interfaces and the fundamental ideas it builds upon.

3.1 Concept

The field of computer vision is progressing with dazzling pace. Many of the fundamental problems like image segmentation, object tracking and object detection have their own respective communities. Similar to many other machine learning communities, they are extremely competitive and new insights emerge on a daily basis. In particular, there are numerous competitions running each year with considerable bounties and prestige attached to it. Simultaneously, gigantic private institutions, like Google, Facebook, Microsoft and Amazon, invest copious amounts of resources into research in machine learning. This state of constant development led to the decision to implement the object detector and tracker as abstract entities.

3.1.1 Building Blocks

Although there are significant internal differences between many trackers and detectors, at a higher level of abstraction they have the same functionality. This allows us to easily keep up with rapid industry developments, it simplifies the implementation and provides a rich baseline for future work.

Object Detection is a fundamental computer vision task. It infers high-level information from images. The task is to localize all instances of predefined classes and return

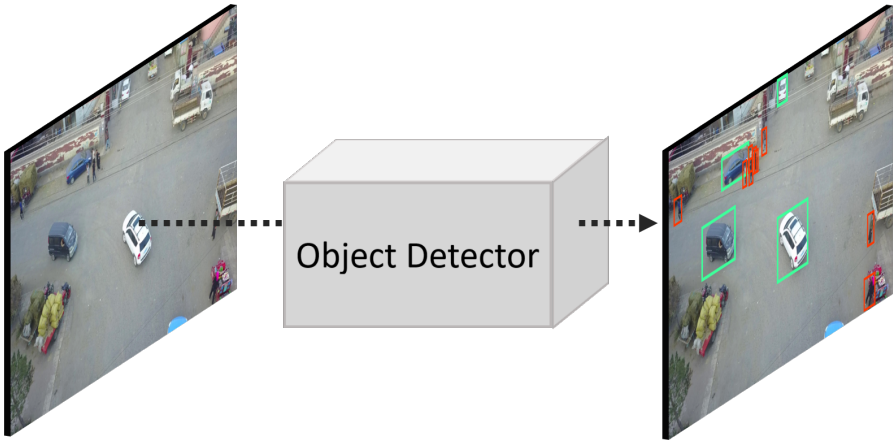


Figure 3.1: Blackbox visualization of the object detector. It takes an image as input and outputs an array containing all predicted objects of given classes.

an array containing the predicted bounding boxes, confidence and class labels. If we treat the detector itself as a blackbox, the process can be formulated as follows. Given an image $I_i \in [0, 255]^{m \times n \times 3}; m, n \in \mathbb{N}$ the detector predicts the array

$$F_i = detect(I_i) = [(c_1, p_1, x_1, y_1, w_1, h_1), ..., (c_k, p_k, x_k, y_k, w_k, h_k)]_i \tag{3.1}$$

in which $c_1, p_1, x_1, y_1, w_1, h_1$ represents the class label, probability, x-coordinate, y-coordinate, width and height of the first object and k is the number of detected objects. For the rest of the chapter we omit the detection confidence of the object detector from the detection array. In practice, we only propagate objects above a certain confidence threshold, but treat all objects alike, independent of the confidence. Figure 3.1 shows the blackbox representation. The object array F_i has been placed on the image for visual purposes.

Object Tracking can be formulated as the problem to predict the trajectory of an object in the image plane along the temporal dimension [70]. At its core, the idea is that tracking is substantially faster than detecting. Thus, it is computationally cheaper to detect objects in image I_i and propagate them to image I_{i+t} instead of detecting objects in both frames. Given an image $I_i \in [0, 255]^{m \times n \times 3}; m, n \in \mathbb{N}$, an object array F_i and the same image some interval t later $I_{i+t} \in [0, 255]^{m \times n \times 3}$ with $t \in \mathbb{N}$ the tracker predicts

$$F_{i \to i+t} = track(I_i, I_{i+t}, F_i). \tag{3.2}$$

Here, the notation $F_{i \to i+t}$ represents the array of objects propagated from image I_i to I_{i+t}. Figure 3.2 shows the relation.

ScopeNet predicts regions in the image which contain objects that have not been detected yet, using only a minimum amount of resources. It is a task that is very easy

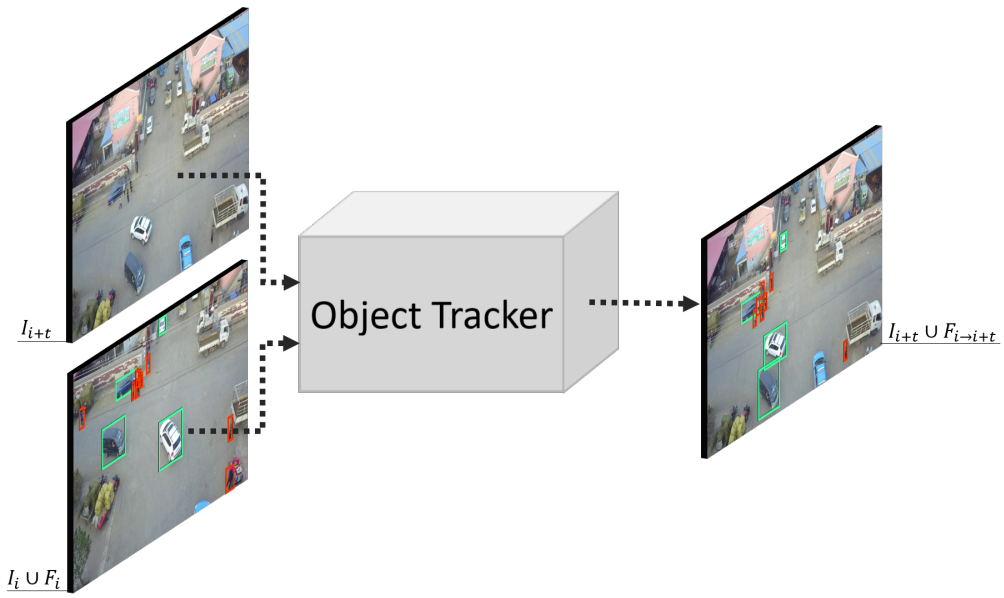


Figure 3.2: Blackbox visualization of the object tracker. As inputs it requires the original frame I_i and original detections F_i as well as a temporally shifted frame I_{i+t}. From these inputs it calculates the trajectory of the objects and returns the temporally shifted object locations $F_{i \to i+t}$

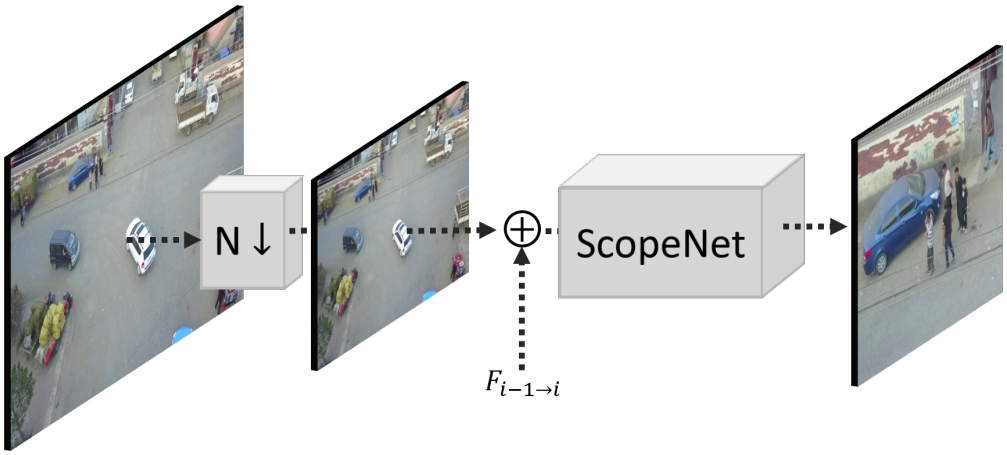


Figure 3.3: Visualization of the mode of operation of ScopeNet. A frame from the camera sensor is read and downsampled to an observation. Then the observation is concatenated with the tracking predictions for the current frame and fed into ScopeNet. ScopeNet uses this information to make a prediction where to zoom in order to increase the detection accuracy.

for humans. If you view a movie for instance, and a new actor appears on the screen, you immediately integrate him into your idea of the scene. A machine which is poised

to learn such behaviour, has to have an unconditional grasp of what was seen before and needs to combine this knowledge with new information in its receptive field. In order to reduce computational cost, ScopeNet is given a downsampled representation $I_{i,\downarrow}$ of a video frame I_i as well as information on the currently tracked objects $F_{i \to i+t}$. Based on this information, it predicts which action a_k maximizes its future reward. This behaviour can be seen in Figure 3.3.

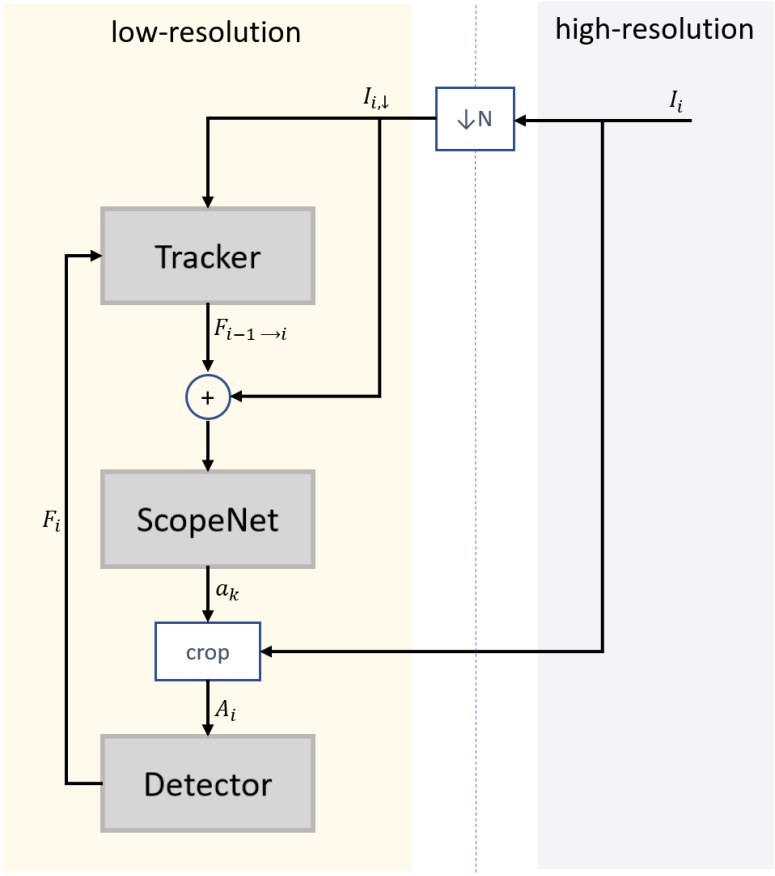


Figure 3.4: The proposition of this thesis can be depicted by this architecture which shows the information flow between the modules and their interaction.

3.1.2 The Big Picture

Finally, after introducing all building blocks, we can assemble the architecture (see Figure 3.4). Once a frame I_i is read from the camera sensor, it is downsampled and concatenated with propagated objects $F_{i-1 \to i}$ from the previous frame. On the basis of this information, ScopeNet chooses an action a_k. The actions comprises different regions in

the frame as well as the action to entirely skip object detection in the current frame. Depending on the action, a region $A_{i,k} = crop(I_i, a_k)$ is extracted from the original frame at full resolution and fed into the object detector. The object detector returns an array of calculated objects $F_i = [(x_1, y_1, w_1, h_1), ..., (x_k, y_k, w_k, h_k)]_i$ and passes it on to the tracker which stores them for the calculation of the propagation trajectory.

3.2 Visual Object Tracking

The field of Visual Object Tracking (VOT) has been in constant unrest for the last couple of years and the problem remains open and challenging. Every year, researchers from the University of Ljubljana [71] host a competition bearing the same name: the VOT Challenge. Such challenges are a great addition to the community because they compare competitive implementations and often publish their code. VOT hosts two different competitions: general tracking and real-time tracking. When CNNs started to revolutionize other fields of computer vision around 2012, it quickly became apparent that tracking will have a similar destiny. Trackers can be categorized depending on the features and visual model. In 2014, features were dominated by HoG [41], intensity, color and other handcrafted features [72]. Video models were built on correlation filter, part-based or regressions. Today, deep convolutional features dominate the field with either correlation filters or neural networks as visual models. However, in the real-time competition there are still some traditional trackers due to the high computational cost of Convolutional Neural Networks (CNN). Most notably, the Discriminative Correlation Filter Tracker with Channel and Spatial Reliability (CSRDCF) [73] is a traditional tracker which is faster than CNN bases methods with comparable accuracy.

3.2.1 Discriminative Correlation Filter

Correlation filters have been a popular signal processing tool since the 80's [74] to solve various problems in the Fourier domain [75]. In 2010 *Bolme et al.* [76] manifested that Discriminative Correlation Filter (DCF) can be a powerful approach to visual object tracking. Four years later, *Henriques et al.* [74] refined the method and it became the basis for most competitive correlation filter in the years to come.

DCF are class-agnostic visual trackers which means that they learn their class online. The objective is, given an initial frame and a bounding box, to find the same object in a frame some time later (see Figure 3.5).

The fundamental idea is to solve the problem in two steps: training and detection. Every initial frame is inherently rich in positive and negative training (see Figure 3.6). These samples are used to train a classifier. Finally, the classifier is used on subwindows of the target frame to find the object, as seen in Figure 3.7.

[1] `https://www.youtube.com/watch?v=Q9PVt-r_y_U`

(a) $t = t_0$ **(b)** $t = t_1$

Figure 3.5: At its core, the objective of visual trackers is to estimate the position of a given object at a later point in time. It requires the initial frame as well as the bounding box. Images rights belong to DJI[1]

First, lets consider the detection. A linear classifier

$$y = \mathbf{w}^T \mathbf{x} \tag{3.3}$$

with weights $\mathbf{w}$ is trained. The position of the object in the target frame is found by evaluating all subwindows $\mathbf{x}_i$, such that

$$y_i = \mathbf{w}^T \mathbf{x}_i. \tag{3.4}$$

It is obviously inefficient to evaluate every subwindow by itself. Luckily, we can use an insight from signal processing to calculate it efficiently. If we concatenate y_i to a vector $\mathbf{y}$, the expression becomes equivalent to the cross-correlation

$$\mathbf{y} = \mathbf{x} \circledast \mathbf{w}. \tag{3.5}$$

Where the $\circledast$ symbol denotes the cross-correlation. The cross-correlation is closely related to the convolution which allows us to use the Convolution Theorem [77]. The theorem states that correlation becomes element-wise multiplication in the Fourier domain. Thus,

$$\mathbf{y} = \mathbf{x} \circledast \mathbf{w} \Leftrightarrow \hat{\mathbf{y}} = \hat{\mathbf{x}}^* \odot \hat{\mathbf{w}} \tag{3.6}$$

where $\hat{}$ denotes the Discrete Fourier Transform (DFT) of the vector, $\hat{\mathbf{y}} = \mathcal{F}(\mathbf{y})$, $\odot$ is the element-wise multiplication and * the complex conjugate. Computing $\mathbf{y}$ in the Fourier domain allows us to reduce the detection computations from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^2 \log n)$.

Figure 3.6: To track the target, one extracts positive and negative training samples from the initial frame and trains a classifier.

Figure 3.7: To find the correct position of the object in the target frame, the trained classifier evaluates all subwindows x_i in the target frame.

In the past, such classifiers were usually trained by randomly sampling the area around the object [78] to gather positive and negative samples, similarly to what is shown in Figure 3.6. However, it turned out that signal processing had a very elegant solution to training as well. In signal processing $\mathbf{w}$ is called a filter and the goal is to train that filter such that the response of the filter has a high peak at the true location of the target and low values elsewhere, as illustrated in Figure 3.8.

Henriques et al. have shown that such a filter can be trained efficiently with machine learning [78, 74]. The linear classifier has the form

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{3.7}$$

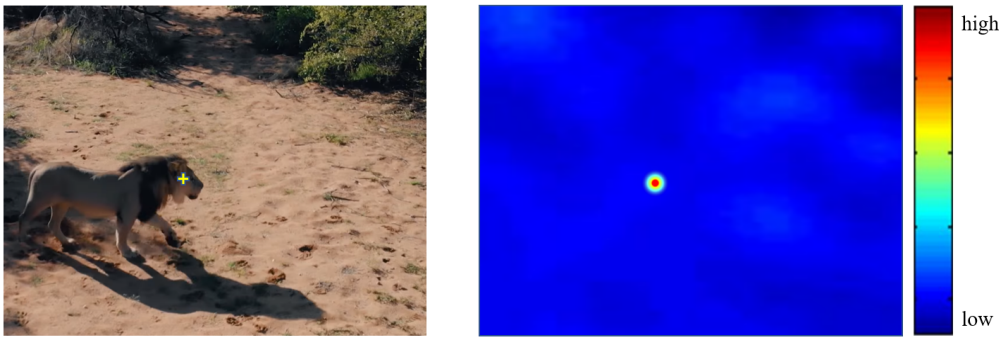


Figure 3.8: The desired response if the original image on the left is correlated with the filter $\mathbf{w}$. Red color represents high peaks and blue color low values.

where $\langle \cdot, \cdot \rangle$ is the dot product, b is the bias and $\mathbf{w}$ are the weights. Then, the goal is to minimize

$$\min_{\mathbf{w},b} \sum_i L(y_i, f(\mathbf{x}_i)) + \lambda \|\mathbf{w}\|^2. \tag{3.8}$$

$L(\cdot, \cdot)$ denotes the loss function, y_i is the ground truth training label and λ is a scalar regularization parameter. To solve this, different losses could be used, like the hinge loss. However, they decided to use the Regularized Least Squares (RLS), known as Ridge Classification, for two main reasons. First, RLS classification can achieve a similar performance as more sophisticated methods, like a Suppor Vector Machine. Second, it offers a simple closed form solution [79]. In the next equation, we use the bias trick and the mean squared error loss of RLS. That leads us to

$$\min_{\mathbf{w}} \sum_i (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|^2 \tag{3.9}$$

in which we can substitute 3.5, such that

$$\min_{\mathbf{w}} \|\mathbf{x} \circledast \mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2. \tag{3.10}$$

This equation is solved by the closed form

$$\mathbf{w} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}. \tag{3.11}$$

Here, the feature matrix X has one training sample $\mathbf{x}_i$ per row, I is the identity and $\mathbf{y}$ are concatenated training labels as earlier. Under normal circumstances, in order to compute the optimal weights a large system of linear equations has to be solved. Luckily, if we choose specific training samples $\mathbf{x}_i$, we can exploit the Fourier domain again and speed up training drastically.

Consider a patch in the original frame which represents the positive base sample, thus the object we want to track is contained in the patch. Instead of using random negative samples from the base frame, we use virtual negative samples. These virtual samples are obtained by translating the positive base sample. In mathematical terms, we apply a cyclic shift operator

$$P = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{3.12}$$

also known as the permutation matrix. P is cyclic, which means that for a vector $\mathbf{x} \in \mathbb{C}^{n \times 1}$

$$P^n \mathbf{x} = P^0 \mathbf{x} = \mathbf{x}. \tag{3.13}$$

This allows us to build a circulant dataset $X = C(\mathbf{x})$:

$$X = \begin{bmatrix} (P^0\mathbf{x})^T \\ (P^1\mathbf{x})^T \\ (P^2\mathbf{x})^T \\ \vdots \\ (P^{n-1}\mathbf{x})^T \end{bmatrix} \tag{3.14}$$

Circulant matrices have a set of important properties. For our case most importantly, they are diagonalized by the DFT. Thus,

$$X = F \operatorname{diag}(\hat{\mathbf{x}}) F^H \tag{3.15}$$

where F is the DFT matrix, $F^H = (F^*)^T$ is the Hermitian transpose. The DFT matrix is a unique, unitary matrix which can be used to calculate the DFT of any vector $\mathcal{F}(\mathbf{z}) = \sqrt{n} F \mathbf{z}$. 3.11 can be extended to the complex numbers, thus we can apply 3.15 to express

$$X^H X = F \operatorname{diag}(\hat{\mathbf{x}}^*) F^H F \operatorname{diag}(\hat{\mathbf{x}}) F^H \tag{3.16}$$

$$= F \operatorname{diag}(\hat{\mathbf{x}}^*) \operatorname{diag}(\hat{\mathbf{x}}) F^H, \tag{3.17}$$

due to F being orthogonal. Which is

$$X^H X = F \operatorname{diag}(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}) F^H. \tag{3.18}$$

Replacing $X^H X$ in 3.11 gives us:

$$\mathbf{w} = \left(F \operatorname{diag}(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}) F^H + \lambda I\right)^{-1} X^H \mathbf{y} \tag{3.19}$$

Finally, by applying some algebra, we receive:

$$\hat{\mathbf{w}} = \operatorname{diag}\left(\frac{\hat{\mathbf{x}}^*}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}\right) \cdot \hat{\mathbf{y}} = \frac{\hat{\mathbf{x}}^* \odot \hat{\mathbf{y}}}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda} \tag{3.20}$$

where the fraction denotes element-wise division. All operations in this equation are element-wise which is bound by $\mathcal{O}(n)$ and the DFT which is bound by $\mathcal{O}(n \log n)$. In Contrast, solving the minimization problem 3.11 regularly has $\mathcal{O}(n^3)$ which is bound by the matrix inversion.

However, it turns out that a regression target with hard constrains $y_i \in \{-1, 1\}$ results in a very sharp peak in the response at the target location. While this fact might be benefitial for location purposes, it is very prone to overfitting and results in generally bad performance. Once again, signal processing offers a solution. Such behaviour was studied for decades [75, 80] and it turns out that a good trade-off between localization and generalization is a gaussian regression target $\mathbf{y} = \mathbf{g}$.

So far, we only considered our base patch to have one channel with real pixel values. In reality, modern DCF trackers use more powerful features, such as HoG [41], SIFT [81], SURF [82] or Color Names (CN) [83], which can have multiple channels. Luckily, the extension of 3.9 from $\mathbf{x}$ to N_c features $\mathbf{f} = \{\mathbf{f}_d\}_{d=1:N_c}$ is straight forward. Similarly, we extend the trained filter $\mathbf{w}$ to N_c target filter $\mathbf{h} = \{\mathbf{h}_d\}_{d=1:N_c}$. Thus, the minimization problem is

$$\min_{\mathbf{h}} \left\| \sum_{d=1}^{N_c} \mathbf{f}_d \circledast \mathbf{h}_d - \mathbf{g} \right\|^2 + \lambda \|\mathbf{h}\|^2 \tag{3.21}$$

which is solved for each filter by

$$\hat{\mathbf{h}}_d = \frac{\hat{\mathbf{f}}_d^{\,*} \odot \hat{\mathbf{g}}}{\sum_{d=1}^{N_c} \hat{\mathbf{f}}_d^{\,*} \odot \hat{\mathbf{f}}_d + \lambda}. \tag{3.22}$$

As above, the fraction is an element-wise division. The solution 3.22 is used in most modern correlation filter trackers.

3.2.2 Channel and Spatial Reliability

The CSRDCF tracker uses two additional concepts: Channel Reliability Weights and Spatial Reliability Maps [73]. Their correlation filter uses 27 HoG channels, 10 CN channels and one grayscale channel, which all have drastically different scales.

Channel Reliability Weights stem from the fact that *Lukezic al.* [73] understood that in 3.22 each d-th filter channel is divided by the sum of all feature channels. The result is that the scale of each feature plays an important role for the target filter channel,

irrespective of its discriminative power. HoG features have generally lower values than CN features. This means that HoG channels might get suppressed by others. To address the issue, *Lukezic al.* propose to scale each channel independently with channel reliability weights $\tilde{w}_d$, such that

$$\min_{\mathbf{h}} \sum_{d=1}^{N_c} \|\mathbf{f}_d \circledast \mathbf{h}_d \cdot \tilde{w}_d - \mathbf{g}\|^2 + \lambda \|\mathbf{h}_d\|^2. \tag{3.23}$$

The weights are composed of two reliability measures $\tilde{w}_d = \tilde{w}_d^{(lrn)} \cdot \tilde{w}_d^{(det)}$. The value of the learning reliability weight is given by the maximum response of a learned channel filter

$$\tilde{w}_d^{(lrn)} = \max(\mathbf{f}_d \circledast \mathbf{h}_d). \tag{3.24}$$

As detection reliability $\tilde{w}_d^{(det)}$, they measure the uniqueness of each channel contribution. They use the ratio of the second and first highest peaks ρ_d of non adjacent responses, such that

$$\tilde{w}_d^{(det)} = \max\left(1 - \frac{\rho_d^{max2}}{\rho_d^{max1}}, 0.5\right). \tag{3.25}$$

Spatial Reliability Maps is a binary mask $m \in \{0, 1\}$ which segments the target object from the background. They use the map to reduce the training region from the bounding box training patch to a deformed training region which fits better to the target. The map is automatically estimated and restricts the correlation filter to the suitable parts for tracking. However, the system does not have a closed form solution anymore and has to be solved iteratively.

To sum this section up, this thesis uses the CSRDCF tracker. It is a reliable, state-of-the-art tracker which was among the top 2 trackers in the 2018 VOT challenge. It does not use CNN features and solely requires traditional machine learning and signal processing methods. The filter is able to run in real-time, works well with small objects and is adaptive to scale changes which makes it a perfect fit for the pipeline. However, there are some consideration which one has to be aware of. These will be addressed in Chapter .

3.3 Deep Convolutional Object Detectors

Object detection is one of the most important tasks in computer vision. It allows a machine to infer high-level knowledge of a scene which is a core problem in huge array of applications. In contrast to VOT, deep learning has fully transformed the field of object detection. All competitive implementations are build on the same principles. The first step is to extract convolutional feature maps of the input image from which the detector gets its discriminative power [84]. This part of the detector is called the **backbone**. The

backbone is usually very deep in layers and subsequently slow. The second step is to perform bounding box regression on the extracted features. The goal of the step is to localize the object within the image. There are two inherently different approaches to the bounding box regression. The first class of object detectors perform sparse object proposals in two stages [85, 86, 17, 87]. The second approach relies on a dense sliding window in multiple branches of the network in a single stage [88, 8, 18, 19].

This section will start off by exploring backbone architectures and how CNNs work. This knowledge is not just of great importance for object detectors, but for ScopeNet as well. Then subsection 3.3.2 details the fundamental differences between one-stage and two-stage object detectors. Next it draws a comparison to conclude which one is most suitable for this use-case. Finally, 3.3.3 introduces the object detector that is used in the pipeline.

3.3.1 Backbone

Backbone networks are CNNs. They are the part of an object detector which extracts discriminative features from the input data and play a pivotal role in the endeavor to achieve a high detection accuracy [7]. Surprisingly, almost every backbone network was developed to excel at the ImageNet classification task [20]. Some of the most prominent backbone architectures are VGG [89], Inception [15], ResNet [16] and MobileNet [90].

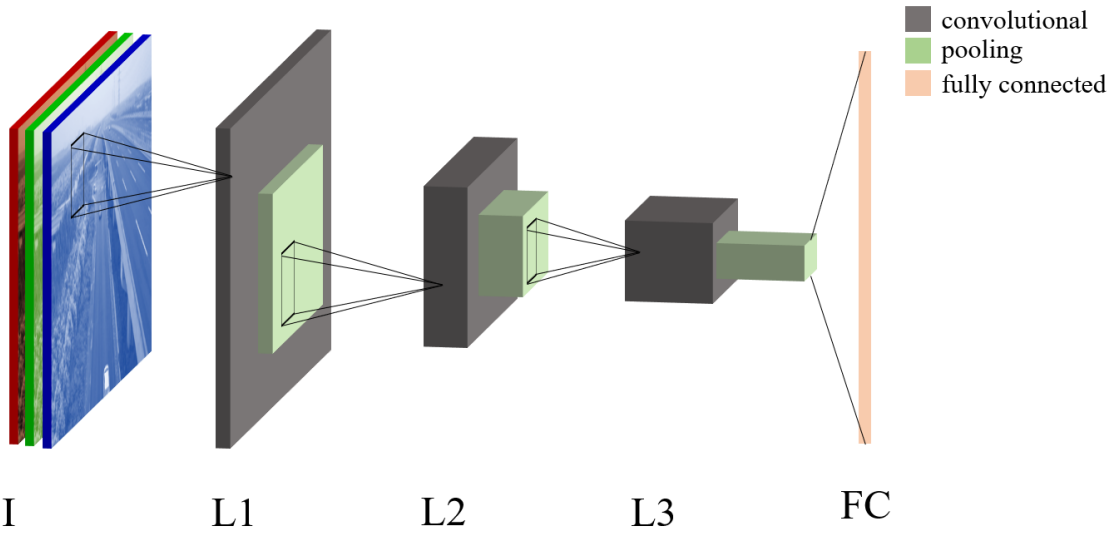


Figure 3.9: A CNN consisting of four hidden layers of which three are convolutional/pooling layers and is a fully connected layer.

Convolutional Neural Networks draw inspiration from the visual cortex of mammals [34]. *Hubel and Wiesel* showed in the 1960s that many neurons in the visual cortex

have a local receptive field [91, 92]. This entails that the neurons in the cortex are only stimulated by a subregion of the visual field. Thus, the entire field is covered by tiles which have a certain amount of overlap. Moreover, they manifested that different neuron groups activate to distinct shapes and orientations. Studies have shown that generally lower layers learn simple shapes, like edges and curves which get increasingly more complicated with the depth of the network. It is this hierarchical architecture which is able to detect complex patterns. For instance, see Figure 3.10 which shows the activations of the first convolutional layer of AlexNet, the first stand-out convolutional neural network which won the ImageNet challenge [14].

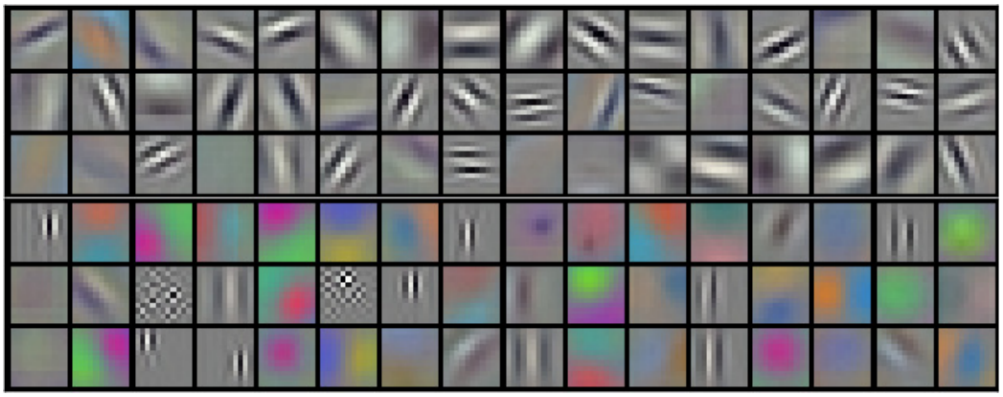


Figure 3.10: Trained weights, or filter, from the first convolutional layer of AlexNet [14]. One can see that the filters are activated on very simple patterns. The layer was trained on the 1000 class image classification task ImageNet

Now finally, after using the term Convolutional Neural Network in literally every single chapter, let us define it. A CNN is a neural network which uses convolution instead of general matrix multiplication. CNNs consist of only a couple of different building blocks: convolutional layer, pooling layer and fully-connected layer. Stacking these layers forms a CNN. Figure 3.9 shows their interaction in a unified architecture.

Convolutional Layer. The convolution of a two-dimensional image I with a two-dimensional kernel K is defined as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.26}$$

It plays a pivotal role in many fields of mathematics and engineering, as seen in Section 3.2. Intuitively, this definition flips the kernel before calculating the values of the result. However, technically most CNNs don't implement the convolution but the cross-correlation which is the convolution without flipping the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i+m, j+n). \tag{3.27}$$

In the following chapters, we will follow this convention and call both operations convolution and specify when the kernel is flipped. A simple example without flipped kernel is shown in Figure 3.11.

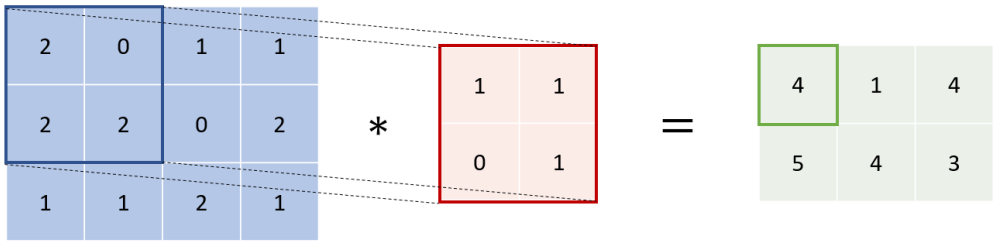


Figure 3.11: Simple example of a convolution without a flipped kernel.

The kernel - or filter - of the convolution operation is the part of the network that is trained. Intuitively speaking, in a forward pass, we slide each kernel across the width, height and depth of the input image and compute the convolution at every position. The result is a volume of two-dimensional activation maps which has a depth depending on the amount of filters that are trained (see Figure 3.9). As stated in the introduction to this section, similar to the visual cortex, this way the neurons are only locally connected to neurons in the input image or layer before. The spatial extend that each local field has depends on the size of the filter kernel. Although the spatial extend is constrained by the size of the kernel, it is always along the entire depth of the input volume. The amount of neurons in the output volume depend on three parameters: the number of filters, the stride and the amount of zero-padding that was applied. The stride refers to the number of pixels that the filter skips when sliding over the input channel. Another important concept that every convolutional layer uses is *parameter sharing*. Training one kernel for every neuron in the output would result in an extremely high number of parameters. Thus, for each output channel only one kernel is trained. As for regular neural nets, every neuron computes it's output by applying an activation function, such as Rectified Linear Unit (ReLU) or sigmoid.

Pooling Layers are commonly inserted into the network after a convolutional layer to gradually reduce the spatial resolution of the output volume. It operates independently on every depth channel of the volume and is usually used with a filter kernel of 2×2 and a stride of 2. The most common implementation is max pooling which simply compares all values covered by the kernel and returns the maximum value, as seen in Figure 3.12

Fully Connected Layer. This layer is a regular neural network layer, every neuron is fully connected to the activations in the layer before.

Training is an integral part to any neural network or learning algorithm. It iteratively changes the values of the weights from an initialized value to a configuration which allows it to solve the desired task. Training a regular neural network and a CNN is very similar.

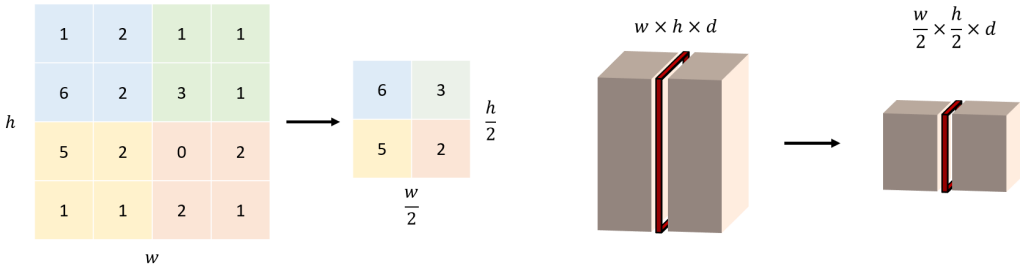


Figure 3.12: Left: Max pooling operation with a 2×2 filter and stride 2; Right: The operation only reduce the width and height of the volume and leaves the depth as is. A 2×2 filter and stride 2 max pooling operation reduces the volume by 75%.

Since it is a supervised learning algorithm, you require labelled training data and a loss function which quantifies the quality of the current set of weights. The loss is used to iteratively update the training weights using an algorithm called *Stochastic Gradient Decent* (SGD). SGD uses a method called *Backpropagation* to compute the gradient of the neural network graph which it then uses to find the updated values. As stated before, most backbone networks are pretrained on a multi-class classification task which means that the loss function is often the cross entropy loss which is defined as

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i} \mathbf{y}_i \log \hat{\mathbf{y}}_i. \tag{3.28}$$

Here, $\mathbf{y}$ are the ground truth class labels and $\hat{\mathbf{y}}$ is the predicted probability distribution, over i samples.

3.3.2 Bounding Box Regression

There are two subtasks to object detection: recognizing an object of a predefined class and localizing it using a bounding box. Modern detectors build on two inherently different philosophies: one-stage and two-stage detection. Two stage object detectors follow the footsteps of traditional object detection algorithms. In their first step they are generate region proposals, which are then classified in the second step [85]. In contrast, one step detectors solve a regression problem and use a unified architecture.

One-Stage Methods directly predict bounding boxes for an image. There is no intermediate task, which results in a simpler and faster model. The model starts off with the backbone network and removes the last fully connected layers, such that the last remaining layer is a stack of feature maps (see Figure 3.13). The feature maps are a representation of the original image. They are commonly very low-resolution due to the multiple pooling layers. However, because it is the very last convolutional layer it

[2] `https://www.jeremyjordan.me`

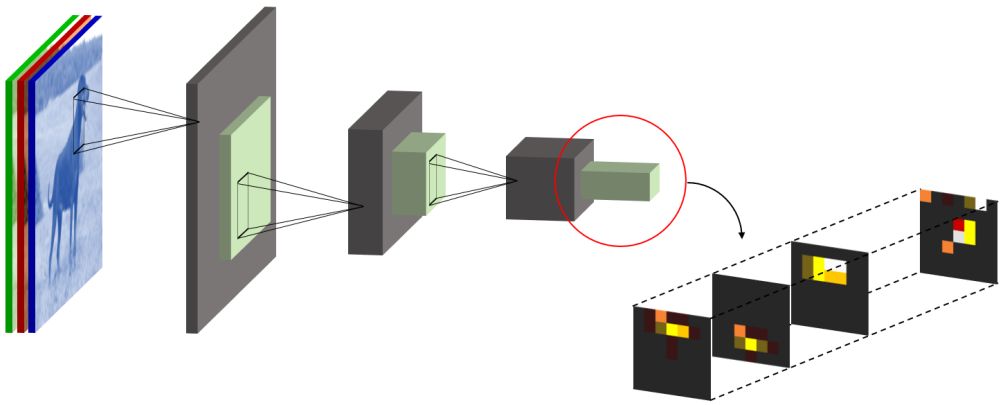


Figure 3.13: One-stage object detector removes the fully connected layer from a classification CNN. Representation inspired by Jeremy Jordan[2].

contains the best estimate of high-level semantics. It is possible to deduce a lot of insights and information from the activations in the feature maps. They roughly correlate with the objects and locations in the image. To transfer the knowledge contained in the channels to the object detection task, the architecture adds another convolutional layer. This convolutional layer learns one kernel each for the likelihood p_{obj} that a cell from the feature map contains an object, (x, y, w, h) of the bounding box and one filter for each of the C classes. Thus, the resulting volume has 5 channels, plus one channel for each class. The structure of the additional convolutional layer is depicted in figure 3.14. To detect multiple objects in one image, the volume of feature maps gets further scales by a factor B. This results in a volume of grid cells that have to be searched to find the most probable objects. To solve that problem, one-stage detectors use a method called *Non-Maximum Suppression*. The goal is to search through this huge volume of cells and find the best bounding boxes. Often there are multiple boxes for one object due to inaccuracies, it is desirable to single out the most confident prediction. In the first step, it is reasonable to filter out any bounding box with a likelihood below a certain confidence threshold. Then, non-maximum suppression selects all remaining bounding boxes and calculates the intersection between those. If boxes overlap a certain amount, the assumption is that they refer to the same object and thus, only the most probable of the boxes is selected.

Two-Stage Methods. Region with CNN features (R-CNN) [85] was one of the first popular deep learning implementations of the two-stage approach to object detection [93]. Although, the method got refined in several iterations afterwards, such as Fast R-CNN [86] and Faster R-CNN [17], the method remains fundamentally the same. First, they propose several regions in the input image. The regions are then fed to a CNN for feature extraction. At last, there is a classifier and regressor for bounding boxes and class labels.

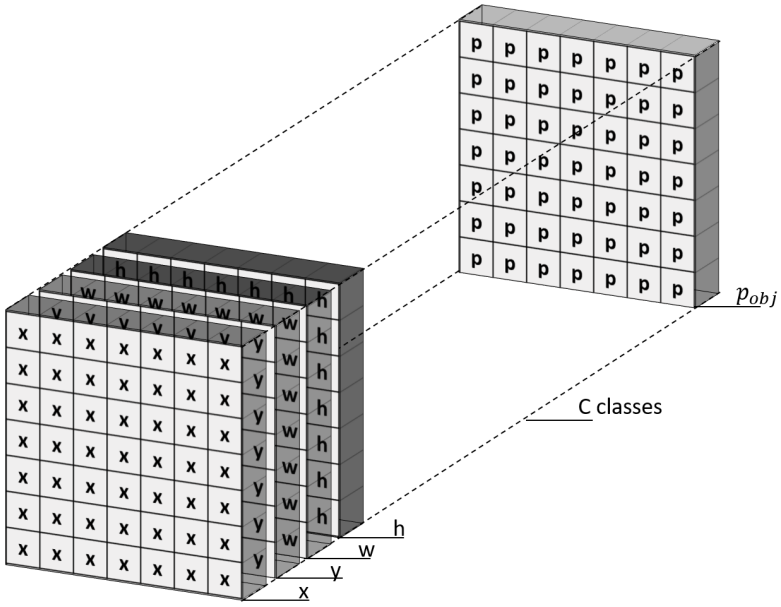


Figure 3.14: One-stage object detector removes the fully connected layer from a classification CNN. Representation inspired by Jeremy Jordan[3].

When R-CNN was first published, they used selective search for the region proposal step, a CNN for feature extraction and a pretrained SVM for classification and localization. One major problem with the approach was, that it builds on three different modules which can't be end-to-end trained. Additionally, the selective search takes around 2 seconds per image which was a major bottleneck of the architecture. Fast R-CNN still used selective search for the region proposals. However, instead of using a shallow SVM for classification, they used fully-connected layers integrated into the CNN, thus unifying feature extraction and classification/localization. The third iteration, Faster R-CNN, finally completed the evolution of the region proposal CNNs by replacing the selective search with a Region Proposal Network (RPN). The RPN slides a 3×3 window over the last feature map of the backbone network and proposes bounding boxes at different aspect rations. The RPN unifies feature extraction, region proposals and classification and finally makes the model end-to-end trainable.

3.3.3 MobileNet SSD

Instead of designing, building and training a new object detector, this thesis will build on a pretrained model. The process of creating a new, better detector is a task that many machine learning giants are committed to. In the scope of this thesis it is simply not reasonable, due to resources and time being two major limiting factors. Luckily, one key

principle of the computer vision community is open-source software development. Thus, a huge variety of excellent models are at our disposal. The goal is to choose a backbone network and bounding box regression technology which suits our use-case best.

As a backbone network, the models we can choose from are ResNet-50 [16], ResNet-101 [16], Inception v1 [15], Inception v2 [94], MobileNet v1 [90] and MobileNet v2 [95]. In regards of the bounding box regression method there is Sing Shot Multibox-Detection (SSD) [88] representing the single-stage detectors and Faster R-CNN [17] or R-FCN [87] for the two-stage detectors. All models were pretrained on the MS COCO dataset [21]. A list comparing the models is shown in Table 3.1.

Table 3.1: Comparison of object detection models [7]. Mean average precision (mAP) measure on the COCO [21] dataset. The speed depicted in the table was calculated with a NVIDIA Geforce Titan X GPU. To evaluate the performance of the model, the mAP was measured on the validation set.

Model Name	Speed (ms)	mAP
SSDLite MobileNet v2	27	22
SSD MobileNet v1	30	21
SSD MobileNet v2	31	22
SSD Inception v2	42	24
Faster R-CNN Inception v2	58	28
SSD ResNet-50	76	35
Faster R-CNN ResNet-50	89	30
R-FCN ResNet-101	92	30
Faster R-CNN ResNet-101	106	32

It is important to note that the speed measurements that are shown in the table are the result of running the object detector on an ultra-high-end GPU, the NVIDIA Geforce Titan X. The performance measure is calculated on the validation set of COCO. Although these measures are almost useless to evaluate how good they will run on our embedded platform and use-case, there are still some insights that can be derived. Single-stage detectors certainly have an edge when it comes to processing speed. Four of the five fastest detectors are build on the SSD technology. Although, two-stage detectors are generally more accurate than one-stage approaches, it appears that on the COCO validation set they perform approximately similar. Nonetheless, our tests showed that in terms of detection accuracy for aerial imagery use-case two-stage detectors are superior[4].

[4] `https://www.youtube.com/watch?v=LXqMVgUPeEw`

We train ScopeNet with the ground truth labels of the dataset instead of the detection results of the object detector. This allows us to view the detector as a modular entity which can easily be replaced in a plug-and-play fashion. Once a new, better implementation is released, we can easily adapt our model. Simultaneously, if an object detector performs poorly in some condition it can easily be switched out.

For the rest of the thesis we will use the *SSD MobileNetV2* for the pipeline. The model was released in April 2018 by Google. It can compete in terms of accuracy with some of the much deeper models and still maintain very low latency. Especially intriguing is the fact that MobileNet was precisely designed for embedded use-cases. The following will shortly introduce the model architecture.

MobileNetV2 is the backbone of the model. It is the successor of MobileNet which was published in 2017. The key differnce that MobileNet builds on is the *Depthwise Seperable Convolution* [90]. To reduce the total number of computations that are needed in a convolutional layer, they split the standard convolution into a depthwise convolution followed by a 1×1 convolution. The process can be seen in Figure 3.15. By replacing

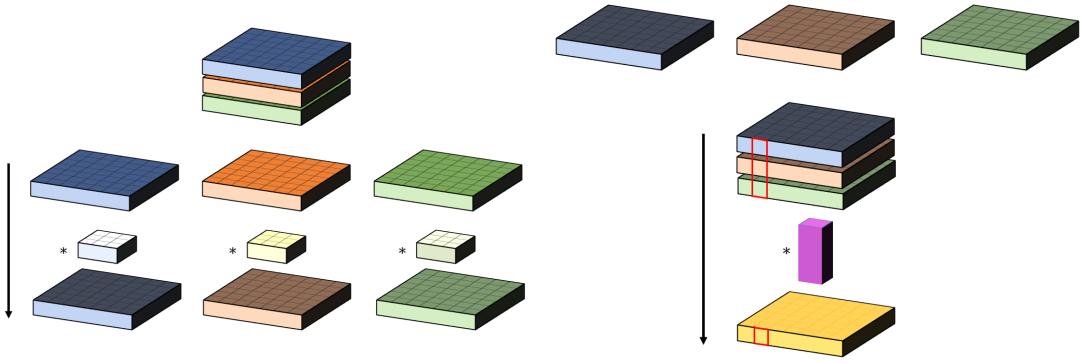


Figure 3.15: The depthwise seperable convolution is a key feature of MobileNet. Left: step 1 is a depthwise convolution, which is a lightweight filtering by applying a single convolutional filter per input channel; Right: a regular 1×1 convolution is then responsible for building new feature by computing linear combinations of the three output channels of the depthwise convolution.

a standard convolution with the two step process of depthwise separable convolution, it reduces the number of computations needed by

$$\frac{1}{d_i} + \frac{1}{w_i \cdot h_i} \tag{3.29}$$

where d_i is the number of output channels and w_i, h_i the width and height of the kernel. Beside that, MobileNet has a simple architecture which makes it intriguing.

Single Shot Multibox Detection builds on the concept of one-stage detection that was covered in Subsection 3.3.2. More specifically, that it is possible to relate the feature maps of convolutional layers to the input image in a gridwise fashion. For instance, if

the feature map at a certain layer is $16 \times 16 \times 128$ then we can relate the 16×16 grid to the original image and each gridcell can be considered a feature. How feature maps of different spatial resolution relate to the input image, can be seen in Figure 3.16. With that

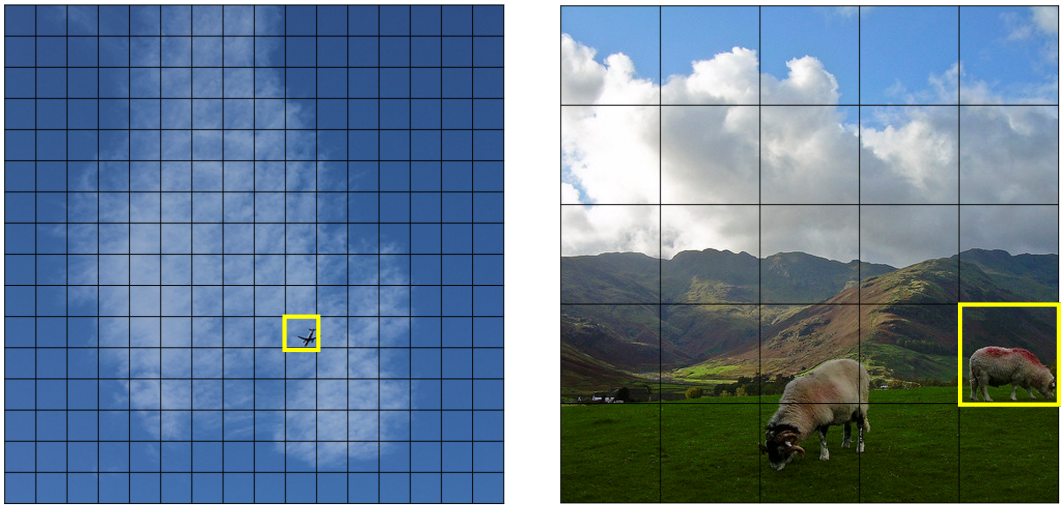


Figure 3.16: SSD uses the idea that it can use shallow layers to detect small objects and deeper layers to detect bigger objects. Both images are from MS COCO [21].

idea in mind, SSD does classification and bounding box regression on multiple different feature maps in parallel, by redirecting the values before they are further processed in the next convolutional layer. To visualize the information flow in an SSD network, Figure 3.17 shows the original architecture from *Lui et al* [88]. The model is taken from their paper and uses a VGG-16 backbone instead of the MobielNet v2.

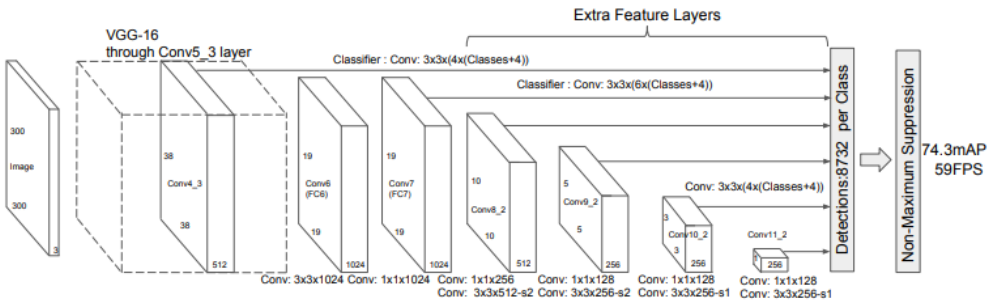


Figure 3.17: The original SSD architecture from *Lui et al.* [88]. In certain intervals, SSD computes classification and regression on feature maps of different spatial resolutions in parallel. This allows the network to detect objects at different scales in parallel.

This concludes the section detailing the object detector which is an integral part of

the visual attention and detection pipeline. At first, we detailed the fundamental functionality of backbone networks. Backbone networks play a central role in the detector because they extract the features upon which the bounding boxes are predicted. Then, the two most relevant bounding box regression methods were introduced. Finally, we reached the conclusion of the chapter by detailing the model that the pipeline uses: SSD MobileNet V2. It is a very recent architecture developed for embedded object detection which still retains very competitive accuracy on popular datasets.

4 ScopeNet

At last, we can embed ScopeNet into the Visual Attention and Detection pipeline. This chapter highlights the central piece of this thesis. Learning the key-frame scheduling policy from raw data is one of the main contributions of the thesis and ScopeNet is the component that controls the information flow in the model. So far there is no "intelligence" in the overall architecture. To sequentially analyze a video stream there needs to be an entity which remembers what happened in the past and can make a decision for the future. Hence, we need an intelligent model which has an internal state that can relate current observations to events from the past and has an understanding of computational resources.

To start off the chapter, we will briefly refresh the goals of ScopeNet and the functionality it has to implement. Then, Section 4.2 covers the *Synchronous Advantage Actor Critic* (A2C) which is the deep reinforcement learning algorithm that ScopeNet is built upon. Afterwards, 4.3 shows how A2C is integrated in three different models: a baseline model, a more sophisticated approach and a final "lessons-learned" model. The subsequent section will then detail the different technologies that are important for the models.

4.1 Goal Description

ScopeNet is supposed to solve two problems in the pipeline:

- We cant apply the object detector directly to the frame, because modern detectors are design for low-resolution images. Upscaling the object detector is unfeasible due to computational cost and downscaling the image does not retain enough information;
- We can't apply the detector directly to *every* frame in the video stream. Available object detector today are slow and designed for images. Video data introduces a variety of intrinsic properties that an image object detectors does not address.

The goal of ScopeNet is to resolve both issues simultaneously with an intelligent, cost-aware agent. The agent observes a drastically downsampled version of the input frame at around 6% of the original size[1]. Then, it has two options: **(a)** If the agent predicts

[1]The average frame of the dataset is around 2560×1440 pixels.

that there are objects in the image, which are currently not tracked (thus, the pipeline is unaware of), he can choose to apply the object detector to that subregion. Such a frame is called a *key-frame*. **(b)** If there is no new information in the current frame he can choose to skip the frame. In the background a low-cost object tracker is running which propagates the objects from observation to observation.

4.2 Synchronous Advantage Actor Critic

This section introduces the A2C algorithm. Beforehand, a short introduction of notation and fundamentals is in order. Reinforcement learning is a class of machine learning algorithms which are neither supervised nor unsupervised [96]. Instead, reinforcement learning is concerned with a problem in which an agent is placed in an environment. The agent learns by interacting with its environment and a scalar reward feedback signal R_t. In other words, Reinforcement Learning (RL) learns sequences of actions to maximize the expected reward of the agent[2].

4.2.1 Fundamentals

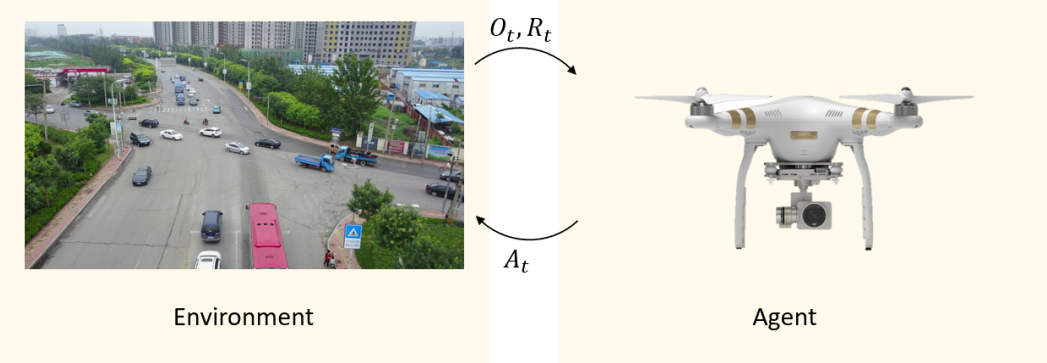


Figure 4.1: The reinforcement learning scenario. A agent interacts with its environment on the basis of observations O_t and is awarded rewards R_t according to its actions A_t.

The general scenario can be seen in Figure 4.1. At time t, the agent executes action A_t, receives a scalar reward R_t and is stimulated with an observation O_t. The environment receives A_t and emits O_{t+1} and R_{t+1}. RL bases on the reward hypothesis [96], that all goals can be described by the maximization of the expected cumulative reward. The goal of the agent is to select the actions that maximize its total future, long term reward.

[2] A big portion of the background which is presented in this chapter is based on the lecture *Advanced Topics in Reinforcement Learning* held by David Silver at University College London. Material can be found here: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`

To make a decision what action to take next, there has to be a state S_t. The state is formally a function of the history of all observations, actions and rewards, $H_t = O_1, R_1, A_1, ...A_{t-1}, O_t, R_t$. Such that

$$S_t = f(H_t). \tag{4.1}$$

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]. \tag{4.2}$$

In other words, the future is independent of the past, given the present. An environment is considered fully observable, if the agent can directly observe the state of the environment S_t^e, such that

$$O_t = S_t^a = S_t^e, \tag{4.3}$$

where S_t^a is the state of the agent. It is called a *Markov Decision Process* (MDP). These circumstances rarely hold in scenarios outside of simulations. For instance, a drone who reads a camera frame, does not have absolute knowledge about its environment. Thus, the environment is only partially observable $S^a \neq S^e$ and called *Partially Observable Markov Decision Process* (POMDP). In such an environment, the agent has to build its own state representation. The environment that we are defining for our problem is such a POMDP. In our case, the agent uses a Recurrent Neural Network (RNN) to construct its state

$$S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o), \tag{4.4}$$

where σ is some non-linearity, W_s are weights for the internal state and W_o are weights of the current observations.

There are different components which influence how the agent interacts. The agent can use a value function $\mathbf{v}_\pi(s)$, according do a implicit policy π, to predict future rewards. This allows it to evaluate the quality of a state s and discriminate between actions a. The value function

$$\begin{aligned} \mathbf{v}_\pi(s) &= \mathbb{E}_\pi\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s\big] && (4.5)\\ &= \mathbb{E}_\pi[G_t|S_t = s] && (4.6) \end{aligned}$$

with accumulated reward

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{4.7}$$

is the expected value of future rewards given a state. $\gamma \in [0, 1]$ a discount factor. $\gamma = 0$ is called a is myopic evaluation. If $\gamma = 1$ the agent evaluates rewards in the infinitely distant future. Similarly, the action-value function

$$\mathbf{q}_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{4.8}$$

is the expected return from state s taking action a and then following policy π. A stochastic policy $\pi(a|s)$ fully defines the behaviour given a state:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \tag{4.9}$$

The goal is to find the optimal state-value or action-value function over all policies $\mathbf{q}_*(s,a) = \max_\pi \mathbf{q}_\pi(s,a)$. The solution is non-linear and is computed iteratively.

Another important concept that distinguishes RL from other machine learning classes, is the explortation exploitation trade-off. In order an agent can complete his task successfully, it needs to explore its environment, even if the actions don't bring him closer to his goal. Once the agent has gathered enough information, he has to exploit the environment to maximize the reward.

How to does one esimate these functions? There are a number of different approaches [96], but the two most popular are Monte Carlo (MC) Learning and Temporal-Difference (TD) Learning.

Monte Carlo methods learns based on full episodes in the environment. The goal is to learn the estimated value function $V(s)$ under the policy π. After a episode is concluded, the total reward is calculated and $V(s)$ is updated, such that after a number of episodes $V(s) \to \mathbf{v}_\pi(s)$. It is easy to see, that if in the episode an action was negative, but the overall reward was positive, the agent can'trealize that he took a bad action. Thus, MC tends to converge very slowly.

Temporal-Difference methods learn from incomplete episodes. It learns $V(s)$ online by bootstrapping under policy π. It iteratively updates its estimate value function

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\delta_t\text{:TD error}}) \tag{4.10}$$

into the direction of the TD error with a learning rate α [96]. TD holds several advantages against MC. It can learn from every step in non-episodic environments. Additionally, as said before MC has a high variance due to hidden actions. In contrast, TD has low variance but introduces bias. Generally, TD learning converges faster than MC learning. The A2C algorithm is a TD method. Thus, from now on we will only consider the TD scenario.

Often, $V(s)$ or $Q(s,a)$ is implemented by lookup-tables that are iteratively updated. Table 4.1 shows such a lookup-table for a simpel MDP with four states and three action. In a TD scenario, the agent can pick the best action at every state and update the values of the table, for instance according to equation 4.10.

However, most interesting problems don't have a small state space, some have millions of states with hundreds of actions. It is obvious, that for such an MDP it is unfeasible to implement a lookup-table. The solution is to generalize from seen states to unseen

Table 4.1: A exemplary Q-Learning look-up table.

	a_0	a_1	a_2
s_0	0.1	0.5	0
s_1	7	2.9	0
s_2	1	0.3	0.1
s_3	0.3	1.7	0

states and estimate the value functions with function approximation iteratively

$$\hat{\mathbf{v}}(s, \mathbf{w}) \approx \mathbf{v}_\pi(s) \tag{4.11}$$

$$\hat{\mathbf{q}}(s, a, \mathbf{w}) \approx \mathbf{q}_\pi(s, a). \tag{4.12}$$

Here, the state value function approximation $\hat{\mathbf{v}}(s, \mathbf{w})$ tells the agent how much reward he can get when he follows the policy π. Similarly, $\hat{\mathbf{q}}(s, a, \mathbf{w})$ tells the agent how much reward he can get along some trajectory, following some policy π if he takes action a.

This thesis, uses neural networks as function approximators. The parameters can then be updated like regular neural networks with some stochastic gradient descent (SGD) method. Consider $J(\mathbf{w})$ to be a differential function of parameter vector $\mathbf{w}$, then its gradient is defined as

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_2} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}. \tag{4.13}$$

As always, the gradient describes the direction of the steepest ascend, thus we can follow the gradient downwards

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{4.14}$$

with step size α.

For now, let us imagine we have a supervisor and know $\mathbf{q}_\pi(S, A)$. To approximate the action-value function with stochastic gradient descent, we need to find parameter vector $\mathbf{w}$ that minimizing the mean squared error (MSE) between $\hat{\mathbf{q}}(S, A, \mathbf{w})$ and $\mathbf{q}_\pi(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_\pi[(\mathbf{q}_\pi(S, A) - \hat{\mathbf{q}}(S, A, \mathbf{w}))^2]. \tag{4.15}$$

Thus,

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{4.16}$$

$$= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} E_\pi[\mathbf{q}_\pi(S, A) - \hat{\mathbf{q}}(S, A, \mathbf{w}))^2] \tag{4.17}$$

SGD then samples this gradient gradient

$$\Delta \mathbf{w} = \alpha \Big(\mathbf{q}_\pi(S, A) - \hat{\mathbf{q}}_\pi(S, A, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{\mathbf{q}}_\pi(S, A, \mathbf{w}). \tag{4.18}$$

However, the true value of $\mathbf{q}_\pi(S, A)$ is unknown. We only have the rewards at our disposal and can substitute the TD target $R_{t+1} + \gamma \hat{\mathbf{q}}(S_{t+1}, A_{t+1}, \mathbf{w})$ for it

$$\Delta \mathbf{w} = \alpha \Big(R_{t+1} + \gamma \hat{\mathbf{q}}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{\mathbf{q}}(S_t, A_t, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{\mathbf{q}}(S_t, A_t, \mathbf{w}). \tag{4.19}$$

The policy is the behaviour of the agent. So far the policy was only defined implicitly. However, since we want to find the optimal behaviour strategy for an agent, it might seem more intuitive to directly optimize the policy. These algorithms are called *Policy Gradient* methods. They aim to directly parametrize the policy

$$\pi_\theta(a, s) = \mathbb{P}[a|s, \theta] \tag{4.20}$$

with respect to θ. Many exciting results in general artificial intelligence and control have been accomplished with policy gradients in recent years [96, 97, 98, 99, 100].

Let $J(\theta)$ be the objective function[3], defined as

$$J(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) V^{\pi_\theta}(s) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{4.21}$$

where $d^{\pi_\theta}(s) = \lim_{t \to \infty} \mathbb{P}(s_t = s | s_0, \pi_\theta)$ is the on-policy distribution under π_θ. Finding the gradient of 4.21 is challenging because it depends on the action selections and the distribution of states in which those selections are made. Both of these are affected by the policy parameter. Nonetheless, *Sutton et al.* [96] found a solution in the form of the policy gradient theorem:

Theorem 1 (Policy Gradient Theorem) *For any differentiable policy $\pi_\theta(s, a)$ the policy gradient is*

$$\nabla_\theta J(\theta) \propto \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(a|s). \tag{4.22}$$

[3] A lot of the information on policy gradients was taken from Lilian Wengs blogpost at `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`

The expression can further be written as:

$$\nabla_\theta J(\theta) \propto \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \tag{4.23}$$

$$= \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \frac{\pi_\theta(a|s)}{\pi_\theta(a|s)} \tag{4.24}$$

$$= \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s,a) \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \tag{4.25}$$

$$= \mathbb{E}_{\pi_\theta} \Big[Q^{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s) \Big] \tag{4.26}$$

Here the log-likelihood trick was used, such that

$$\nabla_\theta \log \pi_\theta(a|s) = \frac{1}{\pi_\theta(a|s)} \cdot \nabla_\theta \pi_\theta(a|s). \tag{4.27}$$

This insight sprouted a variety of different policy gradient algorithms in recent years. One of them is the Actor-Critic which is the basis of ScopeNet.

4.2.2 Actor Critic

When we now want to run a policy gradient algorithm and update the weights θ such that

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) Q^{\pi_\theta}(s_t, a_t) \tag{4.28}$$

the action-value function is unknown. Initial policy gradient methods such as REINFORCE [101] used the return G_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$. This required them to wait for the entire episode to finish, thus being a MC method.

To be able to update the model as TD at every step, we require an approximation of the action-value function

$$Q_{\mathbf{w}}(s,a) \approx Q^{\pi_\theta}(s,a). \tag{4.29}$$

This model, which now uses an action-value function approximator and a policy gradient, is called an **Actor-Critic** model.

The Actor-Critic consists of two models with two sets of parameters θ and $\mathbf{w}$:

- the Critic updates the action-value parameters $\mathbf{w}$ of $Q_{\mathbf{w}}(s,a)$;
- the Actor interacts with the environment and updates the parameters of the policy $\pi_\theta(a|s)$.

Intuitively, the Critic observes the actions of the Actor and provides feedback. Learning from the feedback, the Actor updates its policy. The Critic will also learn to give better feedback.

To update the policy, we now use the approximated action-value function

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_{\mathbf{w}}(s,a)] \tag{4.30}$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a|s) Q_{\mathbf{w}}(s,a) \tag{4.31}$$

and to update the Critic we use the TD-target as in equation 4.19

$$\mu = \mathbb{E}_{\pi_\theta}\big[(Q^{\pi_\theta}(s,a) - Q_{\mathbf{w}}(s,a))^2\big] \tag{4.32}$$

$$\Delta\mathbf{w} = \beta\big(r + \gamma Q_{\mathbf{w}}(s_{t+1}, a_{t+1}) - Q_{\mathbf{w}}(s_t, a_t)\big)\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s_t, a_t) \tag{4.33}$$

where r is the current reward and β is a second learning rate. Moreover, we can establish equality in equation 4.30 if we choose $Q_{\mathbf{w}}$ as in 4.32.

The Advantage of an action a_t in a state s_t, is a concept that can significantly reduce the variance of the policy gradient [97]. It makes our Actor-Critic model to an *Advantage Actor-Critic*. The advantage can be any function $B(s)$. It is used as a comparison to the action-value

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - B(s). \tag{4.34}$$

The advantage does not change the expectation as long as it is independent of a:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s,a)\big] \tag{4.35}$$

$$= \mathbb{E}_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a|s)(Q^{\pi_\theta}(s,a) - B(s))\big] \tag{4.36}$$

$$= \mathbb{E}_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\big] - \mathbb{E}_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a|s) B(s)\big]. \tag{4.37}$$

Where 4.37 follows due to the linearity of the expectation. All that is left to do, is to show that $\mathbb{E}_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a|s) B(s)\big] = 0$.

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) B(s)] = \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \nabla_\theta \pi_\theta(s,a) B(s) \tag{4.38}$$

$$= \sum_{s\in S} d^{\pi_\theta}(s) B(s) \nabla_\theta \underbrace{\sum_{a\in A} \pi_\theta(s,a)}_{=1} \tag{4.39}$$

$$= 0 \tag{4.40}$$

Here, from 4.38 to 4.39 the fact was used that $B(s)$ and ∇_θ do not depend on a and the gradient of a constant is zero.

It turns out, that the state value function $V^{\pi_\theta}(s)$ is a good candidate for $B(s)$. Thus, the policy gradient becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)] \tag{4.41}$$

with advantage function

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s). \tag{4.42}$$

Since both Q^{π_θ} and V^{π_θ} are both unknown, we would need two sets of function approximators to estimate the advantage function

$$A(s,a) = Q_{\mathbf{w}}(s,a) - V_{\mathbf{v}}(s). \tag{4.43}$$

Then, we need to train three approximators, for instance neural networks. Luckily, there is a more elegant way. The idea is that the TD-error is an unbiased sample of the advantage function [102]. If we use the expected TD error as our advantage function

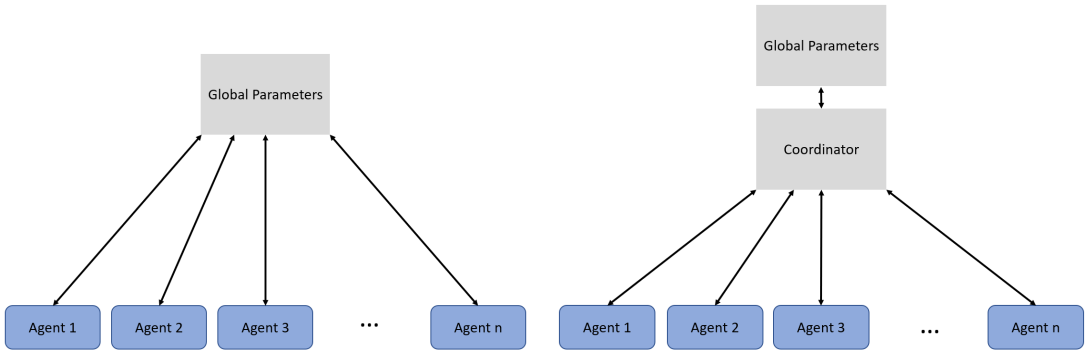
$$A^{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta}|s,a\right] \tag{4.44}$$

the estimate of the advantage becomes

$$A_{\mathbf{v}}(s,a) = r + \gamma V_{\mathbf{v}}(s') - V_{\mathbf{v}}(s) \tag{4.45}$$

where s' is the next state. The policy gradient becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A_{\mathbf{v}}(s,a)\right] \tag{4.46}$$



(a) A3C (b) A2C

Figure 4.2: In the A3C network, all agents update the global network on their own. Thus, it can happen that two agents have different parameters and update the global network. This can lead to undesirable effects. The A2C network (Right) has an intermediate entity which controls the flow. The coordinate waits for all agents to finish their sequence before updating the network.

Synchronous. To fully utilize modern GPUs, researchers wanted to execute multiple environments in parallel. At first *Mnih et al.* [97] proposed an Asynchronous Advantage Actor-Critic (A3C). It learns the value function while multiple actors are trained in

parallel. Every agent has its own set of parameters which get synched at the start of each sequence. The agents interacts with the environment for given time steps $t_{\max}$ and update their gradients individually, before synchronizing again. Due to each agent having its own set of parameters, it is sometimes possible that thread-specific agents are playing with different versions of the functions. This leads to undesirable updates to the global parameters.

At last, the Synchronous Advantage Actor-Critic resolves this issue. It is a deterministic and synchronous version of A3C. In A2C all agents unroll a certain number of time steps $t_{\max}$ and store their experience in minibatches. After all the agents finish their work, the total experience of all agents is used to update the global parameters. Algorithm 1 presents the pseudo-code of the A2C implementation that is used in the thesis. OpenAI researchers stated in a blogpost [4], that in their research A2C uses the GPU more effectively and converges faster than A3C.

Algorithmus 1 : Synchronous Advantage Actor-Critic Pseudocode

//Global shared parameter vectors theta and $\mathbf{v}$
//Global shared counter $T = 0$
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $\mathrm{d}\theta \leftarrow 0$ and $\mathrm{d}\mathbf{v} \leftarrow 0$
  Reset minibatch: $(\mathcal{S}, \mathcal{A}, \mathcal{V}, \mathcal{R}, \mathcal{G})$
  $t_{\text{start}} = t$
  Get state s_t
  //Roll out t_{max} *steps*
  repeat
    Perform a_t according to policy $\pi_\theta(a_t|s_t)$
    Receive reward r_t, value v_t and new state s_{t+1}
    Store s_t, a_t, v_t, r_t in $\mathcal{S}, \mathcal{A}, \mathcal{V}, \mathcal{R}$
  until $t - t_{start}$ *is* t_{max};
  $$G = \begin{cases} 0 & \text{if } s_t \text{ terminal} \\ V_{\mathbf{v}}(s_t) & \text{else } \textit{//Bootstrap from last state} \end{cases}$$
  for $i \in t-1, ..., t_{start}$ **do**
    $G \leftarrow \mathcal{R}[i] + \gamma G$
    Store G in $\mathcal{G}$
  end
  Perform update of θ and $\mathbf{v}$ using the experience in the minibatch
until $T > T_{max}$;

[4] OpenAI blog post arguing for the superiority of A2C against A3C: `https://blog.openai.com/baselines-acktr-a2c/`

4.3 Models

To evaluate the approach proposed in this thesis, we will build three different models:

- Model (A) is a baseline implementation. It is a reference point for the other models;
- Model (B) is a more sophisticated approach. It uses ResNet-18 as backbone and a multi-task loss to aid training convergence;
- Model (C) embodies the lessons-learned from its two predecessors. It combines the best practices and methods;

This way, we have a reference implementation which allows us to make inferences for future research. The baseline solution consists of three different components: the Actor-Critic model, a LSTM and the convolutional head. In addition, (B) and (C) use an multitask model which learns to complete an additional task with an extra regression layer. This section will briefly outline the different components. The implementation of the models is detailed in the next chapter.

4.3.1 Actor-Critic Model

As implied in Section 4.2.2 you can approximate the value function and policy with a variety of different linear or non-linear function approximators. We will use two neural networks to approximate them. They share one fully connected layer, as depicted in Figure 4.3. The main objective is to reduce the memory that is needed to train the network. Additionally, it improves the network's ability to generalize. Nonetheless, it can take longer to train. The first fully connected layer is connected to the feature network. It's size depends on the last convolutional layer of the feature network.

Before we get to the feature network, let us focus on the loss function of the Model. The model is trained with the loss function suggested by [97] and used in the OpenAI baseline implementation[5]. The loss consists of three terms:

$$L = L_\theta + c_{\mathbf{v}} L_{\mathbf{v}} + c_{\mathrm{H}} L_{\mathrm{H}}. \tag{4.47}$$

The objective function $J(\theta)$ was defined as the total reward an agent can achieve under the policy π. Thus, the loss is simply

$$L_\theta = -J(\theta) \tag{4.48}$$

with policy gradient as in equation 4.46

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(s,a) A_{\mathbf{v}}(s,a)\big]. \tag{4.49}$$

[5] `https://github.com/openai/baselines/tree/master/baselines/a2c`

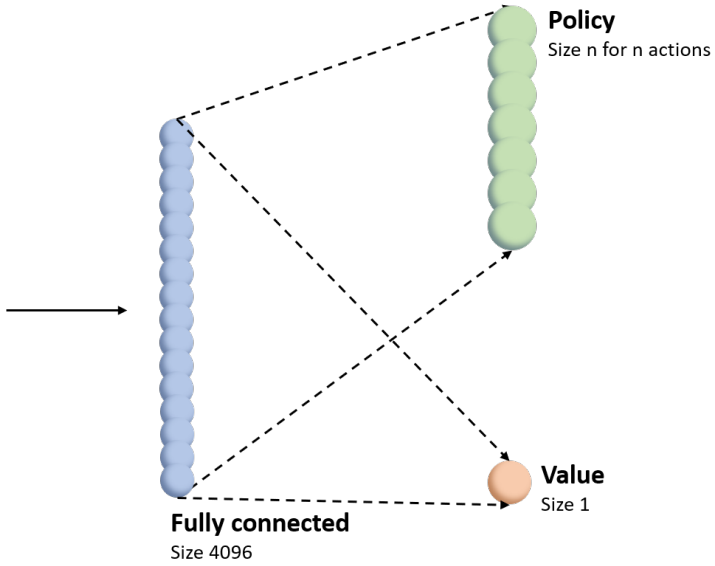


Figure 4.3: The policy and value layer share parameters in one fully connected layer. This improves the generalization and reduces memory usage. However, training can take longer. In this example the last feature map had dimensions $3 \times 3 \times 512$

We can't compute the gradient analytically, but can rewrite the definition[6], such that

$$J_\theta = \mathbb{E}_{\pi_\theta}\big[A_{\mathbf{v}}(s,a)\log\pi_\theta(a|s)\big]. \tag{4.50}$$

Finally, we have to average over all the samples in a batch and the policy loss becomes

$$L_\theta = -\frac{1}{n}\sum_{i=1}^{n} A_{\mathbf{v}}(s_i,a_i)\log\pi_\theta(a_i|s_i) \tag{4.51}$$

To make a distribution from the predicted action logits we use a softmax activation. The value function loss is the regression loss between the predicted value and the ground truth return

$$L_{\mathbf{v}} = \frac{1}{n}\sum_{i=1}^{n}(G(s_i) - V_{\mathbf{v}}(s_i))^2. \tag{4.52}$$

The original A3C implementation by DeepMind [97] uses an additional regularization loss term, which was first proposed by *Williams et al.* [103]. They subtract the entropy of the policy distribution from the loss. This was found to improve exploration, because it penalizes distributions which favor one action. We will use the same trick. The entropy

[6] `https://jaromiru.com/2017/03/26/lets-make-an-a3c-implementation/`

of the distribution is

$$H(\pi_\theta) = -\sum_{a_i} \pi_\theta(a_i|s) \log \pi_\theta(a_i|s) \tag{4.53}$$

The entropy is maximized if the distribution is uniformly distributed. Thus, we try to maximize the entropy. Averaging over all samples in the batch gives us the last term of the total loss

$$L_{\mathrm{H}} = -\frac{1}{n} \sum_{i=1}^{n} H(\pi_\theta(a|s_i)). \tag{4.54}$$

4.3.2 Long Short Term Memory

As briefly stated before, the environment a drone acts in is a POMDP. This means, that information is spatial and temporally limited. Every model that is poised to complete a sequential task in a POMDP needs some understanding of time. Regular feed-forward neural networks don't have an internal state, all they do is feed information forward. At any given time later, a feed-forward neural network has no knowledge of what happened beforehand. In some RL problems a straight forward approach is it to model the temporal dimension with a stack of input frames instead of a single frame [65]. This method has shown some great success even on complex problems like the Atari simulations. However, sometimes events that took place an arbitrary number of frames before is important for the next action.

The problem of temporal dependencies can be solved by a Recurrent Neural Network (RNN) which will be placed within the model [104, 105]. RNNs have successfully been used in a wide variety of use-cases. The recurrent module in our model allows us to change the predictions for the actions and values depending on the temporal pattern of observations.

Long Short Term Memory (LSTM) networks are a special kind of RNN which is capable of learning long-term dependencies. The LSTM was first introduced by [23] and work extraordinarily well on many problems. They are used in some of the most powerful RL algorithms today[7][100]. Luckily, the tensorflow API allows us to implement it as a black box. Thus, this subsection will briefly introduce the key ideas and architecture before implementing it into the ScopeNet model.

A LSTM network consists of LSTM units, or memory units, arranged in a chain strucutre. There are four neural network layer in each cell. The structure can be seen in Figure 4.4. The cell state C_t is propagated through the cells and the neural network layers have the ability to change the information contained in the cell state.

The ScopeNet implementation follows [104] in terms of the number of unit cells. As [100, 105, 104], the LSTM is implemented right before the value and policy function

[7] OpenAI's PPO algorithm with a LSTM recently beat semi-professional Dota2 players: `https://blog.openai.com/openai-five/`

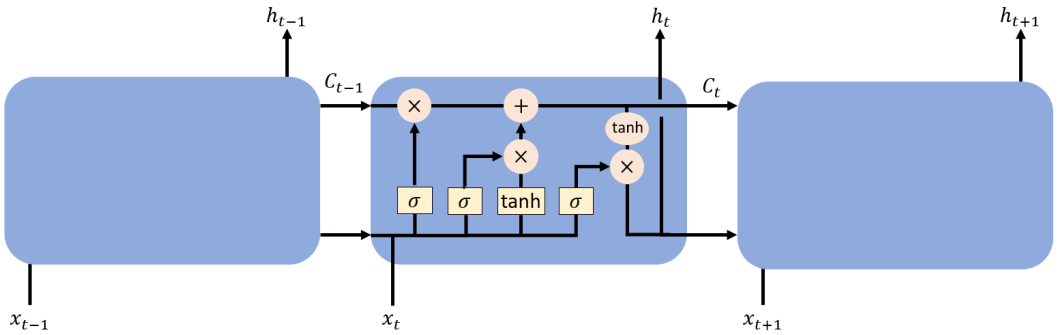


Figure 4.4: A LSTM network with three cell units. LSTMs are excellent recurrent neural networks to model long-term memory. The yellow boxes are neural network layers and the pink circles are pointwise operations. Each line carries a vector.

approximator (see Figure 4.5). However, there were differences in the literature whether to install one last fully connected layer between the last convolutional layer and the LSTM or not. *Lample et al.* [105] flatten their last feature map and feed it into a fully connected layer of size 4608. Oppositely, *Hausknecht et al.*[104] flatten their last feature map and feed it directly in the LSTM. We went with the approach of [105]. If a proof-of-concept is successful, but the model doverfits, the model can easily be reduced in size.

4.3.3 Convolutional Backbone

Although we build two different models, so far model (A) and (B) had the same architecture. Both have the same amounts of neurons in their function approximators as well as a LSTM network. The reason why we chose two different backbones for ScopeNet is as follows: almost every deep RL paper, that was published in recent years and uses a method similar to ours, plays in a simulation. Their observations are usually around 80×80 pixels [97, 65, 104, 105] and due to the nature of retro video games, or simulation in general, the variance in spatio-temporal information is much lower than in real camera footage. One approach that works with camera data uses a VGG-16 [89] pretrained on ImageNet, as a backbone [67, 66]. The network is dramatically more complex than the other backbone networks.

Thus, we implemented both variants. The baseline (A) uses convolutional layers following the design principles of [97, 65, 104, 105] which are vanilla convolutional neural networks. Approach (B) follows the choice of [89] and uses ResNet-18 [16] as a backbone network without the fully connected layers. ResNet-18 shows great results on various complex tasks, such as image classification.

(A) Baseline Backbone. The goal of the backbone is to build discriminative features from the input image that we can feed into the LSTM network. It gradually reduces the

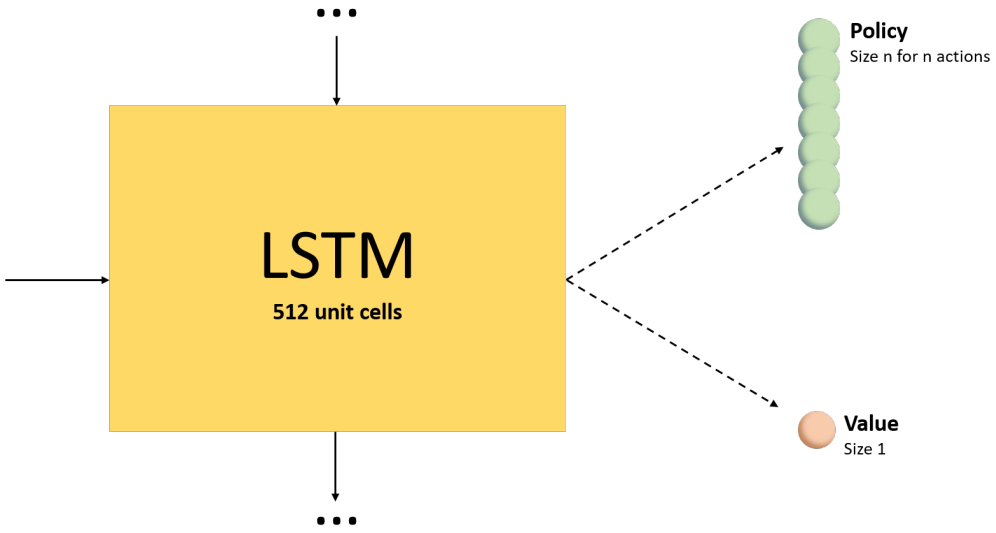


Figure 4.5: The LSTM network consisting of 512 cell units replaces the fully connected layer from Figure 4.3. It is implemented right before the value and policy function approximators.

the spatial dimensions, meanwhile increasing the number of feature maps, as throughly discussed in Chapter 3. Due to the input images having the dimension $640 \times 360 \times 3$ we will need more convolutional layer than the other implementations. Additionally, we break from their design principle in the choice of the kernel size of the filter. [105, 104] both follow the convention of the seminal work in [65] and use a 8×8 kernel for their first layer. Many other non-RL networks have relatively large kernels in the first layer as well. However, our task entails that the network can be sensitive to small objects as well, thus we follow the design principles of backbones like MobileNet [90] that have only 3×3 kernels. This keeps the effective receptive field of the network smaller which can lead to problems when the objects in the frame are overly large. But hopefully, the resulting features have more discriminative power for our aerial imagery use-case. Additionally, most Convolutional Neural Networks (CNN) are optimized on square images. Modern HD cameras often have a 16:9 aspect ratio, which ovver a wider view angle. The dataset was shot in 16:9 as well, thus the feature maps are rectangular. Other networks rectify their feature maps to squares with average pooling. In our case though, the rectangular shape is important, because with our design, the feature maps in the very last convolutional layer directly correlate with the actions that the agent is able to take. The last convolution operation brings the feature maps to the resolution of

3×2, which are exactly the 6 actions that the agent can take. This connection between the actions and the last feature map can be seen in Figure 4.6. The architecture of the backbone is detailed in Table 5.1. More information on the action space is found in section 5.2.2.

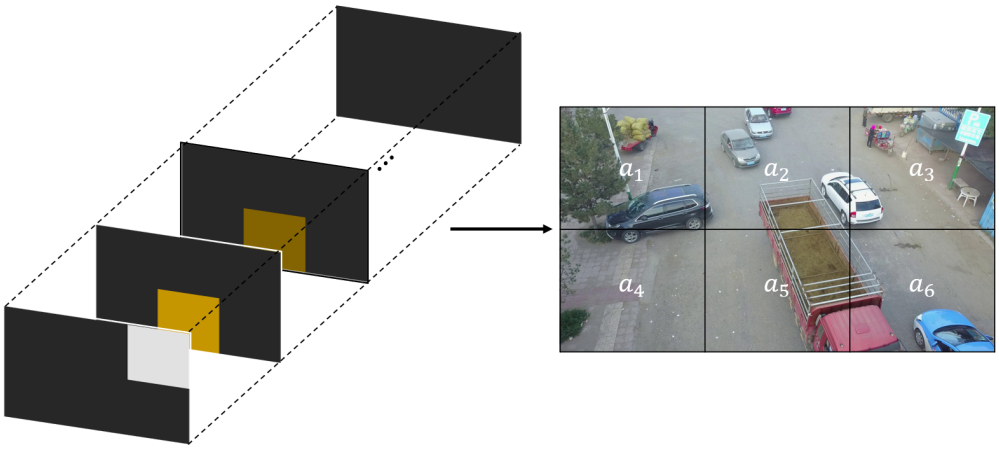


Figure 4.6: The activations in the last feature map of model (A) directly correlation with the action space of the agent. This improves the discriminative power the model.

(B) ResNet Backbone follows the design principle of [67] and uses a "off-the-shelf" architecture which is known to perform well in the ImageNet classification task but is still relatively shallow and fast: ResNet-18 [16]. ResNet won the ILSVRC 2015 and manifested the concept of residual units. A residual unit consists of two convolutional layers with a skip connection (see Figure 4.7). The skip-connection is key to being able to train very deep networks. Conceptually, the unit is very simple and easy to implement. It feeds the input signal forward through a shortcut in parallel to the convolutional layers and adds it to the output. The idea is, when training a neural network, the network learns a target function $h(x)$. If one adds the input x to the output that forces the network to model $f(x) = h(x) - x$ [106]. When a network is trained, the network flows from the output backwards. The skip connections allow the signal to make progress in earlier layers even if the deeper layers have not learned yet. The skip connections allow the gradient to flow more easily through the network and increase training speed considerably. This is the primary reason, why we opted for ResNet-18 instead of VGG that [67] uses. Moreover, ResNet uses batch normalization after every "Layer 2" in the residual unit and ReLU after "Layer 1" and before the output. Batch normalization is a technique which normalizes activations after the layer which can increase training speed and generalization of the model.

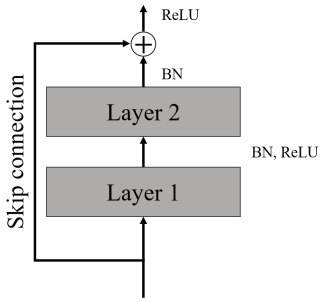


Figure 4.7: The key feature of all ResNet architecture is the residual unit. It employs a skip connection which forces the layer to model $f(x) = h(x) - x$. When training such networks, the skip connection has shown to allow the gradient to "flow" more easily to early layers. This can increase the training time drastically.

4.3.4 Multi-task Learning

Multi-task learning is popular and powerful method to regularize a model [107]. It improves the generalization ability of the network by pooling the examples from several tasks [34]. It puts additional pressure on the parameters in the layers shared among both tasks. Of course, multi-task learning can only improve the performance, and sometimes training speed, of a network if the task have a statistical relationship.

The idea to train the model on an additional task is taken from *Lample et al.* [105] who train a Deep Q-Network on a related classification task. The multi-task learning in their case incentives the CNN to see relevant objects and speeds-up the training time considerably.

It is important to note, that the implications of adding an additional loss to the model are uncertain. In [105], before adding the multi-task loss, the model is trained on only one objective. The A2C loss that ScopeNet learns to minimize already consists of three different terms. How the model will react on an additional loss is difficult to determine, because several factors such as scaling of the added loss factor play an important role and are nearly impossible to properly choose without extensive hyperparameter search.

5 Implementation

An absolutely vital part of the performance of the agent is the environment it interacts with. Most of the work that is currently done in Reinforcement Learning (RL) is focused on simulations. However, we want the algorithm to generalize to live camera footage, thus we settled for a real-world dataset. On top of the dataset, we build an environment in which the agent trains.

At first, Section 5.2 details the dataset, environment structure and reward shaping. Especially the latter is pivotal to the learning ability of the agent. Then, we will briefly cover the software and hardware stack that the experiments were performed on. Afterwards, Section 5.4 will cover the training of the ScopeNet models. At last, Section **??** conducts experiments to evaluate the hypothesis.

5.1 Models

Chapter 4 introduced all the fundamental building blocks of our models. Finally, we can construct their implementation.

5.1.1 Model (A)

To have a reference point, Model (A), uses the most simple architecture for the task. It can be seen in Figure 5.1. It consists of a Convolutional Neural Network (CNN)

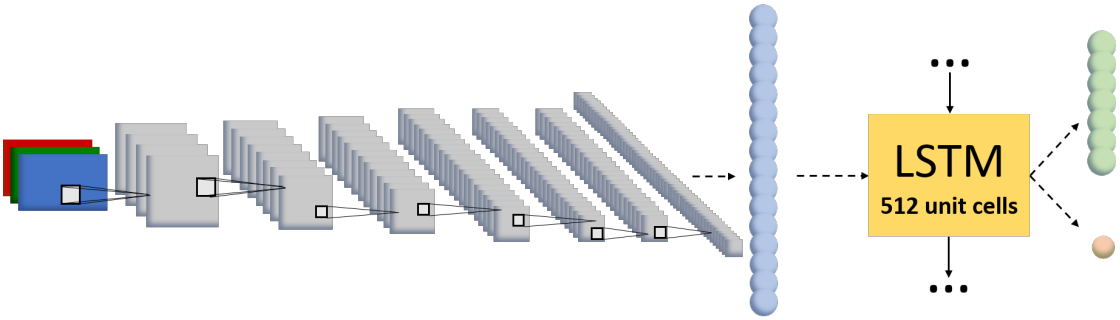


Figure 5.1: Model (A) is the baseline model for the hypothesis test. It consists of 7 simple convolutional layers, one fully connected layer, a LSTM network and the function approximators.

as a backbone network comprised of 7 layers (see Table 5.1). These layers build the features. After the last convolutional layer, the feature map is flattened and fed into a fully connected layer with 6144 neurons (light blue). The output of the layer is connected to a LSTM network with 512 cells. The output of the LSTM is directly connected to the two function approximators for the policy and value function. The details of the convolutional backbone are shown in Table 5.1.

Table 5.1: The (A) backbone which follows the design principles of other RL models [97, 65, 104, 105] and MobileNet [90].

Name	Kernel	Filter	Stride	Input
conv1	5×5	32	4	$640 \times 360 \times 3$
conv2	3×3	64	2	$160 \times 90 \times 32$
conv3	3×3	128	2	$80 \times 45 \times 64$
conv4	3×3	256	2	$40 \times 23 \times 128$
conv5	3×3	512	2	$20 \times 12 \times 256$
conv6	3×3	1024	2	$10 \times 6 \times 512$
conv7	3×3	1024	2	$5 \times 3 \times 1024$
FC8	1×1	flatten	1	$3 \times 2 \times 1024$

5.1.2 Model (B)

Model (B) uses an advanced convolutional backbone and an additional multi-task loss. Instead of the 7 layers of Model (A), Model (B)'s backbone consists of 18 convolutional layers. The architecture can be seen in Table 5.2.

The additional task that we train the model on, is an object count regression. The task of the model is to count the total number of a certain object class within the frame. The ground truth labels for the task are easily acquired from the dataset. As loss, we use the mean squared error

$$L_{\text{reg}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{5.1}$$

where y_i is the ground truth number of objects in frame i of the batch and $\hat{y}_i$ is the predicted number of objects. Additionally, we use an extra fully connected layer to train learn the regression. The regression loss is added to the total loss of the model without a scaling term.

In conclusion, ResNet-18 computes the features in model (B). The last feature map is flattened and fed into both branches of the network. The regression branch uses 512 neurons as proposed by [105], while the branch that leads to the A2C function approximators has 4096 neurons.

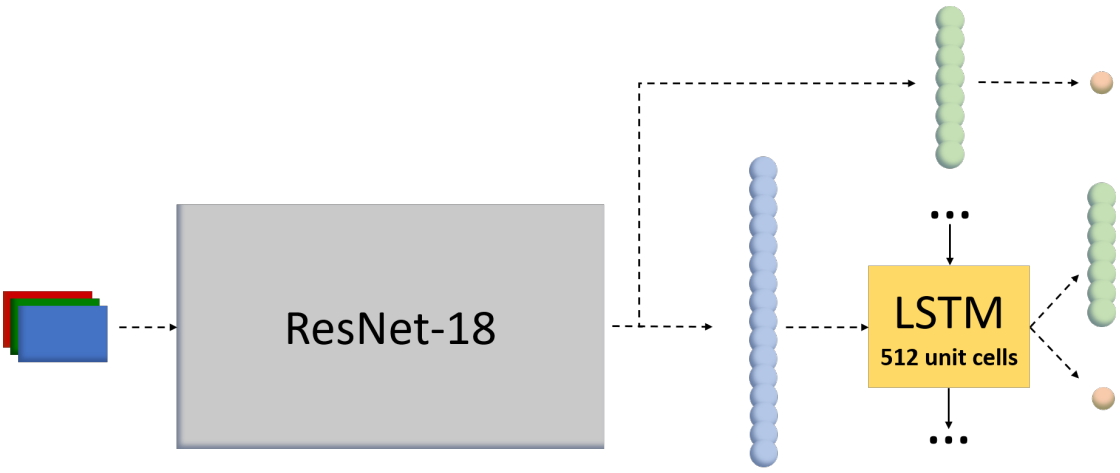


Figure 5.2: Model (B) allows us to apply more advanced technology to the problem. It builds on an established backbone network and an additional regression task.

5.1.3 Model (C)

Model (C) embodies the lesson-learned and best-practices from the two previous models. After training Model (A) and (B) it became apparent that each model had components which worked as intended. However, each had its own respective flaws which led to the creation of Model (C). A thorough analysis of what might have caused the short-comings of (A) and (B) can be read in Section **??**. Model (C) uses a similar backbone as Model (A) which turned out to be more effective. The complexity of it has been reduced slightly, which is reflected in its architecture seen in Table 5.3.

Moreover, it uses the multi-task learning method of Model (B) to increase the training speed of the convolutional layers. We extend the method by adding a second regression to the top branch of the network. Thus, besides approximating the value and policy function, the network is trained on predicting two additional targets. First, it counts the number of a certain object class in the current frame, just as Model (B) does. Additionally, we train it to detect the number of objects that are currently tracked. This information is easily available at training and testing time. Adding the second regression task has some genuinely interesting implications for the training of the model. Intuitively, the model has to distinguish the tracked objects from the not tracked objects. The two regressions allow us to easily monitor its learning progress in that regard. Moreover, our dataset has only 119410 samples. We expect that the object count prediction will be easy to learn for the convolutional layers. In contrast, the tracked objects in the frames are dynamically changing with the actions of the agent. Thus, the regression should be drastically more difficult to learn for the network. We scale both regression losses by a

Table 5.2: The (B) architecture [16, 106] is a slightly adapted ResNet-18. The only adaptation is the rectangular shape of the feature maps due to the aspect ratio of the input image. The filters are made square in the average pooling layer at the very end with a 1×3 filter kernel.

Name	Kernel	Filter	Stride	Input
conv1	7×7	64	2	$640 \times 360 \times 3$
maxpool1	3×3	64	2	$320 \times 180 \times 64$
conv2_1	3×3	64	2	$160 \times 90 \times 64$
conv2_2	3×3	64	1	$80 \times 45 \times 64$
conv2_3	3×3	64	1	$80 \times 45 \times 64$
conv2_4	3×3	64	1	$80 \times 45 \times 64$
conv3_1	3×3	128	2	$80 \times 45 \times 64$
conv3_2	3×3	128	1	$40 \times 23 \times 128$
conv3_3	3×3	128	1	$40 \times 23 \times 128$
conv3_4	3×3	128	1	$40 \times 23 \times 128$
conv4_1	3×3	256	2	$40 \times 23 \times 128$
conv4_2	3×3	256	1	$20 \times 12 \times 256$
conv4_3	3×3	256	1	$20 \times 12 \times 256$
conv4_4	3×3	256	1	$20 \times 12 \times 256$
conv5_1	3×3	512	2	$20 \times 12 \times 256$
conv5_2	3×3	512	1	$10 \times 6 \times 512$
conv5_3	3×3	512	2	$10 \times 6 \times 512$
conv5_4	3×3	512	1	$5 \times 3 \times 512$
avgpool1	1×3	512	2	$5 \times 3 \times 512$
FC	1×1	flatten	1	$3 \times 3 \times 512$

factor of 0.5, such that the overall loss is comparable to Model (C). The resulting model is shown in Figure 5.3.

5.2 Environment

The training environment is paramount to the success of the agent. Sadly, there is not a lot of research, literature or even best practices for the implementation of such an environment. In many regards the best pointers were to look at the implementation of other popular environments such as OpenAI's gym[1], DeepMind Lab[2] and other open-source environments like ViZDoom[3].

[1] `https://gym.openai.com/`
[2] `https://deepmind.com/blog/open-sourcing-deepmind-lab/`
[3] `http://vizdoom.cs.put.edu.pl/`

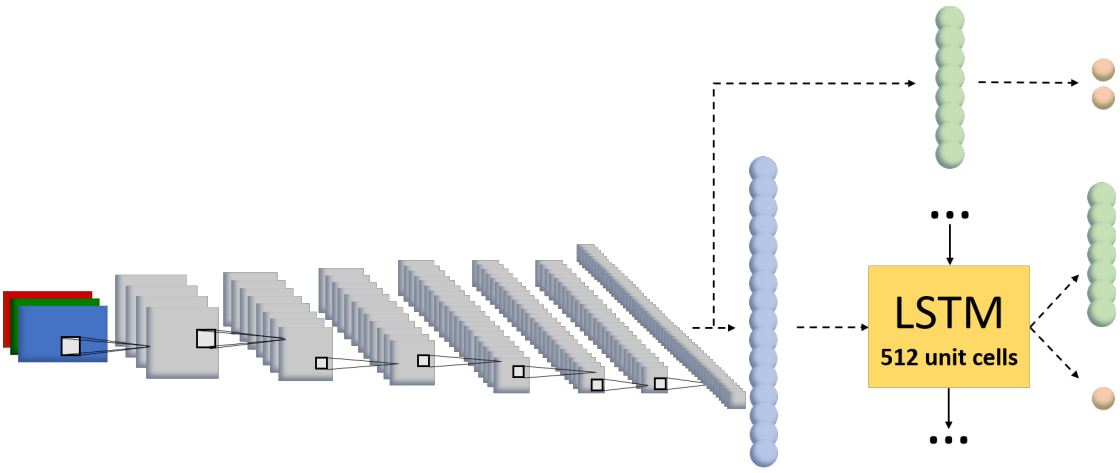


Figure 5.3: Model (C) is the culmination of the lessons we learned during the training of (A) and (B). It consists of a backbone specifically designed for the task and in contrast to (B) it has an additional regression target which is to count tracked objects in the frame.

5.2.1 Dataset

Due to the open-source spirit that the machine learning community has, there is a multitude of openly available, fully labelled datasets. When it comes to the niche of aerial imagery, the options shrink considerably but there are still some solid choices. It follows a shortlist of candidates: Stanford Drone Dataset [108], Mini-drone Video dataset [109], Inria Aerial Image Labeling Dataset [110] or VisDrone [24].

The data that we were looking for was a HD dataset, shot from varying altitudes with a great variance of scenes. The VisDrone Dataset fit the requirements the best, albeit it only being 10 GB. The Stanford Dataset with 70 GB is a great dataset to expand on later. But due to its sheer size a single training epoch takes too much time.

VisDrone is a challenge by Tianjin University. It exclusively contains sequences from UAV, shot from a variety of different heights, spatial resolutions (all HD) and scenes. The scenes are mostly shot in urban areas with 12 different classes of objects. It is fully labeled and the labels are stored in .txt file with the sequence. In crowded shots, the labeling is sadly imprecise for some classes. A set of exemplary frames can be seen in Figure 5.4.

5.2.2 Design

The principle of the design is to mimic the pattern of other environments which have been successfully solved by RL. The dataset is divided into different stages, similar to

Table 5.3: The (C) architecture is comparable to Model (A). However, due to the slow training results of (A) we reduced the filter number of the slightly.

Name	Kernel	Filter	Stride	Input
conv1	5×5	32	4	$640 \times 360 \times 3$
conv2	3×3	64	2	$160 \times 90 \times 32$
conv3	3×3	128	2	$80 \times 45 \times 64$
conv4	3×3	128	2	$40 \times 23 \times 128$
conv5	3×3	256	2	$20 \times 12 \times 128$
conv6	3×3	256	2	$10 \times 6 \times 256$
conv7	3×3	512	2	$5 \times 3 \times 256$
FC8	1×1	flatten	1	$3 \times 2 \times 512$

stages in a game like Super Mario Bros[4]. Characteristic to different stages in Super Mario Bros is that they have a different tile-set and sometimes different enemies. For ScopeNet's environment each stage has its own theme. There is for instance a "highway", "sports" and "river" stage. In total there are 9 different stages, which span 23882 frames. To minimize the correlation between the experiences of different agents, the stage order is randomized after each episode. To present a concise overview of the design choices, this subsection will cover the action space, observation space, reward shaping and data augmentation.

Action Space. The action space defines the actions that the agent can take to complete the task. In our environment, the task is to detect every object of a given class. For instance, the task could be to detect all humans in a video. Keeping the goal of the agent in mind (4.1), there are two primary actions:

- First, zoom in on a subregion of the image, at the cost of resources.
- Second, do nothing, at no cost.

Implementing the latter is straight forward. However, for the zoom-in action, there are several options. Two alluring approaches are detailed below: a hierarchical zoom and a floating zoom inspired by humans. Both approaches can be seen in Figure 5.5.

The simplest solution is to divide the frame into static subregions and attach one action to each subregion. Such a hierarchical approach is easy to implement and easy to extend to more zoom levels. The downside is though, that an inspiration for the agent is how humans view large images. Such a static hierarchical segmentation recedes from human behavior.

A more natural solution, allows the agent to freely move his attention across the the video stream. The zoom area could be either fixed size or adaptable by the agent as well.

[4] `https://pypi.org/project/gym-super-mario-bros/`

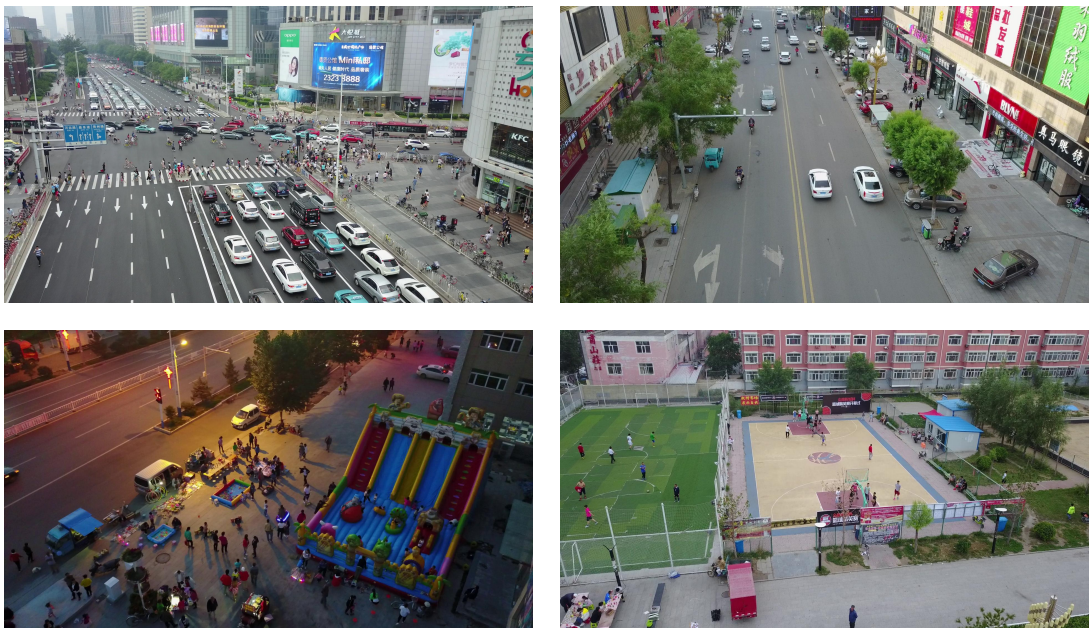


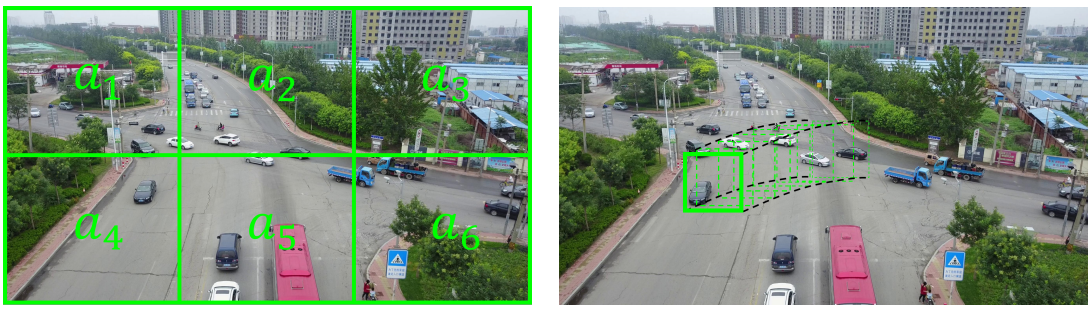
Figure 5.4: Some examples from the VisDrone Dataset [24]. The dataset was designed for a machine learning challenge.

Thus, the actions need to include several continuous actions, like shifting the zoom-window vertically and horizontally by a certain offset and scaling the zoom-window by a scalar factor. Policy gradient methods are able to learn such continuous policies.

The second option is without a doubt the more elegant and probably more effective solution, once the agent is trained. However, finding an optimal policy of a significantly larger, continuous action space will take inestimable more time to train. Thus, for the proof-of-concept we use the most simple solution that meets the requirement of our goal. We choose square subregions, because the receptive field of all SSD "off-the-shelf" object detectors is square and they perform exceedingly poor on rectangular images. Additionally, we settled for 2×3 subregions instead of 3×4 or 4×5. The primary reason for this is again the correlation of the action space complexity to the training time. The final action space used in the implementation is:

- a_0: skip the frame at no cost;
- $a_1, a_2, a_3, a_4, a_5, a_6$: zoom-in on the region according to Figure 5.5 (a).

Observation Space. Based on the observations that the environment emits, the agent chooses its next best action. The true resolution of the video data varies from around 1280×720 (HD) pixels to 3830×2160 (Ultra HD). As stated before, each input



(a) Hierarchical Zoom (b) Floating Zoom

Figure 5.5: Two different variants of defining the action space. Left: a hierarchical approach segments the image into different subregions. The approach is especially simple and easy to extend to multiple levels of zoom hierarchy. Right: a more elegant and human way of inspecting a large image is with a floating attention. The zoom-window can be moved by an offset in horizontal and vertical direction and scaled with a scalar.

frame is downsampled to 640×360, independent of its original frame. This makes our method independent of different camera models.

To find all objects, the agent is required to store a representation of which objects he has already seen in his internal state. Objects that he has already seen are tracked and don't need to be detected another time. Because information of the position of each tracked object is easily available during training and testing, the idea is to give that information to the agent. Although this concept recedes from the mammalian vision, it can speed up learning significantly. To add this information to the observation of the environment, there are two different ways.

We can add an additional, fully-connected input layer to ScopeNet. The input layer is connected the flattened feature layer before the LSTM. Because the amount of objects in the frame can vary drastically, there needs to be an embedding layer that encodes the bounding box positions to a fixed size vector. Embedding layers are an integral part to natural language processing and have been studied excessively [34]. Every bounding box has four scalar parameters x-position, y-position, width and height.

An alternative is to add the information directly to the observation. For instance, by adding a visual clue to every object that is currently being tracked. The advantage of this method is, that there is no need to adapt the architecture of the model or the training data procedure. All that needs to be changed is the environment that emits the observations. Yet, learning the representation of the visual clues might take longer, if it works at all. Artificially changing the observation can conceal important information, thus the method of adding visual clues has to be chosen carefully. In the current implementation of the environment, the visual clues that we add is a 4×4 pixel box in the middle of the

bounding box of the tracked object. There is certainly space for future research to test which approach performs better.

We went with the second approach due to its simplicity and ease of implementation. In conclusion, the environment emits observations of resolution $640 \times 360 \times 3$. Those observations consist of the downsampled camera frame and a visual clue added to every object that is currently tracked (see Figure 5.6).

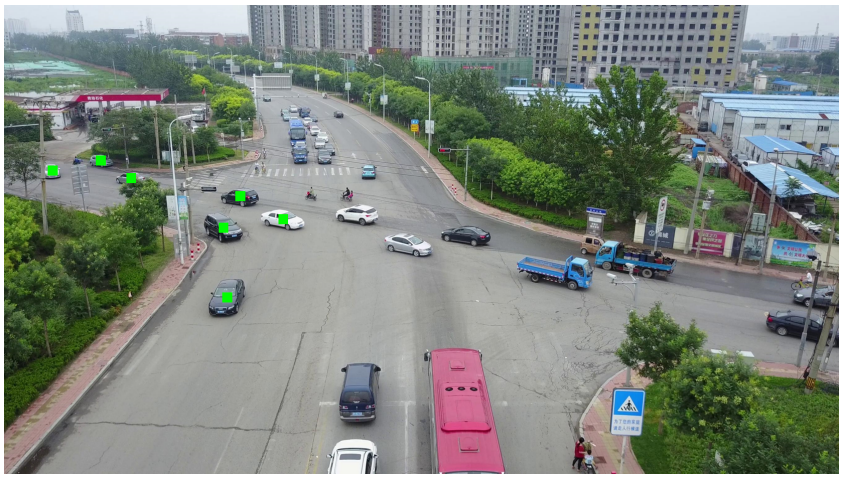


Figure 5.6: The observation space of the environment consists of $640 \times 360 \times 3$ images fed in a stream to the model. To add more information to the observation that is available to us at any time, we add visual clues to objects that are currently tracked. This trick is supposed to help ScopeNet to differentiate tracked of untracked objects.

Reward Shaping is pivotal to the success of the agent. A scalar reward signal is the only guidance that the agent has to evaluate his actions. Reward shaping to increase training efficiency in an open research topic and little information or "best practices" can be found.

We found, that the scale of the rewards has a drastic impact on the loss of the model. We conducted experiments with various different scales, such as scaling all positive and negative rewards. This resulted in the value function quickly dominating the loss function, which led to an unsubstantial entropy term in the loss function (see equation 4.47). The system converged rapidly to a suboptimal action probability distribution.

The principle for designing the reward function is to keep the scaling and payout similar to other environments. Most environments emit rewards in the single digit range. For instance, the gym "Super Mario Bros" environment awards the agent with 1 reward for each distance unit it travels to the right. If the agent dies it gets a -15 reward penalty[5]. For ScopeNet, it is critical to be aware of cost, thus every action besides a_0 costs a fixed

[5] `https://pypi.org/project/gym-super-mario-bros/`

amount of reward. The ViZDoom environment has a similar policy in their death-match environment [6]. In order to keep the agent from blindly shooting non-stop, the agent has a reward penalty for every shot he fires. The ScopeNet environment emits positive rewards whenever the agent zooms on a target that is currently not tracked. Again, this follows the ViZDoom environment, as shooting an enemy awards the agent with a reward.

We decided to employ two different reward functions. Much like the ViZDoom environment, Model (A) and (B) are rewarded in as follows:

- -1 reward penalty for every zoom action $(a_1, ..., a_6)$
- $+2$ reward for zooming-in on every untracked object

Thus, the resulting reward function is

$$r_i = \begin{cases} 0 & \text{if } a_i = a_0 \\ 2k - c & \text{else,} \end{cases} \tag{5.2}$$

where k is the number of untracked objects which were correctly detected and $c = 1$ the cost for each zooming action. With this setup, the agent is awarded a positive reward even if there is only one new object in the zoom-window but has steady cost for all zoom-actions.

For Model (C) we use a slightly more complicated reward function. The design principle for the second reward is to give the agent unmediated feedback after every single action. Sparse rewards have shown to be a problem in reinforcement learning [111]. Although our previous reward function is not "sparse", we still have the opportunity to shape the rewards to stimulate the agent at every step. In the second reward function we reward the agent for skipping a frame correctly. Hence, if all of the objects in the frame are currently tracked and he decides to skip the frame, he is rewarded. Additionally, we penalize him now if he skippes a frame and there are still untracked objects in the observation. In pseudocode the reward can be computed as follows: where t_i is the number of tracked

Algorithmus 2 : Modified reward function

if a_0 **then**
  $r_i \leftarrow 1$ **if** $t_i = n_i$ **else** -1
else
  $r_i \leftarrow d_i^{a_i}$ **if** $d_i^{a_i} \neq 0$ **else** -2
end

objects, n_i is the total number of objects and $d_i^{a_i}$ is the number of correct objects in zoom a_i. This reward function has another valuable property. The maximum of the reward

[6] `http://vizdoom.cs.put.edu.pl/`

function is only met if the agent acts fast. Thus, for the agent to maximize his reward he has to zoom-in on all objects in the video stream as soon as possible. Every frame with undetected objects that passes, the agent "effectively" loses out on reward.

Data Augmentation is a method to artificially increase the size of your dataset by manipulating the data. Moreover, it can increase the performance of your network by adding additional stress on the parameters [112]. Some of the most common augmentation techniques are standard image processing tools, such as flipping, translating and rotating. Recently, Generative Adversarial Networks, a class of generative neural networks, have become very popular to automate the task [34]. For this work, we don't use data augmentation in its technical sense. We don't take samples and manipulate some property of it.

However, what the environment does is the following. As covered before, the environment emits 640×360 pixel observations. The original samples have an average resolution of about 1980×1080 pixels. On average we lose 1843200 pixels per sample. Or put differently, we effectively use about 11.1% of the data. To reduce the data loss we take each sample at full resolution and add 5 different observations to the dataset. Once the original sample and then each quadrant of it. Of course each downsampled to 640×360. This allows us to extend the dataset to 119410 samples. The process in shown in Figure 5.7. This way we can use at most around 55.56% of the dataset, which is only a upper bound because the four cropped observations contain pixels from the the downsampled original.

5.3 Experimental Setup

All experiments were conducted on a desktop PC. The system specifications are:

- CPU: i7-4790 CPU @ 3.6 GHz
- GPU: NVIDIA GeForce GTX 1060 6 GB
- Memory: 16 GB
- Drive: SSD
- OS: Windows 10

The programming language that the tracker, object detector, ScopeNet, environment and multi-agent platform is implemented in is *Python* 3.6.5. The object detector was implemented using the *Tensorflow Object Detection API* [7]. It follows a list of the major dependencies that were used:

- tensorflow-gpu 1.11.0: to build and train all neural networks;
- numpy 1.14.2: standard calculations;

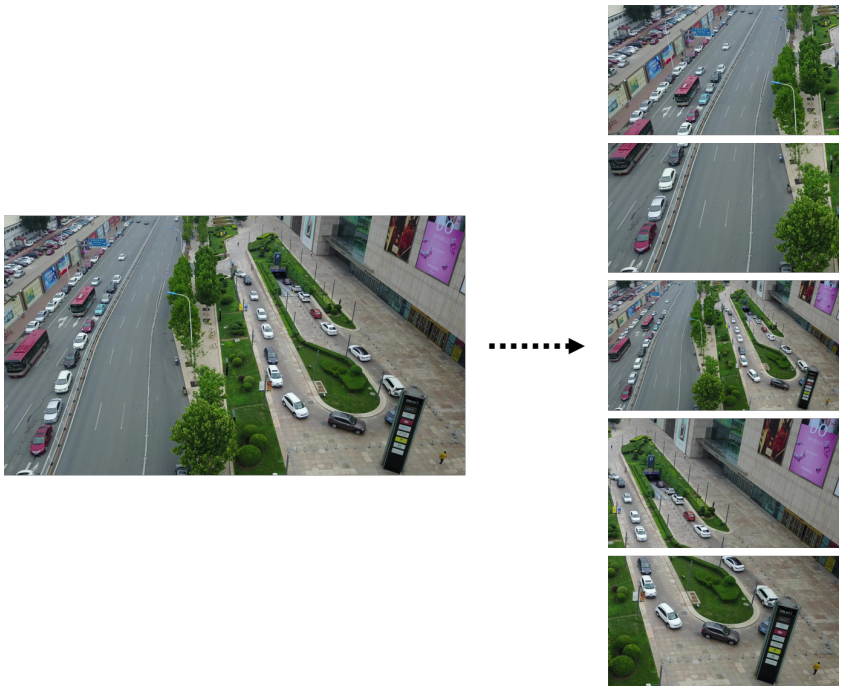


Figure 5.7: To minimize the amount of data we lose during preprocessing the dataset from full resolution to the training size 640×360, we use data augmentation. For each sample at full resolution, we create 5 training samples. One is the original frame downsampled and the other 4 are the four main quadrants.

- opencv-python 3.4.2.17: environment related image manipulation and video capturing of test videos; the CSRDCF tracker is implemented in C++ and accessed via a OpenCV interface;
- gym 0.10.5: a few quality-of-life methods;
- multiprocessing/pickel: parallel multi-agent training;

The project would have been utterly impossible without the open-source spirit of the machine learning community. The main inspirations are as follows:

- `https://github.com/openai/baselines`: OpenAI has a fully fledge A2C implementation published in their git. Their model is the gold standard for most of the other implementations that are found.
- `https://github.com/MG2033/A2C`: a A2C implementation that is based on OpenAI's but without using their libraries;

- `https://github.com/awjuliani`: a A3C implementation that implements a LSTM which helped a lot with our implementation;
- `https://github.com/openai/gym`: OpenAI's gym environments help significantly in setting up the drone environment of ScopeNet;

5.4 Training

Finally, the model and environment is set up. This section outlines the training process of the A2C model.

Frame Skip. Most approaches in RL use a technique called frame-skip to increase training speed [113]. The technique entails that the agent only receives an observation every $k+1$ frames, with k being the number of skipped frames. The agent then picks an action and the action is repeated over all skipped frames. This method can drastically increase the training speed at the cost of network performance. We chose to not use the technique due to the fact that the reason it is used is precisely why we develop ScopeNet. It is used, because in the span of 3 or 5 frames there is little to no spatial variance along the temporal dimension and we want to exploit this very fact with ScopeNet. Thus, we did not use frame-skipping.

LSTM Sequences. Updating the Recurrent Neural Network (RNN) in the model has turned out to be complicated. In non-multi-agent models there is usually a experience buffer [65]. Experiences of the agent get stored in the buffer during the roll-out. After a certain amount of steps, the buffer is randomly sampled to train the network. The method allows the model to train in mini-batches instead of single samples and it decorrelates the data, due to the random sampling. Such a buffer drastically increases the training efficiency and is used in many multi- and non-multi-agent models today. Now, updating a RNN on a minibatch requires an initial hidden state of the RNN. According to *Hausknecht et al.* [104] there are two approaches to this: reset the initial hidden state of the minibatch is or carry over the last minibatch from the previous minibatch. They argue, that both approaches are viable but can converge differently depending on the problem. *Lample et al.* [105] showed, that it is important to mask the gradient of the first few observations when using the first approach.

Both studies use Q-Learning, a single agent model. All open-source multi-agent models use the same approaches, disregarding the different platform. Thus, we decided to use the second approach and carry over the last state from the previous minibatch to the next minibatch and only reset the internal state when the episode is done. There is certainly space for future research, because it may be beneficial to store the internal state of the experience in the minibatch and train the network with different internal states.

Hyperparameters. The original work by *Mnih et al.* [97] uses 16 worker, each unrolling 5 time-steps, before updating the network and a CPU for their multithreading. Our approach diverges here, we used a GPU. Additionally, our input data has a higher

resolution which means it needs more GPU memory to store the same minibatch size. The GeForce 1060 GTX 6 GB was only capable of running 8 environments in parallel with $t_{\max} = 8$ timesteps, which results in a minibatch size of 64. This might have an overall negative influence on the results because there is more temporal correlation in the training data. For all experiments, we used a discount of $\gamma = 0.99$ following *Mnih et al.* [97]. As an optimizer for the SGD, Root Mean Square Propagation (RMSProp) was used with a learning rate of 0.0007, decay factor $\alpha = 0.99$ and $\epsilon = 10^{-5}$. As A2C specific parameters we used entropy coefficient $c_H = 0.01$ and value function coefficient $c_{\mathbf{v}} = 0.5$ (see equation 4.47).

6 Results

This chapter presents the experimental results of this thesis. First we evaluate the training progress of the models in the training environment. One key contribution of this thesis is to evaluate different model architectures and environment parameters. Then, we evaluate the performance of the best model. To test the hypotheses, we first assess the run-time of the model and the pipeline. Finally, we compare the accuracy of the pipeline to other state-of-the-art implementations.

6.1 Training Results

A big drawback of A2C algorithms is their sample efficiency. A study by *Schulman et al.* [114] showed, that A2C agents can take multiple tens of million observations until they learn Atari games. Atari environments can run at a couple hundred frames-per-second (fps), which still adds up to training for a couple of days. After optimizations and minimal logging, the drone environment that we built runs at a maximum of 80 fps on a machine with specifications listed in 5.3. Hence, if we run our algorithm for half the amount frames that [114] proposes, we need about 24 days of non-stop training per model. Unfortunately, in the scope of this master's thesis it is unfeasible to train every model for such a long time. Especially, because there is a multitude of experimental decisions that were involved in creating the environments and there is no best, proven set of parameters.

We trained every model for 360 thousand time steps which amounts to 23 million time steps before we interrupted the training and compared the results. Each epoch takes around 3000 time steps, thus the models each trained for 120 epochs. In the following section, the training progress of each model is detailed before we chose the best performing and evaluate the hypotheses.

6.1.1 Metrics

To evaluate the progress of the model, we log the following metrics:

- action probability distribution of the agent;
- total reward per episode, averaged over all environments;
- loss functions, depending on the model;

- weight and bias distributions of the model.

The expectation for the **action probability distribution** is that the key-frame zoom-actions $[a_1, ..., a_6]$ are used with approximately the same probability. Moreover, we expect the probability for the skip-frame action a_0 to be higher than the rest of the action set. Even more revealing about the progress of the model is the **total reward per episode**. Due to 8 environments running in parallel, we average this value over them. A training agent should steadily increase this value. Another insightful metric is the **loss function** of the model. Each model comprises a different total joint-loss, hence comparing them is challenging. On top of that, Reinforcement Learning (RL) loss function are notoriously noisy which makes their evaluation even more difficult. Nonetheless, the loss can be a powerful metric to determine the progress of the model. At last, the **parameters of the neural network** can be an indication of the learning progress of the model.

6.1.2 (A) Baseline

The baseline implementation (see Figure 5.1) consists of a few convolutional layers, a fully connected layer, a LSTM and the function approximators. To train the model we used the the reward function

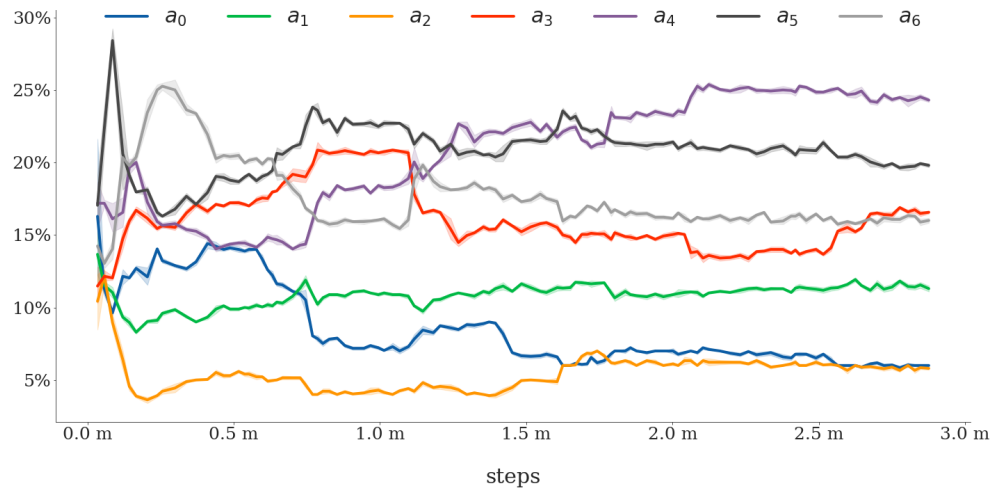
$$r_i = \begin{cases} 0 & \text{if } a_i = a_0 \\ 2k - c & \text{else,} \end{cases} \tag{6.1}$$

where k is the number of untracked objects which were correctly detected and $c = 1$ the cost for each zooming action. Figure 6.1 shows the development of the action distributions over 360 thousand iteration. Each iteration equals 8 unrolled time steps per agent. Even after 2.5 million steps, the action distribution is still changing which is a good indicator that the model is still learning. Nevertheless, action a_0 has only around 7% probability which is drastically lower than what we expected. Apart from this, the action distribution seems to be balanced.

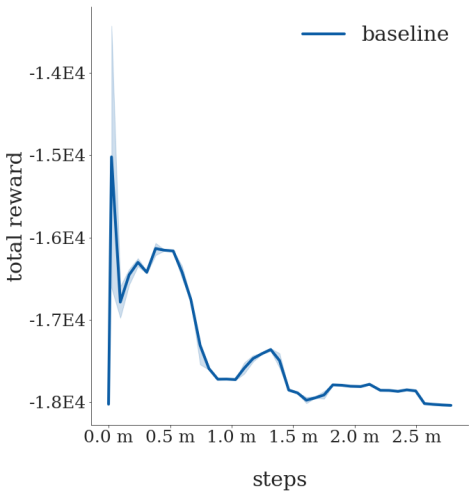
The total loss, averaged over all environments, is probably the metric that is most indicative of whether the agent behaves as intended. Sadly, this metric supports the suspicion that learning did not go as planned. The total reward quickly drops and even after 2.8 million steps it is still declining. Additionally, the loss of the model (Equation 4.47) is shown in Figure 6.1. It is, as expected, very noise and there are no indications that the model is learning. In conclusion, the model did not train successfully.

6.1.3 (B) ResNet-18 Multitask

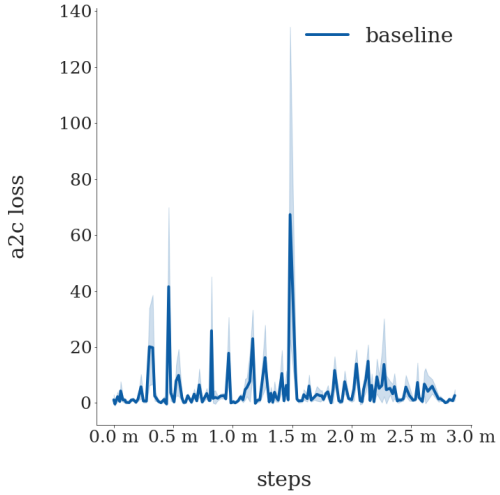
Model (B) is trained on an additional regression loss. It is expected that the parameters of the convolutional layer are trained quicker. This is due to the fact they are trained on counting objects simultaneously to the RL loss. However, ResNet is a more complicated



(a) Action Probability Distribution



(b) Total Reward per Episode

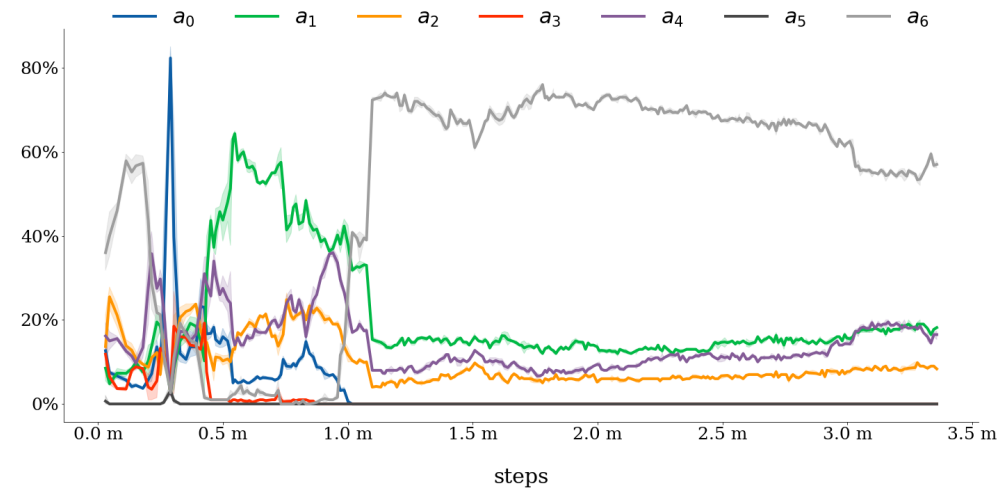


(c) A2C Loss

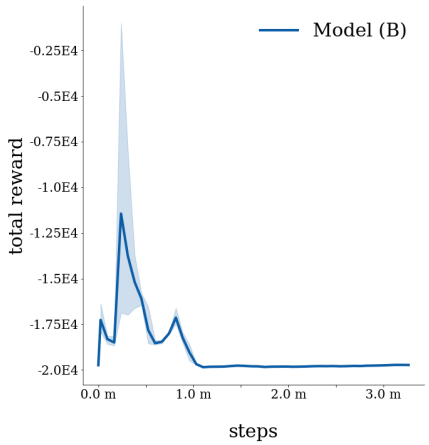
Figure 6.1: (a) The action probability distribution of the baseline model over 3 million steps.(b) The total reward, averaged over all 8 agents. (c) The Action-Critic loss.

backbone than the baseline. Hence, it is challenging to predict the outcome. Model (B) uses the exact same reward function as Model (A).

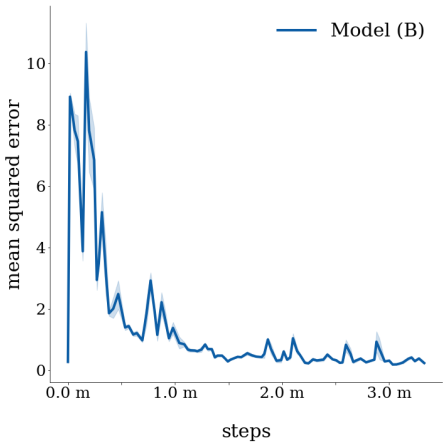
Model (B)'s performance after 3 million steps is even worse than the baseline's. After around 1 million steps, action a_0, a_3 and a_5 sharply drop to zero percent probability, which is the worst case scenario during training. This behaviour can be explored in Figure 6.2. After the sharp decline of the total reward, around 1 million steps, it slightly



(a) Action Probability Distribution



(b) Total Reward per Episode



(c) Object Count Regression Loss

Figure 6.2: (a) The action probability distribution of Model (B). After 1 million steps only 4 actions are actively taken which is a sign for failed training. (b) Shows the total episodic reward of averaged over all agents. (c) As expected, the object count regression mean squared drops quickly.

increases. Nonetheless, the model never recovered, and the action space stays extremely unbalanced. On the bright side, the mean-squared-error of the object count regression decreases steadily and is around 0.5 (over a batch size of 64) after 1.5 million steps and stays low (see Figure 6.2). The A2C loss term without the added object count regression is as noisy as it was in Model (A).

6.1.4 (C) Lessons-Learned

Model (C) is the embodiment of the lessons-learned from the previous two models. Thus, the backbone is designed to match the task. We employ a multi-task regression loss with two regression targets and most importantly, an updated reward function. The action distribution during the training process is visualized in Figure 6.3. Finally, the course of the action distribution is as expected. At first, all actions are sampled at about the same percentage. After 500 thousand time steps a_0 starts to dominate the distribution and rises until roughly 75% to 80%. All other actions have roughly the same probability, around $3\% - 4\%$. If the dataset is balanced enough, every action should be required to find all objects.

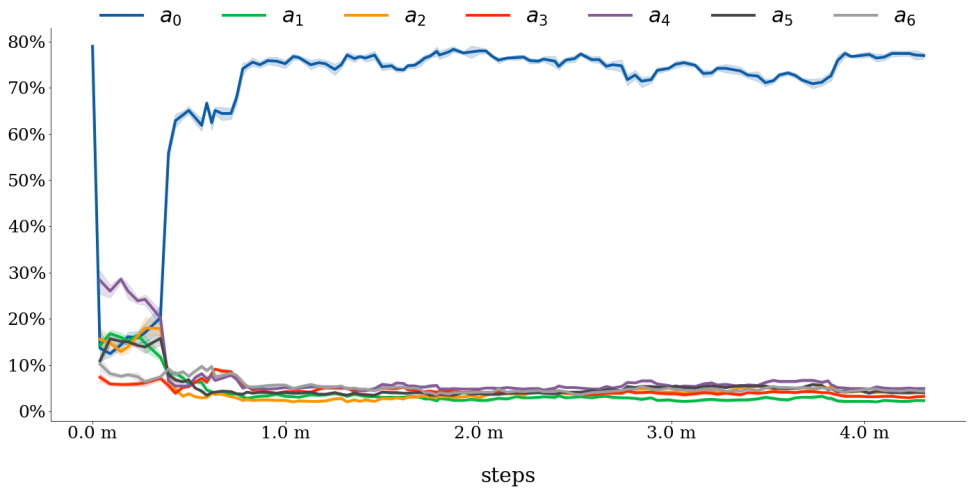
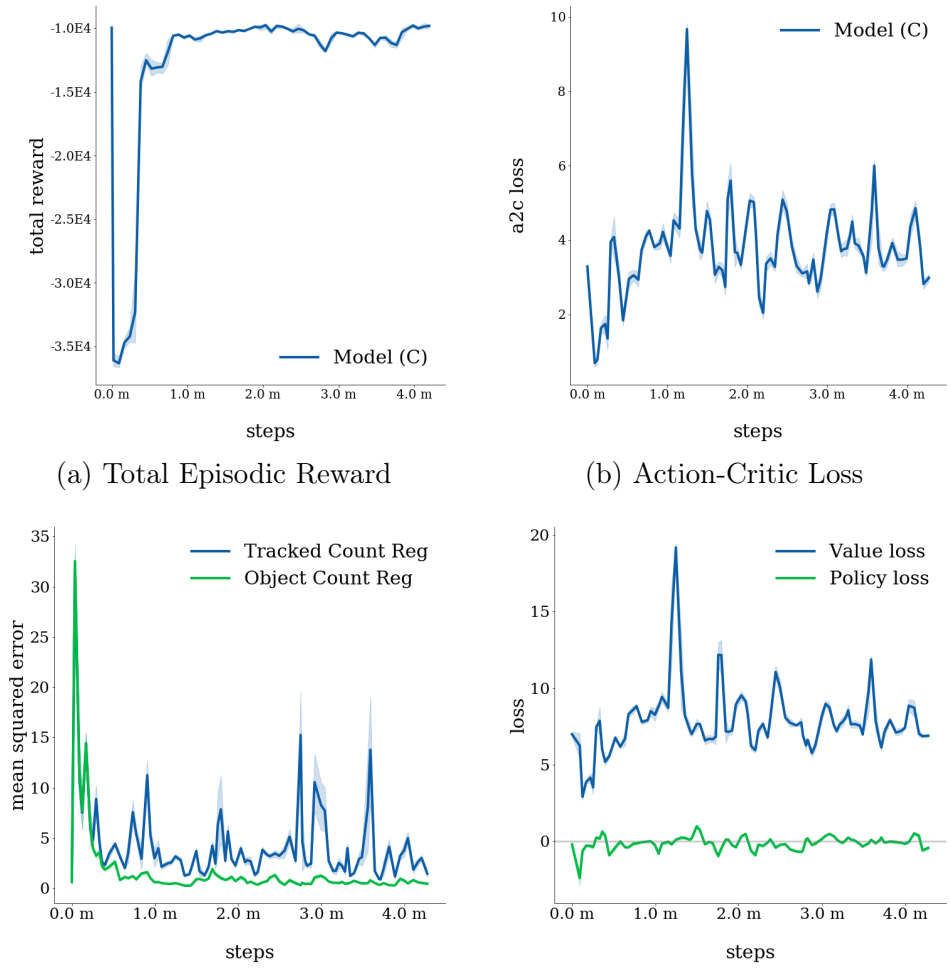


Figure 6.3: Action probability distribution of Model (C). The model converges to a desired state with a_0 being the dominant action. This means, that the model skips around 80% of the frames and that we learned a key-frame scheduling policy.

After 4 million time steps, which took roughly 6 days, the weights of the layers are still steadily changing. This suggests that the training at this point is not over and the model is still learning. For future reference, the Tensorboard layer distributions are published in the appendix 1 - 5.

Figure 6.4 shows the regression graphs of the network. As expected, the object count mean squared error is learned relatively fast. For every training epoch, the objects in each frame remain the same. However, as theorized before, the mean squared error of the tracked object prediction is more volatile. This is due to the fact, that for no epoch the tracked targets remain the same and the layers really have to learn what a tracked object is.

At last we can inspect the total reward of the agent. Additionally, Figure 6.4 shows



(a) Total Episodic Reward

(b) Action-Critic Loss

(c) MSE of Regression Targets

(d) Policy and Value Loss

Figure 6.4: A compilation of training graphs. (a) Shows the total episodic reward of the agent. (b) total loss of the agent consisting of the value, policy and regularization loss as seen in Equation 4.47. (c) Shows both regression mean squared errors. (c) At last, the policy and value function loss.

the loss of the Actor-Critic, the policy gradient loss and the value function loss. The reward seems to saturated after around 2 million steps. This does not mean that the agent can't improve anymore. It is conceivable that he first has to gain the knowledge to overcome a plateau, similar cases exist [100]. At first, the Synchronous Advantage Actor-Critic (A2C) loss ascends and keeps its value for the next 3 million steps. A reason for

this behavior is most likely the multi-task loss. The regression targets are higher at the beginning than the A2C loss. Thus, the back-propagation might prioritize the regression targets. After 4 million steps, at the very end of the training it appears that the A2C loss is finally descending, which could be another hint that the agent may improve with more training. The policy loss is extremely noisy, but it seems that the spikes of the loss reduce in value over time steps.

The training of the model can be considered a success. Therefore, we chose Model (C) to perform the subsequent evaluations of ScopeNet and the entire pipeline.

6.2 Quantitative Evaluation

Finally, we evaluate the performance of our pipeline. To test the hypothesis, we have to investigate two properties. First, we have to determine if the pipeline is able to operate in real-time. This was a critical requirement for the implementation. Then, we compare the speed of our implementation to available baseline object detection implementations. Second, for every object detection algorithm the detection performance is the most important metric. Hence, we compare our implementation to other object detectors.

6.2.1 Latency

The latency is the interval which is needed to compute the detection results for one frame. Three different components in the pipeline are relevant for this metric: ScopeNet, the detector and the tracker. As introduced in Chapter 3, we use the tracker CSRDCF and detector SSD MobileNetV2. Sadly, it was not possible to test the hypothesis on a competitive embedded environment. While the setup described in Section 5.3 is better than most available systems today, non of the video object detection systems covered in the related work section would run on it in real-time. Thus, the evaluation can provide a good intuition on its performance. Table 6.1 shows the latency of each component. It is apparent, that the limiting factors are the detector and tracker. ScopeNet is extremely fast with only around 5 ms inference time.

Table 6.1: Latency of the components in the pipeline.

Name	Latencyt
SSD MobileNetV2	$\sim$ 80 ms
CSRDCF	$\sim$ 20 ms
ScopeNet	$\sim$ 4.7 ms

In the current simulation, ScopeNet skips around 80% of the frames and predicts that only 20% are key-frames. To keep track of the objects, we have to employ the tracker

on every frame. Additionally, we have to apply ScopeNet to every frame, because he controls the information flow in the pipeline. Now we can compute the average latency of the pipeline:

$$t_{\text{avg}} = t_{\text{scope}} + t_{\text{track}} + 0.2t_{\text{detect}} \approx 41 \text{ ms}. \tag{6.2}$$

This adds up to a solid 24.4 frames-per-second. Tabel 6.2 compares the latency to other implementations. All comparison models are implemented with the Tensorflow Object Detection API and run on the same hardware as our pipeline. As stated before, these implementations are **not** optimized for video object detection. To our knowledge, there are no easily available video object detectors.

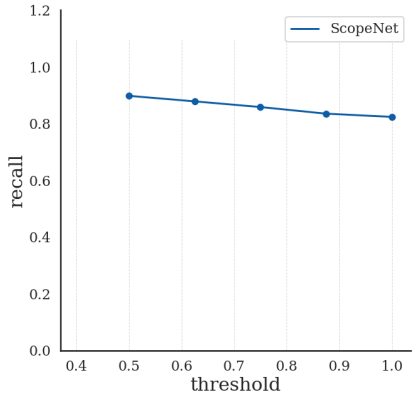
Table 6.2: Frame rate comparison to available object detectors from the Tensorflow Object Detection API. All test were conducted on the same hardware.

Name	fps
Our approach	24.4 fps
SSD MobileNetV2	12 fps
Faster R-CNN Inception v2	7 fps
Faster R-CNN ResNet-101	4.5 fps

We have already shown, that the bottleneck in the pipeline are the tracker and the detector. The difference in latency becomes even worse with the number of objects included in the frame. ScopeNet's latency is independent of the amount of detection targets in the observation. In contrast, the latency of the object detector increases due to the bounding box regression that has to be calculated. Although it increases by a noticeable amount, the backbone which is responsible for the majority of the latency always takes the same amount of time to compute the feature maps. Hence, the multi-target latency increase of the detector is manageable. Sadly, this does not hold for the tracker. As of now, the latency for multi-target tracking increases linear with the number of objects. This hinders the pipeline and emphasizes the fact that the tracker is the limiting factor in our implementation.

6.2.2 Detection Performance

For any detector or detection pipeline, the detection performance is the most important metric. To evaluate the performance of ScopeNet and the Visual Attention and Detection pipeline, we conduct two tests. At first, we evaluate how good the performance of ScopeNet is on its own, without a detector and tracker. Then, for part two of the performance evaluation, we test the performance of the entire pipeline and compare it to baseline object detector implementations. All tests will be conducted with the validation set of the VisDrone dataset and we use the same object classes that ScopeNet



Threshold	Avg. Recall
0.5	0.878
0.625	0.868
0.75	0.858
0.875	0.835
1.0	0.824

Figure 6.5 & **Table** 6.3: Left: A plot of average recall values at different detection thresholds. A threshold of 1 means that the object has to be fully contained in the zoomed region to get detected. Right: Tabular representation of the recall data.

was trained on.

ScopeNet. In this experiment, we want to evaluate the performance of ScopeNet without a detector or tracker. The task of the model is to empower the object detector in the pipeline and zoom-in on all objects as fast as possible. To determine the performance, we need a metric that reflects that task. The most suitable metric for this specific evaluation is the recall. It is a metric that evaluates the ability of a model to find all the relevant cases, in our case ground truth bounding boxes. The definition is:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.3}$$

where TP is a true positive and FN is a false negative. In the case of object detection, a TP is a correct detection and FN is a ground truth not yet detected. For the experiment, we assume that we have an infinitely good detector and tracker. Intuitively, the recall is the percentage of detected objects in the frame. We average the recall over all frames and over all validation videos. Additionally, we vary the intersection threshold required to detect the object from 0.5 to 1. To remove noise, we conduct the experiment 10 times and take the mean. The data is seen in Figure 6.5.

Even if we only assume an object as detected if they fully intersect, the average recall remains above 80%. We conclude that ScopeNet catches around 85% of object in the video stream of the validation set. It should be stated again, that ScopeNet only catches 85% of the objects but only processes key-frames at around 20% of the total frames.

Visual Attention and Detection Pipeline. The most relevant metric to evaluate detection performance is the mean Average Precision (mAP). It is universally used to evaluate object detection models. To calculate the value, you average the maximum precision at varying recall values over all classes. ScopeNet is class specific, thus there is only one class and we will only use AP. The implementation thas is used to calculate the AP is taken from MS COCO[1]. The precision is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.4}$$

where TP is a true positive detection and FP a false positive detection. FP can be considered a wrong detection.

ScopeNet is designed to enhance the performance of the detector in the pipeline. Thus, the experiment is as follows: first, we test the most recent and best available object detectors on the validation set without ScopeNet. Then, we repeat the same test with the same object detectors, but this time they are plugged into the pipeline. For every object detector we compute two AP values, once with ScopeNet and once without and plot the Precision-Recall Curve. One import concept for the evaluation is the Intersection over Union (IoU). An illustration of the IoU of an example can be seen in Figure 6.6. It

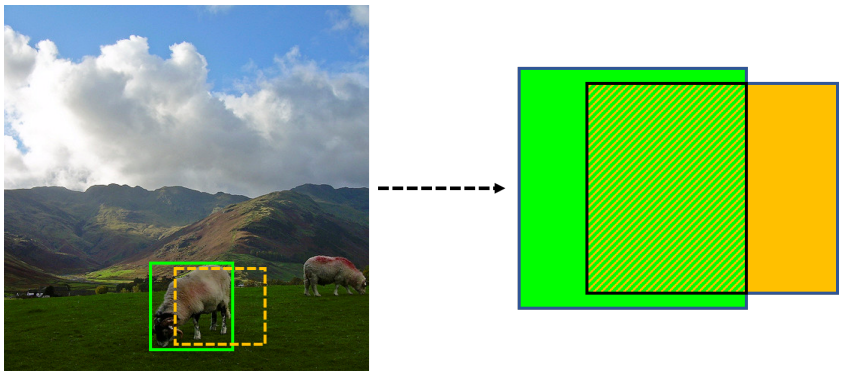


Figure 6.6: The Intersection over Union (IoU) is a common metric to determine the quality of a bounding box prediction.

measures the overlap of two boxes and can be interpreted as the quality of a detection and its ground truth. The value is then computed, such that

$$\text{IoU} = \frac{\text{area of intersection}}{\text{area of union}}. \tag{6.5}$$

To gather data for the computation, the process is as follows:

[1] `https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py`

- No Scope: We apply the detector to the entire validation set and store the detection results and corresponding confidence scores in a dictionary. The ground truths are continuously stored as well. Finally, we calculate the AP over all predictions and ground truths.
- Scope: The OpenCV implementation of the tracker is broken. First, it does not return confidence values, thus we attach the detection confidence to the object. If the object is redetected with a higher confidence, we update the value. Second, there is a boolean value attached to the tracker object which returns if the tracker lost the target. This mechanism currently does not work, thus the tracker is unusable for us. Therefore, once we detect an object with a corresponding confidence score, we propagate the bounding box with the ground truth values, if the IoU is above 0.5. Then, as in the No Scope case, we collect the predictions of scores, boxes and ground truths and calculate the AP.

What the mAP variation of MS COCO normally does, is compute the mean not just over several classes but several IoU thresholds as well. For our implementation this would drastically skew the results, because we use the ground truth trajectory to propagate the boxes, which obviously has a high IoU. Other Video Object Detection challenges [22] compute the mAP with a IoU threshold of 0.5. Hence, we only compute the AP with $\text{IoU} > 0.5$.

The results of the experiment are summarized in Table 6.4. Using the zoom pipeline versus a basic implementation increases the average precision of the detector on average by 237% with a key-frame ratio of $20\% - 25\%$. As expected, especially single-shot models benefit from zoom. These models are known to struggle with large images and on average their AP increases by a stunning 430%. RetinaNet, which is a novel SSD ResNet50 implementation [59], does very well on the validation set for a single-shot detector. Faster R-CNN implementations already do fairly well in their own respect. It should be noted, that these implementations on their own only compute a few frames per second.

Finally, all comparisons are plotted in Figure 6.7. The figures show the precision-recall curve which is generated by taking all predictions and steadily varying the score threshold of detection results. Similarly to the Table 6.4, we observe that SSD implementations benefit greatly from the pipeline. None of the implementations, with or without scope, come close to a recall of 1 because many objects in the ground truth dataset are heavily occluded.

Table 6.4: Comparison of state-of-the-art object detectors on their own and integrated into the pipeline. The Visual Attention and Detection pipeline increases the effectiveness of these object detectors on average by 237% while operating at a key-frame ratio of only $20\% - 25\%$.

Name	No Scope	Scope	Increase
SSD MobileNet	12.17 AP	66.95 AP	446%
SSD MobileNetV2	9.09 AP	35.94 AP	295%
SSDLite MobileNetV2	6.62 AP	44.62 AP	574%
Faster R-CNN InceptionV2	16.73 AP	53.29 AP	219%
Faster R-CNN ResNet101	24.84 AP	53.15 AP	114%
RetinaNet	24.12 AP	61.98 AP	157%
Average	15.6 AP	52.66 AP	237%

6.3 Hypotheses

There were two separate hypotheses that we proposed: **(a)** a agent can learn a policy to sequentially analyze our data and be cost aware; **(b)** the agent, embedded in the pipeline, outperforms baseline algorithms in terms of speed, computations and accuracy.

Hypothesis **(a)** holds true. An agent can be trained to learn a policy to sequentially analyze high-resolution aerial data. Although the policy that we learned is most certainly not optimal, a proof of concept was successful. The truth of hypothesis **(b)** was experimentally partially validated in Subsection 6.2.2. We showed that the pipeline increases the accuracy of all object detectors we tested. It only uses the object detector $\sim 25\%$ of the time which automatically validates that our pipeline uses less computations. At last, we could not validate the part of the hypothesis that the agent outperforms baseline implementations in terms of speed. It does in most cases, however, in the scenario of multi-target tracking where the number of tracked targets reaches a certain point, a baseline object detector might be faster.

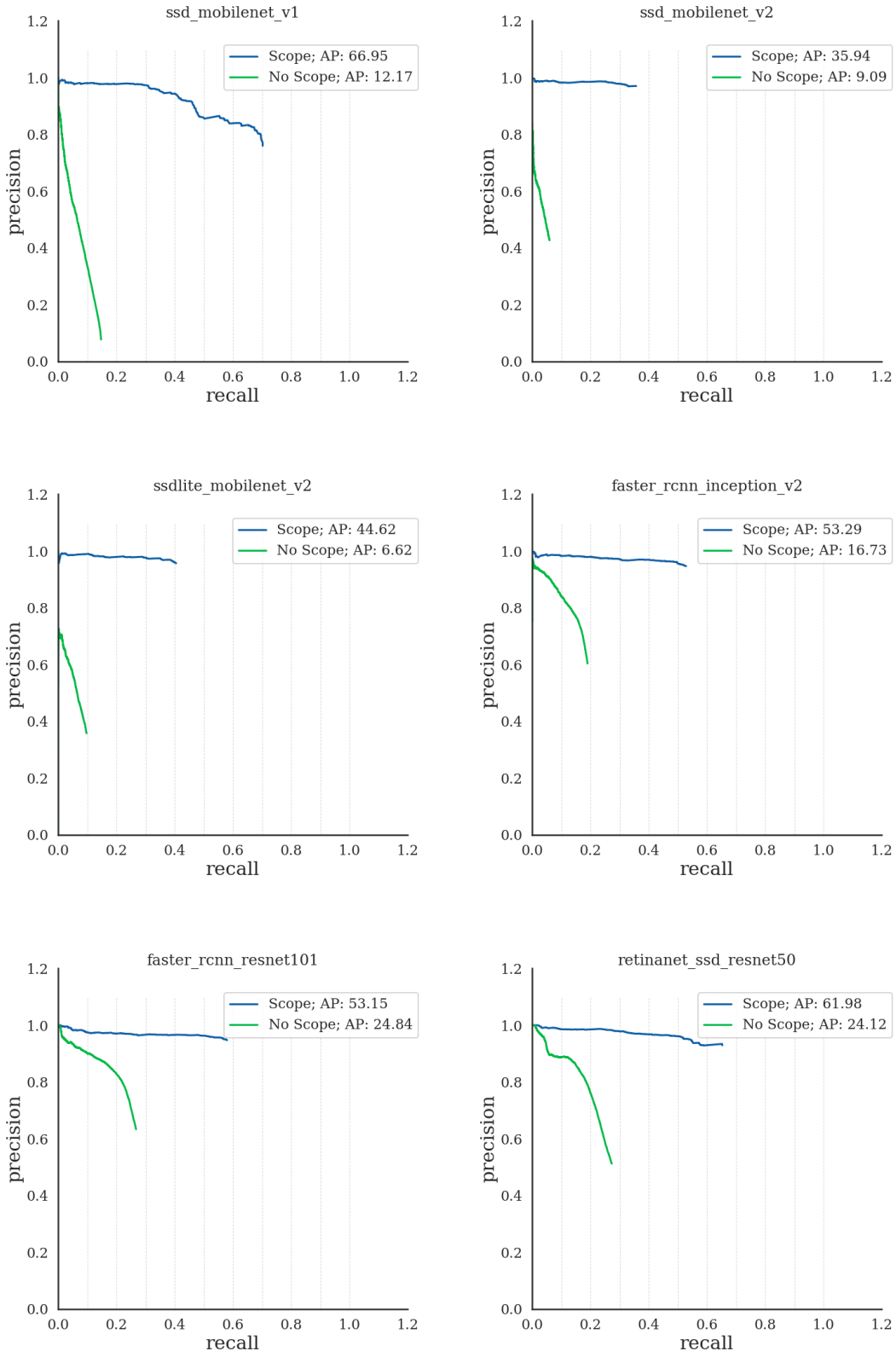


Figure 6.7: These plots show the Precision-Recall Curve of object detectors. Every plot compares an implementation scoped versus unscoped. It is easy to see that the pipeline increases the performance of every single implementation significantly.

7 Discussion

7.1 The Failure of Model (A) and (B)

One of the inherent problems of neural networks is that their behavior is often a black box. This makes them extremely easy to implement. On the other hand, they are very challenging to debug, if their behaviour is not according to your expectations. Nonetheless, there are some interesting takeaways from the training experiments.

Let us first consider Model (B). Although the model did not train successfully, multi-task learning is a powerful regularization tool for deep reinforcement learning and should be used if the training data allows it. The additional regression loss in the total loss function of the model was probably not the reason why the model did not train. The most likely reason, why the model was a failure is the convolutional backbone (see Table 5.2). The suspicion is that the average pooling eradicates too much information. To reduce the size of the last feature map the pooling layer transforms the $5 \times 3 \times 512$ feature map to a $3 \times 3 \times 512$ map. The average pooling uses a 1×3 filter kernel which averages over rows of three. If the actual position in the rectangular last feature map is important for the function approximators, then the average pooling might be fatal. Nevertheless, this is just a hypothesis and future research on optimizing CNN topology for reinforcement learning is required to evaluate it.

On the other hand, Model (A) should have worked. The CNN backbone might have had too many parameters and there is certainly room to optimize the model. However, at least it should have been able to learn some better policy for the training data. When analyzing the TensorBoard layer weights of the convolutional layer, it is apparent, that the network learned very slowly. Many weights remained in their initialization distribution for the entirety of the learning period. Hence, one hypothesis is that the network was simply too deep for the very volatile, fluctuating loss function of A2C.

Another possibility, is that the training environment is not set up correctly, such that the agent does not have the chance to train properly. The success of Model (C) suggests that this is the case. Most of the setup is based on "human" intuition, which might of course not apply to the agents. Both models (A) and (B) are trained in the same environment, thus the reward has an effect on both. The hypothesis is that the reward function

$$r_i = \begin{cases} 0 & \text{if } a_i = a_0 \\ 2k - c & \text{else,} \end{cases} \tag{7.1}$$

is ineffective and flawed. For a human, it seems intuitive to take action a_0 with reward zero, if the alternative is a high risk of -1 reward for a failed zoom-action. The only reason why this is intuitive, is because we have prior knowledge about the problem. An agent does not have this prior knowledge and as long as the reward function does not reflect the positive feedback, he can't learn when it is the right time to skip frames. In short, there is no reward for picking a_0 correctly. **To teach the agent to pick a_0 effectively, there needs to be an unmediated incentive.** In retrospect, this fact seems blatantly obvious. This new insight, led to the formulation of the second reward function. The new loss allowed the agent to learn the intended policy and the model finally displayed the expected behaviour.

The error revealed a problem that was already felt while building the environment. There is little literature on reward shaping and environment design as a whole.

7.2 Shortcoming of the Framework

It is easy to see that the limiting factors of the pipeline are the tracker and the detector. Object detectors have come a long way in the last couple of years and their detection performance is already great. The detector was expected to be the biggest bottleneck in the pipeline, but it turns out that the tracker is more problematic.

7.2.1 Latency

In terms of the latency, there are a few aspects to consider. If there is only one object in the frame, the pipeline is twice as fast as the second fastest implementation (SSD MobileNetV2). This is obviously no realistic assumption. In most realistic use-cases, the goal is to keep track of multiple objects at once. Additionally, the experiment was conducted on the hardware described in Section 5.3 which is most likely more powerful than any embedded system available today. The processing power of embedded systems will improve over the next couple of years, but to get a valid latency evaluation, there is no way around using an actual embedded board.

7.2.2 Detector

The ease with which one can implement powerful object detectors today, is truly magnificent. The entire pipeline builds around catering to the object detector and compensating for its weaknesses. Object detection is such an important problem for so many applications that the technology will continue progressing and those weakness will diminish with time. Nevertheless, applying detectors to high-resolution images is not a focus of the research community right now. This fact is troubling, considering the steady increase in image and video resolution of modern devices.

7.2.3 Tracking

The tracker turned out to be the biggest, albeit unexpected, problem in the pipeline. The tracking community primarily uses MATLAB and python implementations are rare. The easiest way right now to implement a select number of trackers is via an OpenCV interface which directly implements C++ files via Cython. As stated in Section 6.2.2, the OpenCV implementation of the trackers is barely usable right now due to a bug. The problem already existed in May and a change request has been submitted. Although the bug will be resolved eventually, a huge problem with current trackers remains. In most real scenarios, tracking multiple objects is difficult due to the linear growth in computational cost. The margins today for real-time operation are too small to afford spending 10 times the computational cost to track 10 targets. CNN based trackers will be a remedy to this problem once this technology is progressed enough. However, as of now, real-time multi-target object tracking is an open and challenging research topic.

7.2.4 Composition

The structure of the current pipeline does not leave much room for change. As stated numerous times before, the output of the tracker is used to give ScopeNet the information where tracked objects are. To what extend this is beneficial is unknown, but it is unlikely to have a negative effect. Since the information is easily available at run-time and costs no extra resources, there is no reason not to use it. When a real tracker is employed, a realistic scenario is that it can lose targets. Adding information of the currently tracked targets can help ScopeNet to redetect it. The current implementation, which adds the information directly to the input frame, might not be the best way. An extra embedding layer is probably more efficient.

Another unobtrusive component that has to be improved, is the downsampling operation. Right now, the problem is split in a very human way and it has the advantage that it is possible to view and interpret the data flow at every point. However, training the downsampling operation together with ScopeNet, such that the important information is retained after the operation, could be very beneficial to the performance.

7.3 ScopeNet: Strengths and Weaknesses

ScopeNet is, to our best knowledge, the first approach to merge deep learning based object detectors with high-resolution video analysis. The goal of the thesis was a proof of concept which was a success. The idea of sequential video analysis in combination with deep reinforcement learning is powerful and will have a future. Nonetheless, it is in its infancy and there is an incomprehensible amount of work to do, before such a technology can be used in practice.

Generally, the model is still very simple. It consists only of four different parts: a backbone, the LSTM, the multi-task layers and the function approximators. The training period was short in comparison to many other A2C implementation. This begs the question, how much more can the performance improve? Although this implementation will probably not achieve an "optimal" policy, it showed how much potential there is. An optimal policy would most certainly bridge the research gap and bring autonomous drones into reach. Therefore achieving such an optimal policy should be a primary focus in future efforts. Due to its simplicity, the model is extremely fast with only 5 ms inference latency. Therefore, complex extensions of the model are feasible. In this case, multi-task training turned out to be very beneficial and, if the dataset allows it, is a great addition to the model.

The most promising result of this work is the performance of the pipeline. ScopeNet's recall on the validation set is high with $\sim 85\%$. This means that given a very good object detector, it zooms-in on almost every object at only 5 ms latency per observation. The second experiment that was conducted with real object detectors further validates the initial claim. One main hypothesis of the thesis was that, given high-resolution imagery, zooming can greatly boost the performance of object detectors. An average AP increase of 237% leaves no doubt that this is the case. Especially encouraging is the performance increase of SSD implementations. SSD implementations are more inclined to meet the real-time requirements of the embedded system.

However, there are many things in the current model which can be improved. As with all deep learning approaches, the method is only as good as the data. And although the dataset is decent, there is a lot of room for improvement. The labeling is not precise enough to detect smaller objects like humans. Additionally, there is not enough variance in the scenes. Most are very crowded suburban sceneries. Adding another dataset, like the Stanford Drone dataset, will certainly be beneficial. Moreover, the environment doesn't technically use data augmentation which would further improve the quality of the model. Generally, many uncertainties remain when building reinforcement learning models and environments.

The A2C model may not be the best approach for the problem at hand. It lacks an experience replay buffer which greatly benefits models that need to use states from far past observations to build current sates. The buffer would empower the LSTM which could greatly boost the performance of the agent. As explained in Section 5.4, there is no general consensus what the best implementation is. Testing both methods will provide information on which works best for models that need a high number of previous states to build the current one. Previous RNN studies [115, 116, 117] have proven how powerful this method can be, therefore the current LSTM implementation is the biggest uncertainty in the model as it seems to have under-performed in our architecture.

So far, the model is only trained on a few simple classes (cars, busses, trucks and vans). How good the model works for a larger number of classes is unknown. Most realistic use-cases would require the visual system not just to detect objects but to infer high-level

situational awareness. How powerful reinforcement learning is for such scenarios needs to be evaluated.

7.4 Video Object Detection

It is unfortunate that it was impossible to compare our model to actual video object detectors. All the comparisons conducted were against image object detectors. However, the expectation is that video object detectors do not particularly work well with high-resolution aerial video. Current studies on video object detectors follow the trend of small square input resolution. This will certainly change in the future and provide more information on how viable the solution is. Without a doubt, the interpolation method that this thesis uses is only a temporary solution. In the long run, it is most desirable to implement the model end-to-end. Although at this point, a end-to-end architecture with an integrated Visual Attention Network (VAN) is still a long way off. Developing a multi-class VAN would be a great addition to the deep learning tool kit. Similar to the Region Proposal Network (RPN) of R-CNN, a VAN could be a network that can be added to models to enhance their performance on medium- to high-resolution data.

7.5 Aerial Video Analysis

The field of Aerial Video Analysis is pivotal to the success of autonomous drones. Of all autonomous vehicles, UAVs have one of the most restricted environments due to the fact that the weight of the payload and its energy consumption are exceedingly critical to flying robots. For most of this thesis it was assumed that the camera sensor is used as the primary means to collect data and understand the environment. For a multitude of applications this is the case, but there are other use-cases where a simpler processing is sufficient. For instance, the logistic use-case described in the introductory chapter probably uses a minimal amount of computer vision. Other sensors like GPS, LIDAR, accelerometers and altitude sensors are presumably sufficient to fly from position (A) to (B) and deliver a package in a predefined location. Nevertheless, for most sophisticated, multi-dimensional operations, like search operations after a disaster, the camera sensor will play the central role.

A UAV video dataset was used for our training, however no experiments of the validity of the model on real footage were conducted. Therefore the next plausible step will be to decide on a simple use-case, retrain the model for the task, gather data with a drone and evaluate the performance.

8 Conclusion

This thesis presents a novel deep reinforcement learning based approach to high-resolution aerial video analysis. While previous video object detection algorithms based their key-frame scheduling on either fixed intervals or simple heuristics, we learn an adaptive scheduling policy directly from raw data. To tackle high-resolution data, previous work simply downsampled the input and computed detections of the reduced representation. Our approach deals more sensibly with the data by sequentially detecting objects in patches of the video stream at full resolution. We merge both proposals into one, unified architecture: ScopeNet.

8.1 Summary

The problem of high-resolution video object detection is twofold: First, it is unfeasible to naively apply an object detector to every frame in the video because they have a high latency which makes real-time computation impossible. Moreover, they are designed for image data which makes them susceptible to video artifacts. Second, modern object detectors are developed for low resolution data. In practice, high-resolution data gets downsampled before applying the detector. This approach does not retain enough information in the data and has fatal consequences for the detection accuracy. Upscaling the detector to be applicable to large images is unfeasible due to the exponential growth of computational cost and memory restriction on modern GPUs.

Our hypothesis was that sequential analysis can be used to solve the latter problem. By applying an object detector to subregions of the image, we circumvent downsampling the image and can detect objects at full resolution. Moreover, we argued that the first problem can be solved with a detection-tracking pipeline in which we exploit temporal inter-frame redundancies. This approach entails to only apply the detector on sparse key-frames and propagate the results to adjacent frames with a high-speed tracker. ScopeNet unifies these two approaches, it is the control center of the pipeline and has two jobs: (a) deciding which subregion to zoom-in on and (b) determine the key-frame scheduling policy. To realize the required behaviour, we use a recent deep reinforcement learning agent which learns the best policy to achieve the goal from raw data. The agent model is a Temporal-Difference Synchronous Advantage Actor-Critic which learns in 8 environments simultaneously. The environment in which the agent learns, is built on a drone dataset consisting of real-world drone footage.

To validate our hypothesis, it was required to investigate two properties: the detection performance and latency. Our detection performance experiment entails to compare the performance of state-of-the-art detectors once embedded in our pipeline and once on their own. Our results show, that on average the pipeline increases the average precision of an object detector by 237% while only processing around 25% of the frames. The result leaves little doubt that zooming-in can greatly increase the detection performance of current object detectors. Evaluating the latency was critical to the real-time requirement of the system. Our experiment shows that currently the object tracker is the bottleneck, preventing the pipeline from running in real-time in a multi-target scenario. Nonetheless, when tracking only one object our approach is able to operate in real time at 24.4 fps, twice as fast as the second fastest at 12 fps.

8.2 Future Research Directions

Using deep reinforcement learning for video analysis is a novel research field. There are many factors that can have dramatic impact on the results and have yet to be researched. In the future, we aim to investigate the influence of the network topology on the problem. In essence, most backbone networks in current implementations are essentially identical, ignoring the complexity of the problem. Optimizing neural network topology to fit the problem could have great impact on the models behavior. Moreover, deep reinforcement learning models barely use any of the recent technological emergences of computer vision.

To improve the model, we have to exploit the recurrent neural network more efficiently with a different architecture. The multi-agent learning process possibly impaired the models ability to keep states from distant experiences. A simpler model like the Deep-Q Learning with an experience replay might be more suitable for the task. Additionally, the action space was by design very simple. A continuous action space, following the human visual system, is an exciting direction to go forward.

In general, there seems to be little research done on applying new techniques from deep reinforcement learning to real-world problems. Most of the seminal work in general intelligence in recent years focuses exclusively on simulation. Extending those successes beyond simulations would be exceedingly important for real-world applications.

Another direction which could be immensely beneficial to the reinforcement learning community, is fundamental research in environment design. In this thesis, we experienced first hand how important reward shaping, action and observation space of an environment is. Those decisions are mostly experimental at this point and design guidelines and practices can make all the difference.

Generative Adversarial Networks are a class of neural networks which recently have been successfully employed to augment datasets to improve the performance of deep learning models. To what extend such methodology can be beneficial to deep reinforce-

ment learning has yet to be explored. However, if successful, it has the ability to generate more training data and subsequently make the agent more robust.

Outside of reinforcement learning, drone technology has to experience steady improvements and innovations to support autonomy. The visual system has to be integrated in a behavioral system. The *Robot Operating System* (ROS) team has recently published a free UAV simulation in which you can operate a drone. It allows you to implement code on a drone and navigate according to simulated sensor responses. The environment is a perfect playground for future research.

The goal of this thesis was merely a proof of concept, thus there are many open issues to go forward. I am convinced, that sequential problem solving is a powerful concept for a variety of problems. It is a cognitive process that we humans use to solve the most complicated situation. And in the end, those human inspired methods are what is largely responsible for the AI boom in recent history.

Bibliography

1. G. Pajares. Overview and current status of remote sensing applications based on unmanned aerial vehicles (uavs). In *Photogrammetric Engineering & Remote Sensing*, 81(4), pp. 281–330, 2015.

2. A. Capolupo, S. Pindozzi, C. Okello, N. Fiorentino, and L. Boccia. Photogrammetry for environmental monitoring: The use of drones and hydrological models for detection of soil contaminated by copper. In *Science of the Total Environment*, 514, pp. 298–306, 2015.

3. P. Tatham. An investigation into the suitability of the use of unmanned aerial vehicle systems (uavs) to support the initial needs assessment process in rapid onset humanitarian disasters. In *International journal of risk assessment and management*, 13(1), pp. 60–78, 2009.

4. N.J. Cooke. Human factors of remotely operated vehicles. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 50, pp. 166–169. Sage Publications Sage CA: Los Angeles, CA, 2006.

5. D.K. Kim and T. Chen. Deep neural network for real-time autonomous indoor navigation. In *CoRR*, abs/1511.04668, 2015. URL `http://arxiv.org/abs/1511.04668`.

6. N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In *CoRR*, abs/1705.02550, 2017. URL `http://arxiv.org/abs/1705.02550`.

7. J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama et al.. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4. 2017.

8. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788. 2016.

9. Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun. Light-head r-cnn: In defense of two-stage object detector. In *arXiv preprint arXiv:1711.07264*, 2017.

10. X. Zhu, J. Dai, X. Zhu, Y. Wei, and L. Yuan. Towards high performance video object detection for mobiles. In *CoRR*, abs/1804.05830, 2018. URL `http://arxiv.org/abs/1804.05830`.

11. L. Wiskott and T.J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. In *Neural computation*, 14(4), pp. 715–770, 2002.

12. D. Jayaraman and K. Grauman. Slow and steady feature analysis: higher order temporal coherence in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3852–3861. 2016.

13. X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei. Deep feature flow for video recognition. In *CVPR*, volume 1, p. 3. 2017.

14. A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105. 2012.

15. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9. 2015.

16. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778. 2016.

17. S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99. 2015.

18. J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *CoRR*, abs/1612.08242, 2016. URL `http://arxiv.org/abs/1612.08242`.

19. A. Wong, M.J. Shafiee, F. Li, and B. Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. In *CoRR*, abs/1802.06488, 2018. URL `http://arxiv.org/abs/1802.06488`.

20. J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

21. T. Lin, M. Maire, S.J. Belongie, L.D. Bourdev, R.B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. Microsoft COCO: common objects in context. In *CoRR*, abs/1405.0312, 2014. URL `http://arxiv.org/abs/1405.0312`.

22. M. Everingham, S.M.A. Eslami, L. Van Gool, C.K.I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. In *International Journal of Computer Vision*, 111(1), pp. 98–136, January 2015.

23. S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural computation*, 9(8), pp. 1735–1780, 1997.

24. P. Zhu, L. Wen, X. Bian, L. Haibin, and Q. Hu. Vision meets drones: A challenge. In *arXiv preprint arXiv:1804.07437*, 2018.

25. D. Gonzalez-Aguilera and P. Rodriguez-Gonzalvez. Drones—an open access journal. 2017.

26. I. Colomina and P. Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review. In *ISPRS Journal of Photogrammetry and Remote Sensing*, 92, pp. 79 – 97, 2014. ISSN 0924-2716. doi:https://doi.org/10.1016/j.isprsjprs.2014.02.013. URL `http://www.sciencedirect.com/science/article/pii/S0924271614000501`.

27. A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. In *arXiv preprint arXiv:1605.07678*, 2016.

28. J. Everaerts et al.. The use of unmanned aerial vehicles (uavs) for remote sensing and mapping. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37(2008), pp. 1187–1192, 2008.

29. A. Hodgson, N. Kelly, and D. Peel. Unmanned aerial vehicles (uavs) for surveying marine fauna: a dugong case study. In *PloS one*, 8(11), p. e79556, 2013.

30. B. Custers. *Future of Drone Use*. Springer, 2016.

31. Y. Zeng, R. Zhang, and T.J. Lim. Wireless communications with unmanned aerial vehicles: opportunities and challenges. In *arXiv preprint arXiv:1602.03602*, 2016.

32. R. Siegwart, I.R. Nourbakhsh, D. Scaramuzza, and R.C. Arkin. *Introduction to autonomous mobile robots*. MIT press, 2011.

33. A.M. Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pp. 23–65. Springer, 2009.

34. I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

35. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. In *nature*, 521(7553), p. 436, 2015.

36. A.W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. In *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (12), pp. 1349–1380, 2000.

37. R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

38. A. Pentland, B. Moghaddam, T. Starner et al.. In View-based and modular eigenspaces for face recognition, 1994.

39. D.G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pp. 1150–1157. Ieee, 1999.

40. H. Schneiderman and T. Kanade. Object detection using the statistics of parts. In *International Journal of Computer Vision*, 56(3), pp. 151–177, 2004.

41. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pp. 886–893. IEEE, 2005.

42. F. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pp. 829–836. IEEE, 2005.

43. S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pp. 1030–1037. IEEE, 2010.

44. P. Viola, M.J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *null*, p. 734. IEEE, 2003.

45. O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. In *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10), pp. 1713–1727, 2008.

46. W. Pitts and W.S. McCulloch. How we know universals the perception of auditory and visual forms. In *The Bulletin of mathematical biophysics*, 9(3), pp. 127–147, 1947.

47. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning internal representations by error propagation*. Technical Report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

48. G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. In *science*, 313(5786), pp. 504–507, 2006.

49. G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath et al.. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. In *IEEE Signal processing magazine*, 29(6), pp. 82–97, 2012.

50. Y. Le Cun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. In *IEEE Communications Magazine*, 27(11), pp. 41–46, 1989.

51. X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 3. 2017.

52. A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758–2766. 2015.

53. K. Kang, W. Ouyang, H. Li, and X. Wang. Object detection from video tubelets with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 817–825. 2016.

54. K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang et al.. T-cnn: Tubelets with convolutional neural networks for object detection from videos. In *IEEE Transactions on Circuits and Systems for Video Technology*, 2017.

55. C. Feichtenhofer, A. Pinz, and A. Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3038–3046. 2017.

56. W. Han, P. Khorrami, T.L. Paine, P. Ramachandran, M. Babaeizadeh, H. Shi, J. Li, S. Yan, and T.S. Huang. Seq-nms for video object detection. In *arXiv preprint arXiv:1602.08465*, 2016.

57. K. Chen, J. Wang, S. Yang, X. Zhang, Y. Xiong, C.C. Loy, and D. Lin. Optimizing video object detection via a scale-time lattice. In *arXiv preprint arXiv:1804.05472*, 2018.

58. Z. Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *arXiv preprint arXiv:1712.00726*, 2017.

59. T.Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *IEEE transactions on pattern analysis and machine intelligence*, 2018.

60. Z. Meng, X. Fan, X. Chen, M. Chen, and Y. Tong. Detecting small signs from large images. In *Information Reuse and Integration (IRI), 2017 IEEE International Conference on*, pp. 217–224. IEEE, 2017.

61. B. Alexe, N. Heess, Y.W. Teh, and V. Ferrari. Searching for objects driven by context. In *Advances in Neural Information Processing Systems*, pp. 881–889. 2012.

62. P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. In *IEEE transactions on pattern analysis and machine intelligence*, 32(9), pp. 1627–1645, 2010.

63. Y. Zhang, K. Sohn, R. Villegas, G. Pan, and H. Lee. Improving object detection with deep convolutional networks via bayesian optimization and structured prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 249–258. 2015.

64. Y. Lu, T. Javidi, and S. Lazebnik. Adaptive object detection using adjacency and zoom prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2351–2359. 2016.

65. V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski et al.. Human-level control through deep reinforcement learning. In *Nature*, 518(7540), p. 529, 2015.

66. J.C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2488–2496. 2015.

67. M. Bellver, X. Giró-i Nieto, F. Marqués, and J. Torres. Hierarchical object detection with deep reinforcement learning. In *arXiv preprint arXiv:1611.03718*, 2016.

68. Z. Jie, X. Liang, J. Feng, X. Jin, W. Lu, and S. Yan. Tree-structured reinforcement learning for sequential object localization. In *Advances in Neural Information Processing Systems*, pp. 127–135. 2016.

69. M. Gao, R. Yu, A. Li, V.I. Morariu, and L.S. Davis. Dynamic zoom-in network for fast object detection in large images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

70. A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. In *Acm computing surveys (CSUR)*, 38(4), p. 13, 2006.

71. M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin. A novel performance evaluation methodology for single-target trackers. In *IEEE Transactions on Pattern Analysis and*

Machine Intelligence, 38(11), pp. 2137–2155, Nov 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2016.2516982.

72. F. LIRIS. In The visual object tracking vot2014 challenge results.

73. A. Lukezic, T. Vojir, L.C. Zajc, J. Matas, and M. Kristan. Discriminative correlation filter with channel and spatial reliability. In *CVPR*, volume 6, p. 8. 2017.

74. J.F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3), pp. 583–596, 2015.

75. A. Mahalanobis, B.V. Kumar, and D. Casasent. Minimum average correlation energy filters. In *Applied Optics*, 26(17), pp. 3633–3640, 1987.

76. D.S. Bolme, J.R. Beveridge, B.A. Draper, and Y.M. Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2544–2550. IEEE, 2010.

77. Y. Katznelson. *An introduction to harmonic analysis*. Cambridge University Press, 2004.

78. J.F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European conference on computer vision*, pp. 702–715. Springer, 2012.

79. R. Rifkin, G. Yeo, T. Poggio et al.. Regularized least-squares classification. In *Nato Science Series Sub Series III Computer and Systems Sciences*, 190, pp. 131–154, 2003.

80. D. Casasent, G. Ravichandran, and S. Bollapragada. Gaussian–minimum average correlation energy filters. In *Applied Optics*, 30(35), pp. 5176–5181, 1991.

81. D.G. Lowe. Distinctive image features from scale-invariant keypoints. In *International journal of computer vision*, 60(2), pp. 91–110, 2004.

82. H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pp. 404–417. Springer, 2006.

83. J. Van De Weijer, C. Schmid, J. Verbeek, and D. Larlus. Learning color names for real-world applications. In *IEEE Transactions on Image Processing*, 18(7), pp. 1512–1523, 2009.

84. S. Agarwal, J.O.D. Terrail, and F. Jurie. Recent advances in object detection in the age of deep convolutional neural networks. In *arXiv preprint arXiv:1809.03193*, 2018.

85. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587. 2014.

86. R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448. 2015.

87. J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pp. 379–387. 2016.

88. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, and A.C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

89. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv preprint arXiv:1409.1556*, 2014.

90. A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861*, 2017.

91. D.H. Hubel and T.N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. In *The Journal of physiology*, 148(3), pp. 574–591, 1959.

92. D.H. Hubel and T.N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. In *The Journal of physiology*, 195(1), pp. 215–243, 1968.

93. Z.Q. Zhao, P. Zheng, S.t. Xu, and X. Wu. Object detection with deep learning: A review. In *arXiv preprint arXiv:1807.05511*, 2018.

94. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826. 2016.

95. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520. 2018.

96. R.S. Sutton and A.G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

97. V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. 2016.

98. D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*. 2014.

99. G. Barth-Maron, M.W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap. Distributed distributional deterministic policy gradients. In *arXiv preprint arXiv:1804.08617*, 2018.

100. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

101. R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine learning*, 8(3-4), pp. 229–256, 1992.

102. J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *arXiv preprint arXiv:1506.02438*, 2015.

103. R.J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. In *Connection Science*, 3(3), pp. 241–268, 1991.

104. M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *CoRR, abs/1507.06527*, 7(1), 2015.

105. G. Lample and D.S. Chaplot. Playing fps games with deep reinforcement learning. In *AAAI*, pp. 2140–2146. 2017.

106. A. Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

107. R. Caruana. Multitask learning. In *Machine learning*, 28(1), pp. 41–75, 1997.

108. A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In *European conference on computer vision*, pp. 549–565. Springer, 2016.

109. M. Bonetto, P. Korshunov, G. Ramponi, and T. Ebrahimi. Privacy in mini-drone based video surveillance. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*, volume 4, pp. 1–6. IEEE, 2015.

110. E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In *IEEE International Symposium on Geoscience and Remote Sensing (IGARSS)*. 2017.

111. M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M.A. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. In *CoRR, abs/1707.08817*, 2017.

112. L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. In *arXiv preprint arXiv:1712.04621*, 2017.

113. M.G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. In *Journal of Artificial Intelligence Research*, 47, pp. 253–279, 2013.

114. J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning. In *arXiv preprint arXiv:1704.06440*, 2017.

115. A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137. 2015.

116. J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. In *arXiv preprint arXiv:1412.7755*, 2014.

117. K. Gregor, I. Danihelka, A. Graves, D.J. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *arXiv preprint arXiv:1502.04623*, 2015.

118. S. Gruber, H. Kwon, C. Hager, R. Sharma, J. Yoder, and D. Pack. *Handbook of Unmanned Aerial Vehicles*. Springer, 2015.

119. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 86(11), pp. 2278–2324, 1998.

120. W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 4898–4906. 2016.

121. M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*. 2012.

122. A. Graves, N. Jaitly, and A.r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pp. 273–278. IEEE, 2013.

Appendix

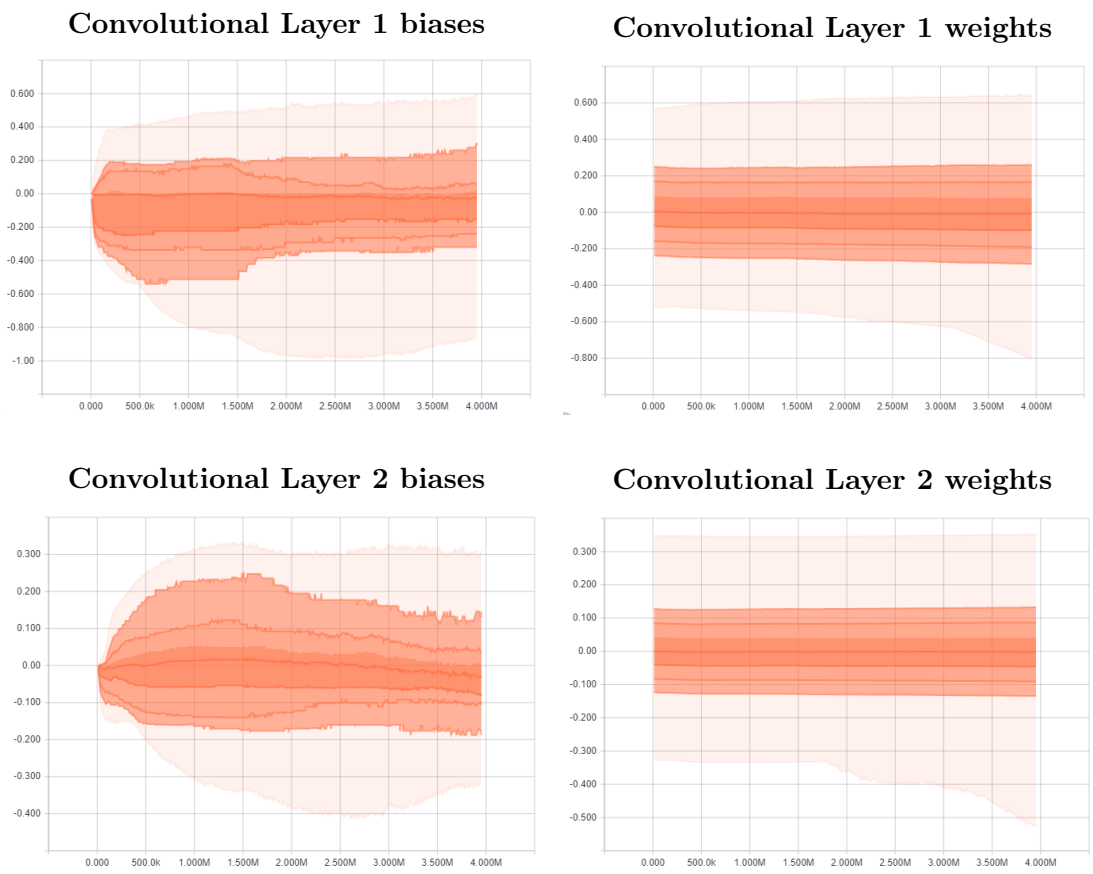


Figure 1: Bias and weight distribution of convolutional layer 1 and 2.

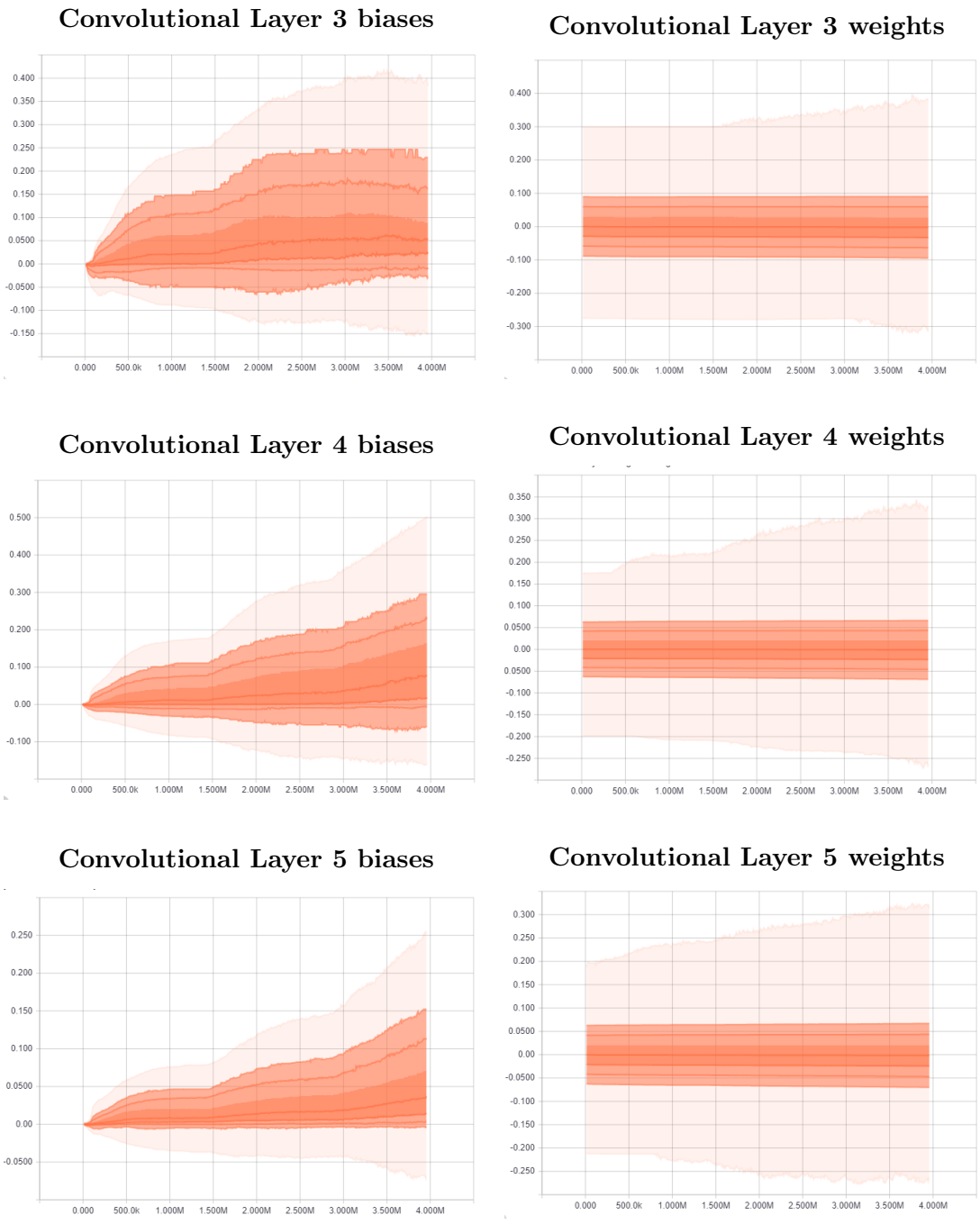


Figure 2: Bias and weight distribution of convolutional layer 3, 4 and 5.

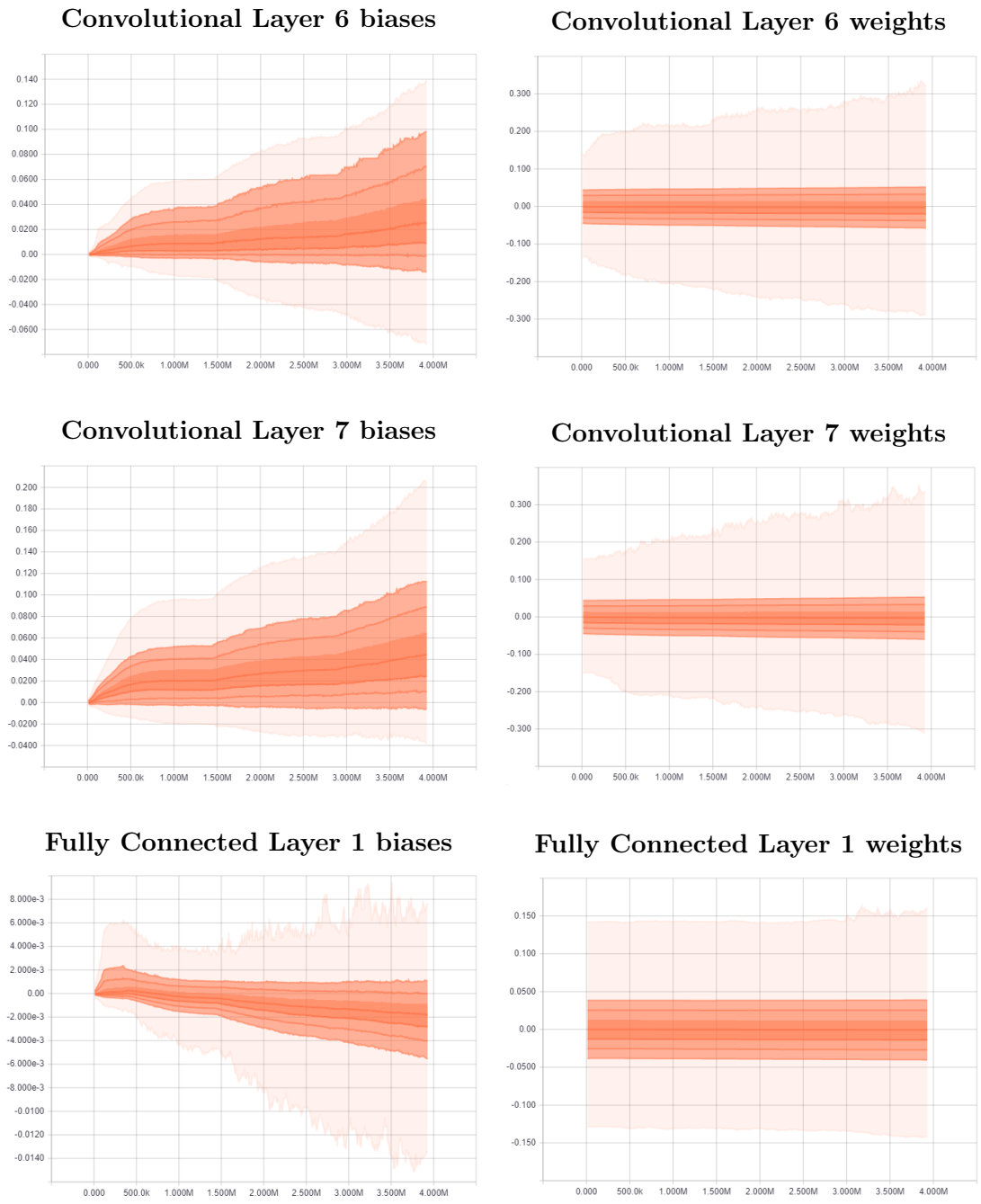


Figure 3: Bias and weight distribution of convolutional layer 6 and 7 the fully connected layer 1.

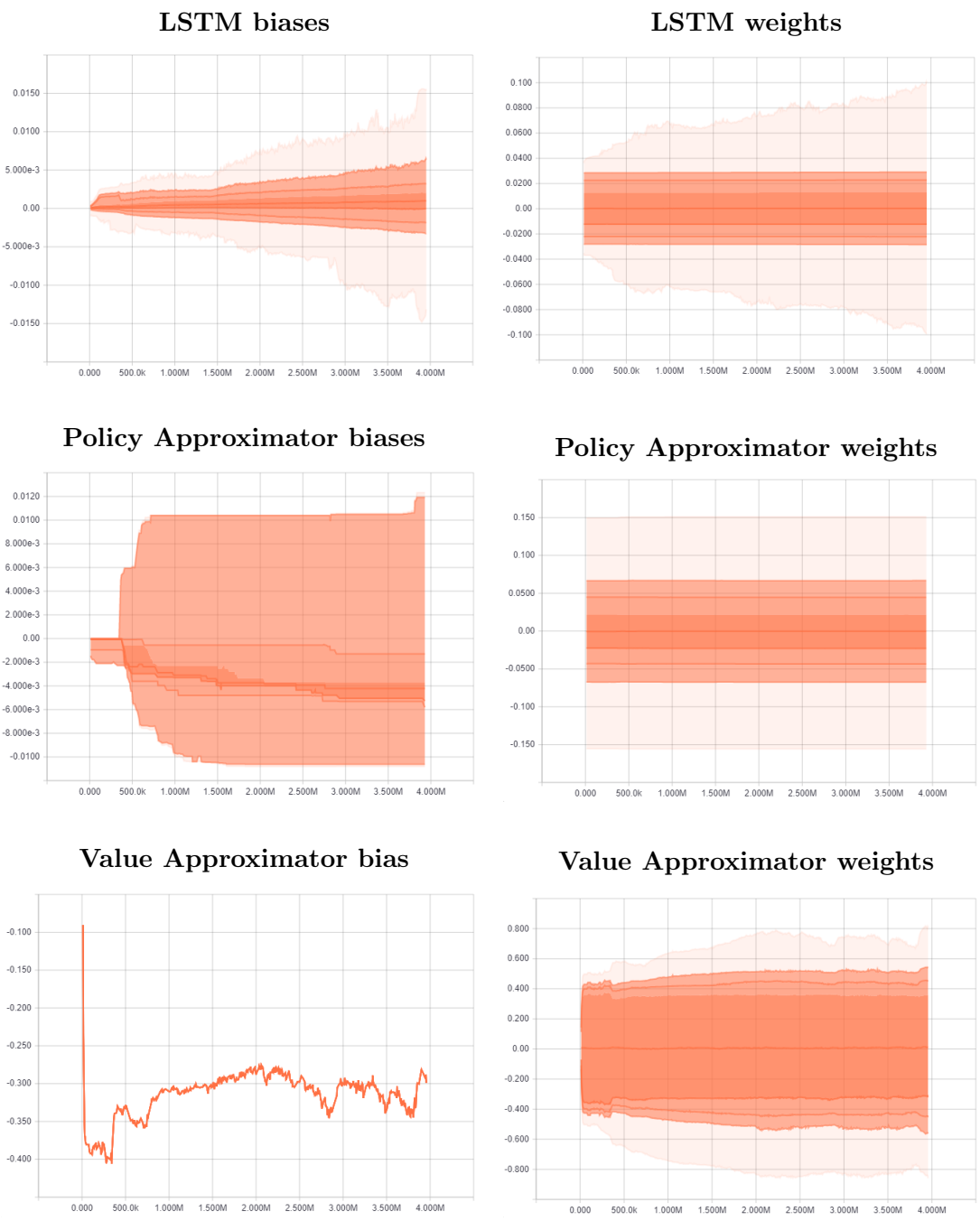


Figure 4: Bias and weight distribution of the LSTM, the policy and value approximator

Figure 5: Bias and weight distributions of the extra regression layer 2, the object and tracked object count layers